DESIGN OF A DIGIT-SLICE ON-LINE ARITHMETIC UNIT

A. Gorji-Sinaki and M. D. Eregovac

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 to show its feasibility 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+1)-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 8+1 digits of precision. Therefore, an output digit can be obtained for each additional input digit. The on-line delay 8 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 multivalued 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 on-line approaches offer 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].

2. ON-LINE ARITHMETIC ALGORITHMS

2.1 On-Line Algorithm

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

\[ P_j = f(P_{j-1}, x_{j-1}, y_{j-1}) \]

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 \( P_j \) and the current inputs.

\[ z_j = \text{SELECT}(P_j, x_j, y_j) \]

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 73, TRI 77]. Also, systematic methods for derivation of on-line addition/subtraction, multiplication and division algorithms appear 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 80a]. 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]:

\[
\begin{align*}
\text{Step 1} \quad \text{[Initialization]:} & \quad P_0 = \sum_{r=0}^{\delta} x_r r^{-1} \quad D_0 = \sum_{r=0}^{\delta} d_r r^{-1} \quad Q_0 = 0 \\
\text{Step 2} \quad \text{[Recursion]:} & \quad \text{for } j = 1, 2, \ldots, m \text{ do:} \\
\text{Step 2.1} \quad \text{[Selection]:} & \quad q_j = \text{SELECT}(P_j, D_j) \\
& \quad Q_j = Q_{j-1} + q_j r^{-j} \\
\text{Step 2.2} \quad \text{[Input Digit]:} & \quad D_j = D_{j-1} + d_j r^{-j} \\
\text{Step 2.3} \quad \text{[Basic Recursion]:} & \quad P_j = r P_{j-1} - q_j D_j + q_j r^{-j-1} - Q_{j-1} - d_j r^{-j-1} \\
\text{Step 3} \quad \text{[End]} & \quad \text{In the above algorithm } N, D, \text{ and } Q \text{ are assumed to be the dividend, divisor, and the quotient, respectively.}
\end{align*}
\]

* Supported in part by the Office of Naval Research Contract No.
N00014-79-C-0866.
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 division 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 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\textsubscript{1} and least significant digits in PE\textsubscript{n}. Output digits are generated by PE\textsubscript{1} in an on-line mode and are placed on the Z-Bus. Each output digit is temporarily stored by all PEs.

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\textsubscript{1} and is placed on the output bus. This operation continues until the required precision is obtained.

![Figure 1 -- Organization of an On-Line Division Unit](image)

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

\[ P_{j} = \sum_{i=1}^{2} p_{i} r_{j-i}^{(1)} \]

\[ D_{j} = \sum_{i=1}^{2} d_{i} r_{j-i}^{(0)} \]

where, for each PE\textsubscript{i}, \( p_{i} \) 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\textsubscript{i}, is defined as:

\[ p_{i}^{(0)} = p_{i}^{(1)} - q_{i} d_{i} + n_{i+8} r_{j-i}^{(0)} - T_{i}^{(0)} + T_{i}^{(1)} \]

where \( n_{i+8} \) is zero in PE, \( T_{i}^{(1)} \) is the transfer digit from PE\textsubscript{i-1} at the j-th step, \( T_{i}^{(0)} \) is the transfer digit to PE\textsubscript{i+1} at the j-th step.

It is obvious that \( a_{i+8} \) is zero in PE, for \( \delta \leq i \leq j-2+\delta \). 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 \( p_{i}^{(0)} \) in PE\textsubscript{i} is represented by an interim partial remainder \( w_{i}^{(0)} \) and a transfer digit \( T_{i}^{(0)} \) such that:

\[ p_{i}^{(0)} = w_{i}^{(0)} + T_{i}^{(0)} \]

![Figure 2 -- Interconnection Between Processing Elements](image)

The transfer function in Eq. (5) is obtained by a series of three transformations \( f_{1} \), \( f_{2} \) and \( f_{3} \) such that:

\[ f_{1} : -q_{i} d_{i} + n_{i+8} r_{j-i}^{(0)} + w_{i}^{(1)} \]

\[ f_{2} : -q_{i} d_{i} - T_{i}^{(0)} + w_{i}^{(1)} \]

\[ f_{3} : -q_{i} d_{i} + n_{i+8} r_{j-i}^{(0)} + w_{i}^{(1)} + T_{i}^{(0)} \]

The transfer digits from PE\textsubscript{i-1} to PE\textsubscript{i} are \( e_{i}^{(0)} \) and \( e_{i}^{(1)} \) resulting from transformations \( f_{1} \) and \( f_{2} \), respectively. Also there is a transfer digit out of the Multi-Input Adder \( e_{i}^{(2)} \). Therefore:

\[ T_{i}^{(0)} = e_{i-1}^{(1)} + e_{i}^{(0)} + e_{i}^{(2)} \]

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

\[ p_{i}^{(0)} = w_{i}^{(1)} + T_{i}^{(1)} - q_{i} d_{i} + n_{i+8} r_{j-i}^{(0)} + w_{i}^{(1)} + T_{i}^{(0)} \]

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 \( r^{2k} \) 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 \( q_{i} d_{i} \) and \( a_{i+8} d_{i} \) are generated by two separate product matrix generators which consist of a \( k \times k \) square array of redundant binary product cells. Each cell performs the product of two redundant binary digits \( q_{i} \) and \( d_{i} \) and its output product digit is also in the digit set \{-1,0,1\}. Figure 5 shows the operation of the digit product generator for radix 16, i.e., \( k=4 \).

Design of a MIRBA

MIRBA is a limited carry/borrow propagation adder which
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:

\[
L = \left[ \log_2 (2(k+1)) \right]
\]

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

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 \(\left(w_{i}^{[j]}\right)\). Three other registers are used to hold the transfer functions \(\left(T_{i}^{[j]}\right)\) 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 inputted to the DPL either directly or through an Output Bus Selector SROB, also in the DPL.

Block Diagram Description of DPL

Figure 8 -- Block Diagram of a Processing Element

As shown in Figure 9, there are input and output ports designated as \(TIP\), \(RIP\), and \(TOP\), \(ROP\), respectively. The input port \(TIR\) carries the 'Transfer' (carry or borrow) from adjacent MIAD and the contents of some register in the Register File of the adjacent \(PE_{i+1}\). \(RIP\) carries the quotient digit from \(PE_{i+1}\). The output ports \(TOP\) and \(ROP\) carry similar information for \(PE_{i+1}\) and \(PE_{i+1}\) 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 MIAR is implemented as a tree structure of BUs (see Fig. 7). Each MIAR requires \(2k+1\) BUs that are arranged in \(L = \log_2(2(k+1))\) levels. Therefore:

\[
G_{MIAD} = k(2k+1)G_{BU} = 26k(2k+1)
\]

(11)

\[
l_{MIAD} = L^8_{BU}
\]

(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 \(\left(w_{i}^{[j]}\right)\). Three other registers are used to hold the transfer functions \(\left(T_{i}^{[j]}\right)\) 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 inputted to the DPL either directly or through an Output Bus Selector SROB, also in the DPL.
where \( G_{\text{MIAD}} \) is the number of gates required for one MIAD, \( t_{\text{MIAD}} \) is the delay of one MIAD, \( G_{\text{BU}} \) is the number of gates required for one BU and \( t_{\text{BU}} \) 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 \( i_{\text{BU}}^{(k)} \) and \( o_{\text{BU}}^{(k)} \) 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 86c]. 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 \( i_{\text{DPG}}^{(k)} \) and \( o_{\text{DPG}}^{(k)} \). The number of pins required for a transfer signal is:

\[
1 + \frac{k(k-1)}{2}
\]  

(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, \( G_{\text{DSE}} \) required by the DSE logic has been found to be [GOY 76]:

\[
G_{\text{DSE}} = 16k
\]

(14)

**Logic Design of Selectors SRI0, SROB, STOP and STIP**

The selector SRI0 is a seven-input multiplexer. It constantly examines the data on D, Q and N Busses. If the data on any of these busses belong to PE\(_j\), it writes this data in the corresponding registers in the register file. It also gates the output of DSE \( w_{j+1}^{(k)} \) to Register RW in the Register File. The transfer function \( T_{j+1}^{(k)} = w_{j+1}^{(k)} \) which should be sent to PE\(_{j+1}\) 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).**

---

76
inside \( PE_i \) is a table look-up device which implements the SELECT function. It examines the \( y \) most significant digits of \( P_{r-1} \) and \( x \) most significant digits of \( D_{r-1} \), in order to select the appropriate quotient digit, \( q_r \).

According to Eq. (9):
\[
P_{r-1} = \sum_{i=1}^{d} w_i^{(r-1)} T_i^{(r-1)} x_r^{(r-1)}
\]
Therefore, the truncated version of \( P_{r-1} \) (i.e., \( \hat{P}_{r-1} \)) is:
\[
\hat{P}_{r-1} = \sum_{i=1}^{d} [w_i^{(r-1)} + T_i^{(r-1)} x_r^{(r-1)}]
\]

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 \( \hat{P}_{r-1} \) 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 (\( PE_i \)). We assume that the time required by the selection process is:
\[
t_i = 48\delta_k
\]

**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:
\[
G_{DPL} = 58k^2 + 79k + 51
\]

**Pin Complexity of DPL**

The pins required for digit processing logic DPL is the sum of the pins necessary for input ports \( TIP \), \( RIP \), and \( TOP \). 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:
\[
P_{DPL} = P_{TOP} + P_{ROP} + P_{TIP} + P_{RIP}
\]
From [GOR 80c] we have:
\[
P_{TIP} = P_{TOP} = P_{ROP} = 2k + 1
\]
and since the information on \( RIP \) is a single digit then:
\[
P_{RIP} = P_{RIP} = k + 1
\]
Substituting (23) and (24) in the equation for \( P_{DPL} \) we get:
\[
P_{DPL} = 10k + 6
\]

**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 PE block and buffer registers in DPL. From Table 1 the number of gates needed for storage is:
\[
G_{STO} = 6(k+1) + 44(2k+1) + 12k + 12 + 2GD
\]

\[
= (k^2 + 13k + 12)GD
\]

Design of The Quotient Selection Unit

The selection of the quotient digits is done by the most significant Processing Element (\( PE_i \)). The quotient digit selector

\[
\begin{align*}
 b &= MAX \left[ k + 1, P_{r-1}^{(k+1)}, P_{r-1}^{(k+1)}, P_{r-1}^{(k+1)} \right] \\
P_{r-1}^{(k+1)} &= \text{Pin Count of } r^{(k+1)} = 2(2k + 1) \\
P_{r-1}^{(k+1)} &= \text{Pin Count of } r^{(k+1)} = 1 + \frac{k(k-1)}{2} \\
P_{r-1}^{(k+1)} &= \text{Pin Count of } r^{(k+1)} = 1 + \frac{k(k-1)}{2} \\
\text{Therefore:} \\
b &= 2(2k + 1) \\
&= (15)
\end{align*}
\]

The logic design of SRIB is similar to that shown in [GOY 76] and the number of gates required is:
\[
G_{SRIB} = 2b + 2(1 + \frac{k(k-1)}{2}) + 4(k+1) = k^2 + 11k + 10
\]

The selector SROD 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, 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 SROD are exactly the same as that of SRIB, that is:
\[
G_{SROD} = k^2 + 11k + 10
\]

The width of selector STOP is equal to the width of output port \( TOP \). The width of \( TOP \) is determined by the maximum length of "Adder Transfers". Therefore, the width of \( TOP \), is given by Eq. (15). Logic implementation of STOP is shown in [GOR 80c]. From the given design we conclude that:
\[
G_{STOP} = 3b + 2(1 + \frac{k(k-1)}{2}) = k^2 + 11k + 8
\]

The selector STIP is actually a four-output demultiplexer. 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:
\[
G_{STIP} = b + k + 1 + 2(1 + \frac{k(k-1)}{2}) = k^2 + 4k + 5
\]

---

<table>
<thead>
<tr>
<th>Register</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 )</td>
<td>( 2(2k+1) )</td>
</tr>
<tr>
<td>( R_2 )</td>
<td>( k+1 )</td>
</tr>
<tr>
<td>( R_{3-4} )</td>
<td>( 1 + k(k-1)/2 )</td>
</tr>
<tr>
<td>( R_{5-9} )</td>
<td>( k+1 )</td>
</tr>
<tr>
<td>IBR</td>
<td>( 2(2k+1) )</td>
</tr>
</tbody>
</table>

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 (\( PE_i \)). The quotient digit selector
$G_D$ is the number of gates required for the realization of a D type flip-flop. Assuming $G_D=6$ [TEX 69] we get:

$$G_{STD}=6k^2+78k+72$$  \hspace{1cm} (26)$$

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

$$G_{PE}=G_{DPE}+G_{STD}$$

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

$$G_{PE}=64k^2+157k+123$$  \hspace{1cm} (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

$$P_{PE}=13k+9$$  \hspace{1cm} (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.

![Diagram](image)

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

$$T_{STEP}=t_s+t_R$$  \hspace{1cm} (29)$$

Each step starts when the digits of the dividend ($n_{i+1}$) and divisor ($d_{i+1}$) appear on the input busses (N-BUS and D-BUS). At the beginning of each step, selection of the quotient digit ($q_i$) is initiated by the quotient selection unit in the most significant Processing Element (PE). This selection is based on the truncated version of the previous partial remainder ($P_{i-1}$) and divisor ($D_{i-1}$).

<table>
<thead>
<tr>
<th>r</th>
<th>k</th>
<th>G_{IMAD}</th>
<th>G_{DPL}</th>
<th>G_{DPE}</th>
<th>G_{SIBR}</th>
<th>G_{STOP}</th>
<th>G_{DPL</th>
<th>G_{PE}</th>
<th>G_{FE}</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>76</td>
<td>9</td>
<td>16</td>
<td>22</td>
<td>22</td>
<td>20</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>260</td>
<td>12</td>
<td>32</td>
<td>36</td>
<td>36</td>
<td>34</td>
<td>17</td>
<td>441</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>456</td>
<td>17</td>
<td>48</td>
<td>52</td>
<td>52</td>
<td>50</td>
<td>26</td>
<td>810</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>936</td>
<td>24</td>
<td>64</td>
<td>70</td>
<td>70</td>
<td>68</td>
<td>37</td>
<td>1295</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>1430</td>
<td>33</td>
<td>80</td>
<td>90</td>
<td>90</td>
<td>88</td>
<td>50</td>
<td>1896</td>
</tr>
<tr>
<td>64</td>
<td>6</td>
<td>2028</td>
<td>44</td>
<td>96</td>
<td>112</td>
<td>112</td>
<td>110</td>
<td>60</td>
<td>2613</td>
</tr>
<tr>
<td>128</td>
<td>7</td>
<td>2730</td>
<td>57</td>
<td>112</td>
<td>138</td>
<td>138</td>
<td>134</td>
<td>82</td>
<td>3446</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>3538</td>
<td>72</td>
<td>128</td>
<td>162</td>
<td>162</td>
<td>160</td>
<td>101</td>
<td>4395</td>
</tr>
</tbody>
</table>

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

$PE_1$ outputs $q_i$ 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 ($P_i$). After a certain amount of time ($t_{PE}$), the next partial remainder will be available in a redundant format ($T_i^{(1)}$ and $T_i^{(2)}$). This process continues until the required precision is obtained. We compute $t_{PE}$ by measuring the time span between the setting of all registers ($R_i$ through $R_{i+1}$) in PE, at step (i) and (i+1). Therefore (29) can be rewritten as:

$$T_{STEP}=t_s+t_{PE}$$  \hspace{1cm} (30)$$

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

$$T_{STEP}=2(t_{SIBR}+t_{DPL}+t_{IMAD}+t_{SM/BUS}+t_{STOP}+t_{SIBR}+t_{STOP})$$

$$+3t_{SIBR}$$  \hspace{1cm} (31)$$

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

The components of $T_{STEP}$ are as follows:

$t_{SIBR}$:

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

$$t_{SIBR}=28$$

$t_{SIBO}$:

From [GOR 80c]:

$$t_{SIBO}=28$$
Also we have:

\[ t_{\text{STOP}} = 28 \delta_e \]
and

\[ t_{\text{ISE}} = 7 \delta_e \]

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 \( r = 2 \), 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 \( G_{\text{PE}} \), 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


