AN ARITHMETIC MODULE FOR EFFICIENT EVALUATION OF FUNCTIONS

M.D. Ercegovac, UCLA Computer Science Department, University of California, Los Angeles

M.M. Takata, Hughes Aircraft Company, Culver City, CA

ABSTRACT

The organization and design of an arithmetic module (Basic Byte-Slice Module -- BBM) is presented. A network of BBM's implements an efficient digit-by-digit method for fast evaluation of polynomial and rational functions. Verification of the BBM design, its feasibility in present LSI technologies and its performance are discussed. The proposed BBM is characterized by a small number of input/output terminals, a uniform internal structure, and simple control and inter-module communication requirements.

I. INTRODUCTION

In this paper the implementation of an efficient method for fast evaluation of polynomial and rational functions at the hardware level is presented. The evaluation method, described previously in [1, 2] as the E-method, is chosen because it has several properties enabling efficient hardware-level implementation. First, the E-method provides a unique iterative algorithm. It allows for parallel execution in such a manner that the time required to simultaneously evaluate a polynomial and all its derivatives or a rational function is linearly proportional to the number of digits in the result. The algorithm has a single primitive operator which allows for easy partitioning of the implementation into identical modules with a small number of input/output connections. The interconnections between modules require very few lines. Some additional characteristics will be discussed in later sections.

The objective of this research was to establish the implementation feasibility of the E-method. The main subject of this presentation deals with the organization and design of a byte-oriented module (Basic Byte Slice Module - BBM) which, together with ROM modules, can be used to implement fast and versatile function evaluation units. It is assumed that the function evaluation is performed by using appropriate polynomial or rational approximations in a fixed-point radix-2 system. The fast polynomial evaluation schemes [7, 8], based on fast primitive operators and a redundant number representation system are versatile but more complex on the level of the basic building block. In addition, the overall control and interconnection requirements for these methods are more complex than the scheme based on the E-method.

This research was supported in part by the National Science Foundation, Grant No. MCS72-03633 A04.

In the following sections, the relevant details of the E-method are discussed and the organization of a function evaluation unit is described in a hierarchial manner. Some details of the BBM design are also presented. A complete description, including a design simulation and verification details, is provided in [3].

II. THE EVALUATION METHOD

In this section the E-method of evaluating polynomial and rational functions [1, 2] is briefly reviewed. The main idea of the method is to replace a given polynomial

$$P_u(x) = \sum_{i=0}^{u} p_i x^i$$

or rational function

$$R_{u,v}(x) = P_u(x)/Q_v(x)$$

by a nonsingular system of simultaneous linear equations $\underline{A}\,\underline{y} = \underline{b}$. This system, subject to certain conditions, can be solved in a parallel manner using a digit-by-digit iterative algorithm. The solution to the linear system directly provides the value of the polynomial (rational function). For an n-th order system where $n = \max(u,v)+1$, a network of n elementary arithmetic units is used to generate the solution in (m+1) carry-free additions where m is the number of digits in the solution.

The transformation of a polynomial

$$P_u(x) = \sum_{i=0}^{u} p_i x^i$$

into a system of n=u+1 simultaneous linear equations $\underline{A}\,\underline{y} = \underline{b}$ is performed as follows. The elements of the coefficient matrix $\underline{A}$ are:

$$a_{ij} = \begin{cases} 1 & i = j; \\ -x & \text{for } j = i+1,\ i \le u; \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

and the elements of the vector $\underline{b}$ are:

$$b_i = \begin{cases} p_{i-1} & \text{for } i = 1,\dots, u+1 \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

so that

$$y_1 = p_u(x)$$

and

$$y_i = \frac{d^{i-1}}{dx^{i-1}} (P_u(x)) \quad \text{for } i = 2,\dots, u. \qquad (3)$$

$$y = (y_1, y_2,\dots,y_n)$$

corresponds to the value of the polynomial and all its derivatives.

Example

To Evaluate $P_3(x) = p_3x^3 + p_2x^2 + p_1x + p_0$

$$\begin{aligned} y_1 - xy_2 &= p_0 \\ y_2 - xy_3 &= p_1 \\ y_3 - xy_4 &= p_2 \\ y_4 &= p_3 \end{aligned}$$

where $y_1 = P_3(x)$. $\square$

The transformation of a rational function

$$R_{u,v}(x) = \frac{P_u(x)}{Q_v(x)}, \quad q_0 = 1$$

into a linear system $\underline{A}\ \underline{y} = \underline{b}$ of order $n = \max(u,v) +1$ is specified as follows:

$$a_{ij} = \begin{cases} 1 & \text{for } i = j \\ q_{i-1} & \text{for } j = 1 \text{ and } i = 2,3,\dots,v+1 \\ -x & \text{for } j = i+1 \text{ and } i=1,2,\dots,n-1 \\ 0 & \text{otherwise} \end{cases} \qquad (4)$$

$$b_i = \begin{cases} p_{i-1} & \text{for } i = 1,2,\dots,u+1 \\ 0 & \text{otherwise} \end{cases} \qquad (5)$$

Then,

$$y_1 = R_{u,v}(x) \qquad (6)$$

Example

To Evaluate

$$R_{2,3}(x) = \frac{p_2x^2 + p_1x + p_0}{q_3x^3 + q_2x^2 + q_1x + 1}$$

the corresponding linear system is

$$\begin{aligned} y_1 - xy_2 &= p_0 \\ q_1y_1 + y_2 - xy_3 &= p_1 \\ q_2y_1 + y_3 - xy_4 &= p_2 \\ q_3y_1 + y_4 &= 0 \end{aligned}$$

where $y_1 = R_{2,3}(x)$.

The linear system $\underline{A}\ \underline{y} = \underline{b}$, obtained by these transformations is solved by using an iterative digit-by-digit algorithm [1,2]. After m+1 iterative steps, the elements of the solution vectory $\underline{y}$ are computed to the precision of m digits. The radix 2 version of this algorithm is shown below:

Algorithm E

```
1. [Begin]
      $\underline{w}^0 \leftarrow \underline{b}$;  $\underline{d}^{(0)} \leftarrow 0$
2. [Recursion]
      For j = 1, 2,..., m+1 do:
          $\underline{w}^{(j)} \leftarrow 2(\underline{w}^{(j-1)} - \underline{A} \cdot \underline{d}^{(j-1)})$
          $\underline{d}^{(j)} \leftarrow S(\underline{w}^{(j)})$
3. [End]
```

where

j is the step index;

m is the number of digits in the solution;

$\underline{w}^{(j)} = (w_1^{(j)},\dots, w_n^{(j)})$ is the scaled residual vector;

$\underline{d}^{(j)} = (d_1^{(j)},\dots,d_n^{(j)})$ is the solution digit vecor with $d_i^{(j)} \in \{-1, 0, 1\}$;

$n = \max(u,v)+1$ is the order of the linear system $\underline{A}\ \underline{y} = \underline{b}$

$S(\underline{w}^{(j)})$ is the digit selection function defined as

$$d_i^{(j)} = \begin{Bmatrix} 1 & \text{if} & w_i^{(j)} \geq 1/2 \\ 0 & \text{if} & |w_i^{(j)}| < 1/2 \\ -1 & \text{if} & w_i^{(j)} \leq -1/2 \end{Bmatrix} \text{for all } i,j \qquad (8)$$

The computed elements of the solution vector $\underline{y}$ are

$$y_i^* = \sum_{j=1}^{m+1} d_i^{(j)} \cdot 2^{-j}, \quad i=1, 2,\dots, n. \qquad (9)$$

The error bound

$$|y_i - y_i^*| < 2^{-m}, \tag{10}$$

holds if the convergence conditions, specified in detail in [1,2], are satisfied. In the case of rational (polynomial) function evaluation in the radix 2 representation system, the sufficient conditions are:

$$|p_i| \leq 1/2 \qquad \text{for all } i$$

and

$$|q_i| + |x| \leq 31/128 \qquad \text{for all } i \tag{11}$$

These conditions are specified so that the precision of the scaled residual

$$w_i^{(j)}$$

required for the digit selection, is 6 bits to the right of the radix point. Thus, the time of the iterative step does not depend on the total number of digits m.

In order to satisfy the convergence conditions, a scaling may be required. The scaling procedures are discussed in [1, 2] and it is assumed here that the coefficients and the argument satisfy the conditions specified in (11). Since a polynomial can be evaluated as a rational function with

$$q_i = 0, \; i > 1$$

the implementation of the algorithm for solving the linear system defined by (4) and (5) is considered. In this case, the elementary recursion to be implemented is:

$$w_i^{(j)} = 2(w_i^{(j-1)} - q_{i-1} \cdot d_1^{(j-1)} + x \cdot d_{i+1}^{(j-1)} \tag{12}$$

Since $d_i^{(j)} \in \{-1, 0, 1\}$, the primitive operator is a 3 - operand addition. The computed results y_i^* are represented in a redundant radix-2 number system [4,5]. Therefore, the selection of digits $d_i^{(j)}$ can be performed on a limited precision as discussed in [1,2]. In particular, $w_i^{(j)}$ is represented in a redundant form so that a limited carry-propagation addition is used. In the present implementation of (12), the scaled residual $w_i^{(j)}$ is represented with the pseudo-sum and carry vectors. In order to perform the selection, the assimilation of carries is performed over 6 most significant non-sign bits. Clearly, the implementation of (12) is easily partitionable into modules.

III. ORGANIZATION OF FUNCTION EVALUATION UNIT

A Function Evaluation Unit (FEU) which implements the algorithm E is described first at the system level. The FEU contains a storage (ROM) for coefficients of polynomial and rational approximations for a number of functions. The FEU consists of several Elementary Evaluation Units (EEU). Each EEU which implements the basic recursion (12) is organized as a cascade of Basic Byte Slice Modules (BBM). The number of EEUs is determined by the desired speed. The fastest evaluation requires n EEUs, n-1 being the maximum degree appearing in the approximations used. The speed/cost tradeoff can be determined from the relationship $T(n/k) = k \cdot T(n)$ where $T(n)$ is the time required to perform computation on n EEUs.

The number of BBMs is determined by the desired precision. Again, the tradeoff between speed and cost can be determined from a relationship similar to the previous one.

The conceptional organization of the FEU is given in Figure 1. A function to be evaluated is selected from a set of 2^h functions and the corresponding coefficients are transferred from the functional memory into the Buffer Section. In order to keep the number of interconnections small, the transfer of coefficients is done in two steps. At the same time, the argument is loaded into its holding registers from an external bus. In the 3rd step, the active registers in the Computational Section are initialized with the contents of the corresponding holding registers and in (m+1) additional steps the result is computed in a digit-by-digit fashion, beginning with the most significant digit. The result appears in redundant representation and can be accumulated and converted to the conventional representation in one extra addition time. However, the conversion is not considered here.

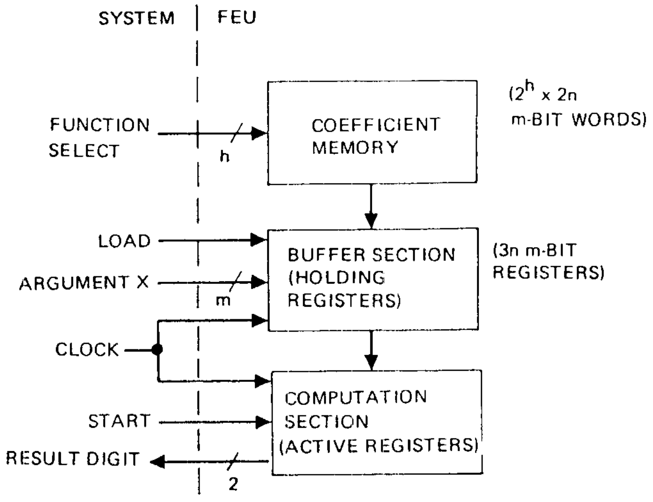

FIGURE 1. CONCEPTUAL ORGANIZATION OF FEU

The Buffer Section is used to eliminate the overhead in initializing the Computational Section since the holding registers can be loaded with new coefficients during the computation cycle. The precision of evaluation is easily controlled by the number of clock periods.

In the next section, the building block for the FEU is described. The organization of the FEU at the level of a network of these Elementary Evaluation Units is shown in Figure 3.

IV. ORGANIZATION OF THE ELEMENTARY EVALUATION UNIT (EEU)

An Elementary Evaluation Unit (EEU) implements the basic recursion (12). It has conceptually the same organization as the FEU. The organization of the EEU is shown in Figure 2. The inputs d_1 and d_{i+1} correspond to the digits generated by EEU_1 and EEU_{i+1}. The organization of the FEU at the level of Elementary Evaluation Units is described in Figure 3. Each EEU consists of an appropriate number of Basic Byte Slice Modules (BBMs). The organization and the design of a BBM is discussed in the next section.

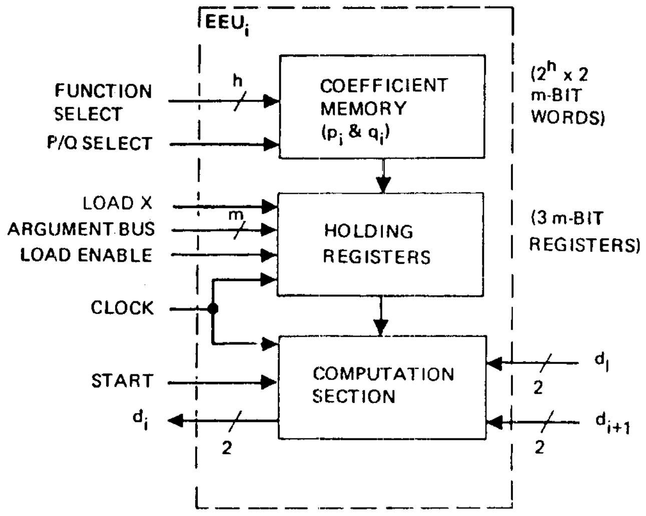

FIGURE 2. CONCEPTUAL ORGANIZATION OF EEU

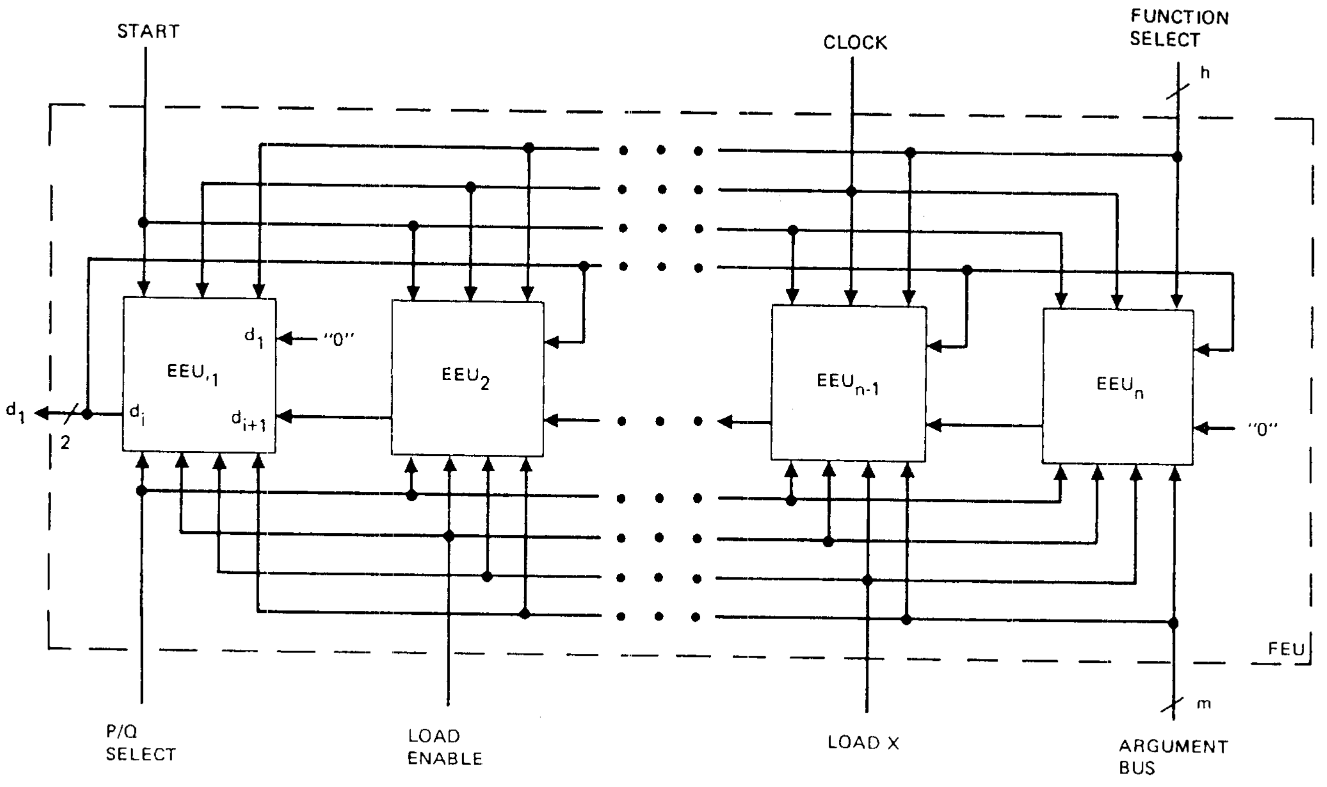

FIGURE 3. INTERCONNECTION OF EEU'S TO IMPLEMENT FEU

V. ORGANIZATION AND DESIGN OF BASIC BYTE-SLICE MODULE (BBM)

The BBM implements an 8-bit slice of an EEU. EEUs can be formed for any precision using only BBMs and a corresponding number of ROMs. The proposed BBM is universal in the sense that it can function in any position in the word format. Figure 4 illustrates how BBMs and ROMs are interconnected to form an EEU. Notice that each BBM and its associated multiplexer and ROM could be defined as another module and realized as a higher level module or possibly as a single IC. The feasibility of this approach is discussed later. A block diagram of an internal design of the BBM is shown in Figure 5.

The inputs to the BBM as indicated in Figure 4 and Figure 5 are briefly described below.

- CLOCK - Used both to load registers in the BBM as well as provide the Basic Cycle clock for generation of the result (DOUT).

- LOADX - Enables loading of the argument Holding Register with the value on the IEB (Initial Entry Bus). This signal overrides the LOADE and REGLS signals. Interpretation of the states of this input are as follows:

  0 = Disables loading of the argument.
  1 = Enables loading of the argument, regardless of the state of the other control lines (LOADE or REGLS).

- LOADE - Enables loading of either the p or q Holding Register (as determined by the REGLS line) with the value on the IEB. No action on the p or q Holding Registers occurs if this line is not active. This signal is overridden by the LOADX signal. Interpretation of the states of this input are as follows:

  0 = Disables loading of p and q Holding Registers.
  1 = Enables loading of p or q Holding Register (provided the LOADX signal is not active).

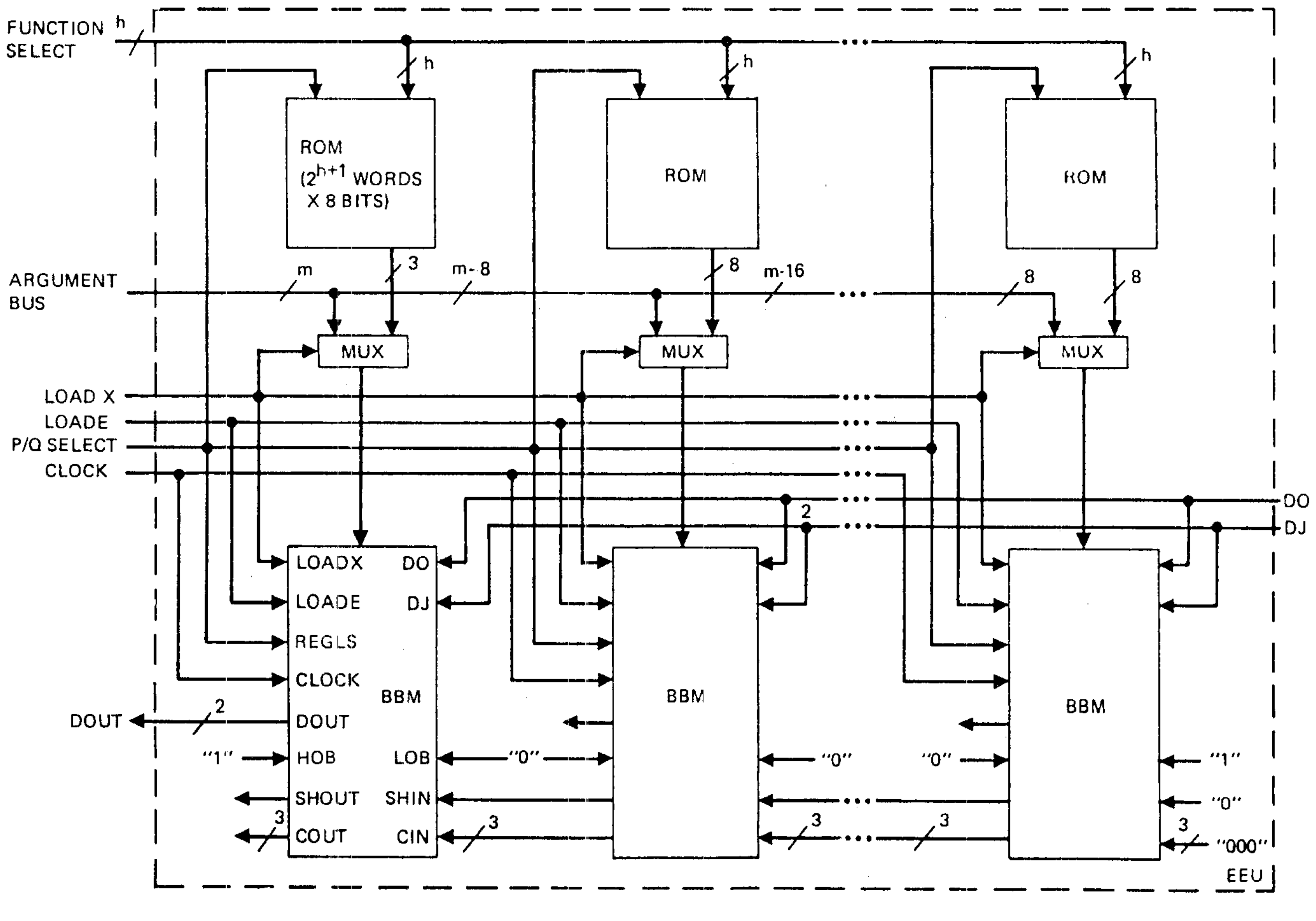

FIGURE 4. INTERCONNECTION OF BBM'S AND ROM'S TO IMPLEMENT EEU

- REGLS - Indicates whether data on the IEB is to be loaded into the p or q Holding Register. If LOADE is 0 or if LOADX is 1, the value of REGLS is irrelevant. Interpretation of the states of this input are as follows:

  0 = Data on IEB is loaded into q Holding Register.
  1 = Data on IEB is loaded into p Holding Register.

- START - Causes contents of the Holding Register to be clocked into the Active Registers of the BBM. If START remains high for more than one consecutive clock, the BBM will produce the first digit of the result on all clock periods following the first. (That is, Basic Cycle 0 is repeated until the Basic Cycle after START goes low.) Interpretation of the states of this input are as follows:

  0 = Current contents of the Active Registers are used to produce results.
  1 = Contents of the Holding Registers are clocked into the active registers.

- IEB (Initial Entry Bus) - Used to enter values into the p, q, and x Holding registers. The IEB is 8 bits wide. The radix point is assumed to be in different positions depending on the value of the HOB input.

  - If HOB = 1, the radix point is assumed to be between the second and third high-order binary digits. The representation is assumed to be in the two's complement non-redundant representation system.

  - If HOB = 0, the radix point is assumed positioned either 6, 14, 22, 30, . . . digits further to the left of the most significant digit on the IEB, depending

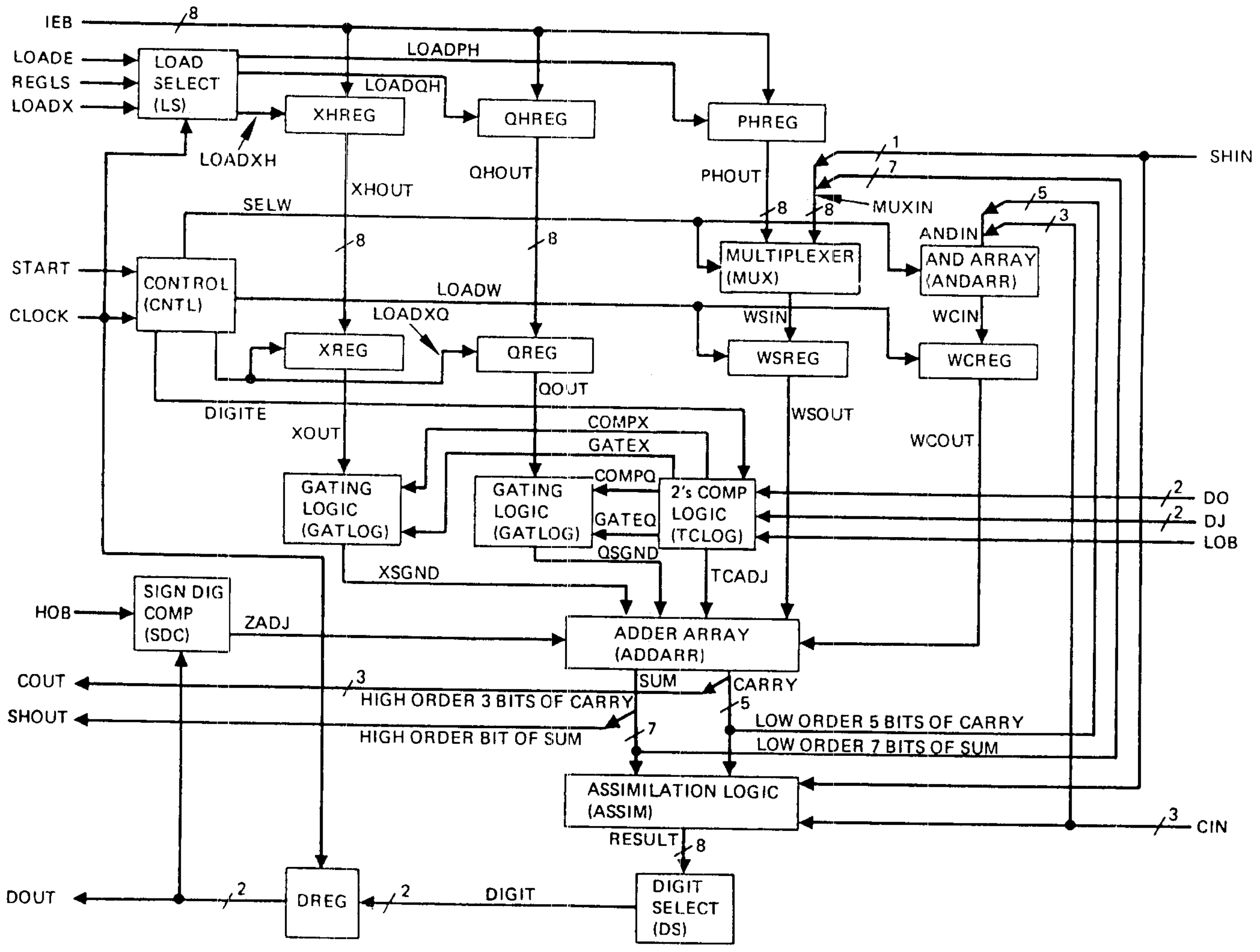

FIGURE 5. INTERNAL BBM FUNCTIONAL BLOCK DIAGRAM

on the interconnections used. In any case, the relative position of the radix point is irrelevant to the operation of the BBM is HOB = 0.

- HOB (High Order Byte) - Indicates that this particular BBM is the high order byte of an EEU.

- LOB - Indicates that this particular BBM is the low order byte of an EEU. It is used internally by the BBM in effecting two's complementation by inserting the appropriate amount of carry-in to the Adder Array.

- DO - Digit serial input of the value of y_1. During the first Basic Cycle (after the START signal), the DO inputs should represent the digit of y_1 with weight 1, with digits having progressively lower weights (1/2, 1/4, 1/8,...) being input during successive Basic Cycles. Since y_1 is represented in redundant form, two input lines are used to represent each digit. Interpretation of these inputs are as follows.

00 =	0
01 =	-1
10 =	invalid
11 =	1

  The selection of this coding was made to facilitate the design of the Adder Array.

- DJ - Digit serial input of Y_{i+1}. During the first Basic Cycle, the DJ inputs should represent the digit of Y_{i+1} with weight 1, with digits having progressively lower weights (1/2, 1/4, 1/8,...) being input during successive Basic Cycles. Since Y_{i+1} is represented in redundant form, two input lines are used to represent each digit as indicated for DO.

- CIN - Inputs the 3 bits of COUT from the next lower order BBM of an EEU. These inputs should be low if this is the lowest order BBM of an EEU (that is, if LOB=1). The need for these inputs is discussed later.

- SIN - Inputs the one bit of SOUT from the next lower order BBM of the EEU. This input should be low if this is the lowest order BBM of an EEU (that is, if LOB=1). The need for this input is discussed later.

The following are the outputs from the BBM.

- DOUT - Digit serial output of the result y_1. This output is only valid if this is the highest order BBM of the EEU (that is, if HOB=1). During the first Basic Cycle, the DOUT output will represent the digit of y with weight 1, with digits having progressively lower weights (1/2, 1/4, 1/8,...) being output during successive Basic Cycles. Since y_1 is represented in redundant form, two output lines are used to represent each digit as indicated before.

- COUT - Outputs 3 bits to the CIN inputs of the next higher order BBM of an EEU. This output is irrelevant if this is the highest order BBM of the EEU (that is, if HOB=1).

- SOUT - Outputs one bit to the SIN input of the next higher order BBM of the EEU. This output is irrelevant if this is the highest order BBM (that is, if HOB=1).

The basic objectives of this proposed BBM design were as follows.

1. Minimize I/O requirements in order to allow for simple module interconnection networks.
2. Design a universal module so that no other module (except possibly memory) is required to realize an EEU.
3. Be compatible with Large Scale Integration techniques.
4. Minimize time required for the Basic Cycle (the iterative step defined by (12)).

The design presented here is not necessarily optimal but does illustrate the feasibility of modular hardware implementation of the E method to perform fast evaluation of functions.

The internal design of the BBM is primarily driven by the need to implement the basic recursion defined in (12). Figure 5 is a block diagram of BBM. Holding Registers for the argument x (XHREG and the p and q coefficients (QHREG and PHREG) are along the top of the diagram. The Active Registers used for the actual computations are XREG, QREG, WSREG, and WCREG. Note that the two registers, WCREG and WSREG, store the $w^{(j)}$'s in the redundant form. The Multiplexer at the input of WSREG and the And Array at the input of WCREG allow the initialization of $w^{(0)}$ with the value of p from XHREG. The Control Logic controls inputs to WSREG and WCREG as well as the use of the DO and DJ inputs.

The multi-operand addition required in (12) is performed in a carry-save manner in the Adder Array with the $w^{(j)}$ result being stored in WSREG and WCREG in redundant form. This eliminates the need to wait for propagation of carries, or the need for carry look ahead logic across the entire precision of the EEU. This is performed in the equivalent of an array of carry-save adders. Alternate implementations of the Adder Array are considered later and in [3].

As mentioned earlier, the required precision of the addition in Equation (12), in order to properly select $d^{(i)}$ (DOUT) is 6 bits to the right of the radix point. Note that the format of the digits in a BBM which is used as the high order byte of an EEU is as follows:

XX.XXXXXX

Thus a complete addition is required only within the high order BBM of an EEU. This addition is performed in the Assimilation Logic. The BBM then performs the digit selection (selection of the correct output digit, DOUT) for the EEU using the Digit Select Logic.

Note that since all digits (DOUT, DO, DJ) have redundant digit sets, inputs to the Adder Array must either be in true or complement form, or zero. Implementation of subtraction with the Adder Array is done by complementing the input and then inserting a carry-in to the array via the TCADJ lines. The Gating Logic, together with the Two's Complement Logic performs this control function. Note that in the case where both q and x need to be subtracted, a TCADJ of 2 is required, meaning that a 1 needs to be added in the second least significant bit of the FEU. Thus, if a BBM is indicated as being the Low Order Byte (LOB=1), there is the provisions for two low order digits to be added in the Adder Array. If a BBM is not the Low Order Byte, the Two's Complement Logic is disabled and always inserts zeroes as carry-ins to the Adder Array regardless of the state of the DO and DJ inputs.

To accomplish subtraction of the digit selected on the previous Basic Cycle, two high order bits must be taken into the Adder Array, providing that the BBM is the High Order Byte (HOB=1) of the EEU. Note that only the high order 2 bits are required to perform this subtraction. ZADJ provides these two bits to the Adder Array. The ZADJ inputs are disabled (always 0) if this is not the High Order Byte. Thus, inputs to the adder array can be envisioned as shown in Figure 6.

The registers, the adder and the selection logic are discussed next. The function of each register is indicated below.

- XHREG - Holding Register for the argument x. This register is loaded from the IEB when LOADX=1 and a clock pulse occurs regardless of the states of LOADE and REGLS.

```
XSGND          XX.XXXXXX

QSGND          XX.XXXXXX

WSOUT          XX.XXXXXX

WCOUT          XX.XXXXXX

ZADJ/TCADJ     XX      XX
               ----------

SUM            XXX.XXXXX

CARRY        XXXXX.XXX
```

FIGURE 6. MULTI-OPERAND ADDITION

- QHREG - Holding Register for the q coefficient. This register is loaded from the IEB when LOADX=0, LOADE=1, and REGLS=0, and a clock pulse occurs.

- PHREG - Holding Register for the p coefficient. This register is loaded from the IEB when LOADX=0, LOADE=1, and REGLS=1, and a clock pulse occurs.

- XREG - Active Register for the argument x. This register is loaded with the contents of XHREG when START=1 and a clock pulse occurs.

- QREG - Active Register for the q coefficient. This register is loaded with the contents of QHREG when START=1 and a clock pulse occurs.

- WSREG, WCREG - Together, these two registers store the value of $w^{(i-1)}$ in redundant form. When START=0 (normal computational mode), these registers are updated with the outputs of the Adder Array, shifted one bit to the left (to effect multiplication by 2), using the SIN and CIN inputs from the next lower order BBM in the proper fashion to fill in the lower order bits. When START=1, $w^{(0)}$ is initialized with the p coefficient which is stored in PHREG. Thus, in this case, the contents of PHREG are loaded into WSREG while WCREG is loaded with zeroes.

- DREG - Stores the value of $d^{(i-1)}$ for output from the BBM during the ith Basic Cycle. The function of forcing $d^{(0)}$ to zero during the first Basic Cycle of each Computational Cycle is performed by the TWO'S COMPLEMENT LOGIC using DIGITE. Note that a value is computed and stored in DREG in each BBM. However, only in the high order BBM (designated by HOB=1) of each EEU does this register contain the proper value of $d^{(i-1)}$.

The Adder Array performs a multi-operand addition producing the result in redundant from as illustrated in Figure 6.

Note that the required result of this multi-operand addition is in a format enabling the use of (5,5,4) and (4,4,4) counters [7]. A possible implementation of such a (5,5,4) counter using a logic array is discussed in [3].

The Assimilation Logic is responsible for producing a non-redundant result from the output of the Adder Array. A classical implementation of this logic is simply an adder (with or without carry look-ahead logic). Array logic implementation of this Assimilation Logic and the Digit Select Logic is discussed later. It should be noted that although this logic is present on every BBM and in fact always produces a result, its result is only meaningful

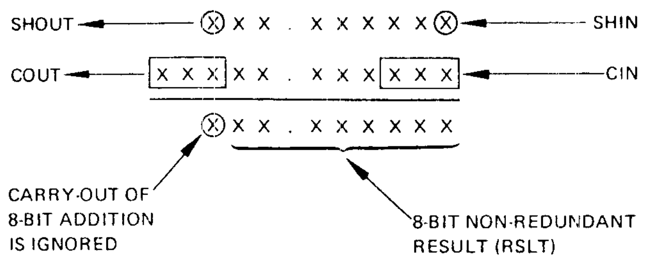
SHOUT
SHIN
COUT
CIN
CARRY-OUT OF 8-BIT ADDITION IS IGNORED
8-BIT NON-REDUNDANT RESULT (RSLT)

FIGURE 7. ASSIMILATION LOGIC FUNCTION

in the high order BBM. The function of the Assimilation Logic is illustrated in Figure 7.

The Digit Select Logic actually selects the digit to be output from the EEU according to equation (8). The inputs and therefore the outputs from this logic are meaningful only in the high order BBM of each EEU. An implementation of the combination of the Digit Select and Assimilation Logic is discussed in [3].

The additional details of the internal design of the BBM are given in [3]. A brief discussion of some alternatives in the BBM design follows. The data input of the BBM provides for transfer of three 8-bit words before each computational cycle. Selection of the 8-bit wide bus (IEB) represents a compromise between the input bandwidth (maximized using a 24-bit wide bus) and the number of interconnections (minimized using a 1-bit wide bus).

The Adder Array can be implemented using the following approaches:

(1) An array of carry-save adders or (3,2) counters;
(2) An array of ROM modules implementing (5,5,4) and (4,4,4) counters;
(3) Logic arrays implementing (5,5,4) and (4,4,4) counters.

The carry-save alternative requires 20 FA's and 4 HA's with the delay of about 12 gate levels and about 250 active elements. The ROM approach requires two (1024x4) and two (256x4) ROM modules or an equivalent ROM of 10,240 bits with 36 inputs and 16 outputs. The alternative based on logic arrays requires on the order of 1000 active elements. A detailed analysis of the logic array approach which is definitely superior to the ROM implementation is given in [3]. On the basis of the required number of active elements, the carry-save approach was selected. Similar analysis of alternatives in implementing Carry Assimilation and Digit Select Logic were performed in [3] and the conventional approach was selected.

VI. PERFORMANCE ESTIMATES

An attempt has been made to provide an estimate as to the speed, power consumption, and size of the proposed BBM. Table 1 breaks down the number of equivalent gates and gate delays required for each function within the BBM. The estimated number of gate delays is for the longest path through the logic function. Specifically, this table assumes use of the Hughes Integrated Injection (I^2L) technology in late 1978. Implementation of the Adder Array, and Assimilation and Digit Select Logic is that of the conventional approach. No logic arrays are used in this estimate. The Basic Cycle time is estimated to be 50 nanoseconds.

FUNCTION	ESTIMATED NUMBER OF GATE DELAYS	TOTAL NUMBER OF COMPONENTS
LOAD SELECT	1	3
CONTROL	1*	11
MULTIPLEXER	3	30
AND ARRAY	2	16
GATING LOGIC	3*	38
TC LOGIC	2*	8
SDC	2	4
ADDER ARRAY	12*	268
ASSIMILATION LOGIC	11*	56
DIGIT SELECT	3*	10
REGISTERS	1*†	580**
TOTALS	33	1024

* INDICATES THOSE FUNCTIONS USED IN ESTIMATING DELAY THROUGH BBM
† INCORPORATED TO ACCOUNT FOR "SET-UP" TIME
** ASSUMES SEVEN 8-BIT REGISTERS AND ONE 2-BIT REGISTER (AT 10 COMPONENTS PER REGISTER)

TOTAL DELAY = (33 GATE DELAYS) x (1.5 ns/GATE DELAY)
= 50 ns

TOTAL CHIP AREA $\simeq$ 5 mm^2 ($\approx$8000 mil^2)
(ASSUMING $\simeq$200 COMPONENTS/mm^2)

TABLE 1. ESTIMATES OF BBM SIZE AND SPEED

Note that the estimated chip size does not include input/output buffer circuits or connector pads which will increase the final chip area. Note, however, that the estimated size and complexity of the chip are well within the realm of current technology.

Using an estimate of 150 microwatts per component, the above I^2L chip would consume approximately 1.54 watts running at maximum speed. This is also within the realm of current technology.

Note from Figure 5 that 29 I/O lines are required by the BBM implying that at least 31 pins on an IC package would be required for an LSI BBM (including two pins for chip power and ground). Currently, 40 pin Dual-Inline packages are used extensively by the microprocessor industry. Thus, the proposed BBM does not require an excessive number of I/O connections for LSI implementation.

As noted earlier, the 8000 square mil chip size is somewhat below the current LSI technology capability. Thus, assuming the availability of more useable surface area, design of the proposed module can be extended to accommodate additional bits in a straight forward manner. It would suffice to extend the "middle" portion of the 8-bit slice to provide the desired number of bits. The required chip area would be nearly a linear function of the number of bits desired. The number of input/output pins on the module appears to be the limiting factor for this expansion. Additional chip area could also be used to incorporate on-chip storage of coefficients with either a ROM or RAM implementation. The latter use of additional silicon area seems to be the more desirable approach since it eliminates the need for an additional chip type (memory) when implementing an FEU.

The use of logic arrays in the implementation would provide a more regular structure on the chip. However, no estimate of the chip area required by such an implementation can be given at the present time.

Using such a high-speed BBM would place fairly strict requirements on the ROMs used in the proposed FEU implementation. However, fairly large ROMs are currently available with address-to-output delays within the required range.

VII. CONCLUSION

The design presented in this paper indicates that implementation of the E-method in hardware is straight forward. The estimated size, power consumption and complexity of the Basic Byte-Slice Module (BBM) are well within the capabilities of current technology. Using 1978 technology, an FEU designed using this proposed BBM would be capable of providing a 64 bit redundant result in about 3 microseconds. In such a case, the implementation of a unit for evaluation of rational functions up to degree 8 would require 72 BBMs and 72 ROMs or 144 ICs.

The major advantages of the function evaluation unit based on the E-method are (a) a large domain and flexibility in specifying the functions to be evaluated, (b) simple control and variable precision capability, (c) high modularity, uniformity and ease of interconnection. Such a unit would require only a simple interface with the other subsystems. If a switching network is provided between BBMs and ROMs, a reconfigurable system with fault-tolerant capabilities can be realized. In particular, the inter-module signals can be effectively protected by low-cost error-codes [9].

The proposed BBM could be used, with minor modifications, to perform evaluation of a given rational function of degree n when the number of EEUs is smaller than n or when the precision requires more BBMs per EEU. In either case, the relationship between the number of BBMs and the evaluation time is linear [1,2]. The FEU is also capable of performing the basic arithmetic operations and the evaluation of certain arithmetic expressions [1].

The main limitations of the described approach is its scaling requirements and its restriction to the fixed-point number system. However, these limitations are not critical in the case of function evaluation where the coefficients can be scaled a priori.

REFERENCES

[1] M.D. Ercegovac, "A General Method for Evaluation of Functions and Computations in a Digital Computer," Ph.D Dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, TR 750, July 1975.

[2] M.D. Ercegovac, "A General Hardware-Oriented Method of Evaluation of Functions and Computations in a Digital Computer," IEEE Transactions on Computers, Vol. C-26, No. 7, pp. 667-680, July 1977.

[3] M.M. Takata, "A Design of A Modular Arithmetic Unit for Polynomial and Rational Function Evaluation," M.S. Thesis, UCLA Computer Science Department, University of California, April 1978.

[4] J.E. Robertson, "A New Class of Digital Division Methods," IRE Transactions on Electronic Computing, Vol. C-26, No. 7, pp. 667-680, July 1977.

[5] A. Avizienis, "Signed-digit Number Representations for Fast Parallel Arithmetic." IRE Transactions of Electronic Computing, Vol. EC-7, pp. 218-222, Sept. 1958.

[6] W.J. Stenzel, W.J. Kubitz, and G.H. Garcia, "A Compact High-Speed Parallel Multiplication Scheme," IEEE Transactions on Computers, Vol. C-26, No. 10, pp. 948-957, October 1977.

[7] C. Tung, "A Combinational Arithmetic Function Generation System," Department of Engineering, University of California, Los Angeles, June 1968.

[8] C. Tung, A. Avizienis, "Combinational Arithmetic Systems for the Approximation of Functions," AFIPS Conference Proceedings, Vol. 36, pp. 95-107, SJCC 1970.

[9] A. Avizienis, "Arithmetic Algorithms for Error-Coded Operands," IEEE Transactions on Computers, Vol. C-22, No. 6, pp. 567-572, June 1973.