Evaluation of Floating-Point Sum or Difference of Products in Carry-Save Domain

A. Wahab, S. Erdogan and A. B. Premkumar
School of Applied Sciences
Division of Computer Technology
Nanyang Technological University
Nanyang Avenue
Singapore 2263
ASABDUL@NTUVAX.BITNET, (65) 7994948

Abstract - **An architecture to evaluate a 24-bit floating-point sum or difference of products using modified sequential carry-save multipliers with extensive pipelining is described. The basic building block of the architecture is a carry-save multiplier with built-in mantissa alignment for the summation during the multiplication cycles. A carry-save adder, capable of mantissa alignment, correctly positions products with the current carry-save sum. Carry propagation in individual multipliers is avoided and is only required once to produce the final result.**

1 Introduction

In evaluating sum of products, the mantissas of the current sum-of-products and the newly examined product are compared. The mantissa of the one with the smaller exponent is aligned first and then added to the mantissa of the other one. One alternative is to use fast shifting units for mantissa alignment. However the cost of such units is prohibitive and may well slow down the overall speed of the system if pipelining has to be used. The technique used in the proposed architecture is to incorporate shifting capability into the multiply/add unit, thus eliminating use of conventional shifters. Sequential carry-save multipliers are used for evaluating the product of the mantissas involved in the floating-point operation [1]. For an n by n multiplication, the result is available in carry-save form after n cycles. Conventional result can be obtained through n additional cycles or a fast carry propagation unit. Since the sum of products evaluation is the main concern in the proposed architecture, carry propagation is deferred until the final stage. Thus, carry propagation is avoided in individual product evaluations. A high throughput is achieved by incorporating the mantissa alignment into the product evaluation cycles.

2 Floating-Point Sum-of-Products Evaluation in Carry-Save Form

The summation of products involves the multiplication of multiplier and multiplicand pairs and subsequent addition of these results. In the floating-point domain, the multiplication itself involves addition of the exponents of operands and multiplication of the mantissas.

Carry-save techniques have been proposed to reduce the timing penalty associated with the propagation of carry during the mantissa multiplication operation.

The multiplication of two floating-point numbers in carry-save domain is achieved by performing a carry- save multiplication of their mantissas and adding their exponents. The exponent part of the product is obtained by the addition of the exponents. The mantissa of the product is the most significant half of the result in carry and sum registers. For a conventional 16-bit mantissa carry-save multiplier, the result is obtained in 16 clock cycles. Additional 16 clock cycles are necessary to produce a conventional result. A fast carry-propagation circuit could also be used for this purpose. However, if the multiplier is to be used in evaluating sum-of-products, the propagation of carry can be deferred till the final stage. If the products in carry-save form are to be used for the evaluation of sum-of-products, carry-save results are satisfactory and timing penalty due to propagation of carry only occurs once as opposed to m times for a sum-of-products with m terms.

The results obtained from two multipliers can be added to produce a carry-save result provided their results are of the same exponent (i.e., their mantissas are correctly aligned first). Further, addition of other products to the current sum is possible provided their mantissas are properly aligned. However, special care must be taken to ensure that the summation result does not overflow.

Let us consider the evaluation of F=A+BW, where A, B, F and W are 24-bit floating-point numbers (16-bit mantissa, excess 64 exponent). First the sum of the exponents of B and W must be obtained. If the exponent of BW is smaller than the one of A, the mantissa of the product should be adjusted prior to addition with A, otherwise the mantissa of A should be adjusted. The mantissa associated with the lowest exponent should be shifted right by the difference in the exponents and added to the other mantissa. In the worst case, a shift of the mantissa by 15 bit positions may be required. The resulting exponent is the higher of the two exponents. In carry-save representation, a correction must be applied, if a carry-out is produced from the most significant digit [2].

One alternative to achieving mantissa alignment is to provide a fast shifting unit. The cost of such a unit is prohibitive and may slow down the overall speed of the system if pipelining is to be used. A 16-bit shifter would require 16×16 connections and sixteen 16 to 1 multiplexers. Since shifting is performed at the end of the multiplication prior to summation a timing penalty occurs. The other alternative would be to incorporate bit shifting capabilities into the summation and multiplier units such that no dedicated shifter would be necessary to sum the products. Furthermore, if this could be achieved with little or no timing penalty during the standard carry-save cycles, a number of multiplier units could be run in parallel and very high summation rates could be obtained. To achieve massive pipelining, multipliers operate on different sets of data and their results are added through a dedicated carry-save adder.

Although the architectural details of an implementation are given in Section 4, the control requirements associated with individual multipliers and the summation unit to allow for massive pipelining are examined below:

1 When a new multiplier and a multiplicand are selected for multiplication, the product's exponent is determined by adding their exponents and is compared with the exponent

of the current sum-of-products.

2.a One in the current sum-of-products, then the multiplication is performed and resulting mantissa is added to the current sum-of-products after necessary mantissa alignment.

2.b If the exponent of the product is higher than the one in the current sum-of-products, then the mantissa of the current sum-of-products is adjusted to its correct position. Since the exponent of the product can be precomputed, the summation unit has 16 clock cycles to correctly align its mantissa with respect to the product while the multiplication is being performed.

The above requirements can be achieved by using a dedicated shifter unit and many multipliers. Pipelining maximizes the usage of the fast shifter. Another alternative is to incorporate mantissa alignment into the multiplication cycles. To achieve this objective, a concept called "global exponent" that is basically the exponent of the current sum-of-products, is proposed in this paper. Instead of recording the exponent of individual products currently being evaluated, their relative difference to the global exponent is recorded. Every multiplier unit and the summation unit have a counter. Since we are dealing with 16-bit mantissas, the maximum relative difference is 15. A unit is required to align its carry-save value one bit at a time as long as its counter is positive. The hardware realization of this alignment for the multiplier and summation is presented in Section 3.

Initially, the global exponent is the first product's exponent and is subsequently updated. The succeeding products are examined one by one, and the relative difference between their exponent and the global exponent are evaluated. The shifting requirement is tagged to the assigned multiplier unit to ensure correct mantissa alignment at the end of 16 clock cycles. For an exponent difference of 16 or above, the multiplier can be set to zero and no further intervention would be necessary. Some sophisticated buffering techniques can also be developed not to schedule this product evaluation at all. The multiplier could therefore operate on a set of operands leading to a significant result. We have considered such an optimization, however, it has not been discussed in the current paper.

Another point of concern is the overflow out of the summation unit. When an overflow is detected, all counters in the system and the global exponent are incremented by one to ensure consistency of mantissa alignment and global exponent. The overflow can be detected before it occurs by examining the most significant carry and sum bits. A correction action is initiated by incrementing all counters in the system and the global exponent.

3 Built-In Mantissa Alignment for Floating-Point Additions

In order to achieve the alignment requirement associated with the sum-of-products, two units are of special interest. The first, is a floating-point multiplier that incorporates alignment into the multiplication cycles and the second is a summation unit that allows for alignment. We first concentrate on the design of the former. The design of the latter is fairly trivial and is described later in this Section.

The conventional carry-save floating-point multiplier as described earlier generates a product in 16 clock cycles for a 16-bit mantissa. To incorporate mantissa alignment into the multiplication operation, this architecture is modified in order to produce a correctly aligned carry-save result. This result is ready for subsequent addition to the current sum-of-products without further manipulation. In the proposed sum-of-products architecture, the alignment requirement of a multiplier unit is determined by a counter. In a 24-bit floating-point environment (16-bit mantissa, excess 64 exponent), a 4-bit counter is necessary to record the shifting requirement. The value of the counter is set prior to the start of a multiplication and is subsequently modified to reflect the requirements of the system.

To achieve alignment required for subsequent summation, the multiplier could be shifted right as many positions as required prior to the actual multiplication cycles and then the multiplication could proceed with the remaining bits of the multiplier. Since fewer bits of the multiplier are examined (starting with the most significant ones) reduced precision may result. Furthermore, the sum-of-products evaluation approach as described in Section 2, requires a multiplier to shift its current product in the middle of a multiplication. This can not be handled simply by preshifting the multiplier prior to the start of the multiplication.

Figure 1 shows a carry-save floating-point multiplier with built-in mantissa alignment capability. The multiplication follows the conventional carry-save algorithm. However additional paths are created to achieve mantissa alignment while multiplication is being performed. Unlike the approach proposed above, the multiplier is examined exactly as in the conventional carry-save algorithm and the multiplication is performed accordingly. However, when shifting is required, the partial product is added to the current product and is fed one bit further right compared to the original implementation. This new configuration ensures correct alignment of the mantissa as specified at the end of the multiplication cycles. Further, to conserve the integrity of the multiplication process, the multiplicand is also shifted by one bit right when a shift is required.

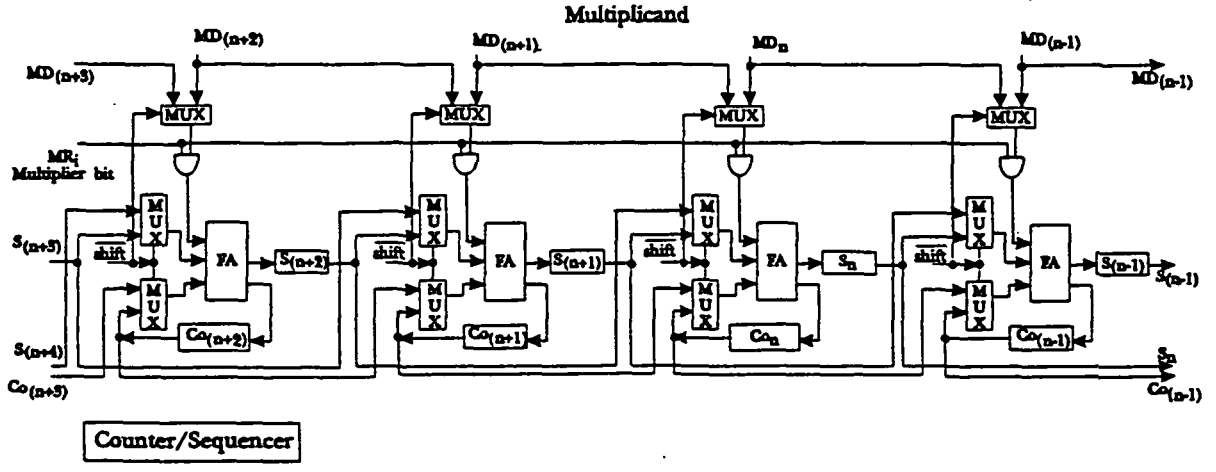

Figure 1: Block diagram of a carry-save multiplier with built-in mantissa alignment capability

Figure 2 shows a carry-save adder for adding the mantissa of products with built-in mantissa alignment capability. N is the number to be added to the current carry-save quantity.

When alignment is not needed, the unit adds an ordinary number to an existing carry-save number. When alignment is required, the carry-save result is fed one unit right compared to the original configuration. This is similar to the approach used in the implementation of the multiplier with built-in mantissa alignment. Two clock cycles are necessary to add the carry and sum components of a recently evaluated product.

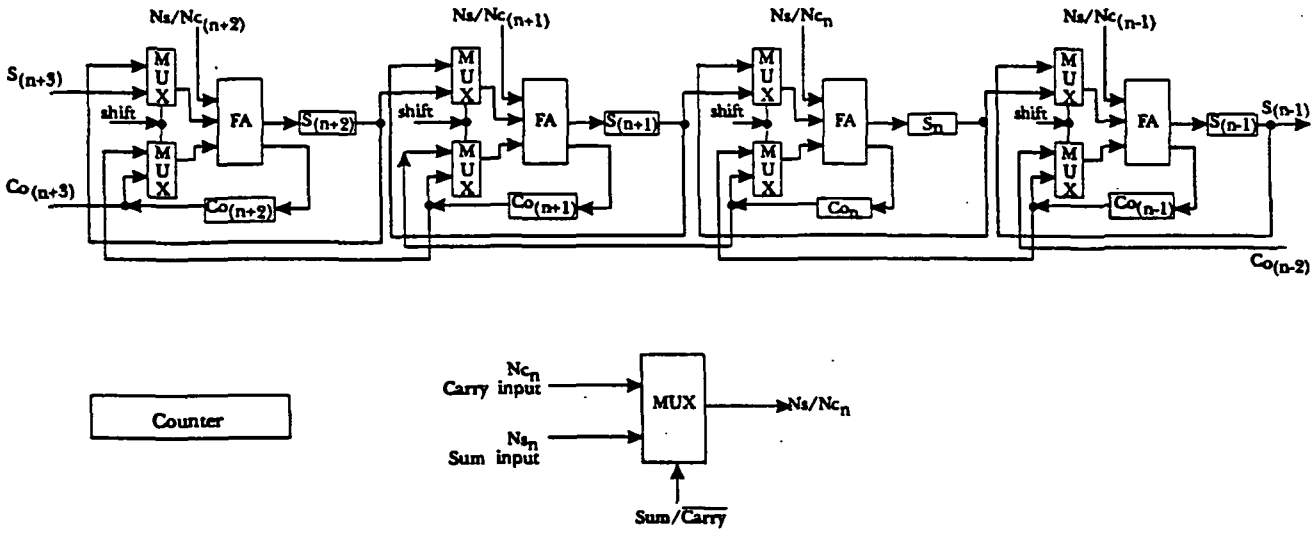

Figure 2: Block diagram of a carry-save adder for summing of products with built-in mantissa alignment capability

4 Architecture for Sum-of-Products Evaluation

The architecture presented in this paper is a novel approach to explore carry-save representation for sum- of-products evaluation as described in Section 2. The two components that are of special concern in this architecture are: a floating-point carry-save multiplier that can align the mantissa of a product during regular multiplication cycles (ready for subsequent summation) and a special floating-point adder with a mantissa alignment capability to sustain continuous addition of products in carry-save domain.

Figure 3 shows the architecture of the sum-of-products module. The module is composed of 8 floating-point multipliers with built-in mantissa alignment capability. The counters associated with individual multiplier units are counter/adders to store the shifting requirements as described in Section 3. Delta exponent is obtained by subtracting the exponents of the product currently examined from the global exponent. When delta exponent is positive (the exponent of a product to be evaluated is less than the global exponent), then the counter of the multiplier is set to reflect this difference. Otherwise, all counters in the system except the one of multiplier unit assigned to the current product evaluation are incremented by that difference.

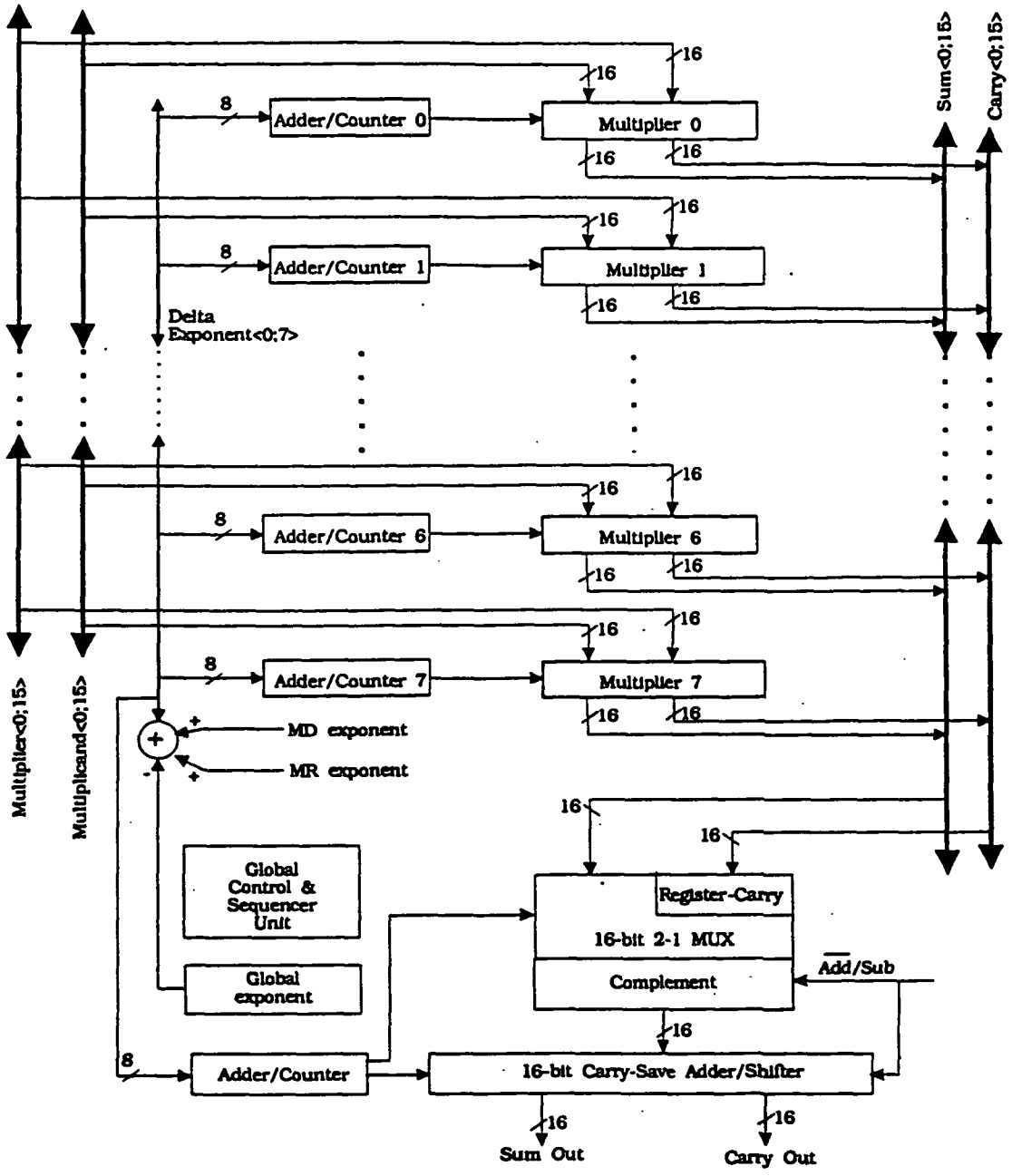

Figure 3: Architecture of a 24-bit floating point (16-bit mantissa, 8-bit exponent) Sum of Products system.

Figure 4 shows the timing characteristics of the architecture. The multiplicand and the multiplier pairs are fed to each multiplier with a phase difference of two clock cycles to ensure that their results can be summed without unnecessary buffering. Depending on the implementation platform, the exponent of the product may need to be pre computed since only two cycles are available for implementing the counter settings in the system. A multiplication result is output to the bus every two clock cycle. Each multiplier requires 16 clock cycles to produce a result.

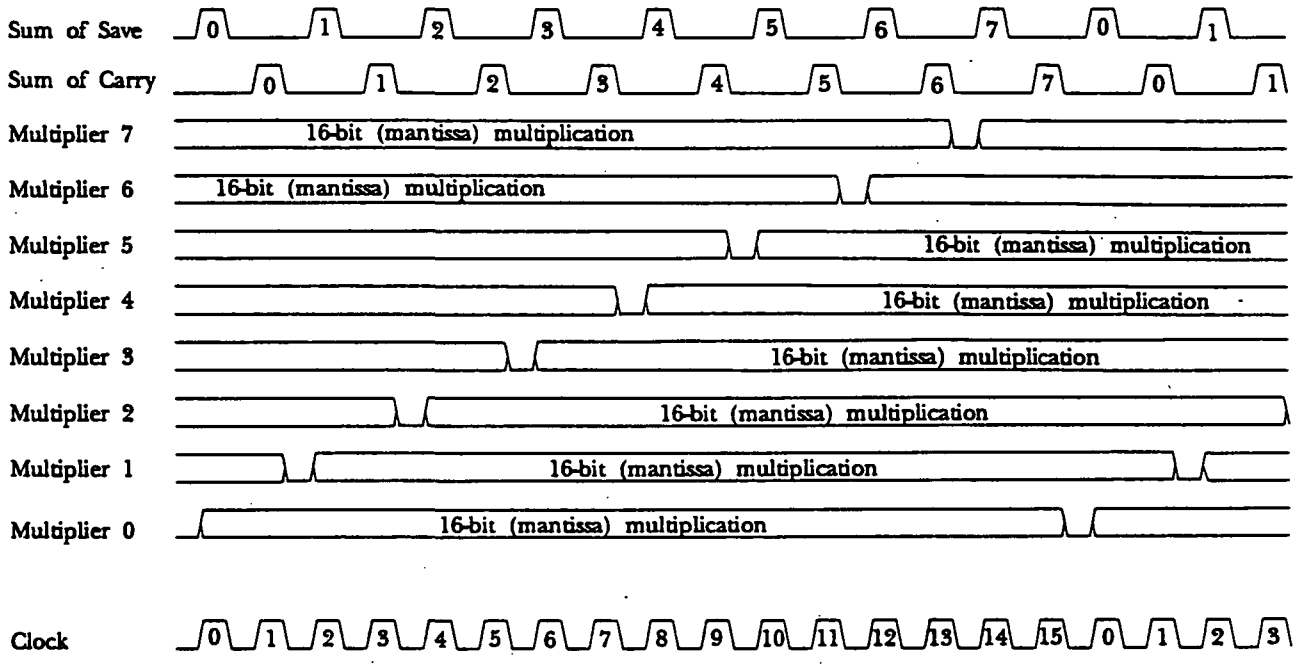

Figure 4: Timing diagram of the pipelined Carry-save Sum of Products architecture

The results obtained through the multipliers are fed through a wide bus to a summation circuit. The 32-bit bus allows for the transfer of a result in one clock cycle to achieve the desired 16-cycle pipelining. The sum component is fed directly to the summation circuit and shifted while being added as required by the counter associated with it. The carry component of the mantissa is moved to a shift register that ensures that its alignment is correct by following the control applied on the summation circuit. The numbers associated with the clock cycles in the summation unit indicate which multiplier is responsible for a particular result.

The carry-save adder shown in Figure 2 is used to accumulate the sum-of-products in carry-save form as required. A radix-2 carry-save configuration is sufficient since a new product is generated every two cycle. When an overflow out of the summation unit is detected, all counters and the global exponent are incremented by one to conserve floating-point representation. Summation of negative products is also supported. Since products are evaluated using positive mantissas, negative products are summed by using one's complement of their mantissas to avoid carry propagation associated with two's complement. Two's complement is later achieved in the summation unit by forcing a "one" to the least significant

carry-in bit of the carry-save adder.

The final result in conventional form is obtained when there are no more products to be summed by propagating the carry. This is achieved by feeding the carry and sum components of the summation through an additional fast carry propagate circuit. The same result can be obtained by allowing the carry to propagate through the summation unit. The 16-bit sum component becomes the conventional result when carry is allowed to propagate for an additional 16 clock cycles. The global exponent is the exponent of the final result and is directly available from the circuit.

The architecture presented in Figure 3 uses 8 multipliers to achieve a throughput of one product summation every two clock cycle. A doubling of this performance is possible by using a radix-4 carry-save adder to accumulate summation results. Sixteen carry- save multipliers or 8 radix-4 multipliers could be used to sum a product every clock cycle in conjunction with this wide adder [3]. Dual prefetching of operands may be necessary to allow for the examination of exponent of a product to meet the timing requirements. Time/hardware tradeoff should be considered for a given implementation environment for maximum efficiency [4].

5 Conclusions

The proposed architecture achieves effective evaluation of floating-point sum-of-products in carry-save domain. A product summation every two clock cycle is performed at steady state at moderate cost by pipelining 8 floating-point carry-save multipliers. Better performance can be achieved by performing 2 bit decode of the multiplier but at the cost of increased complexity of the control. The performance of the proposed architecture may be further increased by detecting and aborting multipliers that will produce insignificant results with respect to the current summation value. Further performance increase is also possible by detecting and not scheduling those products with very low exponents with respect to the global exponent at a very early phase. The merits of such a system will depend on data and should be considered in applications such as artificial neural network simulation hardware [5].

The multiplier with built-in mantissa alignment capability and the special carry-save adder described in this paper can also be used in other environment that requires high performance computing [6]. The carry-save result of the multiplier is useful for applications such as Fast Fourier Transform and complex function evaluations where further computations are necessary prior to the output [7,8]. This concept is similar to the residue-number-system concept where an intermediate form of a number is generated to perform arithmetic operations more effectively [9]. However, we believe that carry-save representation offers a better alternative in sum-of-products evaluation, since the carry-save form is obtained as an intermediate form in the carry-save multiplication and does not require a costly initial conversion [10]. The final conventional result can also be obtained without additional hardware.

References

[1] J. J. F. Cavanagh, Digital Computer Arithmetic, Design and Implementation. McGraw-Hill, 1984.

[2] M. R. Santoro, G. Bewick, M. A. Horowitz, "Rounding Algorithms for IEEE Multipliers", in Proc. IEEE 9th Symp. on Computer Arith., pp. 176-183, 1989.

[3] S. S. Erdogan and Abdul Wahab, "Design and Implementation of a Radix-4 Carry-save Multiplier", submitted to IEEE Trans. on Comp. (December 1991).

[4] A. G. Ferreira, "A Parallel Time/Hardware Tradeoff for the Knapsack Problem", IEEE Trans. Comput., Vol. 40, No. 2, pp. 221-225, February 1991.

[5] "Design of RM-nc: A Reconfigurable Neurocomputer for Massively Parallel-Pipelined Computations", Proc. of the IEEE International Joint Conference on Neural Networks '92 Baltimore, U.S.A., Vol. 2, pp. 33-38, 7 - 11 June, 1992.

[6] "Floating-point Fast Fourier Transform Evaluation in a Parallel-Pipelined Carry-Save Architecture", accepted for publication in the proceedings of the ICARCV '92, Singapore, 15-18 September, 1992.

[7] P. Kornerup, D. W. Matula, "Exploiting Redundancy in Bit-Pipelined Rational Arithmetic", in Proc. IEEE 9th Symp. on Computer Arith., pp. 119-126, 1989.

[8] R. H. Brackert, M. D. Ercegovac, and A N Wilson, "Design of an On-line Multiply-Add Module for Recursive Digital Filters", in Proc. IEEE 9th Symp. on Computer Arith., pp. 34-41, 1989.

[9] F. J. Taylor. "Residue Arithmetic: A tutorial with Examples", IEEE Comp. Magazine, pp. 50-62, May 1984.

[10] K. M. Ibrahim and S. N. Saloum, "An efficient Residue to Binary Converter Design", IEEE Trans. Circuits Syst., Vol. 35, pp. 1441-1444, November 1988.