DESIGN OF A DIGIT-SLICE ON-LINE ARITHMETIC UNIT*

A. Gorji-Sinaki and M. D. Ercegovac

UCLA Computer Science Department
University of California
Los Angeles, California 90024, USA

ABSTRACT -- A gate level design of a digit-slice on-line arithmetic unit is presented. This unit is designed as a set of basic modules, Processing Elements (PE), each of which operates on a single digit of the operands and the results. It is capable of executing four basic operations of addition/subtraction, multiplication and division in an on-line manner. The results are generated during the digit-serial input of the operands, beginning always with the most significant digit. A general (with respect to radix) analysis of the cost and speed of the proposed unit is also given.

1. INTRODUCTION

The subject of this paper is the hardware design of a highly modular on-line arithmetic unit for the four operations of addition/subtraction, multiplication and division. The goal is to investigate the gate complexity of an on-line unit and show its flexibility for VLSI implementation. The logic design of the on-line unit is based on work of Goyal [GOY 76].

We review briefly the basic definitions and concepts of on-line arithmetic. An on-line algorithm generates the j-th leftmost digit of the result after receiving the $(j+\delta)$-th input digit. Therefore, an on-line algorithm is always performed in a digit-serial manner, from left to right. In order to generate the first digit of the result, the inputs must be known to $\delta+1$ digits of precision. Thereafter, an output digit can be obtained for each additional input digit. The *on-line* delay δ is a small integer, typically 1 to 4, depending on the operation and the range of the arguments [TRI 77, GOR 80a].

On-line arithmetic provides a cost-effective approach to achieve higher computational rates by allowing overlap at the digit level between the successive operations [ERC 75, TRI 77]. In particular, on-line arithmetic is highly attractive in high speed multimodule structures for parallel and pipelined computations. Several on-line algorithms have been developed and used in iterative structures for array computations [ERC 80a, GRN 80, ERC 80b]. Typical problems, such as matrix-vector multiplication and solving linear recurrence systems, have been investigated and corresponding solutions using on-line approaches are proposed and evaluated. The main result indicates that the on-line approach offers a speed-up factor of 2-16 with respect to conventional arithmetic while preserving limited interconnection bandwidth, decentralized control and uniform structure. It is especially important that the on-line approach offers a straightforward speed improvement in solving hard problems, such as non-linear recurrences [ERC 80b]. Applications of fault-tolerance techniques to on-line arithmetic have also been considered and the feasibility of low-cost error detection and correction for on-line algorithms has been demonstrated in [GOR 80b].

* Supported in part by the Office of Naval Research Contract No. N00014-79-C-0866.

2. ON-LINE ARITHMETIC ALGORITHMS

In general, an on-line algorithm is specified recursively in terms of on-line representation of operands, results and some internal values. The recursion is usually of the following form:

$$P_j = f(P_{j-1}, x_{j+\delta}, y_{j+\delta}, z_j)$$

where f is a linear function and P_j is the partial internal result. The output digit is obtained by applying a selection function on the truncated version of the partial result $\hat{P}_j$ and the current inputs.

$$z_{j+1} = SELECT(\hat{P}_j, x_{j+\delta}, y_{j+\delta})$$

In order to be able to perform such a selection, it is necessary to use a redundant number system. We assume that all the operands and the results are represented in a symmetric signed-digit number system [AVI 61]. The totally parallel addition/subtraction [AVI 61] can be easily performed in an on-line manner with $\delta=1$. On-line algorithms for multiplication and division were introduced in [ERC 75, TRI 77]. Also, systematic methods for derivation of on-line addition/subtraction, multiplication and, division algorithms appears in [GOR 80a] and for division in [TRI 78].

It has been proved that an on-line division unit is capable of performing on-line addition and multiplication with minor modifications and actually with no increase in hardware [GOR 80c]. Therefore, in the remaining parts of this paper we focus our attention on the design of a digit-slice division unit, assuming that the same unit can also perform addition and multiplication. The design is based on the following on-line division algorithm [TRI 77]:

Step 1 [*Initialization*]:

$$P_0 = \sum_{i=1}^{\delta} n_i r^{-i} \qquad D_0 = \sum_{i=1}^{\delta} d_i r^{-i} \qquad Q_0 = 0$$

Step 2 [*Recursion*]:

for j = 1,2,....,m do:

Step 2.1 [*Selection*]:

$$q_j = SELECT(r\hat{P}_{j-1}, \hat{D}_{j-1})$$

$$Q_j = Q_{j-1} + q_j r^{-j}$$

Step 2.2 [*Input Digits*]:

$$D_j = D_{j-1} + d_{j+\delta} r^{-j-\delta}$$

Step 2.3 [*Basic Recursion*]:

$$P_j = rP_{j-1} - q_j D_j + n_{j+\delta} r^{-\delta} - Q_{j-1} d_{j+\delta} r^{-\delta} \qquad (1)$$

Step 3 [*End*]

In the above algorithm N, D, and Q are assumed to be the dividend, divisor, and the quotient, respectively.

3. ON-LINE DIVISION UNIT

We assume that the on-line unit consists of a linear cascade of identical Processing Elements (PEs). Each PE is a relatively complex logical module capable of performing on-line operation under the control of the Global Control Unit (GCU). Figure 1 shows the schematic organization of the on-line division unit along with the GCU. The module EU performs the exponent calculations. END UNIT allows the last PE to be identical to all the other PEs as far as interface is concerned.

The PEs collectively contain the fractional parts of all active operands, one digit in each PE. Most significant digits are in PE_1 and least significant digits in PE_n. Output digits are generated by PE_1 in an on-line mode and are placed on the Z-Bus. Each output digit is temporarily stored by all PE's.

After receiving the output digit and the transfer information from the right-hand neighbor, each PE starts the computation and generates one digit of the partial remainder. Depending on this partial remainder and the truncated version of the divisor, the next quotient digit is selected by PE_1 and it is placed on the output bus. This operation continues until the required precision is obtained.

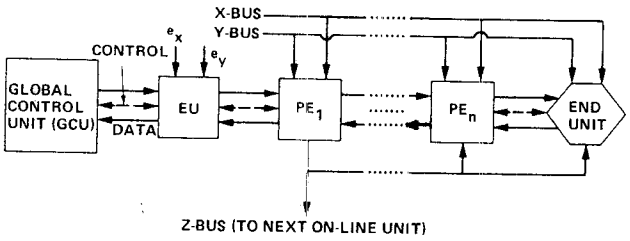


Figure 1 -- Organization of an On-Line Division Unit

In order to determine the operation of each PE, we use the basic recursion formula for on-line division (Eq. 1). Let:

$$P_j=\sum_{i=1}^{m} p_i^j r^{-i} \tag{3}$$

$$D_j=\sum_{i=1}^{j+\delta} d_i r^{-i} \tag{4}$$

where, for each PE_i, p_i^j is the i-th digit of the j-th partial remainder and n_i , d_i are the i-th digit of the operands.

The digit recursion, performed by each PE_i, is defined as:

$$p_i^{(j)}=p_{i+1}^{(j-1)}-q_j d_i+n_{j+\delta}[i=\delta]-q_{i+1-\delta}d_{j+\delta}+T_i^{(j)}-rT_{i-1}^{(j)} \tag{5}$$

where $n_{j+\delta}[i=\delta]$ means $n_{j+\delta}$ will be used only in PE_δ. $T_i^{(j)}$ is the transfer digit from PE_{i+1} at the j-th step $T_{i-1}^{(j)}$ is the transfer digit to PE_{i-1} at the j-th step

It is obvious that $q_{i+1-\delta}$ is zero in PE_i for $\delta \leqslant i \leqslant j-2+\delta$. Using Eq. (5) Figure 2 for the on-line divide unit is obtained.

In order to eliminate carry propagation between the PEs, we assume that each digit of the partial remainder ($p_i^{(j)}$) in PE_i is represented by an interim partial remainder ($w_i^{(j)}$) and a transfer digit ($T_i^{(j)}$) such that:

$$p_i^{(j)}=w_i^{(j)}+T_i^{(j)} \tag{6}$$

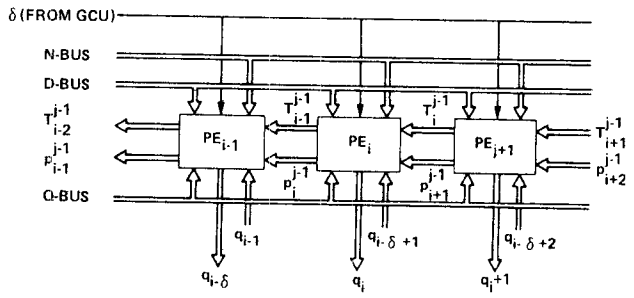


Figure 2 -- Interconnection Between Processing Elements

The transfer function in Eq. (5) is obtained by a series of three transformations f_1 , f_2 and f_3 such that:

$$\begin{cases} f_1 : -q_{i+1-\delta} \cdot d_{j+\delta}=rt_{i-1}^{p_1(j)}+w_i^{1(j)} \\ f_2 : -q_j \cdot d_i=rt_{i-1}^{p_2(j)}+w_i^{2(j)} \end{cases} \tag{7}$$

The transfer digits from PE_i to PE_{i-1} are $t_{i-1}^{p_1(j)}$ and $t_{i-1}^{p_2(j)}$ resulting from transformations f_1 and f_2, respectively. Also there is a transfer digit out of the Multi-Input Adder ($t_{i-1}^{A(j)}$). Therefore:

$$T_{i-1}^{(j)}=t_{i-1}^{p_1(j)}+t_{i-1}^{p_2(j)}+t_{i-1}^{A(j)} \tag{8}$$

substituting (6), (7) and (8) in (5) we get:

$$f_3 : \begin{cases} w_i^{(j)}=w_{i+1}^{(j-1)}+T_{i+1}^{(j-1)}+w_i^{2(j)}+w_i^{1(j)} \\ \qquad +n_{j+\delta}[i=\delta]-rt_{i-1}^{A(j)} \\ T_i^{(j)}=t_i^{p_1(j)}+t_i^{p_2(j)}+t_i^{A(j)} \\ p_i^{(j)}=w_i^{(j)}+T_i^{(j)}=w_i^{(j)}+t_i^{p_1(j)}+t_i^{p_2(j)}+t_i^{A(j)} \end{cases} \tag{9}$$

A block diagram of transformations f_1 , f_2 and f_3 is shown in Figure 3.

Transformation f_3 essentially requires a radix-r multi-input adder which forms the sum of the digits of both signs. This adder is implemented as a k-stage ($r=2^k$) linear cascade of radix-2 multi-input adders where each input of a radix-2 adder can assume three values {-1,0,1}. The organization of this adder is shown in Figure 4 (k=4).

The products $q_j{}^*d_i$ and $q_{i+1-\delta}{}^*d_{j+\delta}$ are generated by two separate product matrix generators which consist of a k*k square array of redundant binary product cells. Each cell performs the product of two redundant binary digits $q_{j_i}^*$ and $d_{i_m}^*$ and its output product digit is also in the digit set {-1,0,1}. Figure 5 shows the operation of the digit product generators f_1 and f_2 (k=4).

Therefore, transformation f_3 requires k MIRBAs (Multi-Input Redundant Binary Adder), each capable of summing 2(k+1) redundant binary inputs, as well as the 'Transfer' from the adjacent MIRBA position [GOY 76]. Figure 6 schematically shows the implementation of f_3 for radix 16, i.e., k=4.

Design of a MIRBA

MIRBA is a limited carry/borrow propagation adder which

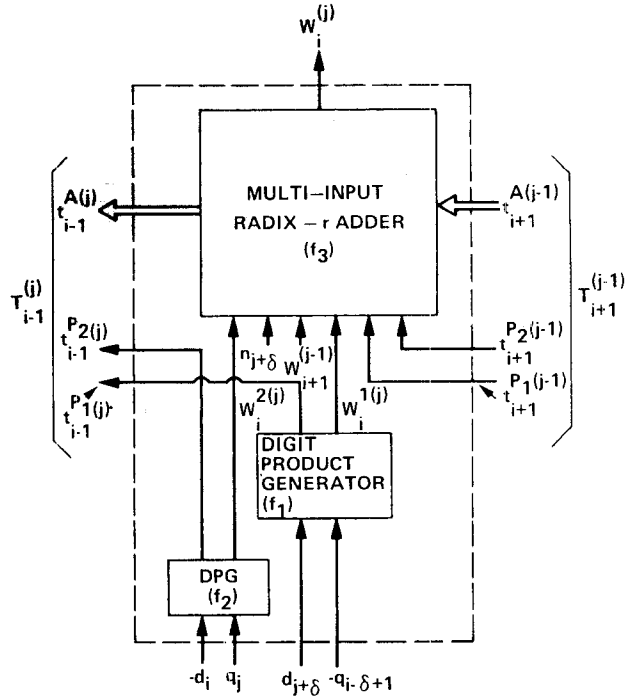


Figure 3 -- Functional Representation of the Digit Algorithm for On-Line Division

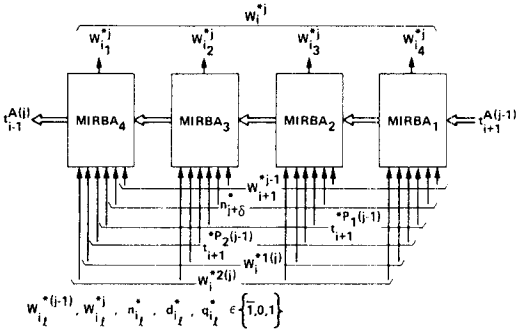


Figure 4 -- Functional Representation of Transformation f^3

accepts several redundant binary inputs (digit set {-1,0,1}) and produces one redundant binary output (with appropriate adder Transfers for more significant adjacent adder stages).

Using Rohatsch's technique [ROH 67], a 10 input MIRBA can be realized with four simple transformations [GOR 80c]. Another way of implementing MIRBA's is the log-sum tree technique. In this scheme each MIRBA can be implemented by a log-sum tree structure of two input redundant binary adders (Borovec Unit (BU) [BOR 68]). For a 2(k+1) input MIRBA, the tree structure has L levels of Borovec Units (BU) such that:

$$L=\left\lceil \log_2 2(k+1) \right\rceil \qquad (10)$$

and the number of BU's required is (2k+1). Figure 7 shows the log-sum tree structure for a 10 input MIRBA.

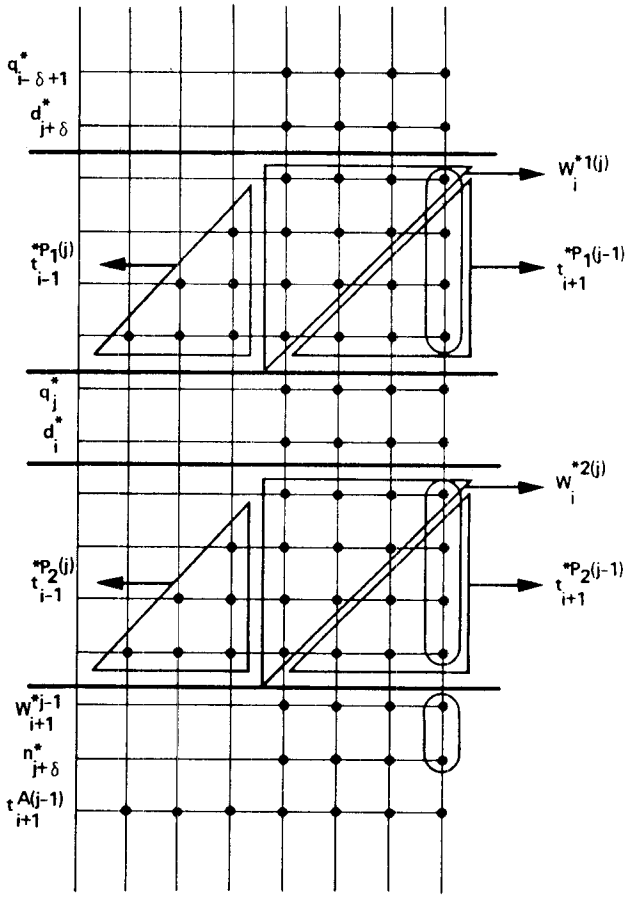


Figure 5 -- Illustration of Adjacent Overlapping Product Matrices

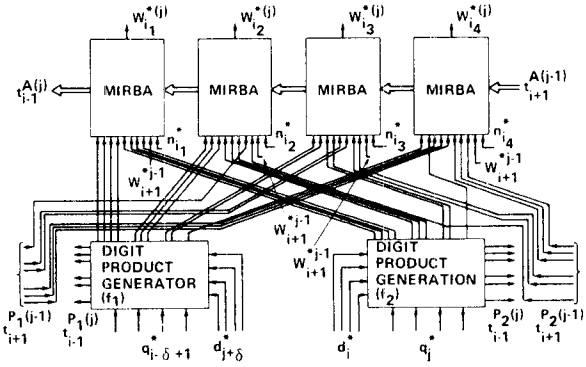


Figure 6 -- Illustration of the Implementation of the Digit Algorithm, Using Redundant Binary Product Matrix Generator (Radix = 16)

Logic Design of The Processing Element

The major components of the PE are the Register File (RF) for the storage of active operands, the Digit Processing Logic (DPL) which is essentially a large combinational network and the Local Control Unit (LCU) which supplies the control signals in proper order to the DPL. Figure 8 shows the schematic block diagram of a Processing Element. The RF comprises a set of digit-wide registers which are used to hold the operand digits and the result digits.

The DPL operates on the operand digits stored in the register file of the PE and the information received from its right neighboring PE. It also generates Transfer information for its left neighbor PE. The LCU issues the timing control signals to the processing logic for sequencing the various steps of the digit algorithm.

The register file is a set of registers that are used to hold the operands and result digits. Each PE retains one digit of each of the active operands. Each register is (k+1) bits long to hold the k-magnitude bits and one sign bit of one sign and magnitude encoded radix-2^k digit.

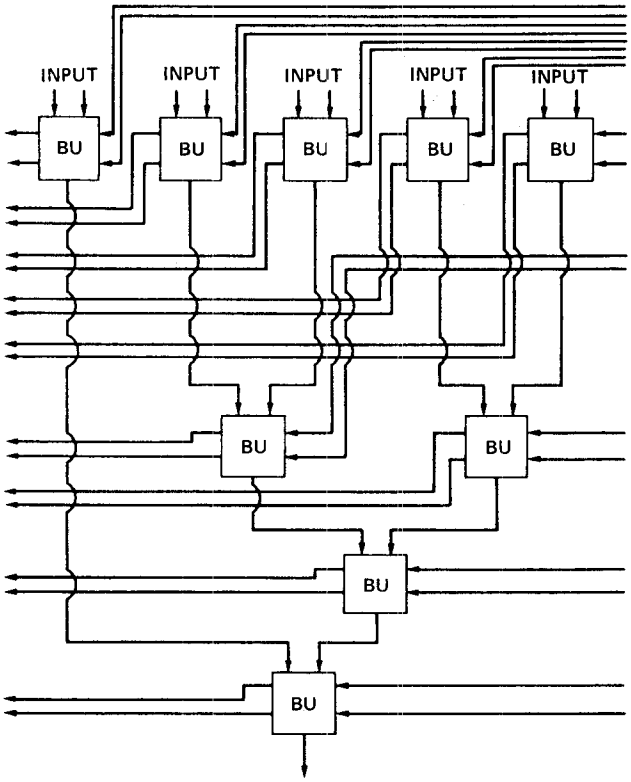


Figure 7 -- Illustration of Log-Sum Tree Structure for a MIRBA Using Borovec Units Only (k = 4)

There must be at least seven registers in a PE. One for the dividend, one for divisor, one for quotient digit and one for interim partial remainder ($w_i^{(j)}$). Three other registers are used to hold the transfer functions ($T_i^{(j)}$) coming from PE_{i+1}. In the next step of the computation (j+1) these functions are gated to PE_{i-1} along with $w_i^{(j)}$. They constitute the operands of PE_{i-1} in step j+1.

The registers in the RF are loaded from a buffer register, IBR whose contents are determined by the internal Register Input Bus Selector, SRIB in the Digit Processing Logic. Similarly, the contents of the registers are inputed to the DPL either directly or through an Output Bus Selector SROB, also in the DPL.

Block Diagram Description of DPL

Figure 9 shows the data flow structure of the Digit Processing Logic (DPL) in a block diagram form. It consists of three major components: the Digit Product Generator, (DPG), a radix-2^k multi-input adder (MIAD), and a Digit Sum Encoder (DSE). DSE converts the redundant binary sum output of adder MIAD to the Sign and Magnitude format for local storage in the Register File, or transfer out of the PE.

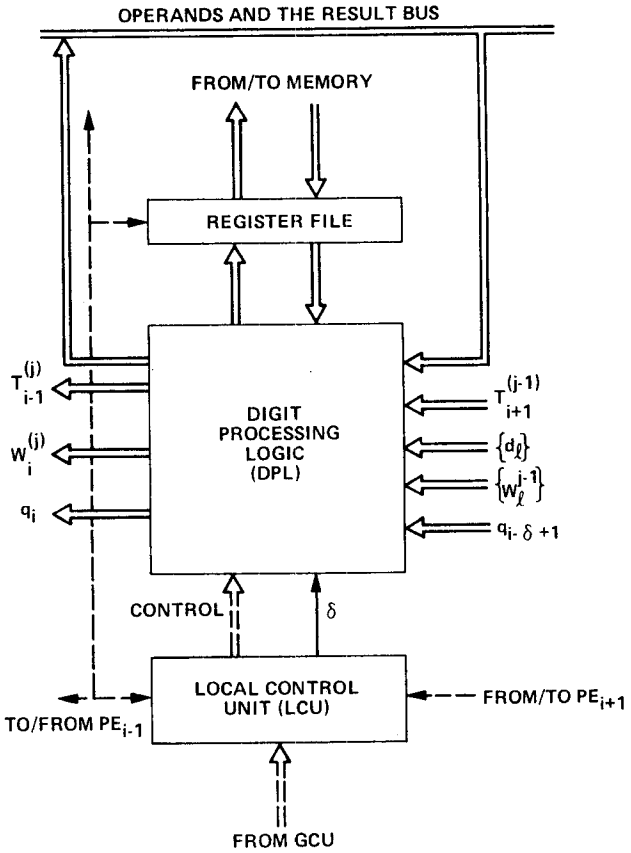


Figure 8 -- Block Diagram of a Processing Element

As shown in Figure 9, there are input and output ports designated as TIP_i , RIP_i and TOP_i, ROP_i, respectively. The input port TIP_i carries the 'Transfer' (carry or borrow) from adjacent MIAD and the contents of some register in the Register File of the adjacent PE_{i+1}. RIP_i carries the quotient digit from $PE_{i+1-\delta}$. The output ports TOP_i and ROP_i carry similar information for PE_{i-1} and $PE_{i+1-\delta}$ respectively.

Logic Design of a Multi-Input Adder (MIAD)

In general, a radix-2^k multi-input adder consists of a linear cascade of k MIRBAs. A 2(k+1) input MIRBA is implemented as a tree structure of BUs (see Fig. 7). Each MIRBA requires 2k+1 BUs that are arranged in $L=\lceil \log_2 2(k+1) \rceil$ levels. Therefore:

$$G_{MIAD}=k(2k+1)G_{BU} = 26k(2k+1) \quad (11)$$

$$t_{MIAD}=L^*\delta_{BU} \quad (12)$$

There must be at least seven registers in a PE. One for the dividend, one for divisor, one for quotient digit and one for interim partial remainder ($w_i^{(j)}$). Three other registers are used to hold the transfer functions ($T_i^{(j)}$) coming from PE_{i+1}. In the next step of the computation (j+1) these functions are gated to PE_{i-1} along with $w_i^{(j)}$. They constitute the operands of PE_{i-1} in step j+1.

There are other registers in a PE which are used to hold the intermediate results. These registers are located in DPL and will be shown later.

The registers in the RF are loaded from a buffer register, IBR whose contents are determined by the internal Register Input Bus Selector, SRIB in the Digit Processing Logic. Similarly, the contents of the registers are inputed to the DPL either directly or through an Output Bus Selector SROB, also in the DPL.

where G_{MIAD} is the number of gates required for one MIAD; t_{MIAD} is the delay of one MIAD; G_{BU} is the number of gates required for one BU and δ_{BU} is the delay of one BU.

For a 2(k+1) input adder, the number of pins required for the input and output adder transfers $t_{i+1}^{A(j-1)}$ and $t_{i-1}^{A(j)}$ are 2(2k+1) each (see Figure 7).

Logic Design of DPG

The Digit Product Generator forms the product array of two signed radix-2^k digits. It accepts the two digits encoded in Sign and Magnitude format and outputs the product array in redundant binary. The logical design of DPG is given in [GOR 80c]. The number of gates required for each DPG is k^2 AND GATES + 1 XOR [GOY 76].

The pins contributed by DPGs to the pin complexity of DPL are those pins which are required for $t_{i-1}^{P_1(j)}$, $t_{i-1}^{P_2(j)}$, $t_{i+1}^{P_1(j-1)}$ and $t_{i+1}^{P_2(j-1)}$. The number of pins required for a transfer signal is:

$$1+\frac{k(k-1)}{2} \qquad (13)$$

Logic Design of Digit Sum Encoder

The Digit Sum Encoder (DSE) transforms the redundant binary sum output of the radix-2^k adder into an algebraically equivalent radix-2^k sum digit in Sign and Magnitude format for either local storage in the Processing Element or transmission out of the PE. The total number of gates, G_{DSE} required by the DSE logic has been found to be [GOY 76]:

$$G_{DSE}=16k \qquad (14)$$

Logic Design of Selectors SRIB,SROB,STOP and STIP

The selector SRIB is a seven-input multiplexor. It constantly examines the data on D, Q and N Busses. If the data on any of these busses belong to PE_i, it writes this data in the corresponding registers in the register file. It also gates the output of DSE ($w_i^{(j)}$) to Register RW in the Register File. The transfer function $T_i^{(j)}$ ($=t_i^{A(j)}, t_i^{P_1(j)}, t_i^{P_2(j)}$) which should be sent to PE_{i-1} in (j+1)-th step is gated through this selector to Register File for temporary storage. The width of the selector is obtained by the following equation:

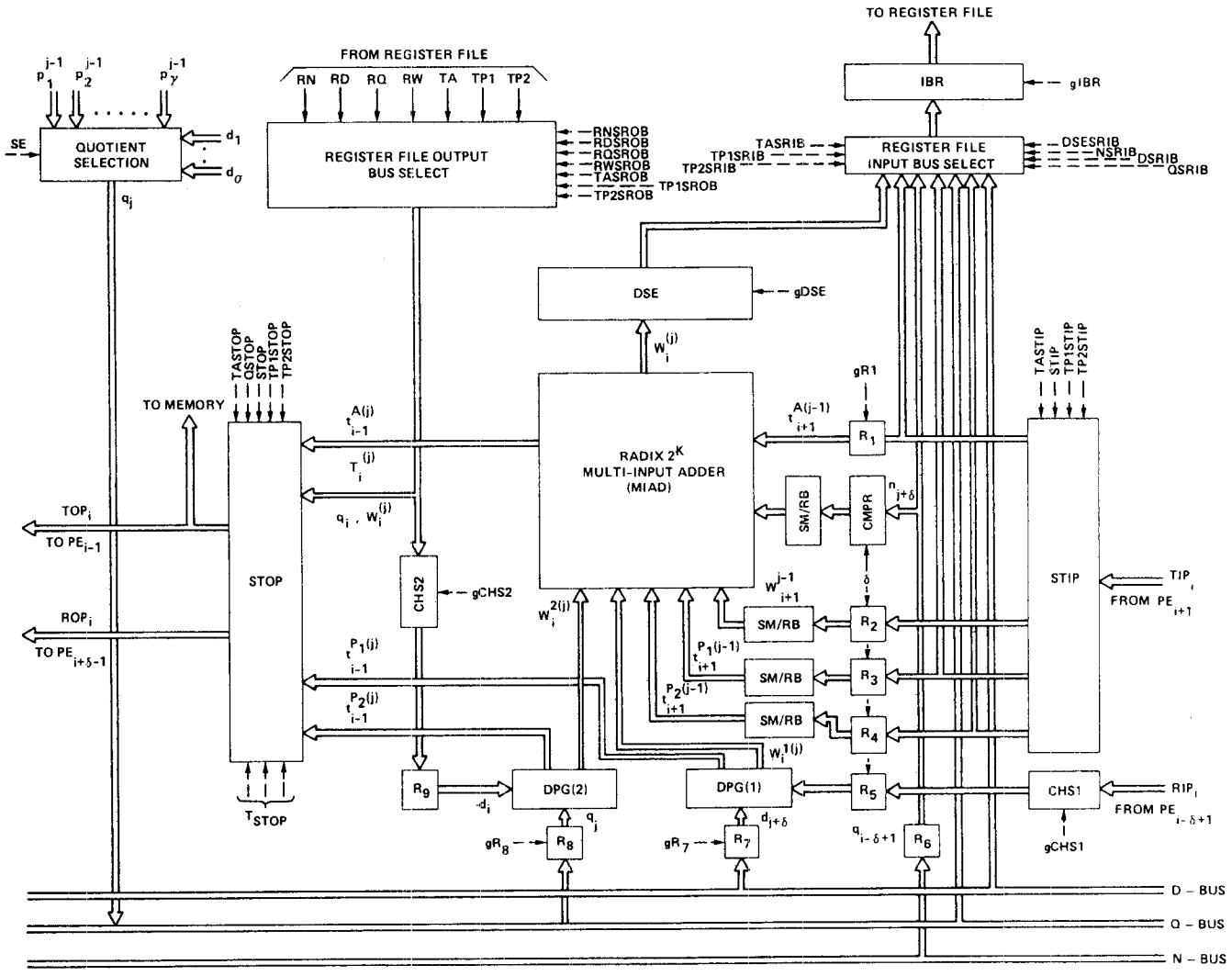


Figure 9 -- Block Diagram of Digit Processing Logic (DPL)

$$b=MAX\left[k+1,P_{t_i^{A(j)}},P_{t_i^{p_1(j)}},P_{t_i^{p_2(j)}}\right]$$

$P_{t_i^{A(j)}}$= *Pin Count of* $t_i^{A(j)}=2(2k+1)$

$P_{t_i^{p_1(j)}}$= *Pin Count of* $t_i^{p_1(j)}=1+\frac{k(k-1)}{2}$

$P_{t_i^{p_2(j)}}$= *Pin Count of* $t_i^{p_2(j)}=1+\frac{k(k-1)}{2}$

Therefore:

$$b=2(2k+1) \tag{15}$$

The logic design of SRIB is similar to that shown in [GOY 76] and the number of gates required is:

$$G_{SRIB}\approx b+b+2(1+\frac{k(k-1)}{2})+4(k+1)=k^2+11k+10 \tag{16}$$

The selector SROB selects the contents of one of the registers of the Register File on to the Register File Output Bus (ROB). The gates required for this network are dependent on the number of registers in the Register File and the bit width of the registers. There are seven registers in the Register File. For radix-2^k , four of them are $(k+1)$ bits wide, one is $2(2k+1)$ bits wide and the other two are $(1+\frac{k(k-1)}{2})$ bits wide. Therefore, the gate requirements of SROB are exactly the same as that of SRIB, that is:

$$G_{SROB}=k^2+11k+10 \tag{17}$$

The width of selector STOP is equal to the width of output port TOP_i. The width of TOP_i is determined by the maximum length of "Adder Transfers". Therefore, the width of TOP_i is given by Eq. (15). Logic implementation of STOP is shown in [GOR 80c]. From the given design we conclude that:

$$G_{STOP}=3b+2(1+\frac{k(k-1)}{2})=k^2+11k+8 \tag{18}$$

The selector STIP is actually a four-output demultiplexor. The width of STIP is exactly the same as that of STOP and is therefore equal to b. The logic implementation of STIP is simple and the number of gates required for this element is:

$$G_{STIP}=b+k+1+2(1+\frac{k(k-1)}{2})=k^2+4k+5 \tag{19}$$

Storage Buffer Registers of DPL

DPL has ten buffer registers, R_1 *through* R_9 and IBR. The width of each of these registers has been indicated in Table (1).

Register	Width
R_1	2(2k+1)
R_2	k+1
R_{3-4}	1+k(k-1)/2
R_{5-9}	k+1
IBR	2(2k+1)

Table (1)- Width of The Registers in DPL

Design of The Quotient Selection Unit

The selection of the quotient digits is done by the most significant Processing Element (PE_1). The quotient digit selector inside PE_1 is a table look-up device which implements the SELECT function. It examines the γ most significant digits of rP_{j-1} and σ most significant digits of D_{j-1}, in order to select the appropriate quotient digit, q_j.

According to Eq. (9) :

$$P_{j-1}=\sum_{i=1}^{m}p_i^{(j-1)}*r^{-i}=\sum_{i=1}^{m}[w_i^{(j-1)}+T_i^{(j-1)}]r^{-i} \tag{20}$$

Therefore, the truncated version of P_{j-1} (*i.e.*, $\hat{P}_{j-1}$) is:

$$\hat{P}_{j-1}=\sum_{i=1}^{\gamma}[w_i^{(j-1)}+T_i^{(j-1)}]r^{-i} \tag{21}$$

This means T_i's and w_i's can be used as the address lines of a ROM device implementing the SELECT function. It is not difficult to see that even for small radices the number of input lines to the device will be prohibitive [IRW 77]. Two techniques to avoid this dilemma are available: 1) Use a PLA, or 2) Perform carry propagation on the most significant portion of $\hat{P}_{j-1}$ to reduce the number of lines required. The number of input lines will be reduced by up to 44% if this technique is used [IRW 77]. In estimating the cost of the Processing Elements we have ignored the cost of the Selection Block because it effectively appears in only one PE (PE_1). We assume that the time required by the selection process is:

$$t_s=4\delta_g$$

Gate Complexity of Digit Processing Logic

The total number of gates we require for the implementation of DPL is the sum of all the gates we require for each of its components:

$$G_{DPL}=58k^2+79k+51 \tag{22}$$

Pin Complexity of DPL

The pins required for digit processing logic DPL is the sum of the pins necessary for input ports TIP_i, RIP_i and output ports TOP_i and ROP_i. The total number of pins, P_{DPL} necessary for logic implementation of DPL is equal to the sum of the pins required for input and output ports.

$$P_{DPL}=P_{TOP_i}+P_{ROP_i}+P_{TIP_i}+P_{RIP_i}$$

From [GOR 80c] we have:

$$P_{TIP_i}=P_{TOP_i}=P_{t_{i+1}^{A(j-1)}}=2(2k+1) \tag{23}$$

and since the information on RIP_i is a single digit then:

$$P_{ROP_i}=P_{RIP_i}=k+1 \tag{24}$$

Substituting (23) and (24) in the equation for P_{DPL} we get:

$$P_{DPL}=10k+6 \tag{25}$$

Overall Logic Complexity of a PE

The total number of gates, G_{PE}, required for the implementation of a PE is the sum of the gates required for the combinational logic of DPL, the gates required for the PE control logic and the gates required for the implementation of storage registers in the PE. The storage registers in a PE comprise the registers in the Register File and buffer registers in DPL. From Table 1 the number of gates needed for storage is:

$$G_{STO}=[6(k+1)+4(2k+1)+k(k-1)+2]G_D$$

$$=(k^2+13k+12)G_D$$

G_D is the number of gates required for the realization of a D type flip-flop. Assuming G_D=6 [TEX 69] we get:

$$G_{STO}=6k^2+78k+72 \tag{26}$$

Ignoring the number of gates needed for PE control, the number of gates required for each PE is:

$$G_{PE}=G_{DPL}+G_{STO}$$

Substituting the values from Equations (22) and (26) we get:

$$G_{PE}=64k^2+157k+123 \tag{27}$$

The pin requirements for each PE is the sum of pins required for the DPL plus the number of pins needed for the input and output busses (ignoring the pins required for control signal from GCU). That is,

$$P_{PE}=P_{DPL}+P_{N-BUS}+P_{D-BUS}+P_{Q-BUS}$$

or

$$P_{PE}=13k+9 \tag{28}$$

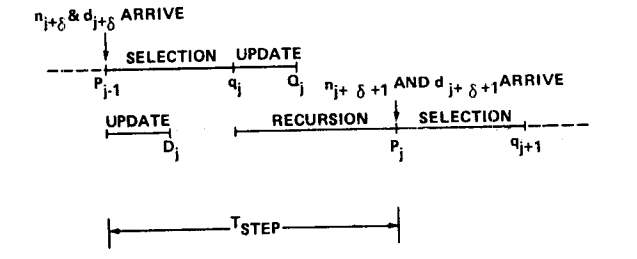
The pin and gate requirements of DPL and PE along with the gate requirement of other PE components have been shown in Table 2.

4. SPEED OF THE DIVISION UNIT

Time required to compute a single quotient digit (k+1 bits) is composed of the following elements:

1. Time to select a quotient digit (t_s)
2. Time to update Q_j and D_j registers (t_u)
3. Time to perform the basic recursion formula (t_R)

The following diagram indicates the relative position of these three delays with respect to one another.



Since usually t_s and t_R are greater than t_u, the total time for one step of the algorithm (T_{STEP}) is:

$$T_{STEP}=t_s+t_R \tag{29}$$

Each step starts when the digits of the dividend ($n_{j+\delta}$) and divisor ($d_{j+\delta}$) appear on the input busses (N-BUS and D-BUS). At the beginning of each step, selection of the quotient digit (q_j) is initiated by the quotient selection unit in the most significant Processing Element (PE_1). This selection is based on the truncated version of the previous partial remainder ($\hat{P}_{j-1}$) and divisor ($\hat{D}_{j-1}$).

r	K	G MIAD	G DPG	G DSE	G SRIB	G SROB	G STOP	G STIP	G DPL	P DPL	G STO	G_{PE}	P_{PE}
2	1	78	9	16	22	22	20	10	188	16	156	344	22
4	2	260	12	32	36	36	34	17	441	26	252	693	35
8	3	546	17	48	52	52	50	26	810	36	360	1170	48
16	4	936	24	64	70	70	68	37	1295	46	480	1775	61
32	5	1430	33	80	90	90	88	50	1896	56	612	2508	74
64	6	2028	44	96	112	112	110	65	2613	66	756	3369	87
128	7	2730	57	112	136	136	134	82	3446	76	912	4358	100
256	8	3536	72	128	162	162	160	101	4395	86	1080	5475	113

Table 2 -- Gate and Pin Complexity of a Processing Element vs the Radix (r).

PE_1 outputs q_j on the Q-BUS. After reception of this quotient digit and some other information from its right neighbor, each PE starts the processing of one digit of the next partial remainder (P_j). After a certain amount of time (t_{PE}), the next partial remainder will be available in a redundant format ($w_i^{(j)}$ and $T_i^{(j)}$). This process continues until the required precision is obtained. We compute t_{PE} by measuring the time span between the setting of all registers (R_1 through R_9) in PE_i at step (j) and (j+1). Therefore (29) can be rewritten as:

$$T_{STEP}=t_s+t_{PE} \tag{30}$$

Graph representation of t_s+t_{PE} is shown in Figure 10. Using this graph T_{STEP} is found to be:

$$T_{STEP}=2t_{SRIB}+t_{DSE}+t_{MIAD}+t_{SM/RB}+t_{STIP}+t_{STOP}$$
$$+3t_{SROB}+t_s \tag{31}$$

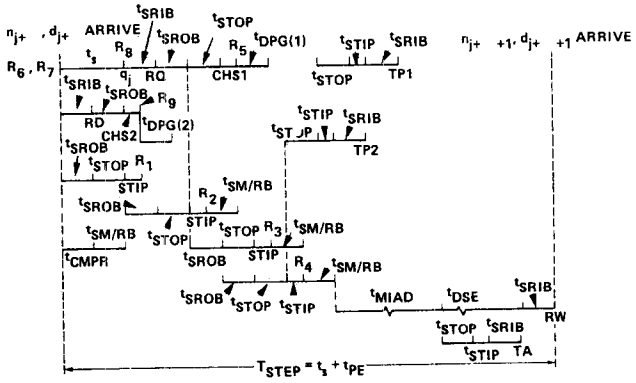


Figure 10 -- Graph Representation of T_{STEP}.

The components of T_{STEP} are as follows:

t_{SRIB}:

Logic design of Register File Input Bus Selector (SRIB) has been given in [GOY 76]. According to this design:

$$t_{SRIB}=2\delta_g$$

t_{SROB}:

From [GOR 80c] :

$$t_{SROB}=2\delta_g$$

Also we have:

$$t_{STOP}=2\delta_g$$

and

$$t_{CHS1}=\delta_g$$

t_{DPG}:

Assuming LVE_3 (Logic Vector Encoding) for the operands [GOY 76], and according to what has been explained in the design of the Digit Product Generator we get:

$$t_{DPG}=t_{XOR}=2\delta_g$$

t_{MIAD}:

According to Eq. (12):

$$t_{MIAD}=L^*\delta_{BU}$$

such that: $L=\lceil \log_2 2(k+1) \rceil$ and δ_{BU} is the time required by one Borovec Unit [BOR 68]. Using LVE_3, δ_{BU} is obtained to be [GOY 76]:

$$\delta_{BU}=7\delta_g$$

Therefore:

$$t_{MIAD}=7\delta_g\lceil \log_2 2(k+1) \rceil$$

t_{DSE}:

From the design given in [GOY 76], t_{DSE} can be estimated to be:

$$t_{DSE}\approx 5k\delta_g+3k\delta_g=8k\delta_g$$

t_{STIP}:

According to the design given in [GOR 80c] :

$$t_{STIP}=\delta_g$$

Therefore, the total time per step is:

$$T_{STEP}=\lceil 8k+7\lceil \log_2(k+1) \rceil+24 \rceil \delta_g \qquad (32)$$

Table 3 shows T_{STEP} and its components. From this table it can be deduced that contribution of "Digit Sum Encoder" (DSE) to the total step time dominates all other components for relatively large radices. But this unit can be eliminated if $w_i^{(j)}$ can be stored in redundant format. That is, RW and R_2 should be made to be double bank registers. Also STIP, SRIB, SROB and STOP blocks should be modified.

r	k	t_{SRIB} (δ_g)	t_{DSE} (δ_g)	t_{MIAD} (δ_g)	t_{STIP} (δ_g)	t_{STOP} (δ_g)	t_{SROB} (δ_g)	t_{STEP} (δ_g)
2	1	2	8	14	1	2	2	29
4	2	2	16	21	1	2	2	54
8	3	2	24	21	1	2	2	62
16	4	2	32	28	1	2	2	77
32	5	2	40	28	1	2	2	85
64	6	2	48	28	1	2	2	93
128	7	2	56	28	1	2	2	101
256	8	2	64	35	1	2	2	116

Table 3 -- Time Required for one step of the Division Process (T_{STEP}) and its Components.

5. CONCLUSION

A detailed design of a digit-slice on-line arithmetic unit was considered. This unit was designed as a set of basic Processing Elements (PE) each of which operates on a single digit of the operands and the results. Assuming that the radix of implementation is $r=2^k$, the number of gates required for one PE has been shown to be proportional to k^2. Also, the number of pins required is proportional to k. Specifically, we showed that the number of gates vary from 350 to 5500 for radices 2 to 256. From the gate count expression G_{PE}, assuming a total precision of B bits, it can be easily determined that r=2 and r=4 require the least total number of gates. The number of external connections per module ranges from 22 to 113. Again, assuming a total of B bits, we determine that the total number of pins (external connections) is a monotonically decreasing function with the maximum at r=2. Similarly, it can be seen that the total time for B bits is also a monotonically decreasing function with the maximum at r=2. These results indicate the expected time-gate count tradeoff.

They also show that the on-line arithmetic is highly suitable for LSI/VLSI implementation. As the number of available gates per chip increases, several processing elements (modules) can be implemented on one chip and thus simplify further the interconnections.

6. REFERENCES

[AVI 61] Avizienis, A. "Signed Digit Number Representation for Fast Parallel Arithmetic", IRE Trans. Electron. Comput., Vol. EC-10, pp. 389-400, 1961.

[AVI 62] Avizienis, A., "On a Flexible Implementation of Digital Computer Arithmetic ", Proceeding of IFIP Congress, Munich, W. Germany, Aug 27 to Sept 1, 1962, pp. 204-211.

[AVI 66] Avizienis, A., " Arithmetic Microsystems for The Synthesis of Function Generators", Proceeding of the IEEE, Vol. 54, No.12, Dec 1966, pp. 1910-1919.

[AVI 70] Avizienis, A. and Tung, C., "A Universal Arithmetic Building Element (ABE) and Design Methods for Arithmetic Processors", IEEE Transactions on Computers, Vol C-19, No. 8, August 1970, pp. 733-745.

[BOR 68] Borovec, R.T., "The Logical Design of a Class of Limited Carry-Borrow Propagation Adders", M.S. Thesis, Report 275, Department of Computer Science, University of Illinois, Urbana, August 1968.

[ERC 75] Ercegovac, M.D., "A General Method for Evaluation of Functions and Computations in a Digital Computer", Ph.D. Thesis, Report No. 750, Department of Computer Science, University of Illinois, Urbana, August 1975.

[ERC 80a] Ercegovac, M.D. and Grnarov, A.L., "On The Performance of On-Line Arithmetic", Proc. 1980 International Conference on Parallel Processing, Harbor Springs, Michigan, August 1980.

[ERC 80b] Ercegovac, M.D. and Grnarov, A., "On-Line Structures for Array Computations", (in preparation), 1980.

[GOR 80a] Gorji-Sinaki, A. and Ercegovac, M.D., "On-Line Division Algorithms: A Systematic Derivation", (in preparation), 1980.

[GOR 80b] Gorji-Sinaki, A. and Ercegovac, M.D., "Error-Coded Algorithms for On-Line Arithmetic", submitted to the 1981 International Symposium on Fault-Tolerant Computing, June 24-26 1981.

[GOR 80c] Gorji-Sinaki, A., "Error-Coded Algorithms for On-Line Arithmetic", Ph.D. Dissertation, Computer Science Dept., University of California, Los Angeles, 1980.

[GOY 76] Goyal, Lakshmi, "A Study in the Design of an Arithmetic Element for Serial Processing in an Iterative Structure ", Ph.D. Thesis, Report 797, Department of Computer Science, University of Illinois, Urbana, May 1976.

[GRN 80] Grnarov, A. and Ercegovac, M.D., "VLSI-Oriented Iterative Networks for Array Computations", 1980 International Conference on Circuits and Computers, New York, October 1980.

[IRW 77] Irwin, M.J., "An Arithmetic Unit for On-Line Computation", Ph.D. dissertation, University of Illinois at Urbana-Champaign, report No. UIUCDCS-R-77-873.

[ROH 67] Rohatsch, Fred A., "A Study of Transformations Applicable to the Development of Limited Carry-Borrow Propagation Adders", Ph.D. Thesis, Report 226, Department of Computer Science, University of Illinois, Urbana, June 1967.

[TEX 69] Texas Instrument Incorporated, "TTL Integrated Circuits Catalog", Dallas, Texas Instruments Catalog CC201, August 1969.

[TRI 77] Trivedi, K.S. and Ercegovac, M.D., "On-Line Algorithms for Division and Multiplication" IEEE Trans. on Comput., Vol. C-26, No.7, July 1977.

[TRI 78] Trivedi, K.S. and Rusnak, J.G., "Higher Radix On-Line Division", Proc. 4th IEEE Symp. Comput. Arithmetic, pp. 164-174, October 1978.

[TUN 70] Tung, C. and Avizienis, A., "Combinational Arithmetic Systems for the Approximation of Functions", 1970 Spring Joint Computer Conf., AFIPS Proc., Vol. 36, Washington, D.C., Spartan, 1970,