INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

On Synthesis and Optimization of Floating Point Units

Syed Yawar Ali Shah

A Thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montreal, Quebec, Canada

Abstract

On Synthesis and Optimization of Floating Point Units

Syed Yawar Ali Shah

This work describes the effect of architectural/system level design decisions on the performance of floating point arithmetic units. By modeling with VHDL and using design synthesis techniques, different architectures of floating point adders, multipliers and multiply - accumulate fused units, are compared using different technologies and cell libraries. Some modifications to recent published works have been proposed to minimize the energy delay product with special emphasis on power reduction. A new low power, high performance, transition activity scaled, double data path floating point multiplier has been proposed and its validity is proved by comparing it to a single data path floating point multiplier. A transition activity scaled, triple data path floating point adder has been compared with a high speed, single data path floating point adder using an optimized Leading Zero Anticipatory logic. Three different architectures of floating point multiply - accumulate fused units are evaluated for their desirability for high speed, low power and minimum area. The findings of this work validate different higher level design methodologies of floating point arithmetic units irrespective of the rapidly changing underneath technology.

Acknowledgments

I am extremely grateful to my Professor Dr. A. J. Al-Khalili for his supervision and support. Without his continuous guidance and invaluable feedback, this work would not have been possible. I would like to thank Dr. R. V. K. Pillai for his suggestions for floating point architectures and Wasim Tout and Ted Obuchowicz for their help in CAD tools. I am indebted to my parents and my brothers, whose prayers and continuous support were always with me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Floating point units are indispensable for the current high performance microprocessors and DSPs. Small dynamic range of fixed point operations limits their desirability for applications where programmers have to put extra efforts for the proper scaling of numbers to maintain sufficient precision and prevent overflows. This work addresses the evaluation of architectural level design approaches for the implementation of low power, high speed floating point adders, multipliers and multiply - accumulators (MACs).

Although full custom implementations and technology specific design optimization at the circuit level is a clear choice for the low power, high speed digital design, the magnitude of power reduction and speed enhancement attainable through higher level approaches is quite significant. The main reason for the exploration of higher level design optimization strategies is the reusability of architectures and algorithms for different technologies. The evaluation of architectural/system level design strategies that are insensitive to technology migration provide results that don't perish in short time.

1.2 Motivation

In applications like digital signal processors (DSPs), microprocessors and computer graphics, the wide dynamic range capabilities of floating point operands is strongly desired. In the past, only large computers were equipped with floating point math-proces-

sors but with the advances in VLSI technology, these days even small computers can perform floating point computations and new CPUs are equipped with floating point units. With the increasing popularity of multimedia entertainment using three-dimensional graphics and the visualization of scientific simulations, computer graphics require floating point units to obtain high quality images.

Although conventional instruction driven CPUs and DSPs offer flexibilities in terms of programmability and re-configurability etc., many new applications demand computational speed which is not attainable by them. Also power consumption for the supporting hardware of instruction driven CPUs like control units, instruction decoders, bus interface units, cache memory, on chip RAM/ROM memories etc., make them suboptimal. So the dedicated system solutions are becoming increasingly popular [1]. When general purpose floating point processors/DSPs are claiming performance heights of a few hundred MFLOPS, dedicated systems have already reached BFLOPS capabilities [2]. The primary concerns of embedded system designers are the performance specifications, area requirements and power budget. They define appropriate data formats, analog/digital or mixed signal design strategies, hardware/software partitions such that the final solution meets the required performance and power goals. Since the integrated circuits generated are application specific (ASICs), the economy of scale and production volumes may not support exhaustive full custom design approaches. Also, the design turnaround time should be minimal in such situations [1]. In this scenario, the use of modern day logic synthesis tools is the viable choice. The evaluation of floating point architectures for power/performance measures, using these tools, gives us a clear picture about their applicability in DSPs and microprocessors.

1.3 Low Power Digital Design and Logic Synthesis

As much of the research efforts in the past twenty years are directed toward increasing the speed of digital systems, present day technologies possess computing capabilities that make possible powerful work stations, sophisticated computer graphics, and multimedia capabilities. But the new digital applications like digital cellular telephones, which employ complex speech compression algorithms and sophisticated radio modems in a pocket-sized device, and notebook and laptop computers, which demand same computational and performance capabilities as found in desk-top machines, require methodologies for low power design.

Power dissipation in digital CMOS circuits can be described as

$$P_{total} = p_t C_L f_{clk} V_{dd}^2 + I_{sc} V_{dd} + I_{leakage} V_{dd} \text{ ..... } (1.1)$$

The first term of equation (1.1) [3] represent the Switching Power which is dissipated by charging and discharging of the load capacitance C_L. Where f_{clk} is the clock frequency, V_{dd} is the power supply voltage and p_t is the probability that a power-consuming transition occurs (the activity factor) and it is equal to $p_g(1-p_g)$, where p_g is probability of finding a logic high at the node under consideration.

The second term is Short-circuit Power which is due to the direct-path short circuit current I_{sc}, which arises when both the NMOS and PMOS transistors of a gate are simultaneously active, conducting current directly from supply to ground. It is also called internal power. These first two terms comprise the Dynamic Power i.e. power dissipated when the circuit is active.

The third term of equation (1.1) represents the Leakage Power. It is the result of source to drain subthreshold leakage current, $I_{leakage}$. This leakage is caused by reduced threshold voltages that prevent the gate from completely turning off. Leakage power is also dissipated when current leaks between the diffusion layers and the substrate and is also referenced as Static Power i.e. power dissipated by a gate when it is not switching.

Switching power comprises 70 - 90% of the power dissipation of an active CMOS circuit [4], thus low power design becomes task of minimizing p_t, C_L, V_{dd} and f_{clk}, while retaining the required functionality.

Low power design approaches reduce the operational power demand through a variety of techniques. It is possible to implement a certain Boolean function in different ways and different implementations can consume different amounts of power. Selection of power optimized implementation by estimating the respective switching activity is a viable approach in the VLSI design of power critical applications. Reduction in operating voltage is another approach to save power. Architectural level power reduction approaches include algorithmic optimizations, technology scaling, clocking strategies, etc. In design synthesis for low power, design decisions related to algorithms, architectures, number systems, clocking strategies and problem redefinitions etc. are applicable while circuit level design decisions are not, because the final implementation may use any available target technology and/or cell libraries [1].

Kurt Keutzer et al. [5] discuss the possibilities of power reduction at various levels of design synthesis, which are summarized in Table 1.1 [1].

Table 1.1: Expected power savings in logic synthesis at various levels of design flow

Level	Transformation	Expected Power Saving
Algorithmic	Algorithm selection	Orders of magnitude
Behavioral	Concurrency	Several times
Register Transfer Level	Structural transformations	10 - 15%
	Clock control	10 - 90%
Data/signal encoding		20%
Technology independent	Extraction/decomposition	15%
Technology dependant	Technology mapping	20%
	Gate sizing	20%
Layout	Placement	20%

The right algorithm gives the maximum amount of power saving which can be orders of magnitude in comparison with other alternatives. Instruction driven architectures versus embedded systems can be considered an example for this type of design decisions. In behavioral level power optimization, the possibility of saving is again several times in comparison with other approaches. RTL level approaches can give 10 to 90% savings while data/signal encodings can save another 20% power. The example of behavioral and RTL level transformation is the use of concurrency and pipelining. These approaches give significant speed enhancements, which can be traded for power, if desired. A proper design of clocking scheme (e.g. multi phase vs. single phase clocking) can save a substantial amount of power. Activities like extraction/decomposition, technology mapping, gate sizing, placement etc. are best handled by the synthesis tools. So more and more designers are relying on them for low level optimization. Efficient power management strategies and

activity reduction through logic minimizations, architectural modifications, low power encodings etc. also result in power-efficient logic implementations [1].

The above discussion show that by using logic synthesis tools, a considerable amount of power saving can be achieved by utilizing correct algorithm/architecture/clocking, without going to circuit level technology dependent details.

1.4 Floating Point Units

The computational algorithms for the realization of floating point operations and the floating point hardware had been around for many years. Early floating point units were built as part of large computers where different implementations were following different floating point number systems. With the introduction of IEEE standards for floating point arithmetics [6], more and more manufacturers are following these standards for the design of their hardware units.

Traditionally, floating point units had been implemented as a combination of integer units. The reason is the availability of good knowledge about integer arithmetic units and the optimized integer building blocks in the form of standard cells. References [7], [8], [9], [10], presents some approaches for high speed addition. A considerable work about high speed multiplication is reported in references [11] through [20]. Many new architectures of integer multipliers have been proposed in [21], [22], [23], and [24], for the significand multiplication of IEEE single and double precision floating point multipliers. A new design for a fast barrel shifter is reported in [25].

In reference [26], the development of a floating point cell library has been reported. The

library contains cells of floating point ALU, floating point multiplier, instruction RAM and register file. These cells can be used for the design of instruction driven processors. Reference [27], presents the implementation of a CMOS floating point processing unit for superscalar processors. Reference [28], reports the design of a high speed floating point multiplier for computer graphics applications.

Some algorithms for high speed rounding of IEEE floating point multipliers are presented in [29]. In reference [30], a super-scaler processor with a floating point unit, which uses both IEEE single and double precision data formats, is reported. It also contains two floating point MACs.

The IBM RISC/6000 reported in [31] and [32] contains the first floating point unit with multiply - accumulate fused architecture. The throughput improvement offered by this scheme is quite attractive. A compound normalization and rounding step is performed instead of separate normalization and rounding for floating multiplication and addition. The authors also present a new leading zero anticipation scheme (LZA) [33] which facilitates concurrent evaluation of significand addition and computation of normalization shift requirements (instead of sequential leading zero counting) so that the latency of the floating point unit is reduced. Suzuki et al. [34] reported a new LZA scheme that outperforms the scheme reported in [33], both in terms of speed and area. The LZA logic is using a simple boolean algebra structure and can be realized by fast logic circuits. The paper also suggested a five stage floating point adder, using new LZA logic, in which normalization shift is performed in parallel with rounding operation.

Recently J. D. Bruguera et al. [35] proposed a Leading - One Prediction (LOP) logic

which can correct the wrong anticipation by earlier LZAs [33] [34] by one bit, in parallel with LOP.

R. V. K. Pillai etc. reported some significant work on the low power, floating point data path architectures in references [1] and [36]-[51]. References [36] through [44] present a new low power, transition activity scaled, triple data path floating point adder with its validity for different DSP applications. The design of a floating point accumulator is reported in [45]. Reference [46] presents the design of a round to nearest - even, Zero Overhead Rounding scheme for floating point adders. Pre-computation of rounded results is performed to decrease the latency of floating point adders. Energy delay measures of different barrel switch architectures for the pre-alignment shift of floating point operands for addition is reported in [47], while [48] presents the evaluation of 1's complement arithmetic for the implementation of low power floating point adders. Reference [49] describes the energy delay analysis of partial product reduction methods for parallel multiplier implementations and reference [50] reports an alternative architecture for floating point multiply - accumulate fused (MAF) unit, compared to that used in IBM RISC/6000 [31], [32], so that the architectural power/delay implications of MAC fusion are optimal. MAFs reported in [31], [32], [50], use a compound rounding operation encompassing multiply and accumulate but in order that the results of multiply - accumulate operation be conformal with IEEE standards, the results of multiplication and addition need separate rounding. Reference [51] presents a new architecture of floating point MAF which delivers IEEE conformal product and sum. It is important to note that in this thesis we have proposed a new architecture of low power, high performance floating point multiplier. The architectures of floating point adder and multiply - accumulate fused unit are proposed by

other authors. We have evaluated these architectures by modeling them with VHDL and thus we do not claim any Intellectual Property, IP, apart from modeling and implementation of these architectures.

1.5 Design Methodology

Comparison of the different architectures of floating point units is done by writing the synthesisable VHDL code of each architecture at RTL level. Developing the architectures using hardware description languages can keep them technology independent. The code is simulated at RTL level, gate level and after placement and routing using Synopsys VHDL System Simulator (VSS) for the functional verification. The code is then synthesized using Synopsys Design Compiler and Power Compiler. As the power consumption of multiple data path architectures is highly dependent on the utilization of different data paths, the role of input data, processed by floating point units, becomes very important and RTL and gate level simulations are done using real data (test vectors generated by the conversion of real audio signals to IEEE single precision data format) and synthetic data (test vectors generated manually using different probabilities of data path utilization to get the figures of the average and worst case power consumption). The switching activity is back-annotated from these simulations to Design Compiler and Power Compiler. Power Compiler uses it for the optimization for low power while Synopsys Design Power uses it for the accurate calculation of power consumption.

1.6 Design Implementations

The synthesis is done towards 0.35 micron CMOS and FPGA (Xilinx XC4085XL) technologies to validate that the findings achieved by the evaluation of different architectures

of floating point units are independent of the technology used. The operating voltage for 0.35 micron CMOS technology is 3 volts and it is 4.75 volts for the FPGA technology. During the comparison, by using the same technology, same cell libraries and same set of constraints for synthesis, it is made sure that it is only the difference in architectures that are affecting the final results. While this is a way to do comparison, each architecture can be further optimized by using the circuit level optimization techniques or by adding certain constraints specific to the architecture during synthesis.

1.6 Thesis organization

The rest of the thesis is organized in the following way. Chapter 2 introduces floating point basics, specially the algorithms that implement floating point arithmetic operations. The IEEE floating point data formats and recommendations for the designing of floating point units are presented. Chapter 3 presents our proposal for a low power, high performance, transition activity scaled, double data path floating point multiplier (FPM). Validity of the proposed architecture is proved by comparing it to a single data path FPM [26] using different technologies. Chapter 4 presents the hardware configuration of a transition activity scaled, triple data path floating point adder [1], [36]-[44], and a high speed, single data path floating point adder [34]. Both adders are then compared to evaluate their performance in DSP applications. Chapter 5 presents the hardware configuration and evaluation of three different architectures of floating point Multiply - Accumulate Fused (MAF) unit, i. e. the MAF used in the IBM RISC/6000 floating point execution unit [31] [33], a low power, multiple data path MAF [1] [50] whose results are compatible with IBM MAF's results, and an IEEE compliant floating point MAF [51]. Chapter 6 concludes the thesis and presents some proposals for future work.

Chapter 2

IEEE Recommendations for Floating Point Arithmetic

2.1 Introduction

Floating point numbers are used to obtain a dynamic range for representable real numbers without having to scale the operands. Floating point numbers are approximations of real numbers and it is not possible to represent an infinite continum of real data into precisely equivalent floating point value.

A floating point number system is completely specified by specifying a suitable base (β), significand (or mantissa) M, and exponent E. A floating point number F has the value

$$F = M\beta^E \text{.....(2.1)}$$

β is the base of exponent and it is common to all floating point numbers in a system. Commonly the significand is a signed - magnitude fraction. The floating point format in such a case consists of a sign bit S, e bits of an exponent E, and m bits of an unsigned fraction M, as shown below

S	Exponent E	Unsigned Significand M

The value of such a floating point number is given by:

$$F = (-1)^S M\beta^E \text{.....(2.2)}$$

The most common representation of exponent is as a biased exponent, according to which

$$E = E^{true} + bias \text{ .....(2.3)}$$

where *bias* is a constant and E^{true} is the true value of exponent. The range of E^{true} using the e bits of the exponent field is

$$-2^{e-1} \leq E^{true} \leq 2^{e-1} - 1 \text{ .....(2.4)}$$

The bias is normally selected as the magnitude of the most negative exponent; i.e. 2^{e-1}, so that

$$0 \leq E \leq 2^{e} - 1 \text{ .....(2.5)}$$

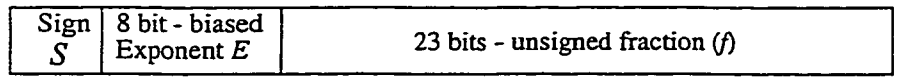
The advantage of using biased exponent is that when comparing two exponents, which is needed in the floating point addition, for example the sign bits of exponents can be ignored and they can be treated as unsigned numbers [52].

The way floating point operations are executed depends on the data format of the operands. IEEE standards specify a set of floating point formats, viz., single precision, single extended, double precision, double extended. Table 2.1 [1] presents the parameters of the single and double precision data formats of IEEE 754 [6] [53] standard.

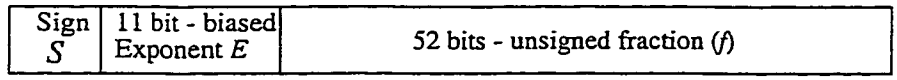
Fig 2.1 shows the IEEE single and double precision data formats. The base is selected as 2. Significands are normalized in such a way that leading 1 is guaranteed in precision (p) data field. It is not the part of unsigned fraction so the significand is in the form *1.f*. This increases the width of precision, by one bit, without effecting the total width of the format.

Table 2.1: Format parameters of IEEE 754 Floating Point Standard

Parameter	Format	
	Single Precision	Double Precision
Format width in bits	32	64
Precision (p) = fraction + hidden bit	23 + 1	52 + 1
Exponent width in bits	8	11
Maximum value of exponent	+ 127	+ 1023
Minimum value of exponent	-126	-1022



(a) IEEE single precision data format



(b) IEEE double precision data format

Fig 2.1 - Single and double precision data formats of IEEE 754 standard

The value of the floating point number represented in single precision format is

$$F = (-1)^S 1.f 2^{E-127} \text{.....(2.6)}$$

where 127 is the value of *bias* in single precision format and exponent E ranges between 1 and 254. $E = 0$ and $E = 255$ are reserved for special values.

The value of the floating point number represented in double precision data format is

$$F = (-1)^S 1.f 2^{E-1023} \text{.....(2.7)}$$

where 1023 is the value of *bias* in double precision data format. Exponent E is in the range $1 \leq E \leq 2046$. The extreme values of E (i.e. $E = 0$ and $E = 2047$) are reserved for special values.

The extended formats have a higher precision and a higher range compared to single and double precision formats and they are used for intermediate results [52].

2.2 Choice of Floating Point Representation

The way floating point operations are executed depends on the specific format used for representing the operands. The choice of a floating point format for the hardware implementation of floating point units is governed by factors like the dynamic range requirements, maximum affordable computational errors, power consumption etc. The exponent bit width decides the dynamic range of floating point numbers while the significand bit width decides the resolution. The dynamic range offered by floating point units is much higher than that offered by fixed point units of equivalent bit width. Larger dynamic range is of significant interest in many computing applications like in multiply - accumulate operation of DSPs. But larger range is not needed in all the applications. The actual bit-width required in many applications need not match with the ones provided by IEEE standards. For example, considering the design of an embedded systems, the choice of IEEE data formats need not give optimal results. In some cases, even IEEE specified rounding schemes may not guarantee acceptable accuracy. That means, depending on the specifica-

tions of a certain application, dedicated system solutions can work with non IEEE data formats as well as rounding schemes such that the real life input/output signals satisfy the data processing goals required by the target application. Although custom specification of floating point format do find quite a lot of applications, in the recent years more and more manufacturers are following IEEE standards for the design of their hardware. IEEE compliance guarantees portability of software between different platforms. Applications that does not need such portability need not stick to IEEE standards [1].

2.3 IEEE Rounding

As not all real numbers can be represented precisely by floating point representation, there is no way to guarantee absolute accuracy in floating point computations. Floating point numbers are approximations of real numbers. Also the accuracy of results obtained in a floating point arithmetic unit is limited even if the intermediate results calculated in the arithmetic unit are accurate. The number of computed digits may exceed the total number of digits allowed by the format and extra digits have to be disposed before the final results are stored in user-accessible register or memory [52]. IEEE 754 standard prescribes some rounding schemes to ensure acceptable accuracy of floating point computations. The standard requires that numerical operations on floating point operands produce rounded results. That is, exact results should be computed and then rounded to the nearest floating point number using the 'round to nearest - even' approach. But in practice, with limited precision hardware resources, it is impossible to compute exact results. So two guard bits (G and R) and a third sticky (S) bit, are introduced to ensure the computation of results within acceptable accuracy using minimum overhead [1].

The default rounding mode specified by the IEEE 754 standard is round to nearest - even. In this mode, the results are rounded to the nearest values and in case of a tie, an even value is chosen. Table 2.2 [52], shows the operation of round to nearest - even, for different instances of significand bit patterns. In this table, X represents all higher order bits of the normalized significand beyond the LSBs that take part in rounding while the period is separating p MSBs of the normalized significand from round (R) and sticky (S) bits. It can

Table 2.2: Round to nearest - even rounding

Significand	Rounded Result	Error	Significand	Rounded Result	Error
X0.00	X0.	0	X1.00	X1.	0
X0.01	X0.	- 1/4	X1.01	X1.	- 1/4
X0.10	X0.	- 1/2	X1.10	X1. + 1	+ 1/2
X0.11	X1.	+ 1/4	X1.11	X1. + 1	+ 1/4

be seen from the table that the average bias (which is the average of the sum of errors for all cases) for the round to nearest scheme is zero. Fig 2.2 illustrates the relative positions of the decision making bits. Rounding to the nearest value necessitate a conditional addition of 1/2 *ulp* (units in the last place). The decision for such addition can be reached through the evaluation of the LSB (M_0) of the most significand p bits of the normalized significand, the round (R) bit and the sticky (S) bit. Rounding is done only if $R(M_0 + S)$ condition is true.

2.4 Floating Point Multiplication

The algorithm of IEEE compatible floating point multipliers is listed in Table 2.3. Multiplication of floating point numbers F_1 (with sign s_1, exponent e_1 and significand p_1) and F_2 (with sign s_2, exponent e_2 and significand p_2) is a five step process. Its block diagram is

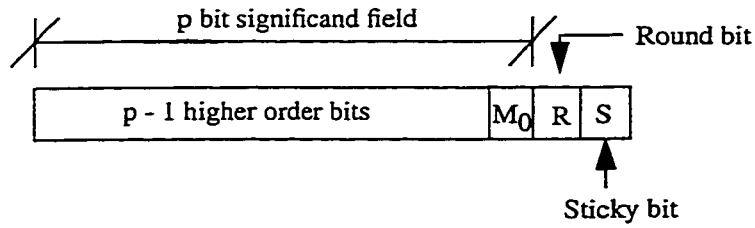


Figure 2.2 - Normalized Significand before rounding

presented in Fig 2.3 [54].

Table 2.3: Floating point multiplication algorithm

Step 1

Calculate the tentative exponent of the product by adding the biased exponents of the two numbers, subtracting the bias, ($e_1 + e_2 - bias$). The bias is 127 and 1023 for single precision and double precision IEEE data format respectively

Step 2

If the sign of two floating point numbers are the same, set the sign of product to '+', else set it to '-'.

Step 3

Multiply the two significands. For p bit significand the product is 2p bits wide (p, the width of significand data field, is including the leading hidden bit (1)). Product of significands falls within range $1 \leq product < 4$.

Step 4

Normalize the product if MSB of the product is 1 (i.e. product of $significands \geq 2$*), by shifting the product right by 1 bit position and incrementing the tentative exponent.*
Evaluate exception conditions, if any.

Step 5

Round the product if R(M0 + S) is true, where M0 and R represent the pth and (p+1)st bits from the left end of normalized product and Sticky bit (S) is the logical OR of all the bits towards the right of R bit. If the rounding condition is true, a 1 is added at the pth bit (from the left side) of the normalized product. If all p MSBs of the normalized product are 1's, rounding can generate a carry-out. In that case normalization (step 4) has to be done again.

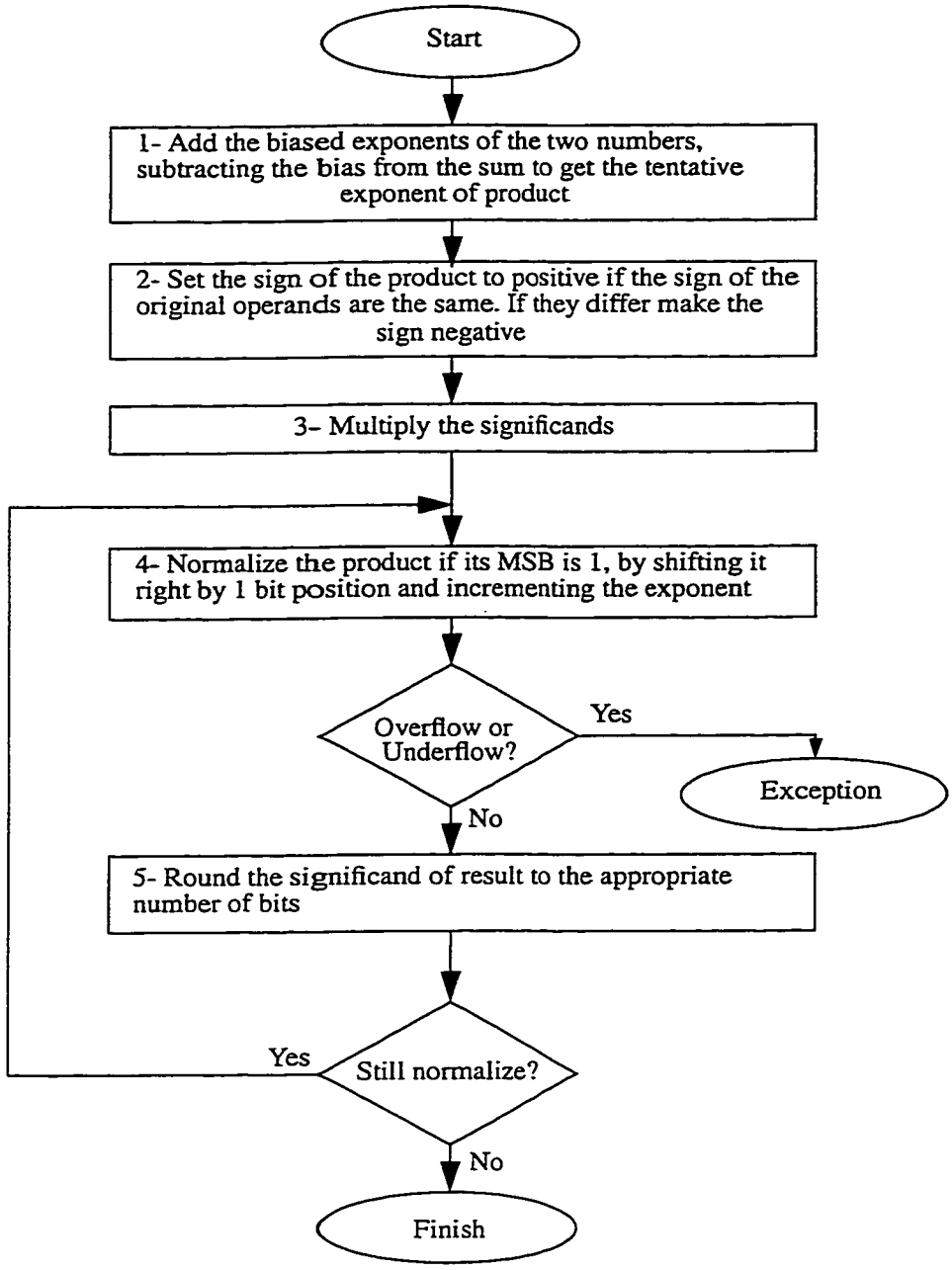


Fig 2.3 - Block diagram of IEEE compliant floating point multiplication

Fig 2.4 illustrate the process of significand multiplication, normalization and rounding.

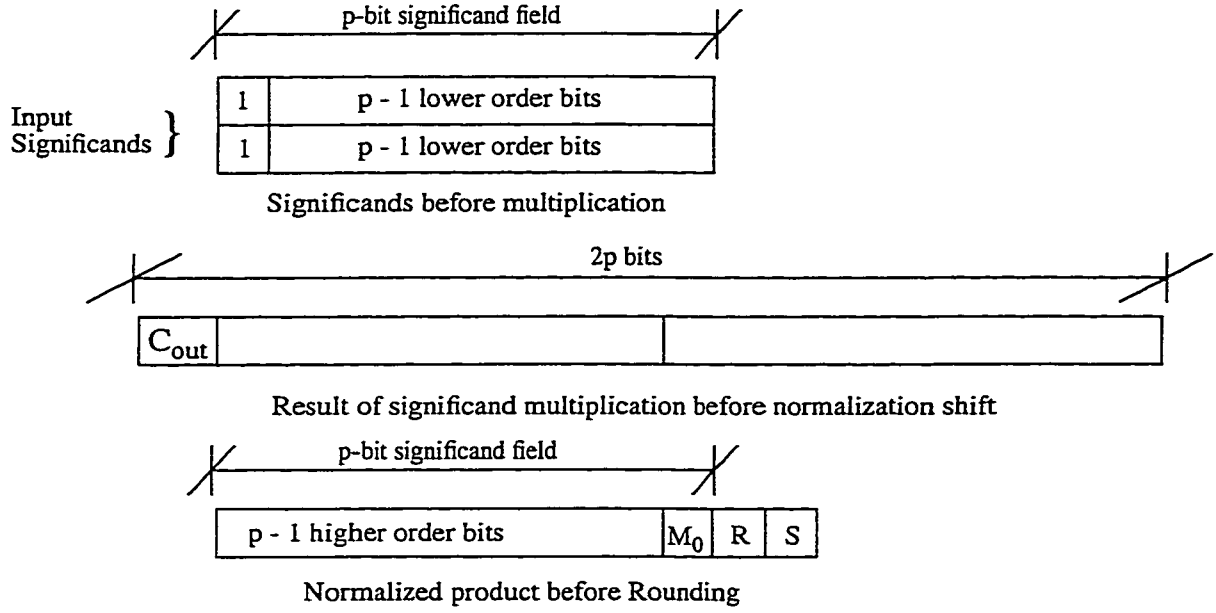


Figure 2.4 - Significand multiplication, normalization and rounding

2.5 Floating Point Addition

For IEEE compliant floating point adders, D. Goldberg etc. [54] has presented an algorithm. This algorithm is applicable to floating point numbers whose significands are presented as sign magnitude format and they are normalized in such a way that a leading one is guaranteed. The algorithm of addition of floating point numbers F_1 (with sign s_1, exponent e_1 and significand p_1) and F_2 (with sign s_2, exponent e_2 and significand p_2) is listed in table Table 2.4 [1], and block diagram is presented in Fig 2.5 [54].

Fig 2.6 [1], illustrates the process of significand addition, normalization and rounding.

2.6 Floating Point Multiply - Accumulate

Floating point multiply-accumulate (MAC) is required in many scientific computing applications and in the design of digital filters. According to IEEE standard, the operation

Table 2.4: Floating point addition algorithm

Step 1

Compare the exponents of two numbers for ($e_1 > e_2$ or $e_1 \leq e_2$) and calculate the absolute value of difference between the two exponents ($|e_1 - e_2|$). Take the larger exponent as the tentative exponent of the result.

Step 2

Shift the significand of the number with the smaller exponent right through a number of bit positions that is equal to the exponent difference. Two of the shifted out bits of the aligned significand are retained as guard (G) and Round (R) bits. So for p-bit significands, the effective width of aligned significand must be p + 2 bits. Append a third bit, namely the sticky bit (S), at the right end of the aligned significand. The sticky bit is the logical OR of all shifted out bits.

Step 3

Add/subtract the two signed-magnitude significands using a (p + 3)-bit adder. Let the result of this is SUM.

Step 4

Check SUM for carry out (C_{out}) from the MSB position during addition. Shift SUM right by one bit position if a carry out is detected and increment the tentative exponent by 1. During subtraction, check SUM for leading zeros. Shift SUM left until the MSB of the shifted result is a 1. Subtract the leading zero count from tentative exponent.
Evaluate exception conditions, if any.

Step 5

Round the result if the logical condition R"(M_0 + S") is true, where M_0 and R" represent the pth and (p + 1)st bits from the left end of the normalized significand. New sticky bit (S") is the logical OR of all bits towards the right of the R" bit. If the rounding condition is true, a 1 is added at the pth bit (from the left side) of the normalized significand.
If p MSBs of the normalized significand are 1's, rounding can generate a carry-out. In that case normalization (step 4) has to be done again.

of multiplication and addition should produce results that are IEEE compatible i.e. both product and sum should be rounded individually [1].

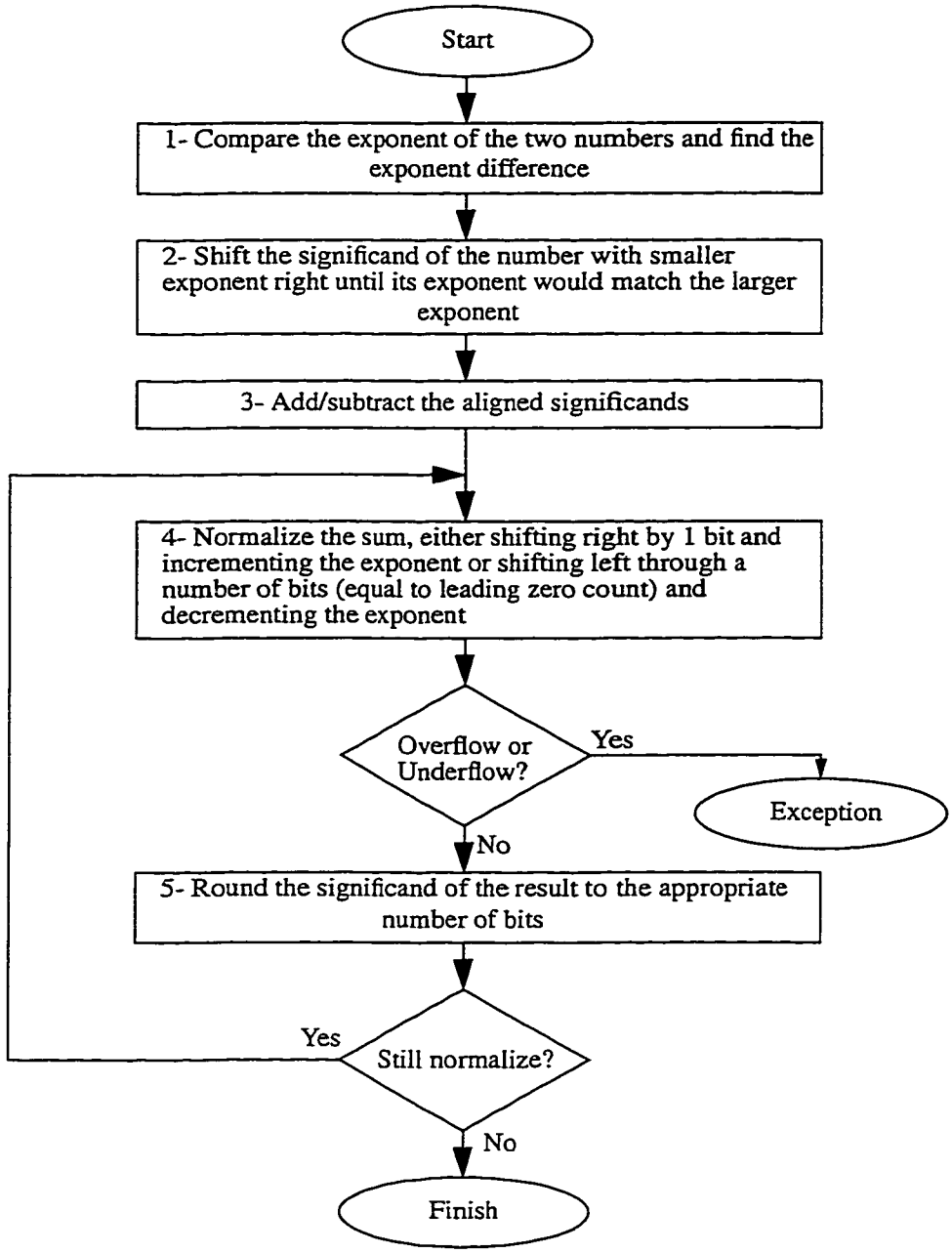


Fig 2.5 - Block diagram of IEEE compliant floating point addition

2.7 Exceptions

For the handling of arithmetic operations on floating point data, IEEE standard specifies

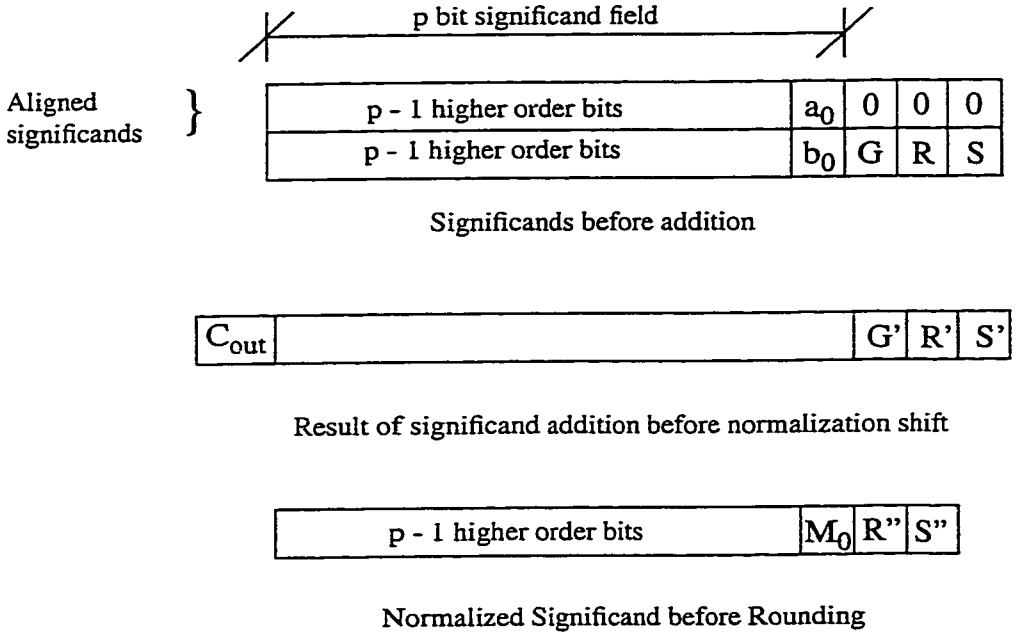


Fig 2.6 - Significand addition, normalization and rounding

some exception flags. Some exception conditions are listed in Table 2.5 [1]. When the result of an arithmetic operation exceeds the normal range of floating point numbers as

Table 2.5: Exceptions in IEEE 754

Exception	Remarks
Overflow	Result can be $\pm\infty$ or default maximum value
Underflow	Result can be 0 or denormal
Divide by Zero	Result can be $\pm\infty$
Invalid	Result is NaN
Inexact	System specified rounding may be required

shown in Fig 2.7 [1], overflow or underflow exceptions are initiated. Detection of overflow

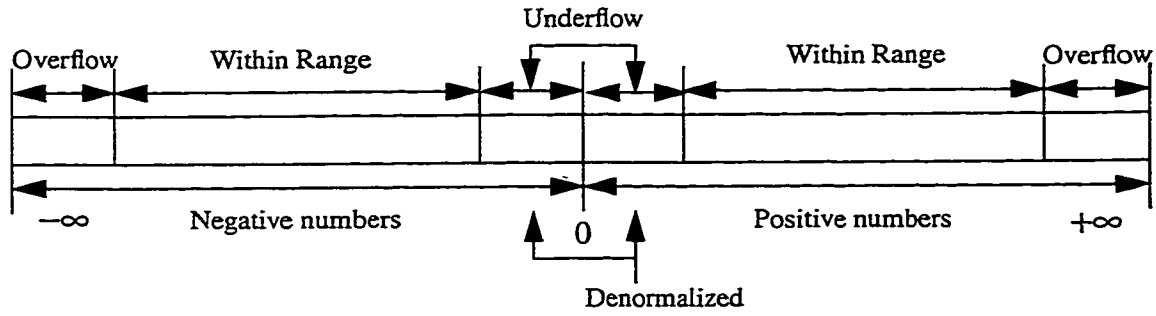


Fig 2.7 - Range of floating point numbers

Table 2.6: Operations that can generate Invalid Results

Operation	Remarks
Addition/ Subtraction	An operation of the type $\infty \pm \infty$
Multiplication	An operation of the type 0 x ∞
Division	Operations of the type 0/0 and ∞/∞
Remainder	Operations of the type x REM 0 and ∞ REM y
Square Root	Square Root of a negative number

or underflow is quite straight forward as the range of floating point numbers is associated with the value of exponents. Table 2.6 [1] lists all possible operations that result in an invalid exception. During invalid exception the result is set to a NaN (not a number). Inexact exceptions are true whenever the result of a floating point operation is not exact and IEEE rounding is required [1]. In IEEE single precision data format, width of exponent field is 8. So 256 combinations of exponent are possible. Among them two are reserved for special values. The value $e = 0$ is reserved to represent zero (with fraction $f = 0$) and denormalized numbers (with fraction $f \neq 0$). The value $e = 255$ is reserved for $\pm\alpha$ (with frac-

tion $f = 0$) and NaN (with fraction $f \neq 0$) [52]. The leading bit of significands (hidden bit) is zero instead of one for all the special quantities.

Chapter 3

Transition Activity Scaled Double Data Path Floating Point Multiplier

3.1 Introduction

Low power, high speed Floating Point Multipliers (FPM) are essential building blocks of modern day digital applications like signal processing, computer graphics and scientific computations. Recent microprocessors, graphic processors and various kinds of DSPs, used for user interfaces, communication or code compression, are equipped with dedicated floating point multipliers. Due to the presence of a relatively high traffic of floating point multiplications in these applications, the power/performance implications of floating point multipliers directly impact the power/performance desirability of the target product. In digital CMOS implementations, the power consumption and speed performance of functional units are highly influenced by algorithmic/architectural design decisions [5]. These decisions influence the switching activities, fanouts, layout complexity, logic depths, operand scalability and pipelinability of functional units. Among them, switching activities, fanouts and layout complexity are directly related to the power consumption. So architectural design for low power operation targets the minimization of these parameters [1].

High speed multiplication of integer data has been a field of research for long time. Some remarkable work has been reported in references [11] through [20]. But applications like DSPs and computer graphics require dynamic range larger than that offered by fixed point

operations. New approaches for high performance floating point multiplication have been proposed in the recent years in [26], [27], and [28]. References [21], [22], [23], and [24] discuss the implementation of integer multipliers for the significand multiplication of IEEE single and double precision FPMs. Floating point multiply - accumulate fused (MAF) units proposed in [31], [32], [50], [1], and [51] are the preferred choice in the applications where there is room for the fusion of multiplication with accumulation but there are still applications even in DSP which require stand alone FPMs.

For the design of FPMs, most of the work has been centered around the performance improvement of significand multiplier. A lot of optimization is possible in other parts of FPMs to minimize critical path delay, latency and power consumption. In this chapter, a new architecture of a low power, high speed Floating Point Multiplier (FPM) is presented. The proposed architecture delivers IEEE compliant product in three clock cycles. The switching activity and hence the dynamic power consumption is reduced by the functional partitioning into two mutually exclusive, clock gated data paths. During any operation cycle only one of the data paths is active and the logic states of the other data path are constant. Critical path delay and latency is improved by using different architectural/logic level approaches like the use of a fast significand multiplier, Zero Overhead Rounding and high speed calculation of sticky bit. Comparison of the proposed FPM with the single data path FPM [26], for power consumption, speed, area and different products of them, is done by synthesizing towards 0.35 micron CMOS technology and FPGAs.

3.2 Architecture of the proposed FPM

Fig 3.1 shows the basic building blocks of the proposed FPM. It can support single and

double precision data formats of IEEE 754 floating point standard. In this Chapter, comparison of single precision FPMs is presented.

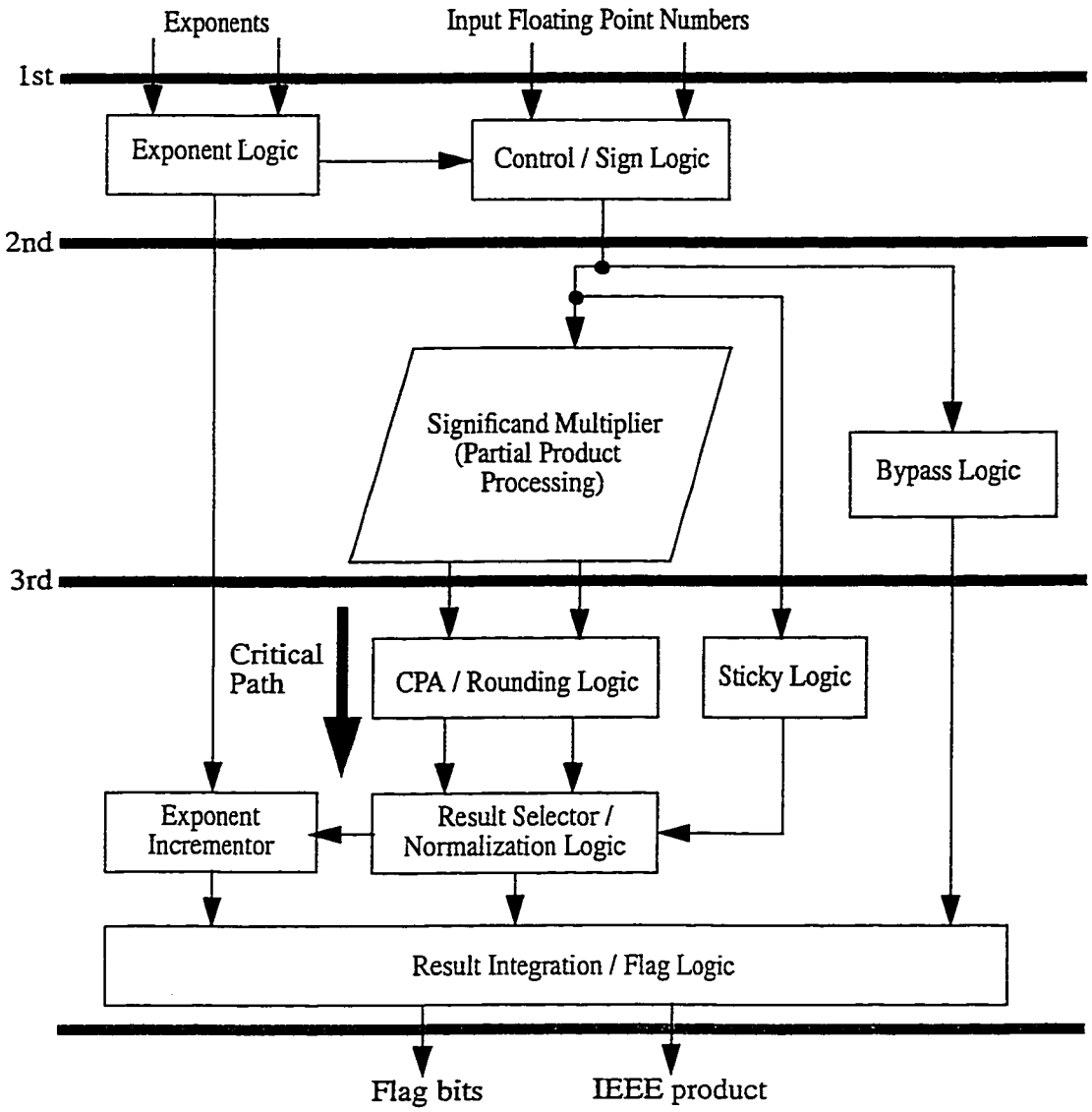


Fig 3.1 - Block diagram of the proposed FPM with 3 execution pipeline stages.

Exponent logic calculates the tentative exponent of the product and also evaluates the conditions for multiplication overflow and underflow using exponents of the input operands.

Control/Sign logic generates the control signals which activate one of the two data paths. It also calculates the sign of the product using a two input XOR gate. Bypass logic is a non-computational unit, which maps floating point numbers to the output during various bypass conditions. Significand Multiplier is composed of partial product generation and compression logic for the multiplication of significands. Partial products are generated by 2 input AND gates. In CPA/Rounding logic precomputation for rounding is concurrently performed with addition. Sticky logic can calculate sticky bit in parallel with CPA. Result Selector/Normalization block selects an appropriate copy of the result in accordance with the rounding/normalization requirements. Normalization shift can be one bit right shift. In case of normalization shift tentative exponent of product is incremented by 1 by the Exponent Incrementor.

3.2.1 Transition activity scaling

In CMOS circuits, the switching activities of functional units exhibit sensitivities towards architectural/algorithmic design decisions and architectural design for low power operation targets the minimization of power consuming transitions through the use of appropriate algorithms, number systems, structural transformation etc. Switching activity reduction through clock gating offers promising results as far as architectural design for low power operation is concerned. The reduction of switching activity implies minimization of effective capacitance being switched per unit of time and minimization of switched capacitance can be achieved through the reduction of either of the factors - capacitance or activity factor. Also, design decisions that increase the area and hence the total capacitance are not considered as bad choices as long as there is a reduction in switched capacitance (per unit of time) [1]. In transition activity scaling [57], the hardware implementation of

algorithms are partitioned into a set of mutually exclusive, distinct, gated clock data paths. During any computing cycle, the operations are mapped to only one clock gated logic block and the logic assertion status of the circuit nodes of all other data paths are preserved at their previous states, so that the switching transitions within these data paths don't exist. If the hardware implementation of an algorithm is partitioned into k data paths. The power consumption of such a scheme can be represented by

$$P = \sum_{i=1}^{k} p(i)P_i \text{.....(3.1)}$$

where $p(i)$ is the probability that the operation is mapped into the ith data path while P_i represents the power consumption of the ith data path. Without activity scaling, the power consumption can be as high as $\sum_{i=1}^{k} P_i$. The mapping of the algorithm into distinct data paths is governed by factors like: area, layout regularity, data path simplifications, delay reduction, pipelineability, functional units reusability etc. [1].

With the proposed architecture, the process of floating point multiplication is partitioned into two distinct categories and a separate activity scaled data path is envisaged for the handling of each. Among the two distinct data paths, one is computing data path while the other is a non-computing data path. The non-computing or bypass data path becomes operational during those situations when the process of floating point multiplication is guaranteed to produce a result that is known apriori. For example, the situation in which either of the two operands is a special quantity (zero, denormal, $\pm infinity$, not a number (NaN)), there is no need to activate the multiplier data path as for operation of type 0 x number, zero can be latched to output, for $\pm\alpha \times number$, an infinity can be latched. During operations when floating point multiplication produces NaN result, NaN can be

latched to the output by the Bypass logic etc. To find that an input floating point number is a special quantity, only their exponents need to be explored. As in IEEE data formats, special quantities are represented by setting the exponents, $e_{min} - 1$ (represents zero if fraction, $f = 0$ and represents denormalized numbers if fraction $f \neq 0$) or $e_{max} + 1$ (represents $\pm\alpha$ if fraction, $f = 0$ and represents NaN if fraction $f \neq 0$). With exponent biasing, $e_{min} = 1$ and $e_{max} = 254$, for single precision floating point numbers. So if exponents are either of the extreme values (0 and 255 for single precision data format), bypass data path will be activated and vice versa. There are also some situations in which based on the exponents of input numbers, multiplication underflow ($e_A + e_B + 1 - e_{bias} < e_{min}$) or multiplication overflow ($e_A + e_B - e_{bias} > e_{max}$) can be found in the exponent logic. In case of multiplication overflow, Bypass logic sets the significand and exponent of the result to $+infinity$ or $-infinity$, depending on the sign of the product. Multiplication underflow is dealt by latching a zero as result. In bypass mode, all the FPM core except the Exponent logic, Control logic, Bypass unit and Result Integration/Flag logic is transition activity scaled and a significant amount of power saving can be achieved in applications where probability of utilization of bypass data path is high.

3.2.2 Exponent logic

During the multiplication of two floating point numbers, A and B, the exponent of the product, e_P is given by

$$e_P = e_A + e_B - e_{bias} + Inc \quad \text{.....(3.2)}$$

In the above equation e_A and e_B represent the exponents of the A and B respectively. Also, e_{bias} represents the value of the exponent bias which is 127 for IEEE single precision data

format and *Inc* represents the MSB of the output of significand multiplication (1 only if the *Product of significands* ≥ 2). Whenever *Inc* is one, the tentative exponent of the product is incremented and the product of significands is subjected to a normalization shift (and rounding). Exponent logic calculates the tentative exponent of product, e_T, as

$$e_T = e_A + e_B - e_{bias} \quad \text{.....(3.3)}$$

Since the number of operands is three, a single row of 3:2 compression logic (carry save adder approach) is required for the processing of input operands e_A, e_B and e_{bias}, as shown in Fig 3.2(a). Third row in Fig 3.2(a) represents the 1's complement of e_{bias}. After the compression (Fig 3.2(b)), 9 least significant bits of two operands are added together with a carry-in of 1. In this way, evaluation of multiplication overflow and underflow conditions is quite straight forward. If carry-out of the addition is 1, there will be a multiplication underflow otherwise if MSB of the result of addition is 1, there will be a multiplication overflow, else 8 LSBs of result present the tentative exponent of product.

0	e_{A7}	e_{A6}	e_{A5}	e_{A4}	e_{A3}	e_{A2}	e_{A1}	e_{A0}
0	e_{B7}	e_{B6}	e_{B5}	e_{B4}	e_{B3}	e_{B2}	e_{B1}	e_{B0}
1	1	0	0	0	0	0	0	0

(a) - Input data for the evaluation of $e_A + e_B - e_{bias}$

0	S8	S7	S6	S5	S4	S3	S2	S1	S0
C8	C7	C6	C5	C4	C3	C2	C1	C0	0

(b) - Input data after bit compression

Fig 3.2 - Data presentation for the evaluation of $e_A + e_B - e_{bias}$

3.2.3 Significand Multiplier

In floating point multipliers using IEEE single precision data format, a 24×24 bit integer multiplier is needed for the multiplication of significands (24 bits wide).

Several schemes, with the objective of improving the speed of the integer multipliers, have been proposed in the past. In the start, multiplication methods were iterative schemes based on the simple shift and add algorithm. Later, Wallace [13] suggested the simultaneous addition of all partial products in a carry free way using carry save adders. Wallace reduced the partial products by connecting the carry save adders in parallel tree structure. A carry propagate addition step is needed in the end. The carry save adders (Wallace scheme) are generalized by Dadda [14], [15], to *(m, n)* counters. He also showed how to minimize the number of counters in a compression tree. Weinberger [16] introduced a 4:2 compressor as a way of reducing the bits in a parallel multiplier array. Booth recoding [11], [12], has been widely used in parallel multipliers and using radix 4 modified Booth algorithm (MBA), the number of partial products can be reduced to $n/2$ for an $n \times n$ 2's complement multiplication but its efficiency has been denied by several authors. In [17], it has been shown that 4:2 compressors can reduce the number of partial products to $n/2$ in less time and using fewer gates compared to radix 4 MBA. The authors in [49], [1], showed that compared to MBA, the energy delay improvement of 4:2 compressor based scheme is better than 20% for a 24×24 bit unsigned multiplier. Also compared to Booth algorithm, 4:2 compressors based partial product compression logic gives a highly regular layout. So the routing lengths and hence the routing delay is reduced which is quite significant in the submicron technologies.

In our significand multiplier, 4:2 compressors are arranged in Wallace tree format, thus only 4 addition stages are needed in order to add 24 partial products as shown in Fig 3.3. The 4:2 compressor has five inputs and three outputs and it adds up four partial products (I_1 to I_4) and one carry-in (C_i) into one sum (S) and two carry-outs (C and C_o). Such a 4:2 compressor is designed to operate faster than two full adders. The XOR operations for the

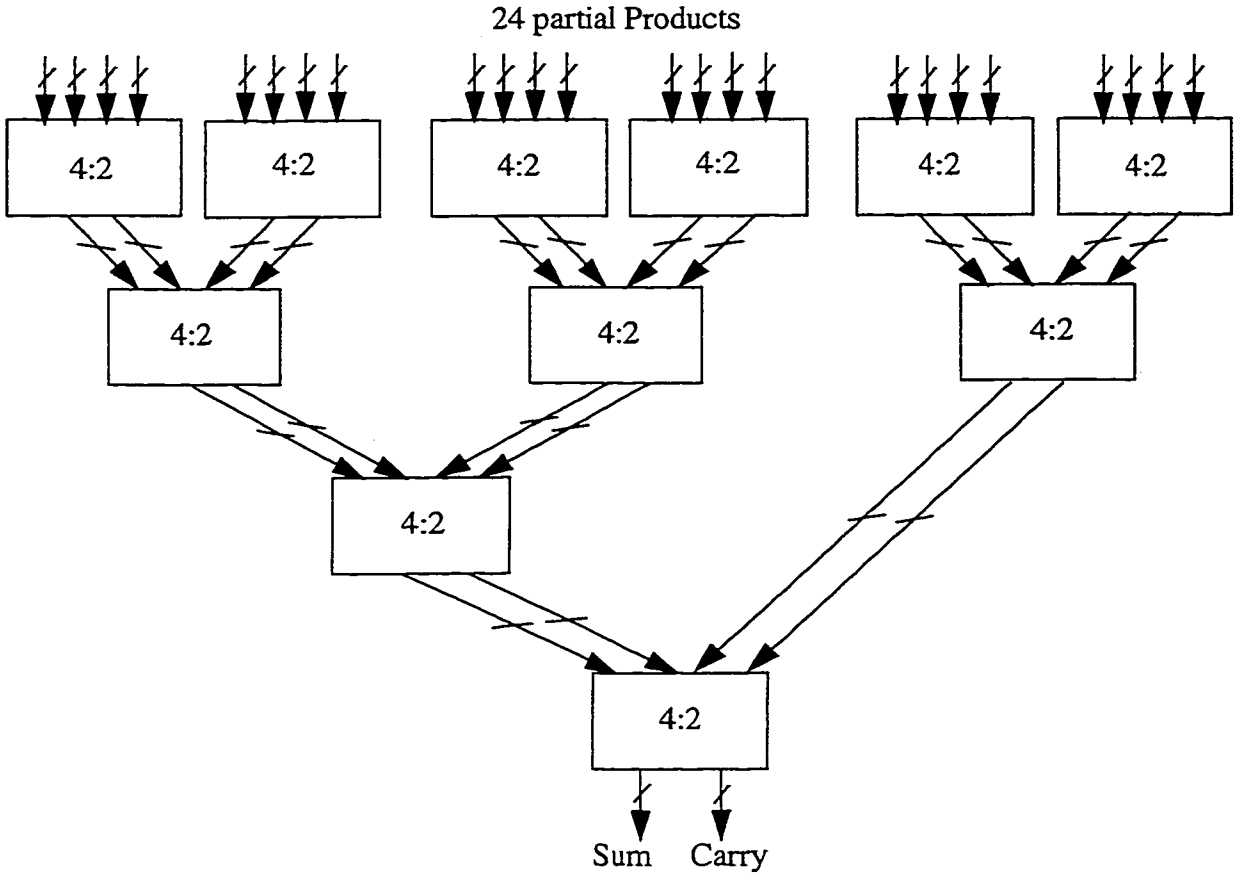


Fig 3.3 - Structure of partial product compression logic using arrays of 4:2 compressors arranged in Wallace tree format

individual two inputs of four are concurrently performed, so four inputs are compress- ed to sum using three XOR gate propagation delay as shown in Fig 3.4 [49] [1]. It takes four XOR gate propagation delay when using an ordinary carry save adder. Also since one of

the two carry-outs (C_o) is independent of the carry-in (C_i), it can be the carry-in of the 4:2 compressor in the same stage of the upper bit without making a long critical path as shown in Fig 3.5 [21] [56].

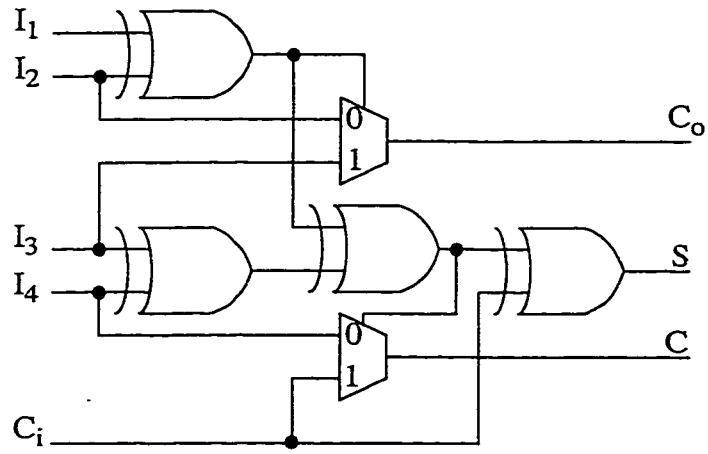


Fig 3.4 - 4:2 compressor

3.2.4 Sticky bit evaluation

For p-bit significands, the width of the product is $2p$ bits. Among them, $p - 3$ ($p - 3$ represents the number of bits towards the right of G (guard) and R (round) bits) LSBs are only computed for the evaluation of the sticky bit of the product, which is the logical OR of these bits. For IEEE multipliers, the sticky bit can also be generated through an evaluation of the trailing zeros of the product. The sticky bit should be set to 1 only if the expected number of trailing zeroes in the product is smaller than the number of the least significand product bits that are discarded i.e. $p-3$. The number of trailing zeros in the product of two binary significands is equal to the sum of trailing zero of the multiplicand and multiplier [52] [1] [28]. Hence sticky bit is 1, if

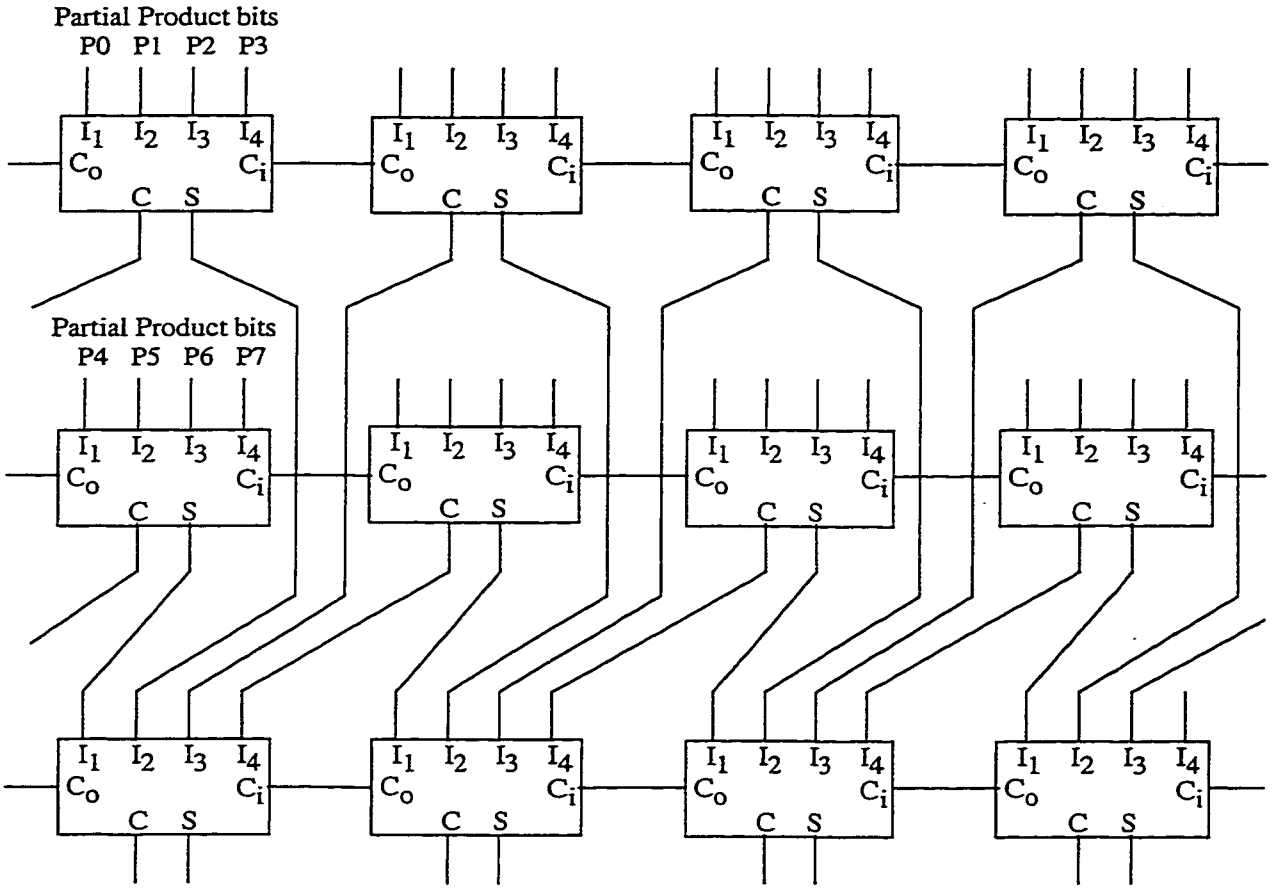


Fig 3.5 - Interconnections of 4:2 compressors

$$TZ_A + TZ_B < p - 3 \text{ .....(3.4)}$$

Where TZ_A and TZ_B represent the trailing zeros of multiplicand and multiplier (significands). For the counting of trailing zeros, a fast, tree type, 32 bit trailing zero counter is used. Its 16-bit version is shown in Fig 3.6 [46]. This type of counter had been proposed by H. Suzuki et al. [34], for leading zero counting. The input to the counter is the 1's complement of the data word whose trailing zeros are to be counted. The basic building block of the counter is a 4-bit tree, which encodes the number of trailing zeros of 4 bits of input data into a 3-bit field. Two 4-bit trees are cascaded to form an 8-bit tree. Two 8-bit trees

can be cascaded to form a 16-bit tree and two 16-bit trees of the form shown in Fig 3.6 can

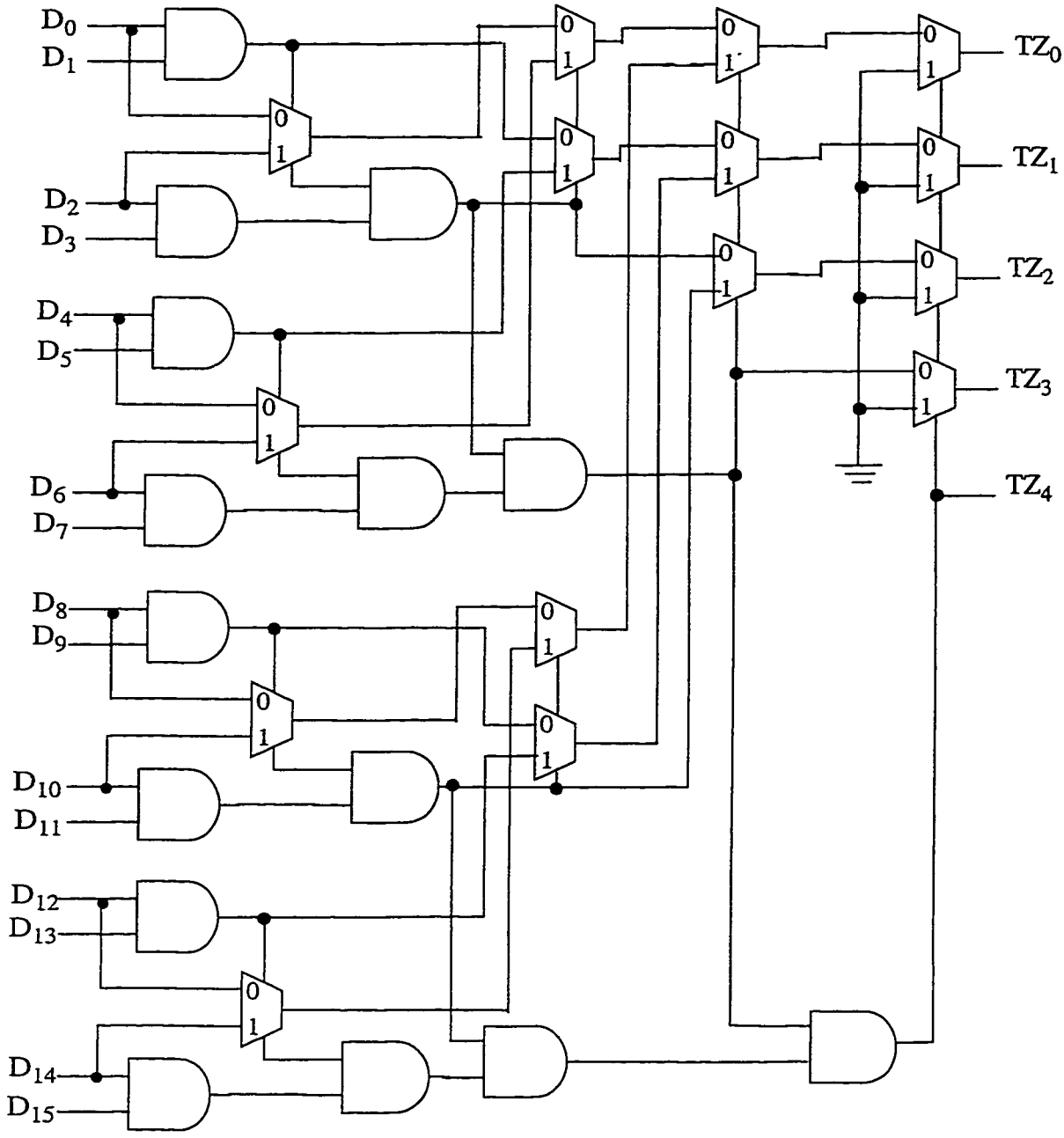


Fig 3.6 - Trailing Zero Counter

be cascaded to form a 32-bit trailing zero counter, which counts and encodes the trailing zeros of a 32-bit data word. As both multiplier and multiplicand are 24 bits wide, remain-

ing bits are asserted as 1's, on the most significant side, before using them in the trailing zero counter.

A high speed comparator, proposed for the sticky bit evaluation in floating point adders [44], can be modified to evaluate the condition in Equation (3.4). Equation (3.4) can be reformulated to $p-3-TZ_A-TZ_B>0$, so that end around carry of 1's complement adder can be used. The power consumption and delay of 1's complement operations are less compared to that of their 2's complement counterparts. Trailing zeros are encoded into 5 bit numbers and are complemented, as shown in Fig 3.7(a). The 3rd row in Fig 3.7(a) represent the value of $p-3+1$ i.e. 22, for the 2's complement addition with TZ_A and TZ_B. These three rows are compressed into sum and carry vectors by a row of 3:2 compressors,

1	$\overline{TZ_{A4}}$	$\overline{TZ_{A3}}$	$\overline{TZ_{A2}}$	$\overline{TZ_{A1}}$	$\overline{TZ_{A0}}$
1	$\overline{TZ_{B4}}$	$\overline{TZ_{B3}}$	$\overline{TZ_{B2}}$	$\overline{TZ_{B1}}$	$\overline{TZ_{B0}}$
0	1	0	1	1	0

(a) - Input data for the evaluation of $TZ_A + TZ_B < p - 3$

	C4	C3	C2	C1	C0	
1	0	S4	S3	S2	S1	∅

(b) - Input data after bit compression

Fig 3.7 - Data presentation for the evaluation of $TZ_A + TZ_B < p - 3$

as shown in Fig 3.7(b). The end around carry of the 1's complement arithmetic operation can be found by making use of a circuit of the type shown in Fig 3.8. The end around carry

is the required sticky bit.

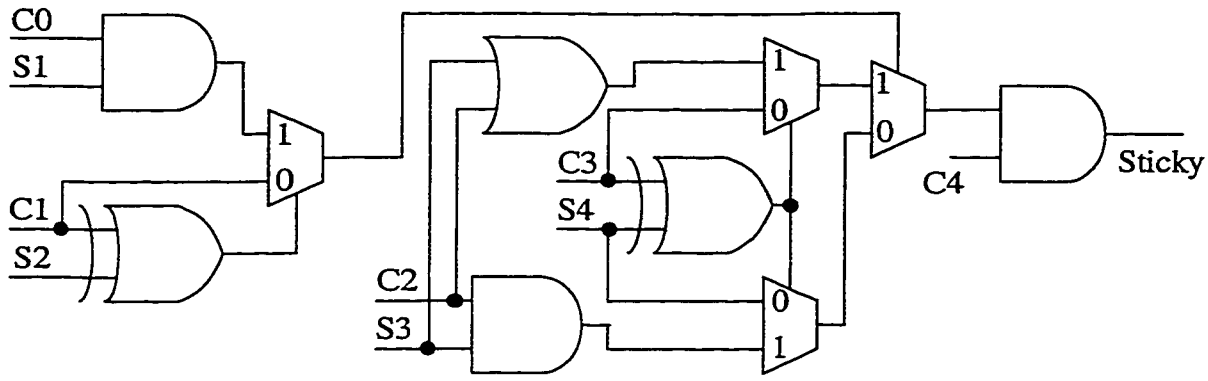


Fig 3.8 - Sticky bit evaluation

3.2.5 CP Adder and Zero Overhead Rounding

During floating point multiplication, decisions regarding rounding become available only at a relatively latter stage, any subsequent delays introduced by rounding adds up with the overall delay of floating point multiplication. In the worst case, the delay due to rounding can be as much as the delay of a significand adder. Pre-computation of rounding results is the best choice as far as the delay reduction of floating point multipliers is concerned. A Zero Overhead Rounding approach for floating point adders has been presented in [46]. Here Zero Overhead Rounding approach for floating point multipliers is proposed. In this approach, $p - 3$ LSBs (bits towards the right of R bit) of sum and carry vectors from Significand Multiplier are truncated from the rest of the MSBs. After the truncation, $p + 3$most significant bits are shown in Fig 3.9.

Bit positions M_1 to *Cout* of sum and carry vectors are treated as a separate block. Since the rounding decisions are not available till the normalization shift is complete, pre-computation of two sets of sum bits is the best choice for Zero Overhead Rounding. So that by the

time the rounding decisions are known, an appropriate set of sum bits can be chosen. The sum bits are evaluated for an incoming carry of zero (*sum0*) as well as one (*sum1*). Conditional Sum/Carry select adders [23] [7] [56] are ideal for this application. As the sticky bit is calculated directly from the significands of the input floating point numbers, $p - 3$ LSBs takes part only in the calculation of carry from these lower order bits to $p + 3$ higher bits, *Cin*. So $p - 3$ LSBs of sum and carry vectors are added together in parallel with the addition of most significant bits (M_1 to *Cout)* to generate *Cin*.

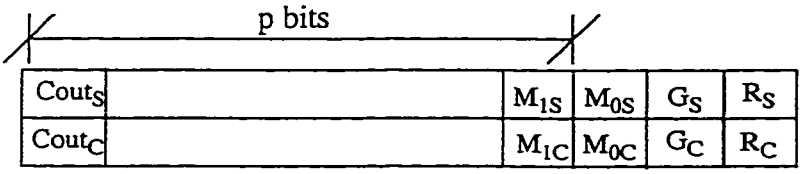


Fig 3.9 - $p + 3$ MSBs of sum and carry vectors of Significand Multiplier

3.2.6 Result Selector/Normalization logic

This unit selects an appropriate copy of conditional sum bits based on different rounding/ normalization conditions. In this unit, M_0, G and R bits of sum and carry vectors (Fig 3.9) are added with *Cin* (carry from lower order bits), which results in S_1, S_2, S_3 and C bits as shown in Fig 3.10.

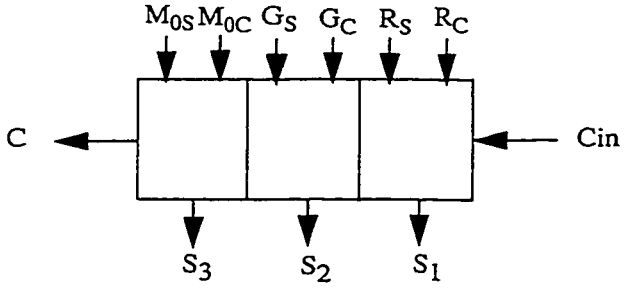


Fig 3.10 - Addition of G, R and M0 bits of sum and carry vectors with carry-in from lower order bits.

This unit selects the sum bits with a carry-in of zero (*sum0*) or sum bits with a carry-in of one (*sum1*) based on the carry injection at the M_1 bit position. Let the most significant bits of *sum0* and *sum1* vectors are *Cout0* and *Cout1*. There are three possible cases for the evaluation of carry injection at M_1 bit position.

Case 1: If the MSBs of *sum0* and *sum1* are one i.e. *Cout0 = 1* and *Cout1 = 1* (in this case product of significands is greater than or equal to 2), the carry injection at M_1 bit position is

$$C_I = C + Round_I$$

Where C is the carry out from the 3-bit addition of M_0, G and R bits of sum and carry vectors, as shown in Fig 3.10, while

$$Round_I = S_3(sum0(0) + S_2 + S_1 + Sticky)$$

Where *sum0(0)* is the least significant bit of *sum0* vector and S_1, S_2 and S_3 are the sum bits of 3-bit addition of sum and carry vectors of Significand Multiplier. If $C_I = 1$, *sum1* will be selected, otherwise *sum0*. Selected sum bits are then normalized by shifting right by one bit position.

Case2: If the MSBs of *sum0* and *sum1* vectors are zeros i.e. *Cout0 = 0* and *Cout1 = 0* (in this case product of significands is less than 2), the carry injection at M_1 bit position is

$$C_{II} = C + Round_{II}$$

Where $Round_{II}$ is

$$Round_{II} = S_2(S_3 + S_1 + Sticky)$$

If $C_{II} = 1$, result selector will select *sum1* otherwise sum0. In this case no normalization shift is required. Rounded bit towards the right of M_I bit, M_0' can be found as

$$M_0' = M_{0S} \oplus M_{0C} \oplus Round_{II}$$

Where M_{0S} and M_{0C} are the M_0 bits of the sum and carry vectors as shown in Fig 3.9.

Case 3: This case represents the situation in which after normalization and rounding, a correction shift is required in conventional FPMs. In this case, *Cout0* = *0* and *Cout1* = *1*. Here again, C_I is checked if it is one, *sum1* bits are selected and normalization (one bit right shift) is performed, otherwise *sum0* bits and M_0' are selected and no normalization shift is needed.

The Zero Overhead Rounding approach considerably decreases the delay introduced by rounding and correction shift.

3.2.7 Exponent Incrementor

When the selected result in Result Selector/Normalization logic is normalized by shifting it right through one bit position, tentative exponent needs to be incremented. Exponent increment can result in an overflow if tentative exponent of product is equal to the maximum representable exponent without overflow, e_{max}, where $e_{max} = 254$ for IEEE single precision data format. 1's complement adders are ideal for this kind of application. With the conditional carry outputs of these adders, evaluation of overflow condition is a trivial operation. During the exponent increment operation, $C_{out}(1) = 1$ implies an overflow,

where $C_{out}(1)$ represent the conditional carry output from MSB position of the adder anticipating an input carry at the LSB position of 1 [48] [1].

Control unit, Exponent logic and Result Integration/Flag logic are common to both the data paths. Control unit selects an appropriate data path and routes the inputs to the selected data path. This unit generates various control signals which control the activation of various data paths by clock gating of registers of pipeline stage 2 and stage 3 (Fig 3.1). Result Integration/Flag logic selects the appropriate copy of result from the relevant data path. It also asserts various status flags subject to the validity of exception conditions. Five different flags are included for zero, *–infinity*, *+infinity*, not a number and denormalized results.

3.3 Comparison with single data path FPM

Comparison of the proposed double data path FPM is done with a single data path FPM [26] by writing VHDL code of both architectures at RTL level. Both floating point multipliers are IEEE compliant using IEEE single precision floating point data format. Both architectures of FPM are pipelined to 3 stages. The code is synthesized using Synopsys Design Compiler and Power Compiler. The synthesis is done towards 0.35 micron CMOS and FPGA (Xilinx XC4085XL) technologies to validate the claim that the proposed FPM gives optimal results independent of the technology used. By utilizing the same technology, same cell libraries and same set of constraints for synthesis, it is made sure that it is only the difference in architectures that are affecting the final results. As the power consumption of the proposed double data path FPM is dependant on the utilization of a certain data path, simulations are done using real data (test vectors generated by the conversion of

real audio signals [55]) and synthetic data (test vectors generated manually using different probabilities of data path utilization). Switching activity from these simulations is back-annotated for the calculation of power consumption.

3.4 Results

Maximum delay, power consumption and area information for both floating point multipliers after synthesizing to 0.35 micron CMOS technology is summarized in Table 3.1 while Table 3.2 presents the same results after synthesizing to FPGAs. The calculated values of Power-Delay (PD), Area-Power (AP), Area-Delay (AT) and Area-Delay^2 (AT^2) are also presented in Table 3.1 and 3.2. In these tables, P_{synth1} represents the power consumption during processing the synthetic data in which the probability of utilization of both data paths of the proposed FPM is the same. Furthermore, P_{synth2} shows the power figures of the FPMs using synthetic data in which the probability of utilization of bypass data path of the proposed FPM is zero. Also, P_{real1} shows the power consumption of the FPMs when real floating point data from two files (each containing 128K data samples, obtained from the conversion of real audio signals) is multiplied, while P_{real2} is the amount of power consumption when real data from one file is multiplied by co-efficients (rotating) of low pass FIR filter of order 8, having cut-off frequency of 0.2 [55].

Table 3.1 shows that using 0.35 micron CMOS technology, when the probability of utilization of both data paths of the proposed FPM is the same, it offers 44.8% reduction in power delay product compared to single data path FPM. The proposed FPM offers 26% reduction in power delay product over single data path FPM during processing the synthetic data in which bypass data path of the proposed FPM is never utilized. Proposed

FPM gives 11.4% reduction in power delay product during multiplication of real data and

Table 3.1: Comparison of the proposed FPM with the single data path FPM [26] using 0.35 micron CMOS technology

Parameters	Proposed FPM	Single data path FPM
Maximum Delay, D (ns)	22.58	24.59
Power using synthetic data, P_{synth1} (mW) at 40 MHz	24.41	44.25
Power using synthetic data, P_{synth2} (mW) at 40 MHz	33.77	45.63
Power using real data, P_{real1} (mW) at 40 MHz	22.67	25.59
Power using real data, P_{real2} (mW) at 40 MHz	46.25	76.35
Area, A (10^4 cell-area)	67.55	48.46
Power-Delay Product, PD (ns.mW)	610.25	1106.25
Area-Power Product, AP (10^4cell-area.mW)	1648.9	2144.36
Area-Delay Product, AT (10^4cell-area.ns)	1525.28	1191.63
Area-Delay^2 Product, AT^2 (10^4cell-area.ns^2)	3.44 x 10^4	2.93 x 10^4

39.42% reduction in power delay product when real data is multiplied with rotating filter coefficients, compared to single data path FPM. Moreover the proposed FPM is 8.2% faster than the single data path FPM and it takes around 28.26% more area. Same kind of results are obtained by synthesizing both architectures of FPM to FPGAs (Table 3.2). The proposed FPM gives 26.9% reduction in power delay product when processing the synthetic data in which the probability of utilization of both data paths of the proposed FPM is

the same. The proposed FPM offers 11.9% reduction compared to single data path FPM when the probability of utilization of bypass data path of the proposed FPM is zero. The proposed FPM is 9.1% faster and it uses 23.8% more CLBs.

Table 3.2: Comparison of the proposed FPM with the single data path FPM [26] using FPGAs

Parameters	Proposed FPM	Single data path FPM
Maximum Delay, D (ns)	151.28	166.4
Power using synthetic data, P_{synth1} (mW) at 5 MHz	503.55	688.95
Power using synthetic data, P_{synth2} (mW) at 5 MHz	637.47	723.8
Power using real data, P_{real1} (mW) at 5 MHz	426.33	435.01
Power using real data, P_{real2} (mW) at 5 MHz	959.96	1000.1
Area, A, Total CLBs (#)	1580	1204
Power-Delay Product, PD (ns.mW)	1.0×10^5	1.38×10^5
Area-Power Product, AP (#.mW)	7.96×10^5	8.29×10^5
Area-Delay Product, AT (#.ns)	2.39×10^5	2.0×10^5
Area-Delay^2 Product, AT^2 (#.ns^2)	3.62×10^7	3.33×10^7

3.5 Discussion and Conclusion

Using both 0.35 micron CMOS and FPGA technologies, the same kind of results are obtained. The proposed architecture of FPM is faster, consumes less power but takes more area compared to single data path FPM [26]. The results indicate that the proposed FPM offers power saving, however the amount of saving is dependent upon the input data. Double data path FPM offers significant amount of saving for the applications where the prob-

ability that the data will go through the Bypass data path is higher. The double data path FPM has more area than the single data path FPM because it has an extra data path, Control logic and Result Integration unit. As the data files are generated from the conversion of real audio signals in which data doesn't change for some time specially in the start and in the end of audio signals, so the values of power consumption, P_{real1}, for both floating point multipliers are less than the values generated by synthetic data. The power consumption of the two FPMs when real data is multiplied by rotating filter coefficients which are changing with every clock cycle, P_{real2}, in this case, are higher than the values of power consumption using synthetic data. While multiplying real data, the chances of bypass are negligible and the proposed FPM offers reduction in power delay product due to its high performance. Floating point bypass becomes important in some DSP applications, where certain arithmetic operations may fail to impact the final result due to the limitations of floating point data formats. Symmetrical band pass/stop filters having a normalized centre frequency of 0.5 exhibit relatively large differences between the exponents of adjacent filter co-efficients [1]. With these applications, the power/delay advantages of the proposed FPM renders it an ideal choice for floating point multiplications.

Chapter 4

Floating Point Adders

4.1 Introduction

The demand for high performance, low power floating point adders has been on the rise during the recent years in the applications like DSPs, image processors and microprocessors. As floating point addition involves operations like pre-alignment shift, addition of significands, normalization, rounding, correction shift and exponent evaluation, the delay and power consumption of floating point adders is quite significant. New design approaches have emerged that incorporate these operations concurrently, to reduce the critical path delay and latency. The significant among such approaches is the concurrent evaluation of leading zeros for normalization shift, with significand addition (leading zero anticipation). The first leading zero anticipatory (LZA) logic, used in IBM RISC/6000 execution unit [33] [31] [32], uses some of the signals of conventional carry look ahead adders (carry generate, propagate and kill signals) and processes these signals concurrently to do its job. Hiroaki Suzuki et al. [34] proposed a new architecture of LZA logic, which is using simple boolean algebra structure and it can anticipate leading zeros using less delay and area compared to the LZA unit proposed in [33]. He also proposed an approach for the concurrent evaluation of rounding and normalization shift in his five stage, single data path floating point adder (FADD). J. D. Bruguera et al. [35] proposed a scheme to correct the wrong anticipation of leading zeros by earlier LZA logics [33] [34] by one bit in parallel with leading zero anticipation, in his Leading-One Prediction (LOP)

logic to reduce the latency of the floating point adder proposed in [34]. R. V. K. Pillai proposed a Zero Overhead Rounding approach [46] for floating point adders in which precomputation of rounded results is performed along with significand addition. Their proposed triple data path floating point adder (TDPFADD) [1], [36]-[44], uses transition activity scaling for power reduction. It also removes one barrel shifter from the computational data paths for the reduction of latency.

In this chapter, hardware configuration and comparison of the two optimized architectures of floating point adder, i.e. TDPFADD proposed in references [1], [36]-[44], and single data path FADD proposed in [34], is presented. The basis of comparison of the two floating point adders is their performance in DSP applications.

4.2 Hardware configuration of triple data path floating point adder (TDPFADD)

In this section, hardware configuration of a triple data path floating point adder (TDPFADD) [1], [36]-[44] is presented, in which transition activity scaling, utilized by the exponent behavior based design partitioning, gives optimized values of time averaged power consumption.

4.2.1 Algorithm and transition activity scaling

At the architectural level, transition activity scaling of the functional units offers a viable approach for power minimization [56]. During the evaluation of floating point addition operation, following observations can be made: *(1)* The leading zero estimation circuits of floating point adders, that handle variable number of leading zeros, generated by signed-magnitude addition of significands, need to be operational only during a limited set of

additions. *(2)* The operation of floating point addition can be bypassed during certain situations.

Based on the above observations, the process of floating point addition is partitioned into three distinct categories and a separate activity scaled data path is envisaged for the handling of each. Among the three distinct data paths, two are computing data paths while the third is a non-computing data path. The non-computing or bypass data path becomes operational during those situations when the process of floating point addition is guaranteed to produce a result that is known apriori. Functional partioning of the floating point adder into three distinct, mutually exclusive, clock gated data paths allows activity reduction. During any computing cycle, only one of the data paths is active, during which state, the logic assertion status of the circuit nodes of the other data paths are maintained at their previous states.

The algorithm of floating point addition that is mapped to the transition activity scaled, triple data path floating point adder (TDPFADD) is listed in Table 4.1 [1], [36]-[44]. Fig 4.1 [1], illustrates the finite state machine representation of the transition activity scaled TDPFADD and Table 4.2 lists the criterion for the assertion of various states of TDPFADD as well the modules that are activity scaled in each state.

State *I* represents bypass conditions. State *J* represents the TDPFADD operations that can result in the generation of a significand with a variable number of leading zeros, which is only possible during the subtraction of one significand from another when the difference between their exponents is zero or one. In this case Leading Zero Anticipatory (LZA) data

Table 4.1: Floating point addition algorithm for TDPFADD

Step 1

compare the exponents of two floating point numbers for (e1 > e2, e1 < e2, e1 = e2) and compute exponent difference |e1 - e2|
evaluate input numbers for special conditions e. g. 0 ± operand, ±∞ ± operand, NaN ± operand etc.
select the tentative exponent of the result
order significands on the basis of the relative magnitudes of exponents
if |e1 - e2| > p or special conditions, go to step 2
if |e1 - e2| ≤ 1 and subtraction and neither |e1 - e2| > p nor special conditions, go to step 3.1
if |e1 - e2| > 1 or addition and neither |e1 - e2| > p nor special conditions, go to step 4.1

Step 2
generate default result
go to step 5

Step 3. 1
align the significands
perform 1's complement addition of aligned significands
perform speculative rounding
count leading zeros of the different copies of result
select result and corresponding leading zero count
go to step 3.2

Step 3.2
normalize significand
compute the exponent of the result
evaluate exception conditions, if any
go to step 5

Step 4.1
align the significands

Step 4.2

perform signed - magnitude addition of aligned significands
perform speculative rounding
evaluate normalization requirements; 0/1 bit left or right shift
select result and perform normalization
compute the exponent of the result
evaluate exception conditions, if any
go to step 5

Step 5

select the appropriate copy of result from the relevant data path

path is active. State K represents the operations of the TDPFADD during those situations when the signed-magnitude addition of significands can produce at the most one leading zero. In this state, floating point addition is performed by Leading Zero Bounded (LZB) data path. The time averaged power consumption of TDPFADD is represented by

$$P = p(I)P_I + p(J)P_J + p(K)P_K \text{ .....(4.1)}$$

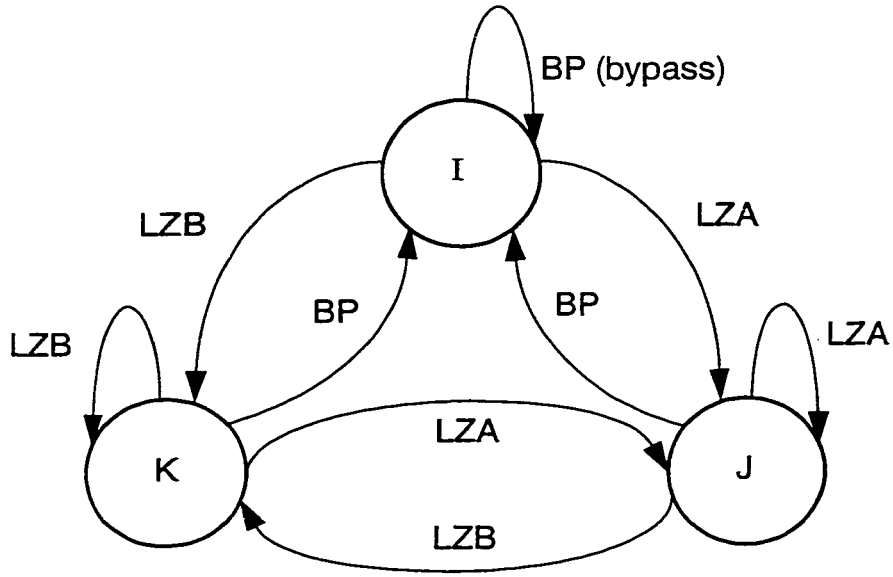


Figure 4.1 - Finite state machine representation of TDPFADD operation

Where $p(I)$, $p(J)$ and $p(K)$ represent the probability that the TDPFADD is operating in states I, J, and K respectively and P_I, P_J, P_K represent the time averaged power consumption of the TDPFADD when it is operating in the respective state. With non activity scaled floating point adders, the power consumption can be as high as $P_I + P_J + P_K$. With the transition activity scaling scheme, the reduction in time averaged power consumption of TDPFADD occurs mainly due to the following:

Table 4.2: State assertion conditions of TDPFADD

State	Active data path	State assertion criterion	Activity scaled blocks
I	Bypass	Either exponent is zero or $e_{max}+1$ or $e_{dif} > p$	Entire TDPFADD except Exponent logic, Control unit, Bypass logic and Result Integration/Flag logic
J	LZA	No Bypass and subtraction and $e_{dif} \leq 1$ ($LZs \leq p$)	Pre-alignment barrel shifter (large)
K	LZB	No Bypass and addition or $e_{dif} > 1$ ($LZs \leq 1$)	LZA logic and normalization barrel shifter (large)

(1) The leading zero estimation circuits that anticipate a variable number of leading zeros are as complex as significand adders. Transition activity scaling ensures that this unit becomes operational only during situations that demand its use. During all other situations, the switching activity of this unit is zero and in floating point additions, the probability that FADDs endure situations that demand estimation of a variable number of leading zeros is marginally low.

(2) In bypass mode, most of the floating point adder core is activity scaled and a significant amount of power saving can be achieved.

(3) Transition activity scaled design partitioning of floating point adder offers power savings also due to the data path simplifications. Since in the LZA data path, the significand pre-alignment shifts are ≤ 1, they can be performed by using a single level of multiplexers (MUXs), however the normalization shift requirement can be anywhere between 0 and p (p represent the width of significand data field, including hidden bit), by virtue of which it is mandatory to have a composite barrel shifter for performing such shifts. In LZB data

path, significand pre-alignment shifts can be any where between 0 and p and a pre-alignment barrel shifter is needed. The normalization shift of this data path can be realized by using a single level of 3X1 MUXs. Due to the complexity of the barrel shifters, they are expensive in terms of power consumption, delay and area and because of transition activity scaling, at the most one barrel shifter is used for the addition of floating point numbers. This minimizes the power consumption, logic depths and circuit delays of the data paths.

4.2.2 Architecture

Fig 4.2 illustrates the data flow architecture of TDPFADD. To distinguish between the two computational data paths, functional sub-units of the LZA data path are shaded darker than that of the LZB data path.

4.2.2.1 Exponent logic

This logic evaluates the relative magnitudes ($e1 > e2$, $e1 < e2$ or $e1 = e2$) of the exponents of input floating point numbers and calculates the absolute value of difference between them ($|e1 - e2|$). A 1's complement adder is used for this application because during operation of the type $|e1 - e2|$, the conditional carry outputs also reveal the truth of the conditions $e1 > e2$, $e1 < e2$, or $e1 = e2$, using no extra hardware as can be seen in Table 4.3 [1] [48]. In Table 4.3, $C_{out}(0)$ and $C_{out}(1)$ represent the conditional carry outputs from the MSB bit position of the adder anticipating an input carry (at the LSB bit position) of 0 as well as 1. Compared to 2's complement scheme, the 1's complement adder can evaluate $|A - B|$ in the same time, using half the hardware. The power consumption of this scheme is around 50% of that of its 2's complement counterpart. The inherent data comparison capa-

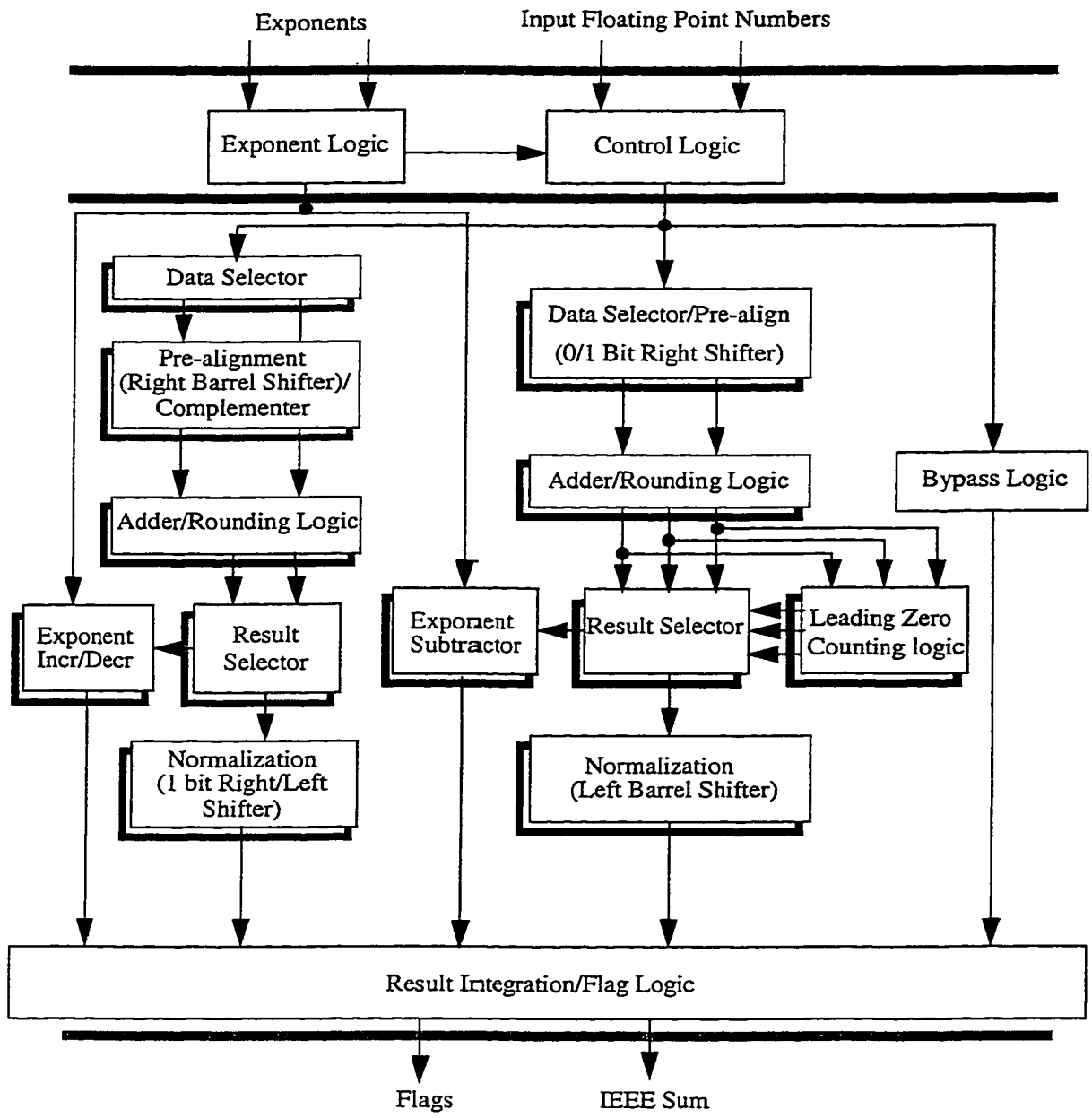


Fig 4.2 - Block diagram of the TDPFADD

bility of the 1's complement adder facilitates the evaluation of conditions like $A > B$, $A \leq B$, $B > A$, $B \leq A$, $A = B$ and $A \neq B$, without any power consumption or use of hardware [1], [48]. Absolute value of exponent difference, $e_{dif} = |e1 - e2|$ is needed by Control logic, for

Table 4.3: Significance of conditional carry outputs of 1's complement adders

$C_{out}(0)$	$C_{out}(1)$	Remarks
0	0	$C_{out}(1) = 0 \Rightarrow A < B$
0	1	$C_{out}(0) = 0$ and $C_{out}(1) = 1 \Rightarrow A = B$
1	0	*Impossible condition*
1	1	$C_{out}(0) = 1 \Rightarrow A > B$

the activation of different data paths and also it is the magnitude of pre-alignment shift in LZA and LZB data paths. Relative magnitude of exponents is used in the selection of the significand of the smaller number for pre-alignment shift by Data Selectors.

4.2.2.2 Control Unit

Control unit selects an appropriate data path and routes the inputs to the selected data path by exploring the exponents of input floating point numbers and exponent difference. This unit generates control signals, according to state assertion conditions discussed in Table 4.2. The Bypass data path is activated not only when either of the inputs is a special quantity but also whenever the magnitude of significand pre-alignment shift i.e. the exponent difference, exceeds the width of significands (p), because in that case the result of floating point addition is definitely the larger number. TDPFADD gives substantial amount of power saving during such 'beyond shift range' conditions due to the limited width of the significand data field.

4.2.2.3 Bypass data path

The bypass data path is a non-computing data path which selects and memorizes an appropriate floating point number during various bypass conditions. Whenever the difference between the exponents of the input floating point numbers is greater than the width of significand, the larger floating point number is latched into the bypass data path. During operations of the type 0 $\pm$ number also the larger floating point number is latched. If both the operands are zeros, a zero is latched. For operations of the type $\pm\infty \pm$ number, an infinity is latched. During situations when a FP addition produces 'NaN' (not a number) result, NaN is latched into the bypass data path. As latency of pipelined TDPFADD (pipelined architecture is presented latter) is fixed, the results of the bypass data path are not immediately presented to the output. The latched data is presented to the output during an appropriate cycle of pipeline operation.

4.2.2.4 Leading Zero Anticipatory (LZA) data path

LZA data path is activated by Control logic when bypass conditions are false and there is a possibility of generation of variable number of leading zeros during signed-magnitude addition of significands, which is only possible if the operation is subtraction and the difference in exponents of the input operands is either zero or one.

In LZA data path, Data Selector/Pre-alignment logic handles the pre-alignment shift of the appropriate significand by 0/1 bit right shifter. Based on the conditional carry outputs of the 1's complement adder, used for the evaluation of $|e1 - e2|$ condition in Exponent logic, significand with the larger exponent is routed directly to the significand adder while the significand with the smaller exponent is right shifted by one bit and is complemented. The

LSB of the shifted out significand is retained as guard (*G*) bit. Fig 4.3 illustrates the pre-aligned significands before addition.

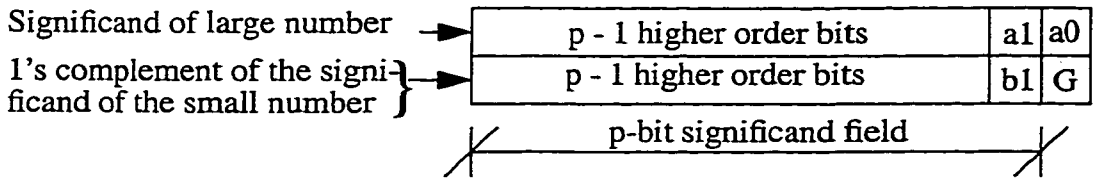


Fig 4.3 - Pre-aligned significands

The data field with a *G* at its LSB position represents the 1's complement of the right shifted significand when the exponent difference between the floating point numbers is 1. The bit *a0* is asserted 1 during the subtraction of a right shifted significand. Effectively, this type of operation guarantees a 2's complement addition of the aligned significands whenever the exponent difference is one, in which case a positive result is guaranteed as significand with smaller exponent is always complemented. Whenever the exponent difference is zero, pre-alignment shift is not required and one of the significands is complemented. In this case, *a0*, *G* bits are not relevant in the signed-magnitude addition.

The Adder/Rounding block performs signed-magnitude addition of significands. Pre-computation for rounding is performed concurrently with addition, so that by the time the rounding decisions are known, an appropriate copy of the result can be selected for normalization shift. Adder/Rounding logic adds the pre-aligned significands without *a0* and *G* bits (LSBs of pre-aligned significands) and generate conditional sum bits for bit positions *M0* to MSB and conditional carry outputs by anticipating block carry inputs of 1 as well as 0. A 1's complement addition using conditional sum/carry select adders is ideal for

this application [23] [7] [56]. Fig 4.4 shows the process of pre-computation, selection and normalization of rounded result in LZA data path while Fig 4.5 illustrates the result of sig-

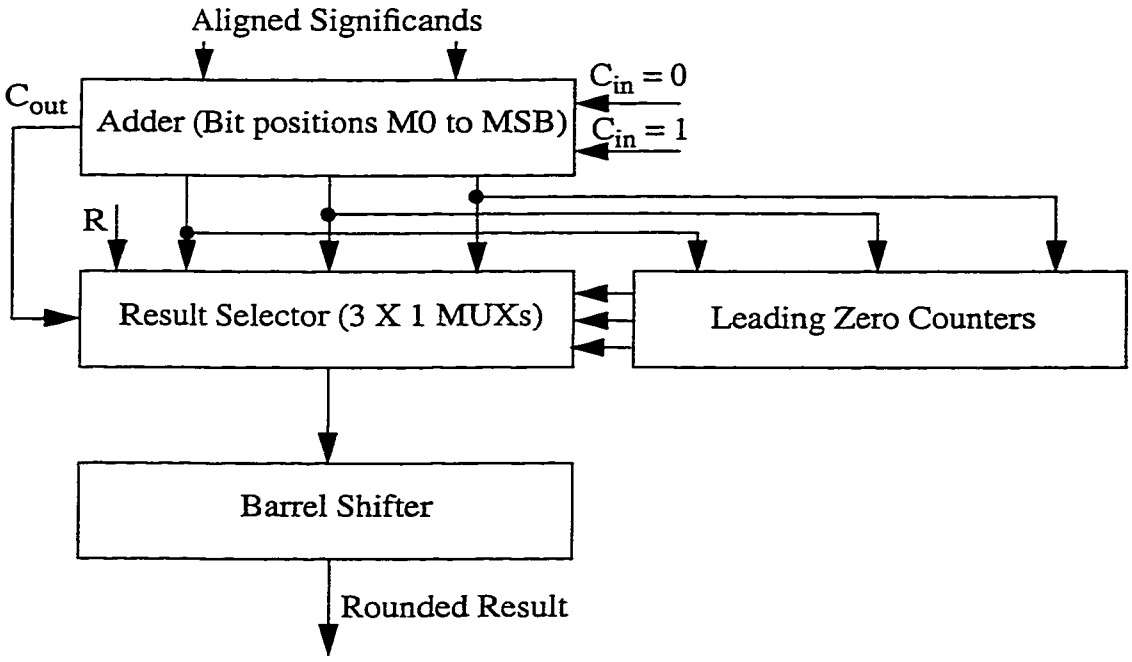


Fig 4.4 - Pre-computation, selection and normalization of rounded results in LZA data path

nificand addition. Note that *Cout* is useful for effecting the selection of the right result after 1's complement addition. The least significand bit, *R*, shown in Fig 4.5, is evaluated separately and is used for the purpose of rounding. The Adder/Rounding logic generates the conditional sum bits with an anticipated carry inputs of one (*sum1*) and zero (*sum0*) as well as the 1's complement of the conditional sum bits with carry-in of zero ($\overline{sum0}$). The $\overline{sum0}$ is required during those situations when the end around carry of 1's complement addition, with equal exponents, is zero. The Leading Zero Counting logic detects and encodes the number of leading zeros of the three results, generated by the 1's complement Adder/Rounding logic. In references [1], [36]-[44], the authors favour the use of serial leading zero counters instead of parallel leading zero anticipatory logic for their

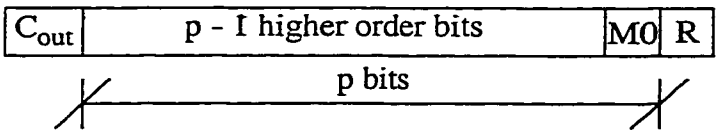


Fig 4.5 - Result of significand addition before normalization shift

TDPFADD architecture. According to their assumption, since the significand pre-alignment operation is faster for the LZA data path (due to the absence of a multi-stage barrel shifter), the adder inputs of this data path get asserted at a relatively earlier time compared to that of the LZB data path. The earlier the arrival of significand inputs, the earlier the completion of addition. Because of this reason, even with a simpler (serial) leading zero counting logic, this data path can complete the process of floating point addition within such time the LZB data path completes an addition. But the speed advantages given by fast significand pre-alignment shift in LZA data path is covered by slower normalization shift, which is done by a barrel shifter. Normalization shift in case of LZB data path is faster because it is a 0/1 bit left or right shift and it can be realized by a single row of MUXs. Hence, even the use of simpler (serial) leading zero counter has the advantages in terms of power consumption and silicon area, the delay introduced by leading zero counter adds up to the critical path delay of TDPFADD and it also increases the latency in pipelined TDPFADD. For the counting of leading zeros, a fast, tree type, 32-bit leading zero counter has been chosen. It is proposed by H. Suzuki et al. [34] and its 16-bit version is shown in Fig 4.6. The basic building block of the counter is a 4-bit section, which encodes the number of leading zeros into a 3-bit data field. Two 4-bit sections are cascaded to form an 8-bit section. Two 8-bit sections can be cascaded to form a 16-bit section and so on.

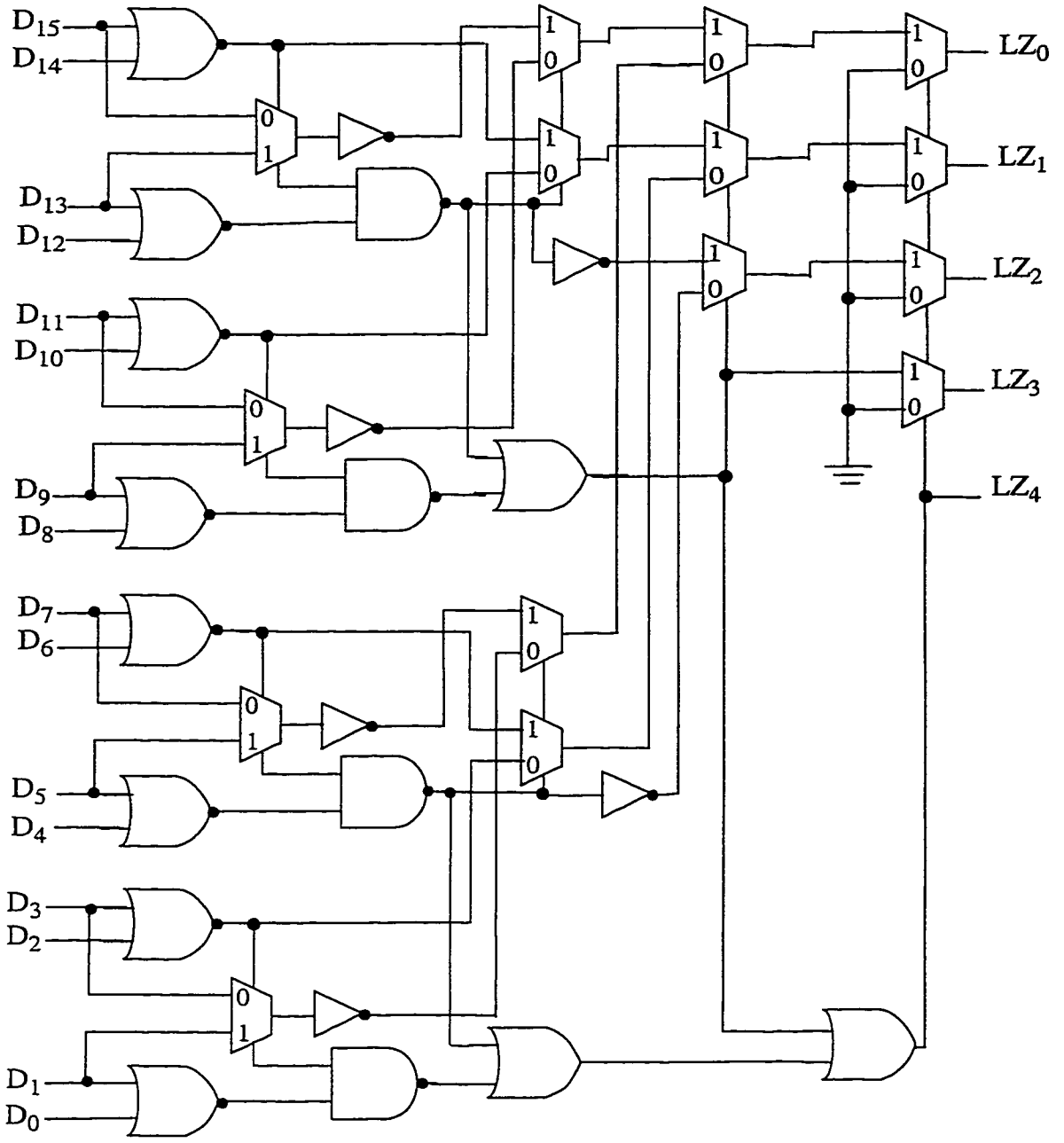


Fig 4.6 - Leading Zero Counter

The Leading Zero Counting logic is composed of 3 such LZ Counters which evaluate the number of leading zeros in *sum0*, *sum1* and $\overline{sum0}$ vectors in parallel. As the conditional

sum outputs of Adder/Rounding logic are 24 bits wide, the remaining 8 LSBs are asserted as 1's before feeding them to leading zero counters.

Result Selector receives three conditional sum vectors and their leading zero counts and selects one of the sum vectors and its corresponding leading zero count in accordance with the rounding/complementation conditions as explained below.

Whenever the exponents of input floating point numbers are equal, a subtraction of one of the significands from the other is guaranteed to produce at least one leading zero, due to the equality of their leading bits. This situation doesn't demand any rounding. Since the relative magnitudes of the significands are not known in the beginning of addition, a 1's complement subtraction can produce either a positive result or a result that is equal to the 1's complement of the magnitude of the result or even a zero result. If the conditional carry-out of 1's complement significand addition, with an anticipated carry-in of zero, is 1 i.e. $Cout(0) = 1$, it means the smaller significand was subtracted from the larger significand and the result is positive. In this case, conditional sum bits with an anticipated carry-in of 1 (sum1) should be selected. If the conditional carry-out, with carry-in of one, is zero i.e. $Cout(1) = 0$, it means the larger significand was complemented and 1's complement of conditional sum bits with carry-in of 0 ($\overline{sum0}$) needs to be selected. When two significands are equal (i.e. $Cout(0) = 0$ and $Cout(1) = 1$), it produces a zero result as the exponents are also equal and the operation is subtraction. The decoding of the conditional carry outputs of 1's complement adders is attractive for the evaluation of such a condition as a direct evaluation of zero result through the decoding of the resulting significand is expensive in terms of silicon area and power. Also in this situation, a zero result is directly mapped to

the output and the normalization shifter is not operational. Transition activity scaling of this barrel shifter results in significant amount of power saving.

During subtraction with an exponent difference of one, the normal carry input at *M0* position is *G*. This carry input is valid as long as no rounding is required, which is guaranteed for all those situations which result in the generation of at least one leading zero and in this case, *R* bit is also required which is the complement of *G*. When the subtraction with exponent difference of one produces no leading zeros, rounding may be required and carry injection at *M0* is $G + (a_1 \oplus b_1)$. Combining both the cases, the actual carry input at *M0* position is

$$Cin = G + (a_1 \oplus b_1)LB(0) \text{.....(4.2)}$$

where a_1, b_1 and *G* are relevant bits of the aligned significands as shown in Fig 4.3 and *LB(0)* is the leading bit of *sum0* vector. If $Cin = 1$, sum bits with anticipated carry-in of 1 (*sum1*) is selected otherwise *sum0* is selected.

Normalization logic is composed of a barrel shifter, which shifts the *(p+1)*-bit wide result, presented by the Result Selection logic, left by an amount equal to the selected value of leading zero count, such that the leading bit of the final result is always a 1. As in case of IEEE single precision data format, the maximum amount of normalization shift is 23, 5 bits are sufficient for effecting the significand shift. $LZ = 24$, presents the case in which result is zero and barrel shifter is transition activity scaled as zero is mapped to the output. In barrel shifters, the dominant component of power consumption is the switching activities of high fanout control path and operand data path nodes. Transition activity scaling of

these nodes is rewarding as far as design for low power operation is concerned. Also the delays of drivers which interface signals to the high fanout nodes, forms a major component of the overall delay. Because of these reasons, the additional power/delay overheads for the evaluation of extra control signals for effecting transition activity scaling is insignificant in contrast to the savings in power wastage attainable through such a control. The barrel shifter is composed of a cascaded array of 2X1 multiplexers. A single stage of nX1 MUXs can also perform the requisite shifts but multi-stage shifters are advantageous, as far as energy delay minimization of barrel shifters is concerned [47] [1]. The 8-bit version of the multistage left barrel shifter [56] used is shown in Fig 4.7. Each 2X1 MUX of the ith stage selects one of the 2^i shifted data and unshifted data. The shifter operates in the order from the shift by the smallest amount to the shift by largest amount. Once the shifting is complete, the LSB of the resultant significand can be discarded. The leading bit can also be discarded, since this bit is guaranteed to be a 1 all the times.

The Exponent Subtractor subtracts the relevant copy of leading zero count from the tentative exponent of the result. The 1's complement adders are appropriate for the evaluation of the exponent of the final result. With the conditional carry outputs of these adders, evaluation of underflow condition is a trivial operation. A $C_{out}(0) = 0$ during the subtraction of a leading zero count from the tentative exponent of the result indicates underflow.

4.2.2.5 Leading Zero Bounded (LZB) data path

This data path handles the signed-magnitude addition of significands in all those situations when the number of leading zeros of the result of such addition is guaranteed to be less than or equal to one. This include all cases of addition (when the sign of input floating

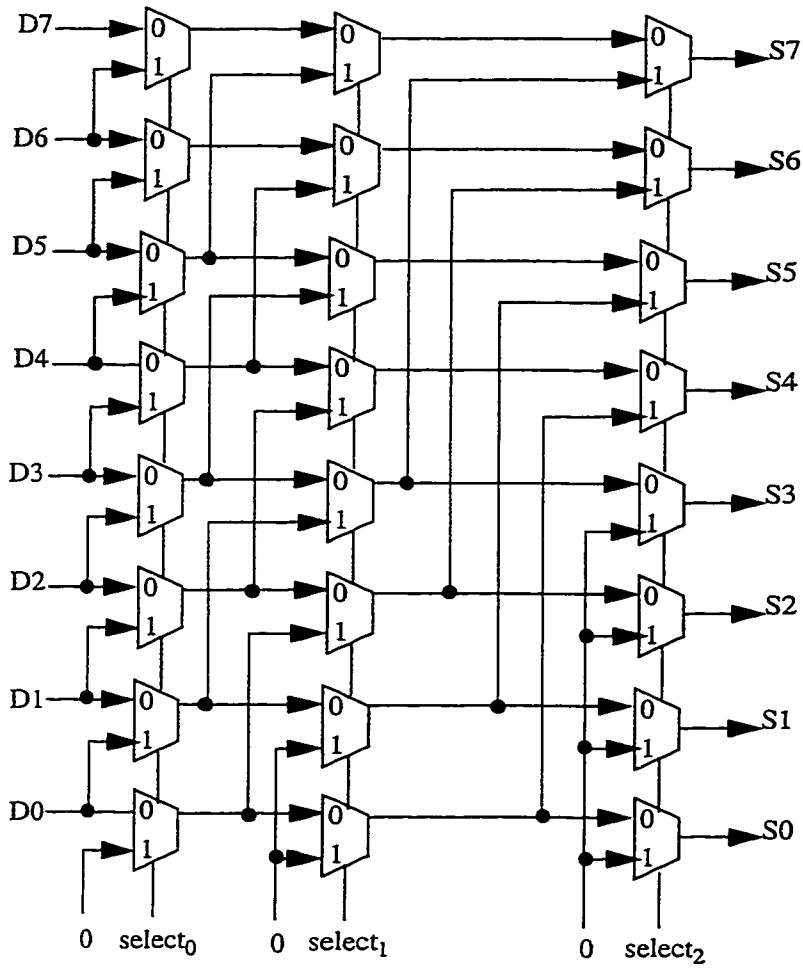


Fig 4.7 - Left barrel shifter

point numbers is equal). The subtraction performed by this data path is, however, restricted to those cases when exponent difference ($|e1 - e2|$) is greater than 1.

In LZB data path, the input Data Selector selects significands for pre-alignment operation. Using conditional carry outputs of the 1's complement adder, used for the evaluation of $|e1 - e2|$ condition in Exponent logic, significand with the smaller exponent is fed to the Pre-alignment logic for right shifting while significand with the larger exponent is routed

directly to the significand adder.

Pre-alignment logic is composed of a right barrel shifter. The barrel shifter shifts the significand of smaller number right by an amount that is equal to the exponent difference $|e1-e2|$. For values of exponent difference greater than the width of significand (p), the aligned significand will be zero. In this condition, Control logic activates the Bypass data path and LZB data path is transition activity scaled. Transition activity scaling of floating point adders during such situations results in power savings. In single precision floating point adders, the width of significand is 24, so the amount of pre-alignment shift is upper bounded by 24 and 5 bits are sufficient for effecting the significand shift. Five LSBs of the exponent difference are realized as the shift control data word. Again a barrel shifter with cascaded array of MUXs is used. The 8-bit version of right barrel shifter is shown in Fig 4.8. Two of the shifted out bits are retained as guard (G) and round (R) bits. In reference [44], authors proposed a sticky bit formation scheme for floating point adders through the evaluation of trailing zeros of the significand of the smaller number. In floating point additions, the magnitude of pre-alignment shift of the significand of the smaller number is given by $SH = |e_1 - e_2|$, where SH represents the magnitude of alignment shift and e_1 and e_2 represent the exponents of the floating point numbers. Since the number of shifted out (discarded) bits is less than the actual shift by an amount that is equal the number of guard bits (G, R bits), the effective shift as far as sticky bit evaluation is concerned is given by $SH_{eff} = SH - 2$. If the effective shift is greater than the number of trailing zeros, the sticky bit can be set to 1, and vice versa. Evaluation of the condition $SH_{eff} > TZ$ can be formulated to $SH - 2 - TZ > 0$, so that a 1's complement adder can be used for the evaluation of

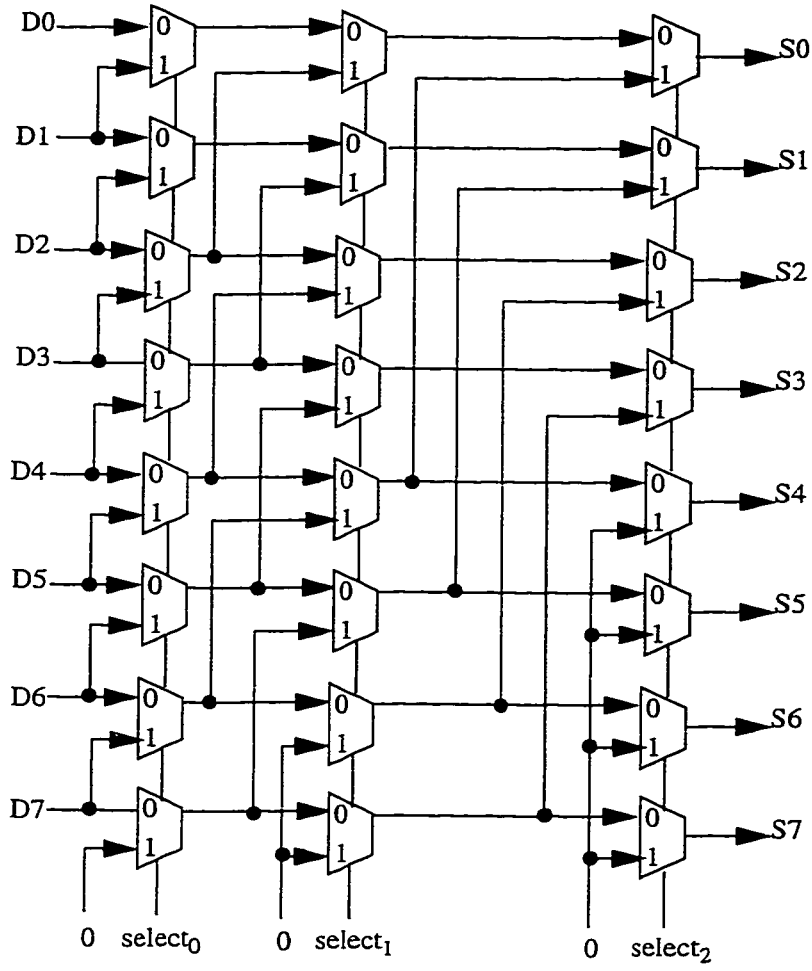


Fig 4.8 - Right barrel shifter

this condition. Trailing zeros are counted by a 32-bit tree type Trailing Zero Counter (16-bit version shown in Fig 3.6). Fig 4.8 (a) illustrates the input data representation for the evaluation of the above condition. The shift magnitude *SH* and trailing zero count, *TZ* are encoded into 5-bit numbers. The third row in Fig 4.9 (a) represents the 2's complement of 2. These three rows can be compressed into sum and carry vectors, as shown in Fig 4.9 (b) by a single row of 3:2 compressors. The end around carry of 1's complement arithmetic

0	SH_4	SH_3	SH_2	SH_1	SH_0
1	$\overline{TZ_4}$	$\overline{TZ_3}$	$\overline{TZ_2}$	$\overline{TZ_1}$	$\overline{TZ_0}$
1	1	1	1	1	0

(a) Input data for the evaluation of $SH - 2 > TZ$

	C4	C3	C2	C1	C0	
1	0	S4	S3	S2	S1	∅

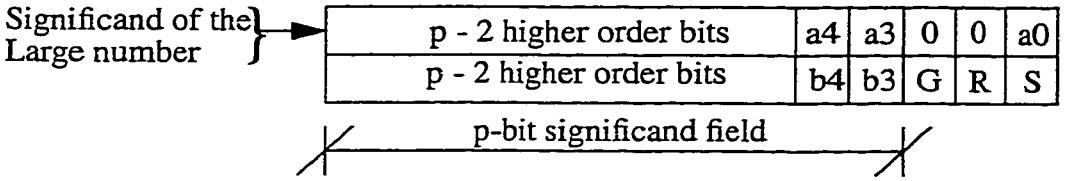
(b) Input data after bit compression

Fig 4.9 - Data presentation for the evaluation of $SH - 2 > TZ$

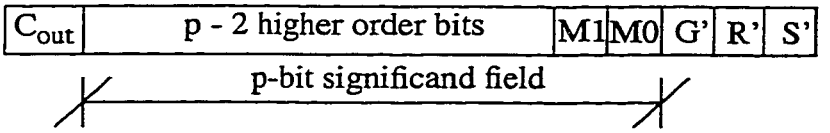
operation gives the required sticky bit by making use of a circuit shown in Fig 3.8. The computation of sticky bit through this approach is attractive for high speed of operation.

The output of the Pre-alignment logic is routed to the significand adder through a complementer which performs an inversion (bitwise) of its input data word during subtraction of one floating point number from another. Fig 4.10 (a) shows the data representation of the aligned significands, where *a0* bit is a 0 during addition and it will be a 1 during subtraction (for 2's complement addition of the aligned significand). The *G*, *R*, *S* and *bi* ($\forall i$) bits are the complements of the bits of the aligned significand during subtraction and are the true bits during addition. Fig 4.10 (b) represents the result of significand addition before normalization shift.

For rounding purposes, bit positions *M1* to C_{out} are treated as a single block. Since the rounding decisions are not available till the normalization shift is complete, pre-computation of two sets of sum bits is the best choice for Zero Overhead Rounding, so that by the time the rounding decisions are known, an appropriate set of sum bits can be chosen. The



(a) Pre-aligned significands



(b) Result of significand addition before normalization shift

Figure 4.10 - Data representation in LZB data path

sum bits are evaluated for incoming carries of zero (*sum0*) as well as one (*sum1*). Conditional sum/carry select adders are ideal for this application. Fig 4.11 gives the block diagram of the significand addition/normalization/rounding scheme for the LZB data path. As far as the bits towards the right of *M1* are concerned, conditional results for all possible cases of addition and rounding are pre-computed. The Result Selector and Normalization Shifter blocks append the rounded bits of the lower order bit positions (towards the right of *M1*) to that of the higher order bit positions (*M1* to MSB).

In LZB data path, rounding for addition and subtraction are distinct problems. These problems are further categorized into sub-problems, depending on whether *Cout* and leading zero occur during the signed-magnitude addition of significands.

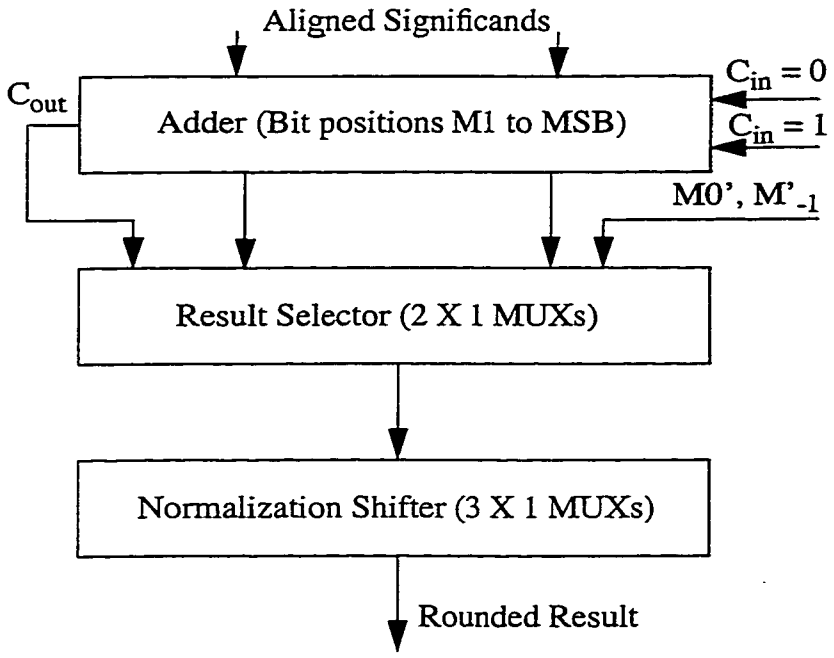


Fig 4.11 - Pre-computation, selection and normalization of rounded results in LZB data path

During addition, *a0* bit is a zero. When *Cout* is 1, the normalized result with new round and sticky bits is shown in Fig 4.12. *M0* of Fig. 4.10 (b) becomes the new round bit while the new sticky bit is the logical OR of all bits towards the right of *M0*. Hence

$$S'' = G + R + S \text{.....(4.3)}$$

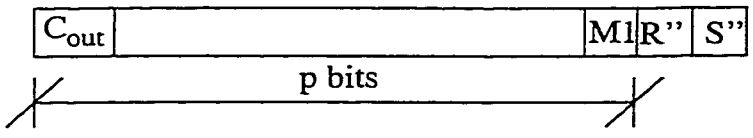


Fig 4.12 - Result of significand addition after normalization shift (Add operation)

$$R'' = M0 = a3 \oplus b3 \text{ .....(4.4)}$$

$$M1 = a4 \oplus b4 \oplus a3b3 \text{ .....(4.5)}$$

In the above equations, *G, R, S, ai* and *bi* bits represent the bits of the aligned significands, as shown in Fig 4.10 (a). The condition for rounding is given by *R"(M1 + S")*. If the rounding condition is true, a 1 must be added to *M1*. The carry in at *M1* bit position due to rounding is given by

$$Cin1 = a3b3 + (a3 \oplus b3)(sum0(0) + G + R + S) \text{ .....(4.6)}$$

In the above equation, *a3b3* represents the normal carry input to the M1 position (irrespective of whether there is any rounding condition or not) while the second term represents a

Table 4.4: Carry injection at M1 bit position

Case	Rounding condition	Carry injection at M1
1	Addition with Cout = 1	$Cin1 = a3b3 + (a3 \oplus b3)(sum0(0) + G + R + S)$
2	Addition with Cout = 0	$Cin2 = a3b3 + (a3 \oplus b3)G$
3	Subtraction with no leading zero	$Cin3 = a3b3 + (a3 \oplus b3)(G + RS)$
4	Subtraction with one leading zero	$Cin4 = a3b3 + (a3 \oplus b3)G(R + S)$

carry input that is generated due to rounding. Table 4.4 [46] lists the carry injection at *M1* bit position for all possible addition and rounding cases. If carry injection at *M1* is 1, Result Selector selects the conditional sum bits with anticipated carry-in of one (*sum1*) else *sum0* is selected.

The rounded bits towards the right of *M1* for all four cases of rounding and addition are given in Table 4.5 [46].

Table 4.5: Rounded M0 and M_{-1} bits

Case	Rounding condition	M0'	M'_{-1}
1	Addition with Cout = 1	Not required	Not required
2	Addition with Cout = 0	$(a3 \oplus b3)\overline{G} + \overline{(a3 \oplus b3)}G(R+S)$	Not required
3	Subtraction with no leading zero	$(a3 \oplus b3)\overline{G}\,\overline{RS} + \overline{(a3 \oplus b3)}G$	Not required
4	Subtraction with one leading zero	$a3 \oplus b3 \oplus G(R+S)$	$\overline{G}R + G\overline{(R+S)}$

The normalization shifter for this data path is a one-bit left/right shifter. Exponent computation of LZB data path is realized by using Exponent Increment/Decrement logic, which conditionally increments (subject to the generation of a carry-out from the MSB position during significand addition) or decrements (because of the generation of a leading zero during subtraction) the tentative exponent. Evaluation of overflow/underflow conditions is straightforward in this data path and it can be easily evaluated by the conditional carry outputs of 1's complement adders. During subtraction, an underflow condition can occur if and only if the tentative exponent is the smallest representable exponent, e_{min} (e_{min} = 1, for single precision floating point adders) and the result of the signed magnitude significand addition contains a leading zero. Underflow is evaluated by *Cout(0)* = *0*, where *Cout(0)* is the conditional carry-out of 1's complement adder with an anticipated carry-in of zero. Similarly, overflow can occur if the tentative exponent is the maximum representable exponent without overflow, e_{max} (e_{max} = 254 in IEEE single precision data format), and the significand addition produces a carry out from the MSB end of the significand

adder. The *Cout(1) = 1*, implies an overflow where *Cout(1)* is the conditional carry-out of 1's complement adder with a carry-in of 1.

4.2.2.6 Result Integration/Flag logic

Result Integration/Flag logic selects the appropriate copy of result from the relevant data path. It also asserts five status flags to represent zero, $-infinity$, $+ infinity$, not a number and denormalized results.

The block diagram in Fig 4.2 shows a low latency architecture of TDPFADD. Input and output registers are needed for the data alignment. A 2nd stage of registers is required for the transition activity scaling of the different data paths. Control logic controls the data presentation to the different data paths by generating control signals which can selectively activate or mute the clock signals to registers of certain data paths.

4.2.3 Pipelined TDPFADD

In [1], [36]-[44], the authors proposed the reduction of latency in TDPFADD by around two pipelined stages compared to the 5 stage single data path floating point adder proposed by H. Suzuki et al. [34]. One stage reduction due to Zero Overhead Rounding and the other due to the reduction in logic depths of the computing data paths by way of eliminating one barrel shifter from the critical path. As TDPFADD is evaluating the number of leading zeros by using a serial leading zero counter while single data path FADD [34] is using an optimized Leading Zero Anticipatory logic in which the delay of LZA logic is completely hidden behind the significand addition and as the first purpose of pipelining is to achieve maximum possible speed of operation without unnecessary increase in power

consumption and area, we present a 5 stage triple data path floating point adder as shown in Fig 4.13. The delay time of the third stage of LZA data path restricts the cycle time of pipelined TDPFADD because the significand adder of the LZA data path has the largest delay time. Clock of register stages 2, 3, 4, and 5 is gated by control signals, generated by Control logic, for the activation/de-activation of different data paths.

4.3 Hardware configuration of single data path floating point adder (FADD)

In this section the hardware configuration of a single data path floating point adder (FADD) [34] which utilizes an optimized approach for leading zero anticipation, and concurrent evaluation of normalization with rounding, for high speed operation, is presented. In this section first the most important feature of single data path FADD i.e. high performance leading zero anticipation is presented and then the hardware configuration of FADD using this LZA logic is discussed.

4.3.1 Leading Zero Anticipatory (LZA) logic

In a floating point addition operation, subtraction of significands can generate leading zeros in the resultant significand. The normalization shifter has to shift the significand to the left until the first '1' appears on the left end as the leading bit. When the two floating point numbers are close to each other, this procedure of normalization can introduce a significant amount of delay. If the consecutive bits of these leading zeros are counted in parallel with the addition/subtraction of significands, the total delay time for the FADD is greatly reduced and it can be helpful in the designing of low latency floating point adders.

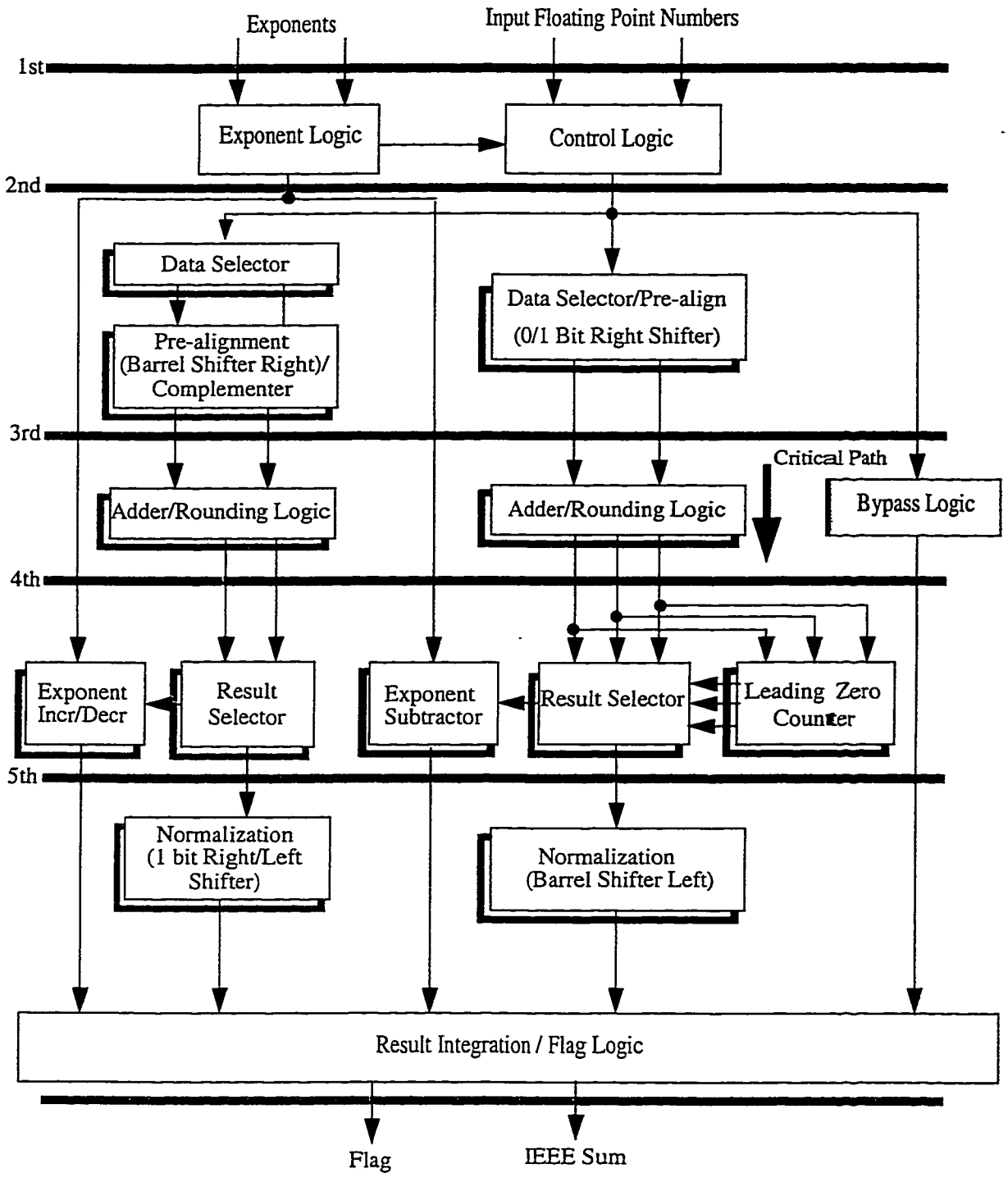
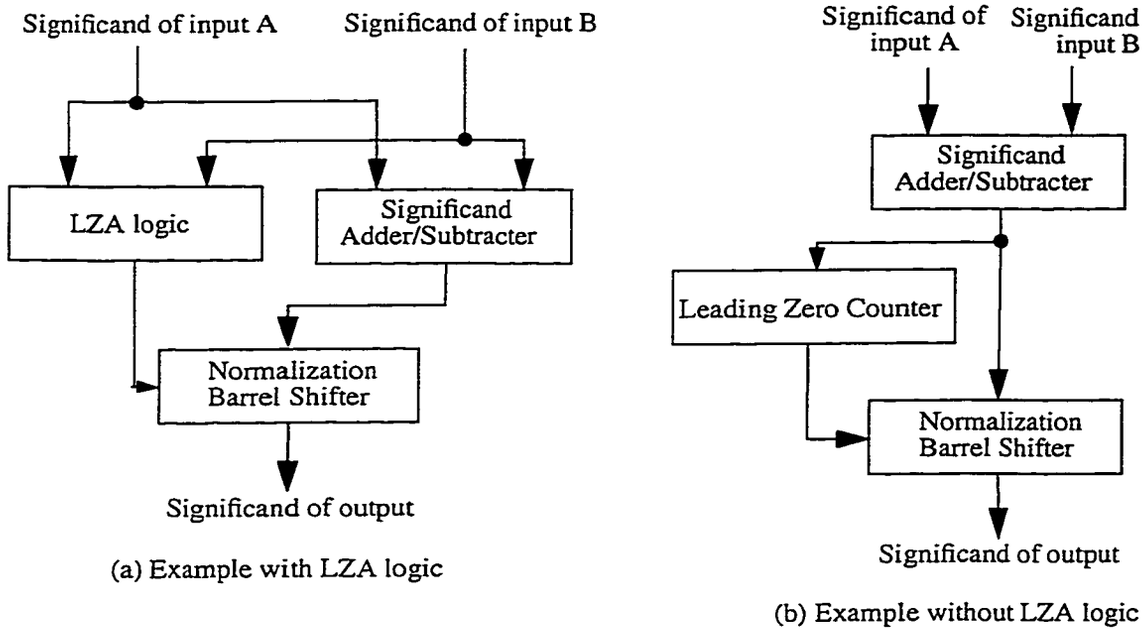


Fig 4.13 - Block diagram of the pipelined TDPFADD with five execution stages

Significand addition and normalization parts of floating point adders are shown in Fig 4. 14. Fig 4.14 (a) shows an example with LZA logic. The LZA logic anticipates the amount of shift for normalization. Because this amount is calculated from the source operands of the addition/subtraction of significands, the LZA performs this operation in parallel with addition/subtraction. Immediately after the addition/subtraction, the normalization shift can be performed since the shift amount is already been determined.

Fig 4.14 (b) illustrates a direct way to perform normalization. Once the addition/subtraction of significands is finished, a serial leading zero counter, counts and encodes the number of leading zeros and then the result is left shifted by the barrel shifter. LZA logic



(a) Example with LZA logic

(b) Example without LZA logic

Fig 4.14 - Addition/Normalization part of floating point adder

improves the performance of floating point adders by hiding the delay of leading zero estimation behind significand adder/subtracter. In reference [33], an LZA logic is proposed which is used in the design of low latency multiply-accumulate fused (MAF) unit [31]

[32]. But due to the complexity of the LZA unit, the performance acceleration comes with power and area penalties. Here overview of an optimized LZA logic, proposed by H. Suzuki [34] is presented.

The process of leading zero anticipation is divided into 2 parts. In the first part, LZA logic prepares the predicted consecutive bits of the leading zeros from the two source operands of significand addition/subtraction. Second, the leading zero counter decodes these consecutive bits to drive the normalization shifter. In this scheme, smaller significand is always subtracted from the larger significand, in which case a positive result is guaranteed. Let A and B are the significands of input floating point numbers. In case of subtraction, if $A \geq B$ then

$$S = A - B = A + \bar{B} + 1 \quad \text{.....(4.7)}$$

Let $Ap = A$ and $Bp = \bar{B}$, where Ap and Bp are the inputs to the LZA logic and significand adder. If $A < B$, then

$$S = B - A = \bar{A} + B + 1 \quad \text{.....(4.8)}$$

In this case, $Ap = \bar{A}$ and $Bp = B$. In case of addition

$$S = A + B \quad \text{.....(4.9)}$$

and $Ap = A$ and $Bp = B$. The leading 1, Ei, is drived as

$$Ei = \overline{(Ap_i \oplus Bp_i)} \bullet (Ap_{(i-1)} + Bp_{(i-1)}) \quad \text{.....(4.10)}$$

A possible gate level schematic of LZA logic, i.e. E_i, is presented in Fig 4.15. E_0 is always asserted as discussed below. The *OR* gate of the $(i - 1)$th bit is shared with the next block of the ith bit. The delay of the LZA logic is significantly less than the adder delay as unlike

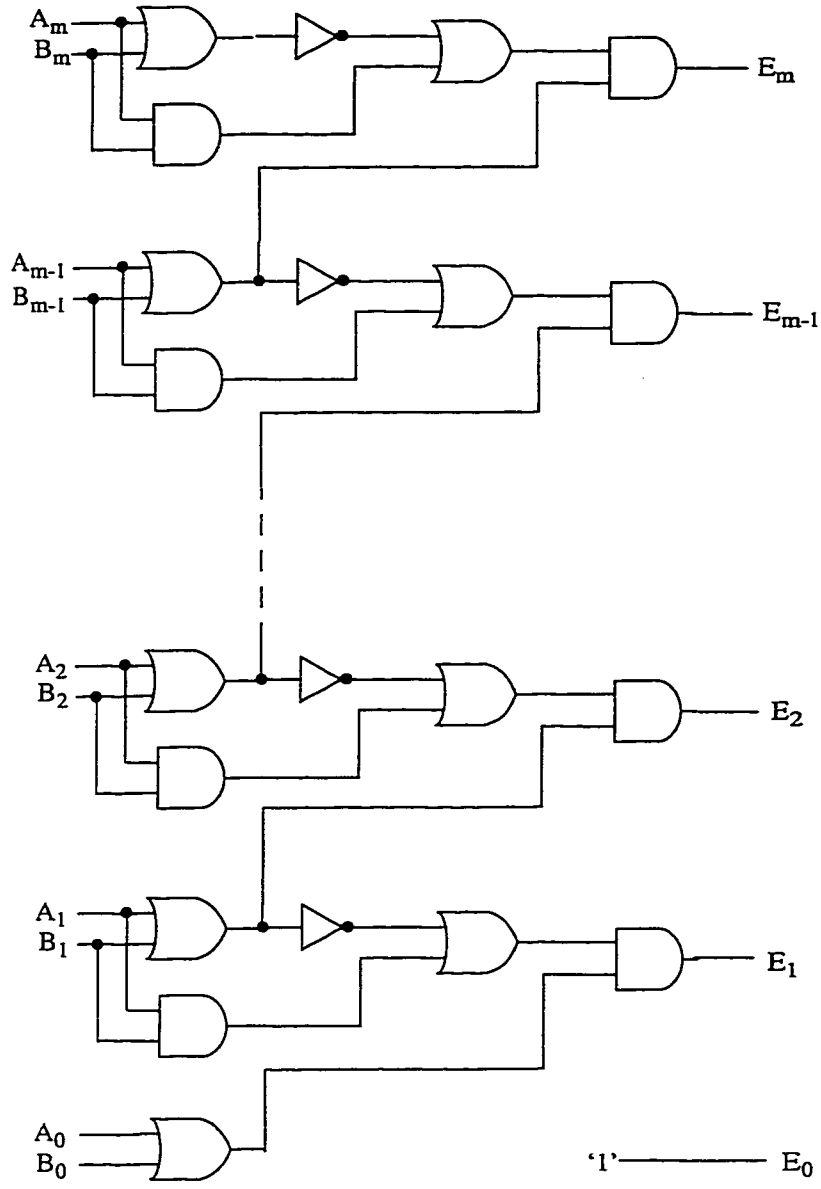


Fig 4.15 - LZA logic

the carry of adders, no signal is spread in more than two bit positions. This LZA logic has the possibility of wrong anticipation of leading 1 by one bit position as can be seen in the

behavior of E and S (sum bits) in the following examples [34]. In these examples, "-" is used to denote E_0 as it can't be evaluated from Equation (4.10). As $A \geq B$, so $A_p = A$, and $B_p = \overline{B}$, in case of subtraction.

Example 1:

A : + 1000 0000 1010 0001
B : — 0111 1111 0110 1001

↓

A_p: +0 1000 0000 1010 0001
B_p: +1 1000 0000 1001 0110 + 1

↓

E : 0 0000 0001 0100 100-
S : 0 0000 0001 0011 1000

In this example, leading bits of significands A and B are complement of each other, which makes the leading bits of A_p and B_p identical. Position of leading 1 is the same for both E and S and uncertainty about E_0 does not affect the result.

Example 2:

A : + 1001 1100 1010 0001
B : — 1001 1100 1000 1001

↓

A_p: +0 1001 1100 1010 0001
B_p: +1 0110 0011 0111 0110 + 1

↓

E : 0 0000 0000 0010 100-
S : 0 0000 0000 0001 1000

Leading bits of A and B are the same, which makes the leading bits of A_p and B_p complement of each other. In this case anticipated leading '1' is not matching with leading '1' of the sum bits. It is one bit left of the correct leading '1'. Status of E_0 is not affecting the

result.

Example 3:

```
A :  +    1001 1100 1010 1000
B :  —    1001 1100 1010 0111
                  ↓
Ap:  +0   1001 1100 1010 1000
Bp:  +1   0110 0011 0101 1000 + 1
                  ↓
E :    0  0000 0000 0000 000-
S :    0  0000 0000 0000 0001
```

In this example, subtraction of 2 significands result in S = "000...001", while E = "000...00-". Status of E_0 needs to be considered in this case. If E_0 is always asserted, E in this case will correctly anticipate the leading '1'. But this will lead to wrong anticipation in case of S = "000...000". But this error represents the anticipated leading '1' to be one bit left of correct leading one, which is the same mode as in Example 2. If E_0 is always de-asserted, this error will be different from the one mentioned in Example 2 and different correction logic (compensation shifter) needs to be designed. So E_0 is always asserted.

Example 4:

```
A :  +    1001 1100 0010 0001
B :  +    0011 1011 1100 1001
                  ↓
Ap:  +0   1001 1100 0010 0001
Bp:  +0   0011 1011 1100 1001 + 0
                  ↓
E :    1  0101 1000 0001 001-
S :    0  1101 0111 1110 1010
```

Example 4 represents a case in which the process is significand addition. In this case E

value at the sign bit will always be 1. So if the LZA logic is extended to sign bit, a no shift signal will be generated in case of addition of significands.

These four examples cover all possible behavior of LZA logic. It can be seen from the examples that LZA logic can either correctly anticipate the leading '1' or a one bit larger than the leading '1' of *S* bit vector. The wrong anticipation can be corrected by a compensation shifter, which can be shared by other shifters of floating point adders.

4.3.2 Architecture of FADD

Reference [34] also presents a single data path floating point adder (FADD) using IEEE double precision data format. For the comparison with TDPFADD [1], [36]-[44], it is modified to work with single precision data format. Fig 4.16 shows the block diagram of the single data path FADD. The operation of floating point addition is divided into five execution pipeline stages.

In the first stage, Exponent logic compares the exponents of two input floating point numbers for their relative magnitude and also calculate the exponent difference, which is the amount of pre-alignment shift. Relative magnitude of exponents helps in the selection of significand of floating point number with smaller exponent for pre-alignment shift. Also the larger exponent is selected as the tentative exponent of the result and sent to Exponent Subtracter in the 4th stage.

In the second stage, Pre-alignment logic performs the right shifting of significand, with smaller exponent, by an amount equal to the exponent difference, using a right barrel

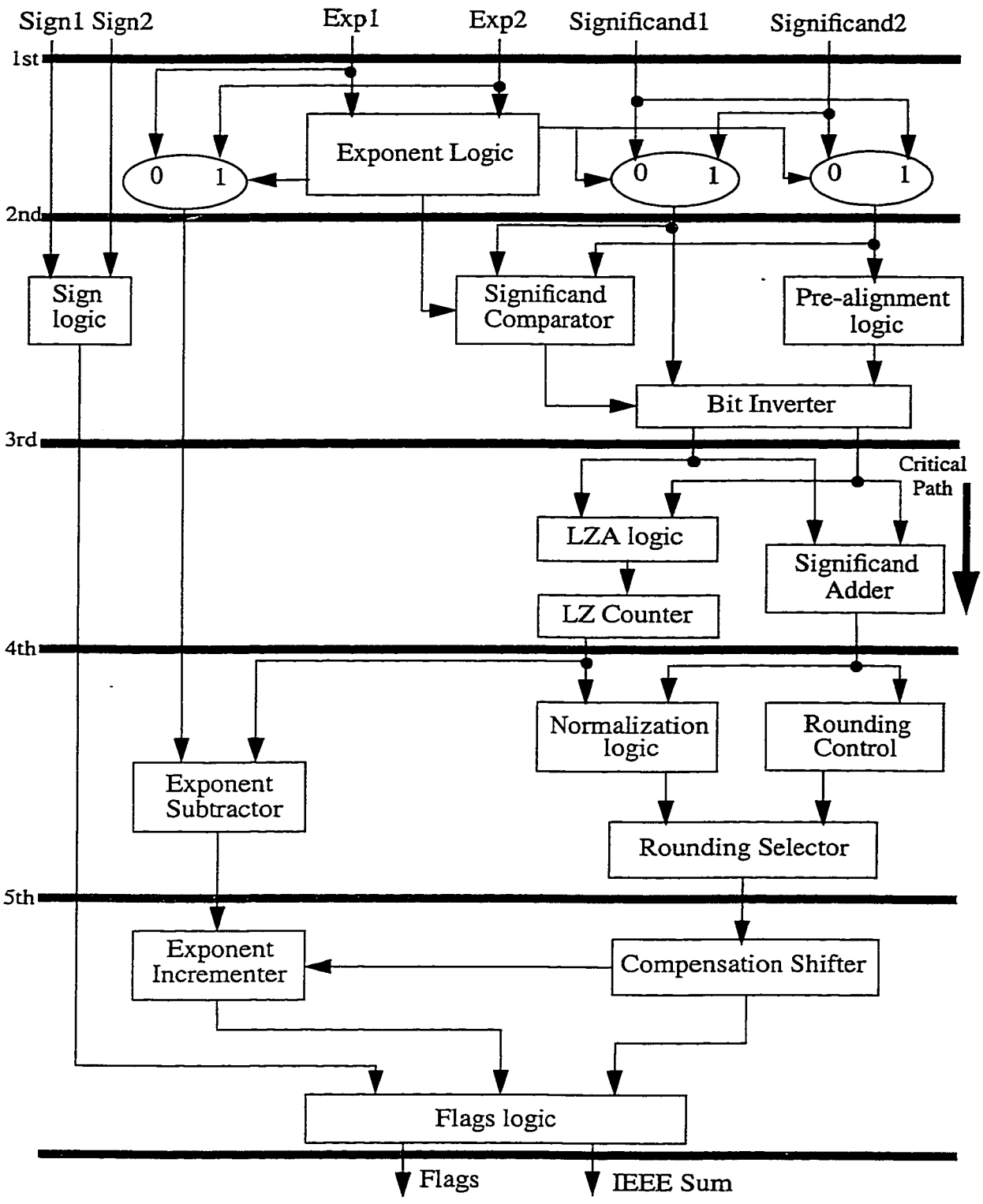


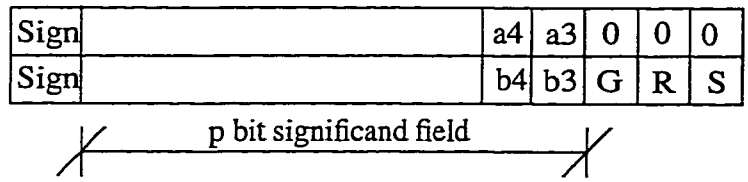
Fig 4.16 - Block diagram of single data path FADD with five execution stages

shifter. To reduce the complexity of barrel shifter, five LSBs of exponent difference are realized as the shift control data word. In case of single precision floating point adders, as the width of significands is 24 bits, any shift larger than 24 bits will result in a zero significand. So if either of 3 MSBs of 8-bit exponent difference is one, significand with the smaller exponent is shifted by a maximum amount which can be encoded by five bit data field i.e. 31, which produces a zero result from the output of barrel shifter. This reduces the delay, power consumption and area of right barrel shifters. The barrel shifter used here is of the same type as used for the pre-alignment shift in the LZB data path of TDPFADD. It is composed of cascaded array of 2X1 MUXs (Fig 4.8).

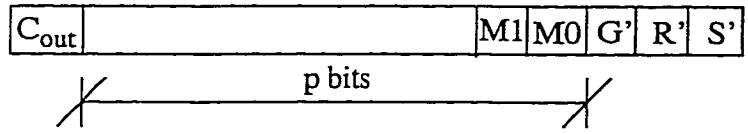
Instead of the rounding proposed in [34], round to nearest-even rounding mode is selected to make the floating point adder IEEE compliant. So two of the shifted out bits of the aligned significand are retained as guard (*G*) and round (*R*) bits. Sticky (*S*) bit, which is the logical *OR* of all shifted out (discarded) bits, can be evaluated by using different techniques. References [46], [1] present a technique for high speed evaluation of sticky bit using the trailing zeros count of significand of input floating point number, which is right shifted. This approach can be realized using high performance/low power/low area hardware as discussed in Section 3.2 of this thesis. The same technique is also used for single data path FADD.

In parallel with the pre-alignment shift, Significand Comparator compares and orders the significands so that subtraction results are always positive. In case of non-zero exponent difference, significand with larger exponent is selected as the larger significand. If the exponent difference is zero, only then two significands are compared for their relative

magnitudes. The LZA logic proposed in [34] can also work with the negative results in case of subtraction, as suggested in [35] but the absolute result of subtraction achieved by the use of Significand Comparator simplifies the Significand Adders and the concurrent rounding with normalization shift. In Bit Inverter, first *G*, *R*, and *S* bit positions of the significand which is not pre-aligned are asserted as "000", as shown in Fig 4.17. In case of subtraction, the Bit Inverter performs the inversion of the smaller significand. Most of the the time the right significand (output of pre-alignment logic) is inverted. The left significand is inverted only if it is smaller than the right significand and the exponent difference is zero. In case of addition, none of the significands is inverted.



(a) Pre-aligned significands before bit inversion



(b) Result of significand addition before normalization shift

Fig 4.17 - Data representation in single data path FADD

Third stage performs the significand addition and leading zero anticipation for normalization shift, in parallel. The optimized LZA logic presented in the previous section prepares the predicted consecutive leading zeros from the two source operands of significand addi-

tion. A 32-bit, tree type, leading zero counter also proposed in [34] (16-bit version shown in Fig 4.6) is used to decode these consecutive leading zeros. The circuit of LZA logic (shown in Fig 4.15) is extended to sign bits. This enables the LZA circuit to generate the unshift signal in case of add operation (*Example 4*, Section 4.3.1). Hence unshifted data will pass through the normalization barrel shifter and is used as rounded-down (not incremented) data during rounding operation. Hence, the extension of the sign bit enables the signal path of the round-down data to be shared with the signal path of the normalization shifter. Significand addition is performed by a 28-bit (24-bit significand, Sign, *G*, *R*, and *S* bits), two stage Carry Look-Ahead (CLA) adder [56], with a carry-in of 1, during subtraction operation for 2's complement addition (as smaller significand is always complemented, a positive result is guaranteed) and a carry-in of zero in case of an addition operation.

Normalization and rounding conditions are evaluated concurrently in the 4th stage. Rounding Controller evaluates the conditions for rounding and generate *Round* signal (*Round* = 1, represents round-up, and *Round* = 0, represents round-down) by examining the various bits of the result of significand addition (Fig 4.17 (b)). By utilizing IEEE default rounding mode of round to nearest-even, the rounding conditions are evaluated as below:

If MSB of result of significand addition is 1, it represents a case in which *Cout* = 1 during the add operation. Normalization logic will right shift the result by one bit and the rounding condition in this case is

$$Round = M0(M1 + G' + R' + S') \text{ .....(4.11)}$$

Where M1, M0, G', R' and S' are the LSBs of the result of significand addition as shown in Fig 4.17 (b). Else if the first bit to the left of MSB of the result of significand addition is one, no normalization shift is required and rounding condition is

$$Round = G'(M0 + R' + S') \text{.....(4.12)}$$

Else if the second bit to the left of MSB is one, it implies one leading zero exists and one bit left shift is needed. The rounding condition is

$$Round = R'(G' + S') \text{.....(4.13)}$$

Else if the third bit to the left of MSB is one, there are two leading zeros and

$$Round = R'S' \text{.....(4.14)}$$

For other possible values of the result of significand addition, round-up condition is always false. Normalization logic performs the normalization shift in parallel. In case of subtraction operation, it shifts the result of significand addition by an amount which is equal to the number of leading zero encoded by the combination of LZA logic and leading zero counter. A left barrel shifter (of the type shown in Fig 4.7) is used for this purpose. In case of addition operation, one bit right shift may be needed in case MSB of the result is one. In case of addition, the extended sign bit generates the control signal to bypass the non-shifted data through left barrel shifter. As round to nearest-even approach using *G*, *R*, and *S* bits is utilized for this single data path FADD, the normalized result has to be incremented because due to the normalization shift the position of conditional increment due to rounding is not the same. Rounding Selector selects the incremented or non-incremented

normalized result based on the value of *Round* signal from Rounding Control logic.

The 5th stage consists of Compensation Shifter and Exponent Incrementer. A one-bit significand shift is performed to compensate for any mis-shift caused by the wrong anticipation of leading zeros. As LZA logic is working on the operands which include the sign bit, it performs the correct anticipation of leading zeros in case the anticipated position of leading one (E) is one bit larger than the leading one of sum (S), as in *Example 2* and *Example 4* (Section 4.3.1). In case the leading one of E matches with that of S, the anticipated leading zero count and hence the left shift is 1 larger than the actual leading zero count (*Example 1* and *Example 3*, Section 4.3.1). The Compensation shifter performs one bit right shift to correct the mis-shift. In cases where p MSBs of the normalized significand are 1's, rounding can result in a carry-out from MSB. Here also a 1-bit right shift is required to bring the significand of the result back into normalized form. Compensation Shifter is shared with the shifter for the rounding procedure. In case of a 1-bit shift, exponent is incremented by one by the Exponent Incrementor.

Delay time of the third stage restricts the cycle time of the FADD as 28-bit Significand Adder has the largest delay time. As the LZA logic and LZ Counter have smaller delay than the adder so the process of leading zero anticipation is completely concealed behind the significand addition.

4.4 Evaluation Process

TDPFADD [1], [36]-[44], presented in section 4.2, and the single data path FADD [34], presented in section 4.3, are implemented by synthesizing towards 0.35-micron CMOS

and FPGA (Xilinx XC 4085XL) technologies. Comparison is done using two different technologies and cell libraries to validate the architectural approaches irrespective of the technology used. Both floating point adders are IEEE compliant using IEEE single precision floating point data format and IEEE default rounding mode of round to nearest - even. The comparison is done for both pipelined structures (presented in Figs. 4.13 and 4.16) and low latency architectures (TDPFADD presented in Fig 4.2, single data path FADD using only input output data alignment registers). Low latency architectures are desirable in applications like digital filters. As the power consumption of TDPFADD is highly dependant on the utilization of a certain data path, simulations are done using real data (test vectors generated by the conversion of real audio signals [55]) and synthetic data (test vectors generated manually using different probabilities of data path utilization to get the average and worst case power consumption). Switching activity from these simulations is back-annotated for the accurate calculation of power consumption.

Data path utilization probabilities of TDPFADD during the filtering of synthetic data are provided in reference [1]. These probabilities, listed in Table 4.6 [1], are also used here for the evaluation of relative magnitude of power consumption of TDPFADD and single data path FADD in different filter applications.

Filters IIR1, IIR2, IIR3, in the Table 4.6, are the direct form I, direct form II, and transposed direct form II realizations of 8th order elliptical filters (low pass), having pass band ripple of 0.03 dB, stop band ripple of -100dB and normalized cut-off frequency of 0.2. While FIR1, FIR2 and FIR3 are low pass FIR filters of order 64, 16 and 8 respectively,

Table 4.6: Data path utilization probabilities of TDPFADD during filtering of synthetic data

IIR1			IIR2			IIR3			FIR1			FIR2			FIR3		
P(I)	P(K)	P(J)	P(I)	P(K)	P(J)	P(I)	P(K)	P(J)	P(I)	P(K)	P(J)	P(I)	P(K)	P(J)	P(I)	P(K)	P(J)
0.0589	0.5952	0.3460	0.0589	0.6116	0.3295	0.0595	0.7049	0.2356	0.2002	0.6755	0.1243	0.1765	0.6779	0.1456	0.1111	0.7415	0.1474
0.0589	0.6406	0.3005	0.0589	0.6373	0.3038	0.0601	0.6952	0.2447	0.2002	0.6963	0.1035	0.1765	0.7571	0.0664	0.1111	0.8485	0.0403
0.0589	0.6180	0.3231	0.0589	0.6162	0.3250	0.0599	0.7031	0.2370	0.2002	0.7033	0.0965	0.1765	0.7227	0.1007	0.1111	0.7938	0.0950

with normalized cut-off frequency of 0.2. The first among the synthetic signals is a sequence of white noise samples (N(0,1) IID RVs) of sample size 128K, while the second and third are auto regressive signals of the same sample size. The AR model of the second signal is $y[n] = x[n] + 0.9*y[n - 1]$ while that of the third signal is $y[n] = x[n] + 0.5*y[n - 1]$. Where *p(I), p(J),* and *p(K)* represents the probabilities when bypass, LZA and LZB data paths are active as shown in Fig 4.1 and Table 4.2. Data path probabilities of TDP-FADD, listed in Table 4.6, are used to make artificial data files, which utilize different data paths of TDPFADD according to the respective probabilities. Both single data path FADD and TDPFADD are simulated using these data files. Switching activity from these simulation is back-annotated to Synopsys Power Compiler and Design Power tools, where these switching activities are used for power calculations. The relative power consumption of TDPFADD and single data path FADD when they are part of these filter programs processing different signals is obtained.

4.5 Results

Table 4.7 presents the comparison of the low latency architectures of both floating point adders using 0.35 micron CMOS technology. Maximum delay, power consumption and area information is obtained from reports generated after synthesis. The calculated values

of Power-Delay (PD), Area-Power (AP), Area-Delay (AT) and Area-Delay^2 (AT^2) are also presented in Table 4.7. In this table, P_{real} presents the relative power figures during the addition of real data. Furthermore, P_a is the power consumption of 2 floating point adders when the average behavior of TDPFADD is considered (i.e. the probability of utilization of all three data paths is the same), while P_w shows the worst case power figures of TDP-FADD (data is always passing through the LZA data path, in which case TDPFADD consumes the maximum amount of power).

Table 4.7: Comparison of low latency architectures of TDPFADD and single data path FADD using 0.35 micron CMOS technology

Parameters	TDPFADD	Single data path FADD
Maximum Delay, D (ns)	40.88	58.64
Average Power, P_a (mW) at 16.7 MHz	26.53	47.18
Worst case Power, P_w (mW) at 16.7 MHz	37.91	46.23
Power using real data, P_{real} (mW) at 16.7 MHz	30.71	41.3
Area, A (10^4 cell-area)	32.64	20.23
Power-Delay Product, PD (ns.mW)	1591.8	2830.8
Area-Power Product, AP (10^4cell-area.mW)	865.94	954.45
Area-Delay Product, AT (10^4cell-area.ns)	1334.32	1186.29
Area-Delay^2 Product, AT^2 (10^4cell-area.ns^2)	5.45 x 10^4	6.96 x 10^4

It can be seen from the results that the average power delay product of TDPFADD is 43.77% less than that of single path FADD. Reduction in the worst case power delay prod-

uct given by TDPFADD over FADD is about 18% while power delay product of TDP-FADD is 25.64% less than that of single data path FADD during the processing of real data. Moreover, TDPFADD is around 30.29% faster and takes 38.02% more area than FADD.

Using the data path utilization probabilities of TDPFADD during the filtering of synthetic data, presented in Table 4.6, the percentage reduction in power delay product offered by TDPFADD over single data path FADD are presented in Table 4.8.

Table 4.8: Percentage reduction in power-delay product offered by low latency TDPFADD over FADD during the filtering of synthetic data (using 0.35 CMOS technology)

Sl. #	IIR1	IIR2	IIR3	FIR1	FIR2	FIR3
1	37.45	39.44	39.93	46.66	46.70	42.60
2	38.06	38.06	39.93	45.07	48.42	48.40
3	38.58	38.58	39.93	45.07	48.00	43.21

Comparison of Table 4.6 and Table 4.8 indicates that TDPFADD offers power saving. However the amount of saving is highly dependent on the input data. Whenever the bypass is used instead of LZA data path more saving in power is obtained. Among the results, TDPFADD is offering maximum reduction (48.42%) in power delay product when second signal is filtered through the FIR filter of order 16 (FIR 2) because the probability that the data will pass through LZA is quite low (approximately 6%) and the probability of Bypass is higher (around 18%). Also, TDPFADD is offering minimum reduction in power delay product (37.45%) when first signal is processed by the direct form I, IIR filter because the probability of Bypass is minimum (around 6%) and the probability that the data will pass through LZA is maximum (around 35%).

In DSP applications, certain arithmetic operations can't affect the final result due to the limitations of floating point data formats. In these circumstances, bypass in floating point addition becomes important. Symmetrical band pass/stop filters having a normalized centre frequency of 0.5 exhibit relatively large differences between the exponents of adjacent filter co-efficients. With these filters, the probability that the TDPFADD endures bypass is relatively significant. During precision limited IIR filtering of white noise samples, probability of bypass data path utilization is around 53% and the probabilities of utilization of LZA and LZB data paths are nearly 11% and 36% respectively [1]. In this case, TDPFADD consumes 21.35 mW power at 16.7 MHz and offers 52.82% reduction in power delay product over FADD, whose power consumption is 45.25 mW. These results are also reported in [44].

Table 4.9: Comparison of low latency architectures of TDPFADD and single data path FADD using FPGA technology

Parameters	TDPFADD	Single data path FADD
Maximum Delay, D (ns)	213.81	327.64
Average Power, P_a (W) at 2.38 MHz	1.0249	1.8365
Worst case Power, P_w (W) at 2.38 MHz	1.7686	1.8506
Power using real data, P_{real} (W) at 2.38 MHz	1.2476	1.8533
Area, A, Total CLBs (#)	1035	664
Power-Delay Product, PD (ns.10mW)	4.3×10^4	7.71×10^4
Area-Power Product, AP (10#.10mW)	1.06×10^4	1.22×10^4
Area-Delay Product, AT (10#.ns)	2.21×10^4	2.18×10^4
Area-Delay^2 Product, AT^2 (10#.ns^2)	4.73×10^6	7.13×10^6

Results of comparison of low latency architectures of floating point adders using FPGA technology are presented in Table 4.9. Average power delay product of TDPFADD is 44.19% less than that of single path FADD. Note that TDPFADD gives 4.43% reduction in power delay product over FADD in the worst case. Using real data, TDPFADD offers 32.68% reduction in power delay product compared to FADD. Also, TDPFADD is 34.74% faster and utilizes 35.85% more CLBs than FADD.

Table 4.10 gives the reduction in power delay product offered by TDPFADD over FADD, using FPGAs, during the filtering of synthetic data. Again TDPFADD gives substantial amount of power savings when bypass conditions are true more often and the chances of utilization of LZA data path are less. Considering Precision limited IIR filtering, TDP-FADD offers 60.31% decrease in power delay product compared to single data path

Table 4.10: Percentage reduction in power-delay product offered by low latency TDPFADD over FADD during the filtering of synthetic data (using FPGA technology)

Sl. #	IIR1	IIR2	IIR3	FIR1	FIR2	FIR3
1	37.75	40.17	43.14	53.20	51.35	47.46
2	41.98	41.98	43.14	52.71	51.35	54.49
3	42.70	42.70	43.14	52.71	52.34	48.00

FADD. Both architectures of floating point adders are also pipelined to 5 stages. Table 4.11 presents the comparison between pipelined structures of TDPFADD, shown in Fig 4.13, and single data path FADD, shown in Fig 4.16, using 0.35 micron CMOS technology. In this case, reduction in average power delay product offered by TDPFADD over

single data path FADD is 35.53%. Worst case reduction offered by TDPFADD over FADD is 27.96%. Using real data, TDPFADD gives power delay product that is 28.28% less than that of FADD. Performance of TDPFADD is 9.07% better than that of single data path FADD, while it utilizes 18.8% more area compared to FADD.

Table 4.11: Comparison of pipelined architectures of TDPFADD and single data path FADD using 0.35 micron CMOS technology

Parameters	TDPFADD	Single data path FADD
Maximum Delay, D (ns)	17.34	19.07
Average Power, P_a (mW) at 50 MHz	34.84	54.04
Worst case Power, P_w (mW) at 50 MHz	40.64	56.41
Power using real data, P_{real} (mW) at 50 MHz	35.53	49.54
Area, A (10^4 cell-area)	49.19	39.94
Power-Delay Product, PD (ns.mW)	696.8	1080.8
Area-Power Product, AP (10^4cell-area.mW)	1713.78	2158.36
Area-Delay Product, AT (10^4cell-area.ns)	852.95	761.66
Area-Delay^2 Product, AT^2 (10^4cell-area.ns^2)	1.48×10^4	1.45×10^4

Table 4.12 gives the percentage of reduction in power delay product offered by pipelined TDPFADD over pipelined FADD during filtering of synthetic data. During precision limited IIR filtering of white noise samples, pipelined TDPFADD consumes 30.477mW at 50 MHz, while power consumption of pipelined FADD is 52.2382 mW at the same frequency, which accounts for 41.66% reduction in power delay product by TDPFADD.

Table 4.12: Percentage reduction in power-delay product offered by pipelined TDPFADD over FADD during the filtering of synthetic data (using 0.35 CMOS technology)

Sl. #	IIR1	IIR2	IIR3	FIR1	FIR2	FIR3
1	31.05	33.07	32.63	36.03	35.90	32.58
2	31.00	31.00	32.63	34.44	35.67	35.19
3	31.35	31.35	32.63	34.44	35.51	35.52

Table 4.13 lists the comparison results of pipelined floating point adders using FPGA technology. The reduction in power delay product offered by TDPFADD over FADD consider average and worst case behavior and real data are 20.63%, 5.51% and 15.7% respectively. Pipelined TDPFADD is 25.24% faster and utilizes 28.85% more CLBs compared to pipelined single data path FADD.

4.6 Discussion

Results presented in the foregoing section show similar trends. Energy delay implications of transition activity scaled, triple data path floating point adder (TDPFADD) [1], [36]-[44], are always better compared to single data path FADD [34], using any of the two available technologies and for both pipelined and low latency structures. Area of TDPFADD is always more than FADD.

Due to the transition activity scaling, whenever exponent differences are greater than the width of significands, TDPFADD endures bypass data path and gives a large amount of power saving. In TDPFADD, shift operations are controlled by a few LSB bits of the

Table 4.13: Comparison of pipelined structures of TDPFADD and single data path FADD using FPGA technology

Parameters	TDPFADD	Single data path FADD
Maximum Delay, D (ns)	101.11	135.24
Average Power, P_a (W) at 5 MHz	0.8035	1.0123
Worst case Power, P_w (W) at 5 MHz	1.0201	1.0796
Power using real data, P_{real} (W) at 5 MHz	0.8661	1.0274
Area, A, Total CLBs (#)	1324	942
Power-Delay Product, PD (ns.10mW)	1.61×10^4	2.02×10^4
Area-Power Product, AP (10#.10mW)	10.64×10^3	9.54×10^3
Area-Delay Product, AT (10#.ns)	1.34×10^4	1.27×10^4
Area-Delay^2 Product, AT^2 (10#.ns^2)	1.35×10^6	1.72×10^6

exponent difference. With single data path FADD, even during situations when the exponent differences are greater than the width of significand, the significands may be shifted through certain finite number of bit positions, as the barrel shifters are not activity scaled. Also with TDPFADD, because of the simplification of barrel shifters, the number of layers of nodes undergoing toggling activity is reduced. Compared to conventional FADD schemes, the computing data paths of the TDPFADD activate only one barrel shifter while all other schemes activates two of them. The pre-alignment shifting of LZA data path as well as normalization shifting of LZB data path are handled by single stage MUXs. Compared to multi-stage barrel shifters, the data path switching activities of these single stage

shifters are insignificant. In contrast to other schemes, the leading zero counting logic of the TDPFADD is operational only during situations that demand its use and it can be seen from Table 4.6 that the probability that FP additions produce a significand with a variable number of leading zeros, is relatively small.

In single data path FADD, some optimizations made to achieve high performance also results in the reduction of power consumption of this unit. The simplicity of the optimized LZA logic, which not only hides the delay of estimation of leading zero count completely behind the delay of significand addition but its power consumption and area are also significantly less than the conventional LZA logic [33]. But efforts made in TDPFADD for low power outperforms the single data path FADD as far as the relative power implications of the two architectures are concerned. Relative power savings given by TDPFADD over single data path FADD are less when the two architectures are pipelined. The reason behind this is the power consumption of a large amount of registers used in the pipelining of 3 data paths of TDPFADD.

Performance improvement was the main goal of the designing of single data path FADD. For this, optimization of LZA logic and concurrent evaluation of rounding along with normalization shift, were introduced. But the data path simplifications and Zero Overhead Rounding approach of TDPFADD makes its performance better than single data path FADD even though it is using a serial leading zero counter. One factor which increases the delay and latency of TDPFADD is the extra delay imposed during the evaluation of many activity scaling conditions. If this delay is overcome by instruction driven processors, the performance and latency of TDPFADD can be even better.

The optimized LZA logic used in the single data path FADD is also area efficient but the area implications of FADD are increased by the requirements for dedicated significand comparator and correction shift logic. With TDPFADD, the merging of rounding operation with significand addition and the absence of a significand comparator offers area reduction. The use of 1's complement adders for the evaluation of functions like $|e1 - e2|$ (as well as $e1 > e2$, $e1 = e2$ and $e1 < e2$) involving integer data $e1$ and $e2$ is attractive as far as power and area reduction are concerned. Still the introduction of extra data paths for transition activity scaling makes the area requirements of TDPFADD more than that of the the single data path FADD.

4.7 Conclusion

Comparison of two different architectures of floating point adders is performed using different technologies. The results obtained show the validity of TDPFADD over single data path FADD for power/performance critical applications where there is room for extra data paths.

Chapter 5

Floating Point Multiply - Accumulate Fused Units

5.1 Introduction

Multiply - Accumulate operation is essential in many applications like DSPs and ALUs. Because of increase in application programs that envisage arithmetic operations involving dot product computations, modern day ALUs are equipped with dedicated multiply accumulators (MACs). Since the number of computational operations in dot product process are more than one - evaluation of a product and summation of this product with another operand - the time complexity of dot product operations are relatively high. The execution time of such operations may, however, be reduced by incorporating concurrency. Architectures that use concurrency for the performance acceleration of MAC operations are termed multiply - accumulate fused (MAF) architectures. Since the most common use of floating point multiplication is for dot product operations, a single unit which performs the multiply - accumulate operation $(A \times B) + C$ as an indivisible operation would produce a significand reduction in latency, chip busing, number of ports and distinct functional units while increasing the performance of floating point operations.

Fusion of multiply - accumulate operation involving fixed point operands is straight forward, the operand to be added with the product is treated as any other partial product. With floating point operands, because of the requirements for significand alignments during addition, the fusion of MAC operations is quite complicated. Though the complexity of floating point hardware units that envisage fusion of multiply - accumulate operations is

relatively high in comparison with traditional approaches of MAC, MAF architectures are still the preferred choice for time critical applications.

The first floating point multiply - accumulate fused (MAF) architecture was presented in [31], [32], as part of IBM RISC/6000 floating point execution unit. While IBM MAF demonstrates the feasibility of floating point multiply - accumulate fusion, this MAF has certain limitations as far as compliance with IEEE floating point standards [6] is concerned. Also in this MAF, due to the increase in the width of significand data path, the amount of power consumption is high and the Leading Zero Anticipatory (LZA) logic [33], used in this MAF is suboptimal in terms of speed, power, area and complexity.

R. V. K. Pillai [1], [50], proposed a low power, high performance, transition activity scaled architecture of multiply - accumulate fused unit which consists of four data paths. The results produced by this MAF are compatible with IBM MAF's results. In this architecture, both partial products of significand multiplication and previously accumulated sum can be shifted for pre-alignment, unlike IBM MAF's approach of always shifting the previously accumulated sum (left or right) with respect to the positions of the partial products of significand multiplication. This gives power-optimal results due to the reduction in the width of significand data paths. Critical path delay and latency are reduced in this MAF by incorporating data path simplifications and Zero Overhead Rounding [46].

The above two MAFs do not produce results that conform to IEEE standards, though the numerical accuracy of results is better than that of IEEE conformal schemes. In order that the results of multiply - accumulate operation be conformal with IEEE standards, the result of multiplication and addition need separate rounding. With the IBM scheme,

rounding is performed only once, which is rather a compound rounding operation encompassing multiply and accumulate. R. V. K. Pillai et al. [51] also proposed a low power floating point multiply - accumulate fused architecture which produces results that comply with IEEE standards.

In this chapter, comparison of the above three state of the art multiply - accumulate fused units is presented. The comparison is done by synthesizing each architecture using different technologies to evaluate the effect of different architectures on parameters like speed, power consumption and area. All three MAFs are designed to work with IEEE single precision data format. The rest of the chapter is organized in the following order. Section 5.2 presents the architecture and hardware configuration of low power, transition activity scaled, IBM compatible MAF unit [1] [50]. Section 5.3 presents the implementation of IEEE compatible MAF [51]. Section 5.4 presents the hardware configuration of IBM MAF [31] [32]. Section 5.5 highlights the evaluation process for the above 3 MAFs. Section 5.6 presents the results. Section 5.7 presents a discussion highlighting the salient features of different architectures of MAF unit. Section 5.8 concludes the chapter.

5.2 Hardware configuration of IBM Compatible MAF Unit

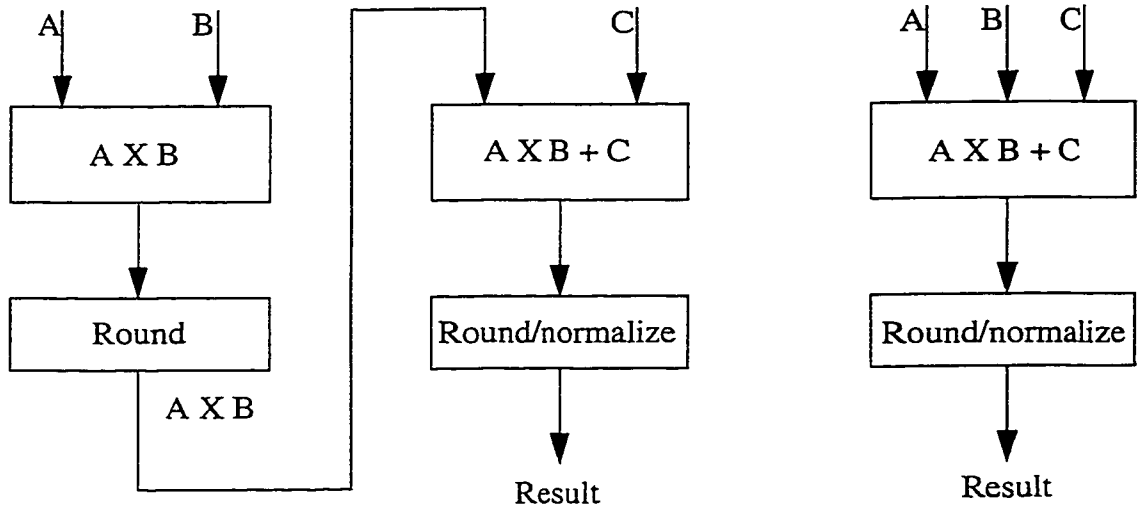
In this section first the approach used for the design of an optimized multiply - accumulate fused architecture [1], [50] is discussed and then the hardware configuration and analysis of this MAF are presented.

5.2.1 The multiply - add fused (MAF) scheme

The concept of multiply - accumulate fusion, used for the designing of this MAF unit and

also the IBM MAF, is illustrated in Fig 5.1 [31]. In the classical approach of multiply accumulation, the operation of multiplication and summation are performed sequentially and floating point product and sum are separately rounded. In multiply - accumulate fusion, the multiplication and addition operations are merged together and a compound rounding operation is performed in the end. This rounding operation provides additional accuracy but the compatibility with IEEE standards is suffered which require separate rounding of results of multiplication and addition.

In IBM MAF, the position of product of signifcands is taken as reference and significand of previously accumulated sum (C) gets aligned all the time irrespective of the value of its exponent. For those values of exponents of C that are greater than that of the product $A \times B$, the significand of C is left shifted through an appropriate number of bit positions



(a) Classical MAC (b) MAC fusion

Fig 5.1 - Multiply - accumulate fusion

and vice versa as shown in Fig 5.2. This results in an increase in the width of significand data path which demands the use of barrel shifters, CPAs and leading zero estimators of

large bit width. Floating point MAF [1], [50], discussed in this section tries to reduce the width of data field and hence the power/delay implications.

According to this approach, both significand of C and product of significands can be pre-aligned (right shifted) with respect to each other depending on the relative magnitude of their exponents. Shifting of product is accomplished through the shifting of one of the significands and then forming the product as is evident from the following equation [1], [50].

$$\frac{AB}{2^{edif}} \equiv A\frac{B}{2^{edif}} \text{.....(5.1)}$$

A right shift of the product of the significands of A and B is equivalent to dividing this product by $2^{e_{dif}}$, where e_{dif} represents the absolute value of the difference between the exponents of the product and C i.e. the amount of pre-alignment shift.

Multiplication of p-bit significands produce a product that is $2p$ bit long. The *Inc* bit repre-

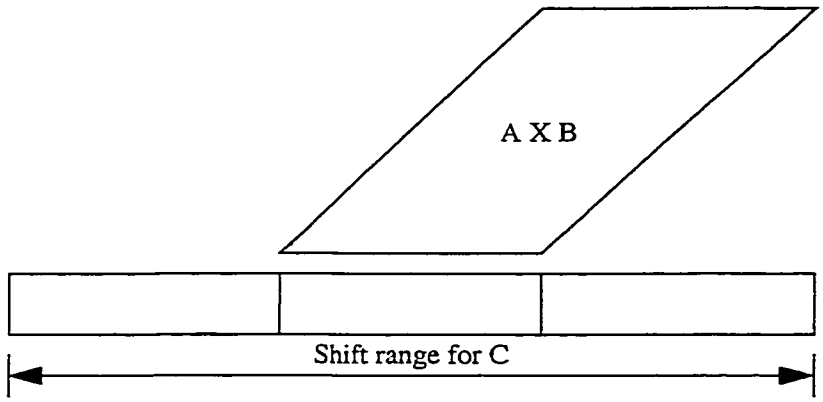


Fig 5.2 - Pre-alignment shift in IBM MAF

sents the MSB of the pre-normalized product, which occurs at the 2nd bit position towards the left of the binary reference point (*Inc* is one only if the *product of significands* ≥ 2). The shift magnitude, e_{dif} is given by

$$e_{dif} = |e_T - e_C| = |e_A + e_B - e_{bias} - e_C| \quad \text{.....(5.2)}$$

where e_C and e_T represent the exponent of C and the tentative exponent of product. e_A and e_B are the exponents of A and B and e_{bias} represents the exponent bias. The tentative exponent of the product for the purpose of shift evaluation is

$$e_T = e_A + e_B - e_{bias} \quad \text{.....(5.3)}$$

Since IBM compatible MAF (the MAF scheme presented in [1], [50]), envisages a conditional shifting of the product of significands through the shifting of one of the component significands, it selects significand of multiplicand for shifting instead of multiplier's significand. In this case, the number of partial products is always fixed - with p-bit significands, the number of partial products is p. Fig 5.3 illustrates the approach of pre-alignment used in the IBM compatible MAF.

If all of the shifted out bits of the significand are retained for partial product generation, the width of the partial products is $e_{dif} + p$. However, the upper bound for shifting the multiplicand significand is $p + 1$. With $(p + 1)$-bit shift of the multiplicand, there is still a chance of non-zero product bit at G bit's position as long as *product of significands* ≥ 2. This bit is used in the evaluation of conditional rounding of the significand of C. Shifting of the multiplicand significand beyond $p + 1$ bits is guaranteed to produce a product

whose summation with the significand of *C* produces a result which is no different than *C*.

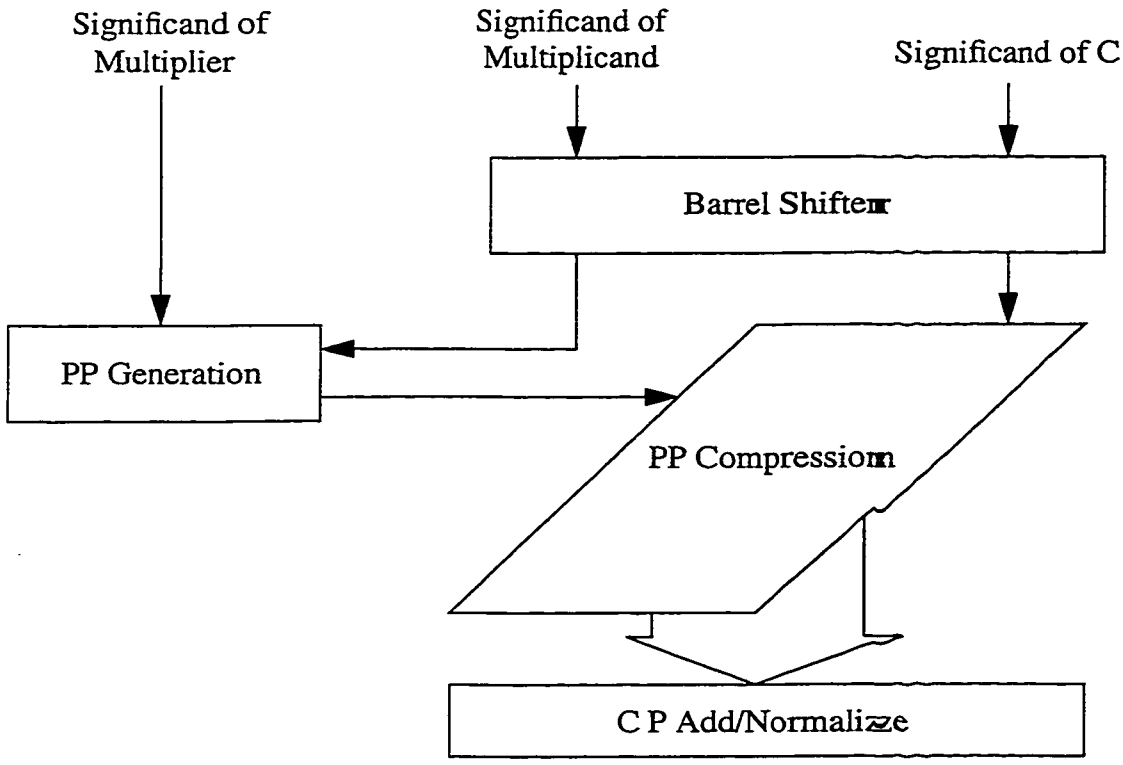


Fig 5.3 - Pre-alignment approach of IBM compatible MAF

If all the shifted out bits of the multiplicand significand are retained, the width of the multiplicand is $2p + 1$ and the width of the aligned product becomes $3p + 1$ as shown in Fig 5.4. Among $3p + 1$ bits, $2p - 3$ of the LSBs are seldom used in the subsequent addition with the significand of *C*. As the MAC operation envisages a single rounding step (instead of rounding the product and rounding the sum), these bits are only computed for the evaluation of sticky bit of the aligned product and the carry injected towards the left of the sticky bit. For IEEE multipliers, the sticky bit can be generated through an evaluation of the trailing zeros of the product. So the complexity of the partial product array can be reduced by discarding some of the shifted out bits of the aligned significand. In reference [1], [50], the authors suggested to keep all the shifted out bits for the applications which

require high precision floating point computations. They also gave another scheme that

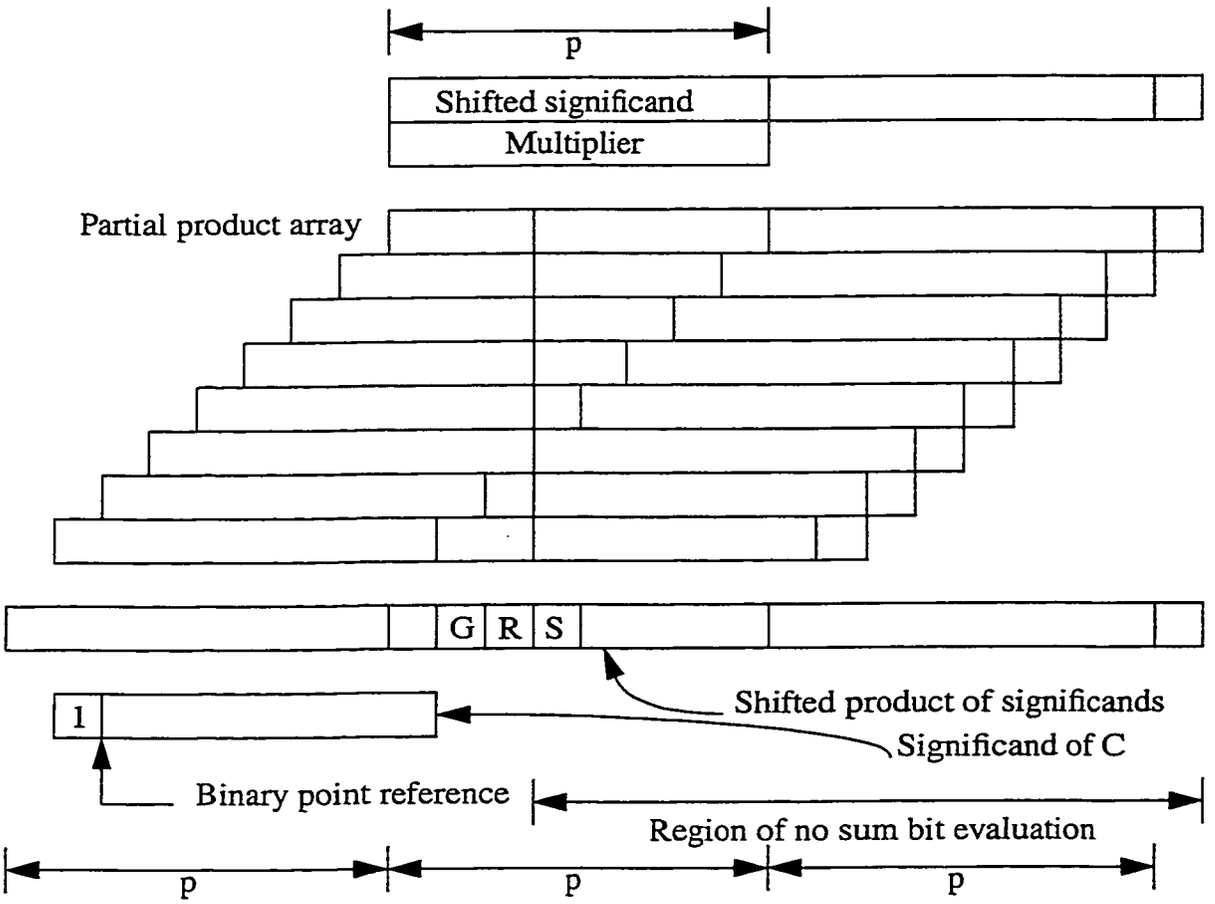


Fig 5.4 - Partial product array by keeping the $p + 1$ shifted out bits of multiplicand

retains four of the shifted out bits which results in compact partial product arrays, that is attractive as far as power/delay/area considerations are concerned. In the hardware configuration of this MAF, 8 of the shifted out bits of the multiplicand were kept as shown in Fig 5.5. In this case, width of the partial products is $p + 8$ and the width of the aligned product becomes $2p + 8$. In general, truncation produces a negative bias, and in a worst case scenario, this bias can accumulate but in practice, the rounding error produced in one MAC

operation may get cancelled in a subsequent MAC operation, and hence the accumulation of truncation error is never a one to one addition. The accumulation of truncation error can be reduced, by extending the width of the accumulator [1]. The effect of above multiplicand truncation is equivalent to the subtraction of 1/128 ULP (unit in the last place) from the product (worst case), which is tolerable in most of the target applications.

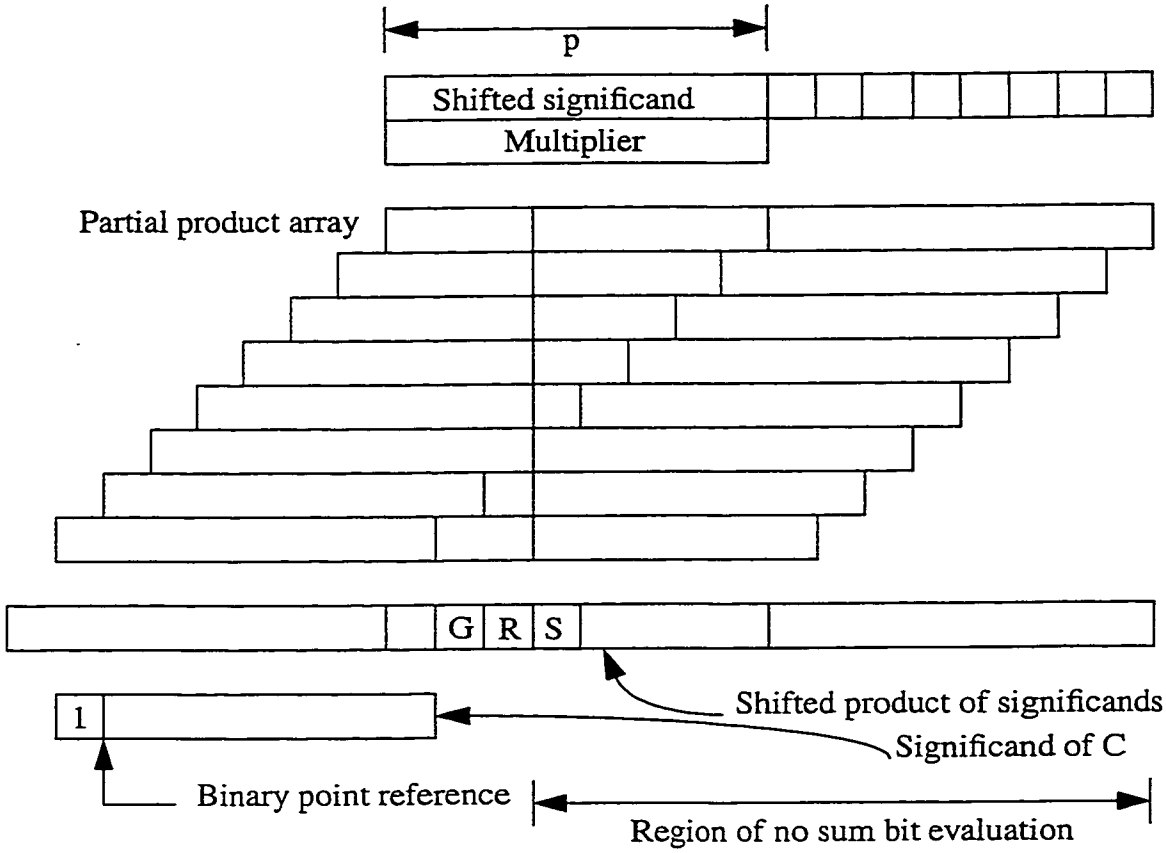


Fig 5.5 - Partial product array by keeping 8 of the shifted out bits of multiplicand

5.2.2 Transition activity scaling

In References [1], [36]-[44], the authors present the architectural power minimization of floating point adders through transition activity scaling as discussed in Section 4.2. They

also used this approach for the design of floating point MAFs. The switching activity function of the IBM compatible MAF is represented as a four state FSM as shown in Fig 5.6. During any given operation cycle, only a limited set of functional sub-units are active, during which time, the logic assertion status of the circuit nodes of the unused functional units are maintained at their previous states. Table 5.1 lists the criterion for the assertion of various states as well as the modules that are activity scaled in each state.

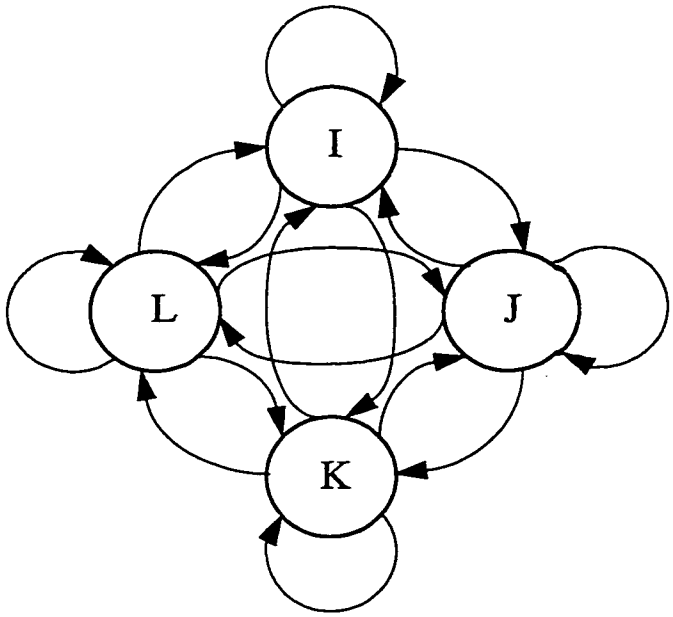


Fig 5.6 - FSM representation of IBM compatible MAF

In state *I*, the transition activity scaled MAF operates in a full bypass mode. During those situations when the difference between the exponents of the current sum and product ($e_{dif} = e_C - e_T$) exceeds $p + 1$, the results of a MAC operation is not different from the current sum. In such a situation, neither the evaluation of the product nor the addition of the product with the current sum is required. Essentially, the whole of the MAC core except the Control unit, Exponent logic, Bypass data path and Result Integration/Flag unit is

activity scaled during this situation. Bypass conditions also include special quantities for input operands and product overflow and underflow.

In state *J*, the exponent difference is still greater than the width of the significand, but the current sum is relatively insignificant compared to the product. That means, it is essential that the product be evaluated, but the summation of this with the current sum is not required. Whenever, the current sum has no effect on the MAC operation, which also include conditions when addition underflow can occur (current sum is zero or a denormalized ($< f_{min}$) number), addition part of MAC which involves significand adder, large barrel shifters and LZA logic can be activity scaled.

Table 5.1: State assertion conditions of transition activity scaled IBM compliant MAF

State	Active data path	State assertion criterion	Activity scaled blocks
I	Bypass	Exponent of multiplicand or multiplier is zero or $e_{max}+1$ or Exponent of accumulated sum is $e_{max}+1$ or Product overflow or product underflow or $e_T < e_C$ and $e_{dif} > p+1$	Entire MAC except Exponent logic, Control unit, Bypass logic and Result Integration/Flag logic
J	Partial Bypass	No Bypass and Exponent of accumulated sum is 0 or $e_T > e_C$ and $e_{dif} > p$	LZA logic and large barrel shifters
K	LZA	No Bypass and no Partial Bypass and subtraction and $e_{dif} \leq 2$ (LZs $\leq p$)	Pre-alignment barrel shifter (large)
L	LZB	No Bypass and no Partial Bypass and addition or $e_{dif} > 2$ (LZs ≤ 1)	LZA logic and normalization barrel shifter (large)

In state *K*, the MAC operations can result in a significand with a variable number of lead-

ing zeros. In significand addition, a variable number of leading zeros can be generated during the subtraction of one floating point number from another only if the exponent difference is not large. Table 5.2 [1] lists various situations during which a variable number of leading zeros can appear at the MSB end of the result significand. It is evident from Table 5.2 that more than one leading zero are only possible during the subtraction of significands when the exponent difference is less than or equal to 2, so the barrel shifter for significand pre-alignment is activity scaled for power reduction.

In state *L*, the MAC performs floating point operations that is guaranteed to produce, at the most, one leading zero. The normalization shifts in this state are upper bounded - a maximum right shift of 2 and a maximum left shift of 1. Since the normalization shift distances are limited, there is no need to excite the large barrel shifter. Activity scaling of this barrel shifter gives power reduction.

Table 5.2: Generation of variable number of leading zeros during the subtraction of significands

Exponent difference	Remarks
$edif = 0$	*Inc* bit of product of significand can be 0 or 1
$edif = 1$ AND $e_C < e_T$	Variable number of LZs iff *Inc = 0*
$edif = 1$ AND $e_C > e_T$	*Inc* can be 0 or 1
$edif = 2$ AND $e_C > e_T$	Variable number of LZs iff *Inc = 1*

The partitioning of MAC operations for transition activity scaling also leads to data path simplifications. Since the significand pre-alignment shifts are less than or equal to 2 during situations when the MAC operation produces a pre-normalized significand with a variable number of leading zeros, significand pre-alignment operations of this data path can be

effected by using a single level of MUXs. However, this data path requires a normalization barrel shifter that can handle a maximum right shift of 1 and a maximum left shift of p bits. With LZB data path, significand pre-alignment shifts can be anywhere between 0 and $p+1$. Normalization shifts for this data path are bounded, a maximum right shift of 2 bit positions and a maximum left shift of 1 bit position. Effectively, the proposed data path partitioning results in data path simplifications. For both LZA and LZB data paths, only one barrel shifter is present, by virtue of which the power consumption, logic depths and circuit delays of the data paths are minimized. Fig 5.7 and 5.8 illustrate the organization of prealignment and normalization barrel shifters of the adder segment of MAF.

5.2.3 Architecture

Data flow architecture of transition activity scaled IBM compatible MAF is presented in Fig 5.9.

Exponent logic calculates the exponent of the product ($e_T = e_A + e_B - e_{bias}$) using a 3:2 compressor and 1's complement adder with which evaluation of product overflow and product underflow conditions is a trivial operation. Exponent difference ($e_{dif} = |e_T - e_C|$) is also calculated by a 1's complement adder whose end around carry reveal the truth about conditions like ($e_T > e_C$, $e_T < e_C$ or $e_T = e_C$). Based on the relative magnitude of e_T and e_C, larger value among e_T or e_C is selected as tentative exponent of the result.

Control logic generates control signals, according to the state assertion conditions of Table 5.1, which are used for the selection of an appropriate data path and routing the inputs to the selected data path.

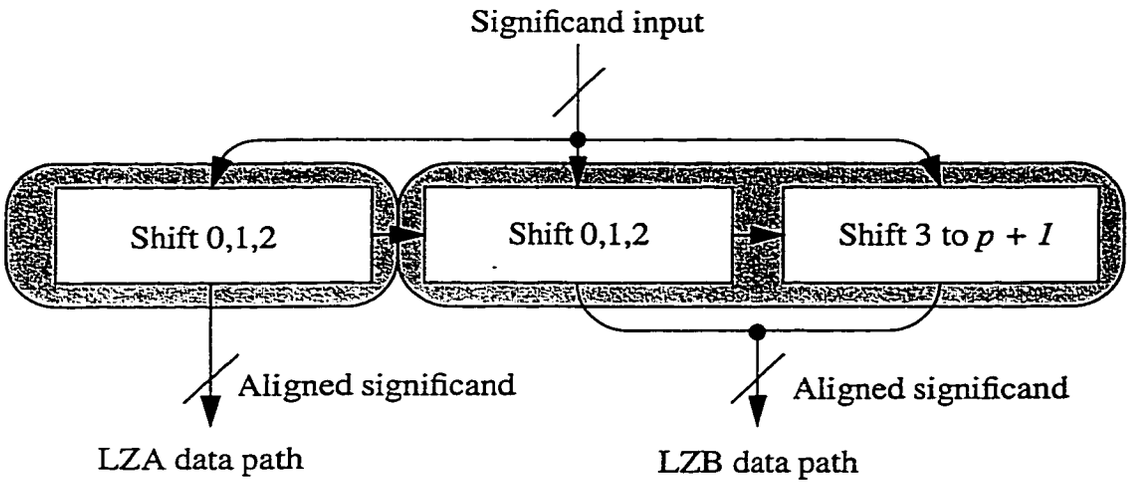


Fig 5.7 - Transition activity scaled pre-alignment barrel shifter for IBM compatible MAF

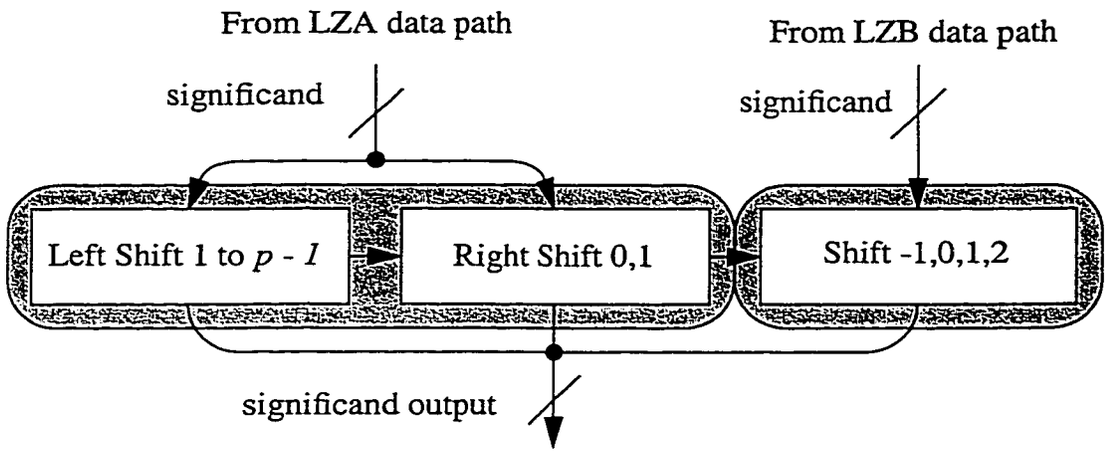


Fig 5.8 - Transition activity scaled normalization barrel shifter for IBM compatible MAF

The bypass data path is essentially a non-computing data path which maps an appropriate floating point number during various bypass conditions. During operations in which either multiplicand or multiplier is zero, the multiplication is guaranteed to produce a zero result. The MAC can be bypassed under such situations as the result is known apriori i.e. the previous sum. For the condition $number1 \times number2 + \alpha$, an *infinity* can be latched to bypass

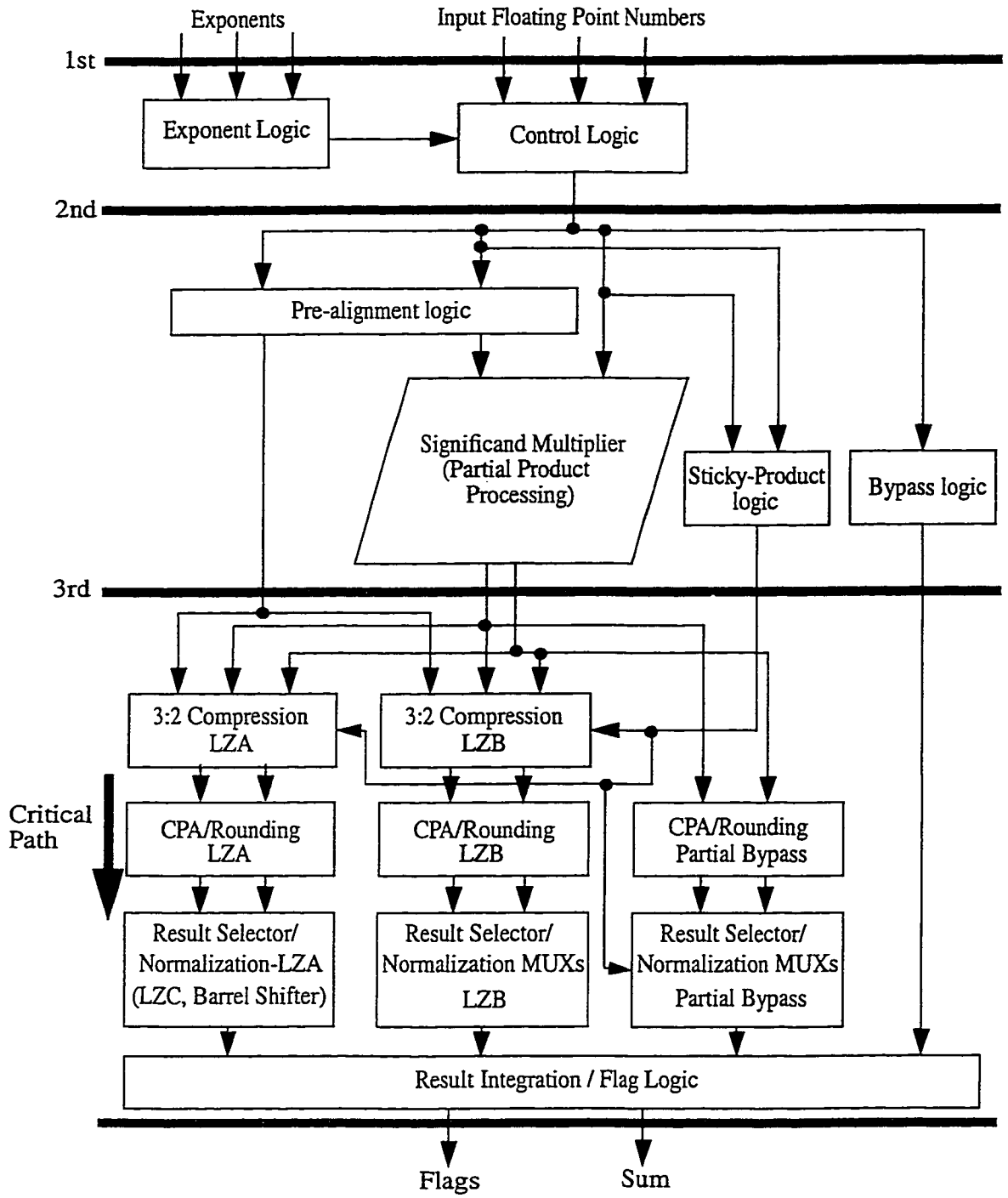


Fig 5.9 - Data flow diagram of IBM compatible MAF with 3 execution stages

data path. In case of multiplication underflow, previous sum is latched. The latched data is

presented to the output during an appropriate cycle of pipeline operation.

The pre-alignment logic conditionally right shifts the significand of C or significand of multiplicand by an amount that is equal to e_{dif}. The pre-alignment unit contains two sub-units. For LZA data path, significand pre-alignment shift is upper bounded by 2. For LZB data path, one sub-unit is handling the significand pre-alignment between 3 to $p+1$ (by right barrel shifter) while the other handles significand alignment that is upper bounded by 2. So prealignment barrel shifter becomes operational only if selected data path is LZB and shift amount is > 2. During situations when shift amount is larger than $p+1$, bypass data path is active and pre-alignment barrel shifter is activity scaled. During this situation, the shift imparted by the barrel shifter is the last shift and hence for IEEE single precision data format 5 of the LSBs of e_{dif} are used for the effecting the pre-alignment shift.

In LZB data path, when the amount of shift can be larger than 2, and significand of C is right shifted, the sticky bit due to this operation can be asserted if the condition $e_{dif} - 2 > TZ_C$ is true. The evaluation of this condition using 1's complement arithmetic has been discussed in Chapter 4.

The significand multiplier is composed of partial product generation and compression logic for the multiplication of significands. Shifting of multiplicand prior to partial product generation results in partial products of larger width, but the number of partial products stays the same as p (24 for IEEE single precision data format). Thus by arranging 4:2 compressors in Wallace tree format only four addition stages are needed in order to compress 24 partial products into sum and carry vectors as shown in Fig 3.3. The horizontal carry-out of the 4:2 compressors is not dependant on carry-in (Fig 3.4, 3.5), so the increase

in the width of partial products due to keeping the shifted out bits of the aligned multiplicand will not affect the speed of partial product compression.

For the high performance evaluation of sticky bit, Sticky-Product logic makes use of the trailing zeros of the product. For floating point MACs, the sticky bit of the shifted product can be set to 1, if the following relation is true.

$$TZ_A + TZ_B < p - 3 + ke_{dif} \text{.....(5.4)}$$

Where TZ_A and TZ_B represent the trailing zero counts of the significands of floating point numbers A and B while k is a 0/1 integer variable, which is zero whenever the exponent of the product is greater than or equal to that of C and vice versa. When $e_T \geq e_C$ i.e. significand of C is shifted, the sticky bit can be found by evaluating the condition $TZ_A + TZ_B < p - 3$, as explained in Section 3.2.4. Whenever $e_T < e_C$, the condition for the evaluation of sticky bit of the product is

$$TZ_A + TZ_B < p - 3 + e_{dif} \text{.....(5.5)}$$

A high performance approach for the evaluation of this condition is proposed here. Equation (5.5) can be reformulated to $p - 3 + e_{dif} - TZ_A - TZ_B > 0$. Trailing zeros are encoded into 5 bit numbers and are complemented, as shown in Fig 5.10(a). The 3rd row in Fig 5.10(a) represents the value of $p - 3 + 1$ i.e. 22, for the 2's complement addition with TZ_A, TZ_B and e_{dif}. These four rows are compressed into a set of sum and carry vectors by a row of 4:2 compressors, as shown in Fig 5.10(b). The end around carry of the 1's complement arithmetic operation can be found by using a circuit shown in Fig 5.11 which is the required

0	e_{dif4}	e_{dif3}	e_{dif2}	e_{dif1}	e_{dif0}
1	$\overline{TZ_{A4}}$	$\overline{TZ_{A3}}$	$\overline{TZ_{A2}}$	$\overline{TZ_{A1}}$	$\overline{TZ_{A0}}$
1	$\overline{TZ_{B4}}$	$\overline{TZ_{B3}}$	$\overline{TZ_{B2}}$	$\overline{TZ_{B1}}$	$\overline{TZ_{B0}}$
0	1	0	1	1	0

(a) - Input data for the evaluation of $TZ_A + TZ_B < p - 3 + e_{dif}$

C5	C4	C3	C2	C1	C0	∅
	S5	S4	S3	S2	S1	

(b) - Input data after bit compression

Fig 5.10 - Data presentation for the evaluation of $TZ_A + TZ_B < p - 3 + e_{dif}$

sticky bit. Partial bypass data path is active during those situations when the previously accumulated sum is zero or very small compared to the product of floating point numbers. Addition operation can be bypassed and product forms the result of floating point MAC operation. The CPA/Rounding block of partial bypass data path performs the carry propagate addition and pre-computation for rounding. The Result Selector/Normalization logic selects an appropriate copy of result based on rounding/normalization conditions and performs the normalization shift.

The 3:2 Compressors in LZA and LZB data path compress the sum and carry vectors of Significand Multiplier with pre-aligned significand of C to 2 vectors and prepare them for carry propagate addition. The operation of rounding is merged with addition. Normaliza-

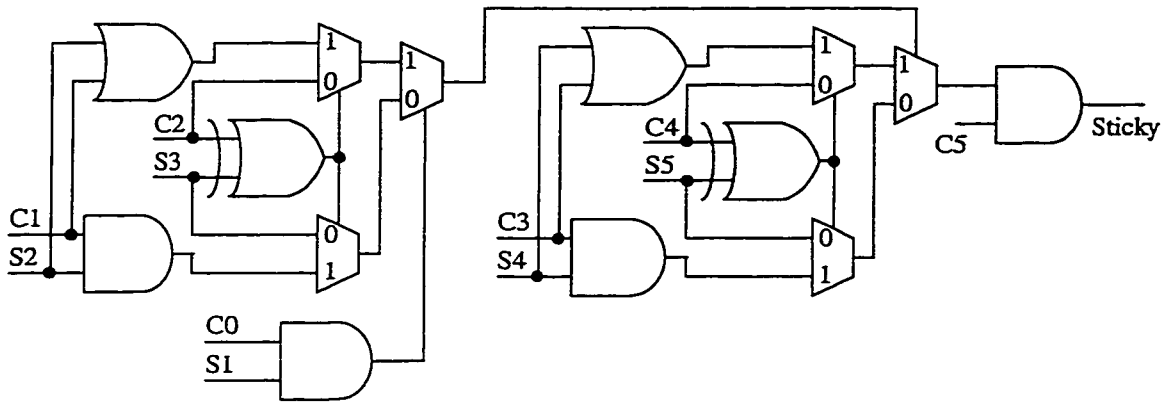


Fig 5.11 - Sticky bit evaluation

tion logic of LZA data path includes, a left barrel shifter that can shift through 0 to p - 1 bit positions, based on the result of leading zero counter as well as 0/1 bit right shifter. Shift control signals of left barrel shifter are transition activity scaled during situations when the result is zero. The normalization shifter in LZB data path is a left/right shifter which can handle a maximum left shift of 1 and a maximum right shift of 2 as shown in Fig 5.8. The Result Integration/Flag logic integrates the results from various data paths and assert flags.

5.3 Hardware Configuration of IEEE Complaint MAF

As discussed earlier, IBM approach of a compound rounding in floating point MAFs makes their results non-compatible with IEEE standards even if the accuracy of the results is increased. R. V. K. Pillai et al. [51] proposed a floating point MAF which delivers IEEE conformal product and sum. Other salient features of this architecture of floating point MAF that makes it an ideal choice for DSP applications as well as general purpose computing, include the functional partitioning of the MAC into 4 data paths for transition

activity scaling and data path simplifications and Zero Overhead Rounding approach for performance enhancement.

5.3.1 Transition activity scaling

Functional partioning of IEEE compatible MAF into four transition activity scaled data paths gives power saving. Finite state machine representation of IEEE compatible MAF is the same as that of IBM compatible MAF as shown in Fig 5.3 but criterion for the assertion of various states is changed as can be seen in Table 5.3.

Table 5.3: State assertion conditions of transition activity scaled IEEE compliant MAF

State	Active data path	State assertion criterion	Activity scaled blocks
I	Bypass	Exponent of multiplicand or multiplier is zero or $e_{max}+1$ or Product overflow or product underflow	Entire MAC except Exponent logic, Control unit, Bypass logic and Result Integration/Flag logic
J	Partial Bypass	No Bypass and Exponent of accumulated sum is zero or $e_{max}+1$ or $e_T > e_C$ and $e_{dif} > p$ or $e_T < e_C$ and $e_{dif} > p+1$	LZA logic and large barrel shifters
K	LZA	No Bypass and no Partial Bypass and subtraction and $e_{dif} \leq 2$ ($LZs \leq p$)	Pre-alignment barrel shifter (large)
L	LZB	No Bypass and no Partial Bypass and addition or $e_{dif} > 2$ ($LZs \leq 1$)	LZA logic and normalization barrel shifter (large)

In state *I*, the MAF works in full bypass mode. For applications that need an IEEE sum and an IEEE product, irrespective of the relative magnitude of these results, the question of transition activity scaling of the multiplier segment is not very relevant. In situations

when previously accumulated sum is large compared to the product and IEEE sum result is the previously accumulated sum, the multiplier segment will still work to deliver IEEE product. When the product is known apriori, i.e. multiplier or multiplicand is a special quantity or product overflow or underflow occurs, neither the evaluation of product nor the addition of product with the current sum is required. In these conditions IEEE sum and product can be mapped to Bypass data path and both multiplier and addition segments of MAF are activity scaled.

When current sum is relatively insignificant compared to the product or the product is insignificant compared to the sum, state *J* is acquired. In this state, IEEE product needs to be evaluated so the product segment is active but adder part can be activity scaled.

State *K* represents the case when during subtraction of significands, variable number of leading zeros can be generated. Transition activity scaling of pre-alignment barrel shifter gives power reduction.

In state *L*, possibility of generation of variable number of leading zeros, during significand addition, is upper bounded by 1 and LZA logic and normalization barrel shifters are activity scaled. In states *K* and *L*, IEEE product is generated by the multiplier segment of the MAF.

5.3.2 Architecture

Fig 5.12 gives the data flow diagram of IEEE compliant MAF.

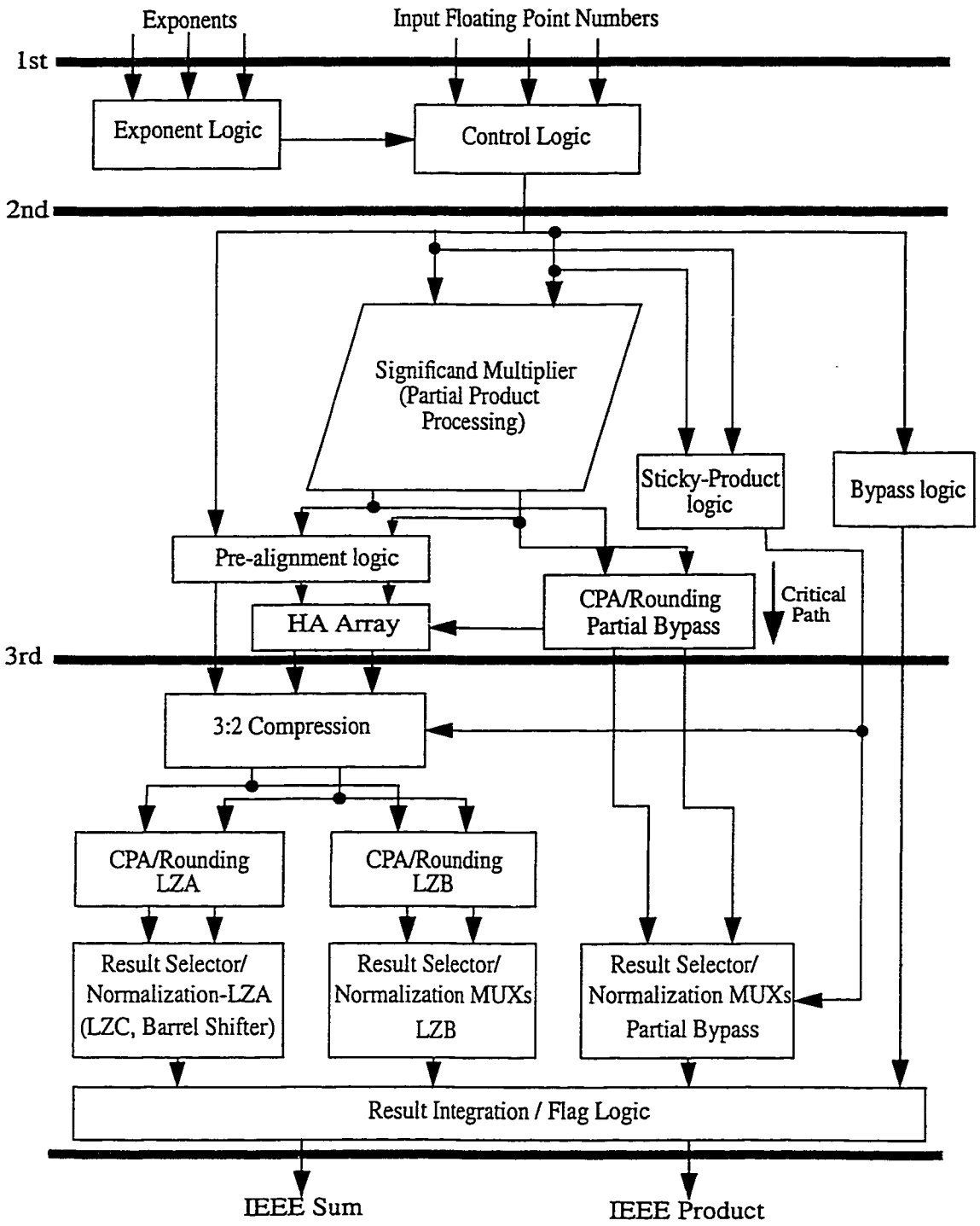


Fig 5.12 - Data flow diagram of IEEE compliant MAF

Exponent logic evaluates the tentative exponent of the product, product overflow and

underflow conditions, amount of pre-alignment shift ($e_{dif} = |e_T - e_T|$), relative magnitude of e_T and e_C and the tentative exponent of the sum result as discussed in Section 5.3.2. Based on the results of exponent logic and the values of input operands, Control logic controls the activation and de-activation of different data paths. Bypass logic maps the floating point numbers as IEEE sum and IEEE product without performing any computation.

Significand Multiplier performs the multiplication of significands of multiplier and multiplicand (non-aligned). CPA/Rounding logic of the multiplier segment performs the carry propagate addition and pre-computation of rounding. Once the final result taking into account the rounding/normalization decisions is arrived at, the rounding information of the product is sent to the floating point addition segment of the MAF, where it is used to produce IEEE compliant sum. In this scheme, in contrast to IBM MAF, the significands of both previously accumulated sum and product can be aligned in accordance with their relative magnitudes. During situations when the exponent of C is larger than that of AB, the sum and carry vectors from the partial product compression logic are simultaneously right shifted, as shown in Fig 5.13, by using a double barrel shifter. When the exponent of AB is greater than that of C, the significand of C is right shifted by a single barrel shifter. The pre-alignment barrel shifter that shifts C is merged with the double barrel shifter by incorporating suitable significand selection scheme. Since in LZA data path the significand pre-alignment shifts are less than or equal to 2 during situations when the MAC operation produce a pre-normalized significand with a variable number of leading zeros, significand pre-alignment operations of this data path are realized by a single level of MUXs without activating pre-alignment barrel shifters. In LZB data path significand pre-alignment shift can be anywhere between 0 to $p + 1$ but barrel shifters are activated only if the shift

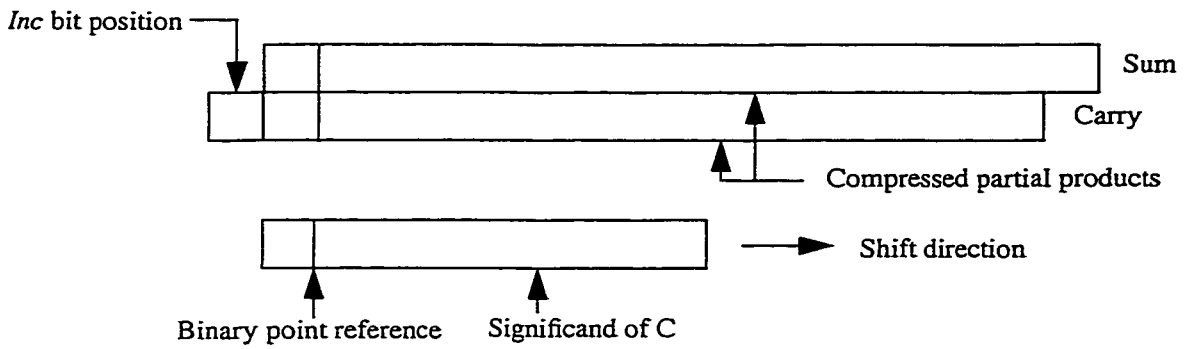


Fig 5.13 - Alignment of compressed partial products

amount exceeds 2 bit position. The use of double barrel shifters for simultaneous shifting of sum and carry vectors of Significand Multiplier, in LZB data path, when $e_C > e_T$ and the shift amount is more than 2, results in increase in power consumption as with floating point adders, the power consumption of the shift control lines of barrel shifters account for the most dominating component of the overall power consumption of these units. But in IEEE compliant MAF that generate both IEEE sum and IEEE product, multiplication of significands of multiplier and multiplicand needs to be performed prior to the pre-alignment shift of the multiplicand as in this architecture of MAF, unlike IBM compatible MAF discussed in Section 5.2 of this chapter, product segment is active during all the states except when bypass conditions are true (state *I*), to produce IEEE product and to generate conditional rounding signals of product. If pre-alignment shift is performed before the significand multiplication, the evaluation of product and the rounding signals of product can't be achieved during those situations when multiplicand is shifted. In IBM compatible MAF, CPA/ Rounding and Normalization units of product segment are only active during partial Bypass conditions when no pre-alignment shift is performed.

Multiplication of significands before pre-alignment shift results in partial products with smaller sizes compared to that of IBM compatible MAF. Compact partial product generation and compression logic makes the power/area/delay considerations of the multiplier segment of IEEE compliant MAF quite attractive compared to IBM compatible MAF. In order to reduce the power/area/delay overheads of IBM compatible MAF, some of the LSBs of the shifted significand (and hence LSBs of the partial products) were truncated, which makes it less desirable for applications that demand higher precision for MAC computations. But this is not the case in IEEE compatible MAF. No truncation of partial product is needed and hence no rounding errors associated with it are introduced.

Fusion of multiply - accumulate operation is quite challenging when IEEE compatibility is also taken into account, as the rounded product (which require CPA, normalization and rounding of the product) needs to be added with previously accumulated sum. In this architecture of MAF, following efforts are being made to add the rounded product with previous sum but still keeping it as a fused unit.

(1) Rounding control signals and sum and carry vectors of Significand Multiplier, instead of rounded product are sent to the adder segment. These control signals are generated as soon as the results of carry propagate addition of multiplier section is complete without the normalization shift of the significand of product. These rounding control signals carry information regarding conditional increment at two bit positions based on the fact that the *Inc* bit (MSB of the result of the significand multiplication, 1 if *product of significands* ≥ 2) is 0 or 1. These signals are introduced in the adder segment by using rows of half adders. A row of half adders, partly adds the sum and carry vectors of the Significand Multiplier

leaving an empty hole at the LSB position as shown in Fig 5.14 [29]. Conditional increment bit from the multiplier segment can be placed into this hole.

(2) Delay of the pre-alignment shift is hidden behind the delay of the carry propagate addition of the multiplier segment for the fusion of multiplication with addition.

Sum and carry vectors from rows of half adders are compressed with the aligned significand of C using 3:2 compressors. CPA/Rounding units of LZA and LZB data path perform the pre-computation of different copies of results taking into account the various rounding/ normalization requirements. Conditional sum / carry select adders are ideal for such applications. Result selectors selects an appropriate copy of sum bits based on the normalization/rounding conditions.

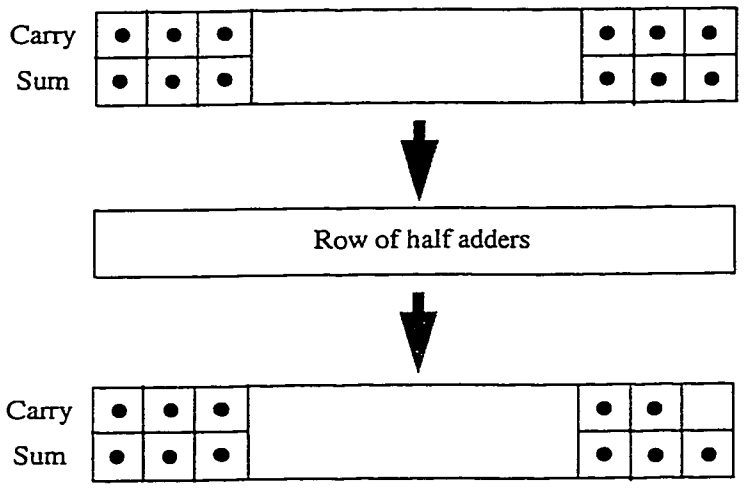


Fig 5.14 - Introduction of conditional increment (for rounding) bit in sum and carry vectors

Normalization shifts of LZA data path are realized by single row of MUXs in case of 0/1

bit right shift and by a leading zero counter and barrel shifter in case of left shift of 1 to p - 1 bits. During situations when the result is zero, the shift control signals of barrel shifter are activity scaled. Normalization shifts of LZB data path are confined to a maximum right shift of 2 bit positions and a maximum left shift of one bit position and they are performed by MUXs. Absence of barrel shifter results in significant amount of power saving.

The design partitioning of floating point MAF also gives data path simplification. From LZA data path, pre-alignment barrel shifter is removed and in LZB data path, normalization barrel shifter is not required. Pre-alignment shift in LZB data path can activate a double barrel shifter but they performed the required shift in parallel. So the delay of the adder segment of MAF is reduced by one barrel shifter delay.

5.4 Hardware Configuration of IBM MAF

R.K. Montaye et. al. [31], [32], proposed IBM RISC System/6000 floating point unit in which performance and accuracy is improved by including a unified floating point multiply - add fused (MAF) unit, which executes the double precision MAC instructions as an indivisible operation, with no intermediate rounding. Fusion of multiplication with accumulation allows one-cycle throughput and two-cycle latency, producing a MAC result with one rounding error. It also reduces the number of adders/normalizers by combining the addition required for fast multiplication with accumulation. Internal busing requirements and number of ports are also minimized.

In this section the hardware configuration of IBM MAF is presented. For the comparison with low power IBM compatible MAF (presented in Section 5.2) and IEEE compatible

MAF (Section 5.3), IBM MAF is configured to work with IEEE single precision data format.

Data flow diagram of IBM MAF is presented in Fig 5.15. The first cycle of pipeline is shared by multiplication of significands, the exponent calculation, and the pre-alignment shift.

Exponent logic calculates the tentative exponent of product ($e_T = e_A + e_B - e_{bias}$) and the amount of pre-alignment shift i.e. exponent difference between the tentative exponent of product and the exponent of C ($e_{dif} = |e_T - e_C|$). As during pre-alignment shift, Significand of C is shifted left or right depending on the relative magnitude of e_C and e_T ($e_T \geq e_C$ or $e_T < e_C$), it also needs to be evaluated. Significand of C is aligned irrespective of the value of its exponent, so the tentative exponent of product also represents the tentative exponent of final sum.

Pre-alignment logic is composed of a left-right barrel shifter. Significand of C is shifted by an amount which is equal to the e_{dif}. The direction of shift is decided by the relative magnitude of e_T and e_C (left if $e_T < e_C$, right otherwise). Right shift is realized by 6 LSBs of exponent difference while during left shift only 5 LSBs of exponent difference are used as shift control data word. In case of right shift, a shift larger than $2p$ bits results in a significand which can't effect the significand of product during subsequent carry propagate addition and normalization. If the left shift is larger than $p + 1$ bits, it results in a significand in which leading one is shifted out. This case is realized by generating a product underflow signal as the previously accumulated sum is so large that the product can not

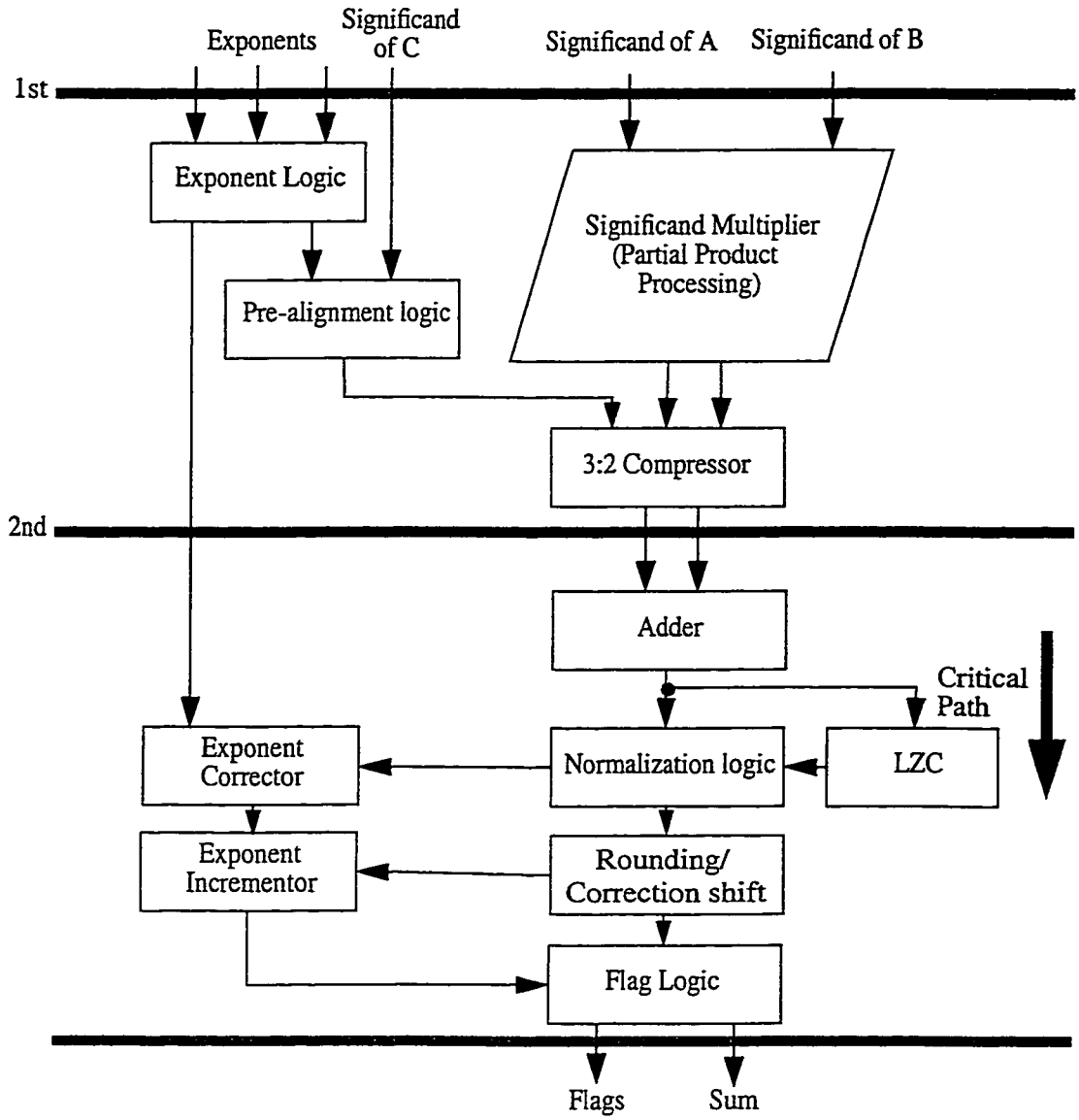


Fig 5.15 - Data flow diagram of IBM MAF

affect the final result. In this situation Result/Flag logic maps previous sum as the result of MAC. A fast left-right barrel shifter which is composed of cascaded array of 2×1 MUXs is used which hides the combined delay of Exponent logic and Pre-alignment unit behind the delay of partial product generation and compression array of Significand Multiplier.

In reference [31], [32], multiplier array utilizes two techniques i.e. modified booth encoding (MBA) and extended Wallace tree (composed of (7,3) counters) for the generation and compression of 56 partial products for double precision MAF. Many authors [17], [49], [1], favour the use of 4:2 compressors for the partial product compression compared to the MBA for its energy delay improvements and less wiring complexity and gate count. Also partial product array organization using (7,3) counters presented in reference [32] introduces more gate delays compared to 4:2 compressors based partial product compression scheme (presented in Section 3.2.3) for the compression of 24 partial products (using IEEE single precision data format), as far as the gate level implementation is concerned (Reference [32] also presents an optimized transistor level design of (7,3) counter which can't be utilized in this work as using logic synthesis tools, the transistor level design decisions can not be acquired). So the partial product compression array, composed of 4:2 compressors arranged in Wallace tree format (Fig 3.3), which require four addition stages, is also used here. In reference [31], [32], it is suggested to add the aligned significand of *C*, during the partial product compression, due to the availability of extra pin for an extra operand, without sacrificing the cycle time. Some extra pins are also available in the Wallace tree of 4:2 compressors, as compression of 32 partial products can be realized using 4-stage array. But another issue is involved here. Due to the left right shift of significand of *C*, its effective width is $3p$ bits. In order to add it to $2p - 1$ bit wide partial products, zeros should be appended on the most significand end of all partial products. It results in an increase in the width of partial product array which increases the power consumption and area of Significand Multiplier. With a small sacrifice in terms of speed, the addition of pre-aligned Significand of *C* is performed using a row of 3:2 compressors of bit width $3p$.

Addition of $3p$-bit wide operands is performed using conditional sum/carry select adder [7] [23] [56], in a 1's complement form, which involves an end around carry. Conditional sum bits with an anticipated carry-in of zero (*sum0*) and carry-in of one (*sum1*) are evaluated. If the result is positive (can be found by evaluating the condition $Cout(0) = 1$, where $Cout(0)$ represent carry-out of addition with anticipated carry-in of zero), *sum1* is selected as the result of addition. If the result is negative (evaluated by $Cout(1) = 0$, where $Cout(1)$ is carry-out, when the injected carry-in during addition is 1), bits of *sum0* vector are complemented. So that the sign magnitude result is correctly output in one more gate delay.

After the addition, normalization shift is required to bring the leading one to the left of binary point reference. In this scheme, due to the alignment of significand of C all the times irrespective of the relative magnitudes of exponents of C and AB, leading 1 after significand addition can occur within a range of $2p + 2$ bits. Reference [31], [32], [33], also present a leading zero anticipatory logic for its use in IBM MAF. But the power/delay/area considerations of this LZA logic have been proved sub-optimal by many researchers [34], [35], [1]. As hardware configuration of IBM MAF is envisaged for its comparison with power/delay optimal MAFs which are using a fast tree type leading counter, whose simplicity makes it an ideal choice for power sensitive designs. The same type of 64 bit tree type leading zero counter is also used here.

The leading one can occur to the left or to the right of binary point reference (BPR). If the condition $LZ < p + 1$ is true, leading 1 is to the left of BPR, where $p + 1$ represents the number of bits of aligned significand of C towards the left of BPR. A right normalization shift is required in this case. If leading 1 is to the right of BPR, evaluated by $LZ > p + 1$, a

left shift is needed for normalization. The shift amount is calculated as $|p+1-LZ|$ and a left right barrel shifter, with a 5 bit wide shift control signal, can perform the necessary normalization shift. In parallel with the normalization shift, $p + 1 - LZ$ amount is added to the tentative exponent of result in Exponent Corrector logic. Rounding/Correction shift logic performs the rounding operation using round to nearest-even approach, which sometimes result in a carry-out from MSB, in which case, a correction shift (right) of one bit position, and exponent increment, is performed. Flags logic asserts five flags to represent special results.

5.5 Evaluation Process

Comparison of 3 different architectures of floating point multiply - accumulate fused (MAF) unit is done by writing VHDL code and synthesizing towards 0.35 micron CMOS technology and FPGA technology (Xilinx XC4085XL). Selection of two completely different technologies is made to show that optimization strategies at the architectural level give results that are almost unaffected by the technology change. All three architectures are configured to work with IEEE single precision data format. As the power consumption of transition activity scaled 4 data path IBM compliant MAF and 4 data path IEEE compatible MAF, is highly dependant on the utilization of a certain data path, simulations are done using real data (test vectors generated by the conversion of real audio signals [55]) as well as synthetic data (test vectors generated manually using different probabilities of data path utilization to get the average and worst case power consumption). While using the real data for simulations, data from 2 files, each containing 128K data samples, is applied to the multiplier and multiplicand inputs of MAFs and the result of MAF is fed back to input C, so that MAFs are performing the multiply - accumulate operations and C input is

the previously accumulated sum.

5.6 Results

Comparison of the 3 different architectures of MAF unit using 0.35 micron CMOS technology is presented in Table 5.4. Maximum delay, power consumption and area information is obtained from reports generated after synthesis. The calculated values of Power-Delay (PD), Area-Power (AP), Area-Delay (AT) and Area-Delay^2 (AT^2) are also presented in Table 5.4.

Table 5.4: Comparison between MAFs using 0.35 micron CMOS technology

Parameters	IBM compatible MAF	IEEE compliant MAF	IBM MAF
Maximum Delay, D (ns)	44.55	49.18	49.3
Average Power, P_a (mW) at 20 MHz	58.46	78.25	122.96
Worst case Power, P_w (mW) at 20 MHz	68.67	90.92	122.96
Power using real data, P_{real} (mW) at 20 MHz	37.98	55.01	76.84
Area, A (10^4 cell-area)	120.57	112.47	89.77
Power-Delay Product, PD (ns.mW)	2.92×10^3	3.91×10^3	6.15×10^3
Area-Power Product, AP (10^4cell-area.mW)	7.05×10^3	8.8×10^3	11.04×10^3
Area-Delay Product, AT (10^4cell-area.ns)	5.37×10^3	5.53×10^3	4.43×10^3
Area-Delay^2 Product, AT^2 (10^4cell-area.ns^2)	2.39×10^5	2.72×10^5	2.18×10^5

It can be seen from the results that the reduction in average power delay product (figures obtained by considering the average behavior of IBM compatible and IEEE compatible MAFs, (i.e. there is an equal chance of data path utilizations through all 4 data paths)

offered by transition activity scaled IBM compatible MAF [1], [51], over IBM MAF [31], [32], is 52.46% and it offers 25.21% reduction in average power delay product compared to IEEE compliant MAF [51]. The reduction in average power delay product given by IEEE compliant MAF compared to IBM MAF is 36.36%. Considering the worst case for transition activity scaled architectures of MAF (data is always passing through the LZA data path, in which case, both IBM compatible and IEEE compliant MAFs, consume the maximum amount of power), IBM compatible MAF gives 44.15% and 24.47% reduction in power-delay product compared to IBM MAF and IEEE compatible MAF respectively and the worst case reduction in power delay product offered by IEEE compliant MAF compared to IBM MAF is 26.06%. While performing the multiply - accumulate operation on real data, IBM compatible MAF offers 50.57% and 30.46% reduction in power delay product compared to IBM MAF and IEEE compatible MAF respectively. With real data, IEEE compatible MAF gives 28.4% reduction in power-delay product compared to IBM MAF.

Also IBM compatible MAF is 9.63% and 9.41% faster than IBM MAF and IEEE compatible MAF respectively. Speed of IEEE compatible MAF is slightly better than IBM MAF. These improvements came at a cost. Area of IBM compatible MAF is 25.55% more than that of IBM MAF and it is 6.72% more than the area of IEEE compatible MAF. IEEE compatible MAF occupies 20.18% more area than IBM MAF. The table also gives Power-Delay (PD), Area-Power (AP), Area-Delay (AT) and Area-Delay^2 (AT^2) products for more detailed evaluation of these architectures.

Table 5.5 presents the results of comparison of 3 MAFs using FPGA technology. Reduc-

tion in average power delay product given by IBM compatible MAF is 50.14% and 31.24% compared to IBM MAF and IEEE compliant MAF respectively. IEEE compatible MAF is offering 27.48% reduction in average power delay product compared to IBM MAF. Considering the worst case for IBM compatible and IEEE compliant MAFs, the reduction in power delay product given by IBM compatible MAF is 37.47% compared to IBM MAF and 26% compared to IEEE compliant MAF. Worst case reduction in power delay product given by IEEE compliant MAF is 15.5% compared to IBM MAF.

Table 5.5: Comparison between MAFs using FPGA technology

Parameters	IBM compatible MAF	IEEE compliant MAF	IBM MAF
Maximum Delay, D (ns)	287.32	301.25	361.54
Average Power, P_a (W) at 2.5 MHz	1.2729	1.8513	2.5529
Worst case Power, P_w (W) at 2.5 MHz	1.5964	2.1571	2.5529
Power using real data, P_{real} (W) at 2.5 MHz	0.7646	1.4014	1.6602
Area, A, Total CLBs (#)	3293	3222	2392
Power-Delay Product, PD (ns.10mW)	5.09×10^4	7.41×10^4	10.21×10^4
Area-Power Product, AP (10#.10mW)	4.19×10^4	5.96×10^4	6.1×10^4
Area-Delay Product, AT (10#.ns)	9.46×10^4	9.71×10^4	8.65×10^4
Area-Delay^2 Product, AT^2 (10#.ns^2)	2.72×10^7	2.92×10^7	3.13×10^7

Using real data, IBM compatible MAF gives 53.95% and 45.44% reduction in power delay product compared to IBM MAF and IEEE compliant MAF respectively, while IEEE compliant MAF gives 15.59% reduction in power delay product compared to IBM MAF.

Also using FPGA's, the delay of IBM compatible MAF is 20.53% less than that of IBM MAF while it is 4.62% less compared to the delay of IEEE compliant MAF. IEEE compliant MAF is 16.68% faster than IBM MAF. IBM compatible MAF requires 27.36% more CLBs compared to IBM MAF and 2.16% more CLBs compared to IEEE compliant MAF. IEEE compliant MAF requires 25.76% more CLBs than IBM MAF.

The amount of power consumption of all three MAFs when processing real data is less compared to the amount of power consumption when working on artificial data for both ASIC and FPGA netlists. The reason behind this reduction is that the artificial data generated manually is changing with every work cycle while real data stays the same for quite some time, specially in the start and end of audio signals. This results in reduction in switching activity and power consumption in all three MAFs.

5.7 Discussion

As anticipated, the results of synthesis to both FPGA and 0.35 micron CMOS technologies present the same kind of trends. The IBM compatible MAF consumes the minimum amount of power, then comes the IEEE compliant MAF and power consumption of IBM MAF is the worst. IBM compatible MAF is faster than IEEE compliant and IBM MAF but it requires more area compared to the other two MAFs. Area overhead of IBM MAF is minimum compared to both IBM compatible and IEEE compliant MAFs. As the differences in the architectures were evaluated at the logical/architectural level without going to circuit level details, the results are also obtained which are almost technology independent.

Transition activity scaled, IBM compatible MAF and IEEE compliant MAF outperforms the IBM MAF as far as the relative power consumption of the architectures is concerned. Due to the modifications in significand pre-alignment operation in IBM compatible and IEEE compliant MAFs, the width of their significand data path is around half of the IBM scheme. With transition activity scaling, during situation when IBM compatible and IEEE compliant MAFs endure bypass and partial bypass conditions, a substantial amount of power saving is achieved. In general, the power consumption of pre-alignment barrel shifters accounts for the most dominating component of power consumption in FADDs and MAFs. Because of the large fanout of the barrel shifter control lines, the capacitive loading of these lines are large. Compared to normalization, the activity of pre-alignment barrel shifters is relatively large. With IBM MAF, the pre-alignment of the significand of C is handled by a non-activity scaled, bidirectional barrel shifter in which fanout of shift control signals is around twice than that of IBM compatible and IEEE compliant MAFs. In IBM MAF, even during situations when the exponent differences are greater than the width of significand, the significands may be shifted through certain finite number of bit positions as the barrel shifter is not activity scaled. During floating point adder part of MAFs, normalization shifts through a large number of bit positions are required only during situations when the process of significand addition results in a large number of leading zeros. During all other situations, normalization shifts are limited. However, with the IBM MAF scheme, normalization shifts can be large even during other situations. With this scheme, with p-bit significands, the leading 1 after significand addition can occur within a range of $2p + 2$ bits. The reason for such behavior is simple. In this scheme, the significand of C is aligned all the time irrespective of the relative magnitudes of the exponents of

C and *AB*. With such a scheme, the leading zero estimation logic has to work with the (2*p* + 2)-bit result and hence the normalization shifts are usually large. With transition activity scaled MAFs, leading zero estimation circuit becomes operational during a limited set of input data. In IBM MAF, a CPA of larger width is required for the handling of significand additions. In general, the wider the width of the adder, the higher the switching activity.

In IBM compatible MAF, the shifting of the multiplicand prior to partial product generation results in certain undesirable effects as far as the switching activity of the multiplier segment of MAF is concerned. The width of the partial product array is increased which results in extra switching activity during the partial product generation and compression. But in this MAF, during the processing of input data, at the most one barrel shifter becomes active. In IEEE compliant MAF, pre-alignment shift is performed after the significand multiplication so the switching activity of the multiplier segment is less but pre-alignment shift in this scheme may require the use of a double barrel shifter. This results in a substantial increase in power consumption and the reduction in power consumption of the multiplier section of the IEEE compliant MAF is offset by the increase in power consumption during pre-alignment shift. Also in IEEE compliant MAF due to the generation of IEEE product, irrespective of its value compared to IEEE sum, the conditions in which MAF is in full bypass mode, are limited, compared to IBM compatible MAF. Unlike IBM compatible MAF, CPA/Rounding logic and Normalization unit of multiplier segment of IEEE compliant MAF stay operational during situations when LZA or LZB data paths are active to generate IEEE product and conditional rounding signals of product required to generate IEEE compatible sum result.

In majority of computing applications, MAF bypasses are not very relevant but this is not the case for some DSP applications. For example, with the order 64 and 16 FIR filters, the probability that the exponent of the product is negligible compared to that of the sum is between 10 and 20%. With symmetrical band pass and band stop filters (the normalized pass/stop bands centered at 0.5), the probability that the MAC operation produces a result that is not different from the current sum is around 47% [1]. In these applications the use of transition activity scaled MAFs can give significant amount of power savings.

With IBM compatible and IEEE compliant MAF, due to reduction in the width of significand data path, speed performance is also improved. Removal of one barrel shifter from LZA and LZB data paths of these architectures also give speed advantages. Also the fanouts of barrel shifter control lines are half than that of the IBM scheme so the set up of these signals are faster. This is true with normalization shifters as well. Also, the process of significand addition is faster in these schemes owing to the reduction in significand data path width, in contrast to that of IBM MAF. However, the evaluation of many of the activity scaling conditions impose extra delay. Evaluation of IEEE compatible results in IEEE compliant MAF adds the extra delay of CP addition of the multiplier segment. But with above measures to increase the performance of IEEE compliant MAF, its speed is slightly better than the IBM approach.

Introduction of new data paths for transition activity scaling in IBM compatible and IEEE compatible MAFs results in an increase in the area, but the area measures of significand adders, barrel shifters and leading zero counters of the transition activity scaled MAFs are less than that of the IBM scheme. Also the handling of certain arithmetic operations by

using 1's complement arithmetic units in the multiple data path MAFs results in area/ power reductions. Since the largest share of MAF circuitry is in the significand multipliers and due to the pre-alignment of multiplicand prior to significand multiplication in IBM compatible MAF, the width of partial product compression increases, which makes the area of the IBM compatible MAF the largest among the three architectures of MAF.

Another issue is accuracy and IEEE compatibility. The requirements for IEEE compatible floating point results is mandatory in many computing applications. The availability of IEEE product and sum makes the IEEE compliant MAF widely accepted. Energy/delay implications of IBM compatible MAF makes it an ideal choice for applications that doesn't require IEEE compliant floating point results. Due to the truncation of some of the LSBs of the shifted multiplicand, IBM compatible MAF introduces a small rounding error, which is acceptable in most of the applications. If the target application is very sensitive to rounding errors, then all of the shifted out bits of the multiplicand need to be retained. In this case transition activity scaled IBM compatible MAF still gives power/ delay optimized results composed to IBM MAF but the amount of savings is reduced.

5.8 Conclusion

Three different architectures of floating point multiply - accumulate fused unit are evaluated by synthesizing them to ASIC and FPGA technologies. The results obtained during this comparative study gives a clear picture of the feasibility of a certain architectural approach in a particular application.

Chapter 6

Conclusions and Future Work

6.1 Contributions

Contributions towards the evaluation of high level design decisions for power/performance optimal realization of floating point adders, multipliers and multiply - accumulators were presented in the preceding chapters.

We developed a low power transition activity scaled double data path architecture for floating point multiplications. We validated the proposed architecture by comparing it with a conventional floating point multiplier by modeling the architectures with VHDL and synthesizing using 0.35 micron CMOS and FPGA technologies.

We compared the two recently proposed architectures of floating point adders. We evaluated the performance of the pipelined and low latency structures of both floating point adders in DSP applications. The findings are reported in [44].

Using VHDL, we modeled and evaluated three different architectures of floating point multiply - accumulate fused unit. The generated results can help in the selection of a suitable architecture for different applications. IEEE compatibility and the accuracy of results in different architectures is explored.

Results obtained in this work by exploration/implementation of floating point units at the logical/architectural/algorithmic level, are almost technology independent.

Netlists of floating point units, generated in this work are used in the implementation of floating point digital filters.

6.2 Future work

The next step is the development and evaluation of these floating point units at the circuit level, where using transistor/full custom level strategies, further optimization, specific to the particular technology, is possible. For the implementation of the transition activity scaled data paths, a stage of registers was always inserted, which is the cause of increase in latency and power consumption in these units. This can be addressed by implementation at the circuit level. Another solution to this problem is to calculate control signals, required for the activity scaling, using instruction driven processors. Functional verification is performed using simulations at various level of design flow. These simulations require a very large number of test vectors for reasonable coverage of functionality in these designs. This can be addressed by utilizing new verification techniques like Formal Verification.

References

[1] R. V. K. Pillai, "On low power floating point data path architectures", Ph. D thesis, Concordia University, Oct. 1999.

[2] Lisa A Coleman, "DSPs break through technology road blocks", Military and Aero space Electronics, pp. 24-28, April 1994.

[3] Anantha P. Chandrakasan, Samuel Sheng and Robert W. Brodersen, "Low-power CMOS digital design", IEEE Journal of Solid State Circuits, Vol. 27, pp. 473-483, April 1992.

[4] Power Product Reference Manual, documentations of Synopsys Inc.

[5] Kurt Keutzer and Peter Vanbekbergen, "The impact of CAD on the design of low power digital circuits", in Proceedings of the 1994 IEEE Symposium on Low Power Electronics, pp. 42-45, 1994.

[6] "IEEE Standard for Binary Floating - Point Arithmetic", ANSI/IEEE Std 754-1985, New York, The Institute of Electrical and Electronics Engineers Inc., August 12, 1985.

[7] J. Sklansky, "Conditional Sum Addition Logic", IRE Transactions on Electronic Computers, Vol. EC-9, No. 2, pp. 226-230, June 1960.

[8] M. Lehman and N. Burla, "Skip Techniques for High-Speech Carry-Propagation in Binary Arithmetic Units", IRE Transactions on Electronic Computers, pp. 691,

December 1961.

[9] Luigi Dadda and Vincenzo Piuri, "Pipelined Adders", IEEE Transactions on Compuers, Vol. 45, No. 3, pp. 348-356, March 1996.

[10] Makoto Suzuki, Norio Ohkubo, Toshinibu Shinbo, Toshiaki Yamanaka, Akihiro Shimizu, Katsuro Sasaki, and Yoshinobu Nakagome, " A 1.5-nS 32-b CMOS ALU in Double Pass-Transistor Logic", IEEE Journal of Solid State Circuits, Vol. 28, pp. 1145-1151, November 1993.

[11] Andrew D. Booth, "A signed binary multiplication technique", Quarterly J. Mechan. Appl. Math., 4: 236-240, 1951.

[12] O. L. MacSorley, "High speed arithmetic in binary compters", Proc. IRE, Vol. 49, pp. 67-91, 1961.

[13] C. S. Wallace, "A suggestion for a Fast Multiplier", IEEE Trans. Electronic Computers, Vol. 13, pp. 14-17, Feb 1964.

[14] L. Dadda, "Some schemes for Parallel Multipliers", Alta Freq., 34: 349-356, 1965.

[15] L. Dadda, "On parallel Digital Multipliers", Alta Freq., 45: 574-580, 1976.

[16] Weinberger A., "4:2 Carry save adder module", IBM Tech. Disclosure Bulletin, 23, 1981.

[17] D. Villeger and V. G. Oklobdzija, "Evaluation of Booth encoding techniques for par-

allel multiplier implementation", Electronics Letters, Vol. 29, No. 23, pp. 2016-2017, 11th November 1993.

[18] Kazuo Yano, Toshiaki, Yamanaka, Takashi Nishida, Masayoshi Sito, Katsuhiro Shimohigashi, and Akihiro Shimizu, "A 3.8-ns CMOS 16X16-b Multiplier Using Complementary Pass-Transistor Logic", IEEE Journal of Solid State Circuits, Vol. 25, pp. 388-395, April 1990.

[19] Vojin G. Oklobdzija, David Villeger and Simon S. Liu, "A Method for Speech Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach", IEEE Transactions on Computers, Vol. 45, No. 3, pp. 294-306, March 1996.

[20] S. Shah, A. J. Al-Khalili, D. Al-Khalili, "Comparison of 32-bit Multipliers for Various Performance Measures", to be presented at ICM 2000, Nov. 2000.

[21] J. Mori, M. Nagamatsu, M. Hirano, S. Tanaka, M. Noda, Y. Toyoshima, K. Hashimoto, H. Hayashida, and K. Maeguchi, "A 10-ns 54X54-b Parallel Structured Full Array Multiplier with 0.5 micron CMOS Technology", IEEE Journal of Solid State Circuits, vol. 26, No. 4, pp. 600-606, April 1991.

[22] Paul J. Song and Giovanni De Micheli, "Circuit and Architecture Trade-offs for High-Speed Multiplication", IEEE Journal of Solid State Circuits, vol. 26, No. 9, pp. 1184 - 1198, Sep. 1991.

[23] N. Ohkubo, M. Suzuki, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, Y. Nakag-

ome, "A 4.4 ns CMOS 54X54-b Multiplier Using Pass-Transistor Multiplexer", IEEE Journal of Solid State Circuits, vol. 30, No. 3, pp. 251 - 257, March 1995.

[24] Hesham A. Al-Twaijry and Michael J. Flynn, "Technology Scaling Effects on Multipliers", IEEE Transactions on Computers, vol. 47, No. 11, pp. 1201 - 1215, Nov. 1998.

[25] G. M. Tharakan and S. M. Kang, "A New Design of a Fast Barrel Switch Network", IEEE Journal of Solid - State Circuits, Vol. 27, No. 2, pp. 217-221, February 1992.

[26] H. Fujii, C. Hori, T. Takada, N. Hatanaka, T. Demura, and G. Ootomo, "A Floating-Point Cell Library and a 100-MFLOPS Image Signal Processor", IEEE Journal of Solid State Circuits, vol. 27, No. 7, pp. 1080 - 1088, July 1992.

[27] N. Ide, H. Fukuhisa, Y. Kondo, T. Yoshida, M. Nagamatsu, J. Mori, I. Yamazaki, and K. Ueno, "A 320-MFLOPS CMOS Floating-Point Processng Unit for Superscalar Processors", IEEE Journal of Solid State Circuits, vol. 28, No. 3, pp. 352 - 361, March 1993.

[28] H. Makino, H. Suzuki, H. Morinaka, Y. Nakase, K. Mashiko, T. Sumi, "A 286 MHz 64-b Floating Point Multiplier with Enhanced CG Operation", IEEE Journal of Solid State Circuits, vol. 31, No 4, pp. 504 - 513, April 1996.

[29] Mark R. Santoro, Gary Bewick and Mark A. Horowitz, "Rounding Algorithms for IEEE Multipliers", Proc. 9th Symp. on Computer Arithmetic, pp. 176-183, 1989.

[30] Craig Heikes, Glenn Colon-Bonet, "A dual Floating Point Coprocessor with an FMAC Architecture", 1996 IEEE International Solid-State Conference, pp 354-355, 1996.

[31] R. K. Montoye, E. Hokenek and S. L. Runyon, "Design of IBM RISC system/6000 floating point exceution unit", IBM J. Res. Develop., Vol. 34, pp. 59-70, 1990.

[32] Erdem Hokenek, Robert K. Montoye and Peter W. Cook, "Second Generation RISC Floating Point with Multiply - Add fused", IEEE Journal of Solid State Circuits, Vol. 15, pp. 1207-1213, October 1990.

[33] Erdem Hokenek and Robert K. Montoye, "Leading-zero anticipator (LZA) in the IBM RISC System/6000 floating-point execution unit", IBM J. Res. Develop., Vol. 34, pp. 70-77, 1990.

[34] Hiroaki Suzuki, Hiroyuki Morinaka, Hiroshi Makino, Yasunobu Nakase, Koichiro Mashiko and Tadashi Sumi, "Leading-Zero Anticipatory Logic for High-Speed Floating Point Addition", IEEE Journal of Solid State Circuits, Vol. 31, No. 8, pp. 1157-1164, August 1996.

[35] Javier D. Bruguera, Tomas Lang, "Leading-One Prediction with Concurrent Position Correction", IEEE Transactions on Computers, Vol. 48, No. 10, pp. 1083-1097, October 1999.

[36] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "A Low Power Approach to Floating Point Adder Design", in Proceedings of the 1997 International Conference on

Computer Design, pp. 178-186, 1997.

[37] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "On the Power Implications of Floating Point Addition in IIR Filters", Proceedings of the 10th International Conference on Microelectronics, pp. 200-203, December 1998.

[38] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "On the Distribution of Exponent Differences During Floating Point Addition", Proceedings of the 1998 Canadian Conference on Electrical and Computer Engineering, Vol. 1, pp. 105-108, 1998.

[39] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "Power Implications of Additions in Floating Point DSP - an Architectural Perspective", Technical report of the Department of Electrical and Computer Engineering (No. 1998-2-pillai), Concordia University, Montreal, Canada, 1998.

[40] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "Power Implications of Additions in Floating Point DSP - an Architectural Perspective", In Proc. AFRICON'99, pp. 581-586, Cape Town, Oct. 1999.

[41] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "On the Structure Driven Power Behavior of Additions in Floating Point IIR filters", Technical report of the Department of Electrical and Computer Engineering (No. 1998-3-pillai), Concordia University, Montreal, Canada, 1998.

[42] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "A Low Power Approach to Floating Point Adder Design", Int. Conf. on Computer Design ICCD'97, Oct. 1997, Aus-

tin, Texas.

[43] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "Power Implications of Precision Limited Arithmetic in Floating Point FIR Filters", 1999 International Symposium on Circuits and Systems, Orlando, Florida, pp. 165-168, May 1999.

[44] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, S. Y. A. Shah, "A Low Power Approach to Floating Point Adder Design for DSP Applications", accepted for publications in the Journal of VLSI Signal Processing, Feb. 2000.

[45] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "A Low Power Floating Point Accumulator", Proceedings of the Eleventh International Conference on VLSI Design, pp. 330-333, January 1998.

[46] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "Zero Overhead Rounding for Floating Point Adders", Technical report of the Department of Electrical and Computer Engineering (No. 1998-1-pillai), Concordia University, Montreal, Canada.

[47] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "Energy Delay Measures of Barrel Switch Architectures for Pre-Alignment of Floating Point Operands for Addition", in Proceedings of the 1997 IEEE International Symposium on Low Power Electronics and Design, pp. 235-238, Aug. 1997.

[48] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "Evaluation of 1's Complement Arithmetic for the Implementation of low power CMOS Floating Point Adders", Proceedings of the 1997 Canadian Conference on Electrical and Computer Engi-

neering, Vol. 1, pp. 153-156, May 1997.

[49] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "Energy Delay Analysis of Partial Product Reduction Methods for Parallel Multiplier Implementation", Digest of Technical Papers - 1996 International Symposium on Low Power Electronics and Design, pp. 201-204, Aug. 1996.

[50] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "Low Power Architecture for Floating Point MAC Fusion", Accepted for publication in Proceeding of IEEE Computers and Digital Techniques, To appear in the October 2000 issue.

[51] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "An IEEE Compliant Floating Point MAF", in proc. of VLSI'99, Portugal, Dec. 99, pp.165-168. System on Chip, edited by L. M. Silveria, S. Devadas and R. Reis. Kluwer Academic Publishers, ISBN 0-7923-7731..

[52] Israel Koren, "Compuer Arithmetic Algorithms", Prentice Hall, Englewood Cliffs, 1993.

[53] David Goldberg, "What every computer scientist should know about floating-point arithmetic", ACM Computing Surveys, Vol. 23, No. 1, pp. 5-48, March 1991.

[54] David Goldberg, "Computer Arithmetic", in Computer Architecture: A quantitative approach, D. A. Patterson and J. L. Hennessy, Morgan Kaufman, San Mateo, CA, Appendix A, 1990.

[55] Wasim Tout, A. J. Al-Khalili, "Digital Filter Structures using Pipelined Floating Point Components", Technical report of the Department of Electrical and Computer Engineering (No. 2000-1-tout), Concordia University, Montreal, Canada.

[56] K. C. Chang, "Digital Systems Design with VHDL and Synthesis - An Integrated Approach", IEEE Computer Society Press, Los Alamitos, California.

[57] Anantha P. Chandrakasan, Randy Allmon, Anthony Stratakos, and Robert Brodersen, "Design of Portable Systems", in Proceedings of the 1994 IEEE International Solid - State Circuits Conference, pp. 259-266, 1994.

Appendix A

List of Abbreviations

VLSI	Very Large Scale Integration
IEEE	Institute of Electrical and Electronics Engineer
MFLOPS	Millions of Floating Point Operations per Second
BFLOPS	Billions of Floating Point Operations per Second
VHDL	Very High Speed Integrated Circuit Hardware Description Language
ASIC	Application Specific Integrated Circuit
CAD	Computer Aided Design
RTL	Register Transfer Level
VSS	VHDL System Simulator
CLB	Configurable Logic Block
NaN	Not a Number
LSB	Least Significand Bit
MSB	Most Significand Bit
MUX	Multiplexors
TDPFADD	Triple Data Path Floating Point Adder
LZA	Leading Zero Anticipatory Logic, LZA data path
LZB	Leading Zero Bounded data path
MAC	Multiply - Accumulator
MAF	Multiply - Accumulate Fused Architecture

Appendix B
VHDL Code of Floating Point Multipliers

VHDL code of the proposed transition activity scaled double data path FPM and the single data path FPM, along with test benches, simulation results, dc-shell scripts to run Design Compiler and Power Compiler and reports generated after synthesis to 0.35 micron CMOS technology and FPGA technology (containing results which are summarized in Section 3.4) are in the discs attached.

Structure of the VHDL code of the proposed FPM is shown in Fig B.1. Top level entity *DDPFPM-PIPELINE* also contains registers for 3 stage pipelining as shown in Fig 3.1. In *SIGNIFICAND-MULTIPLIER*, 4:2 compressors are arranged in Wallace tree format (Fig 3.3). *STICKY5* is the entity containing 1's complement logic, shown in Fig 3.8, for the evaluation of Sticky bit.

Structure of VHDL code of single data path FPM is shown in Fig B.2.

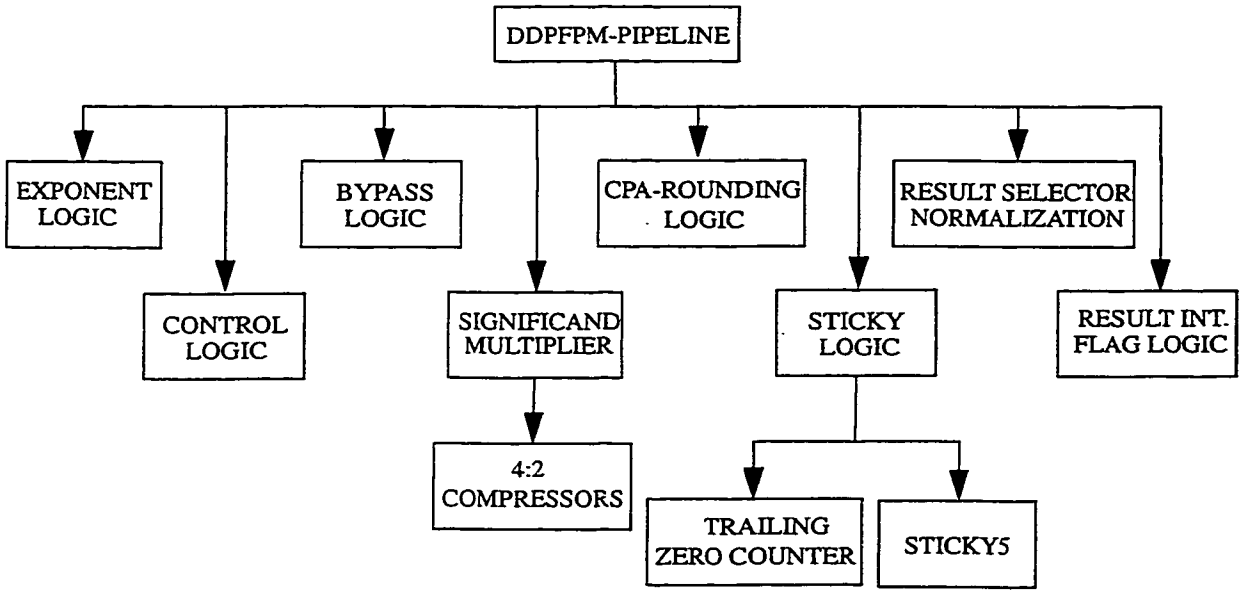


Fig B.1 - Structure of VHDL code of the proposed transition activity scaled FPM

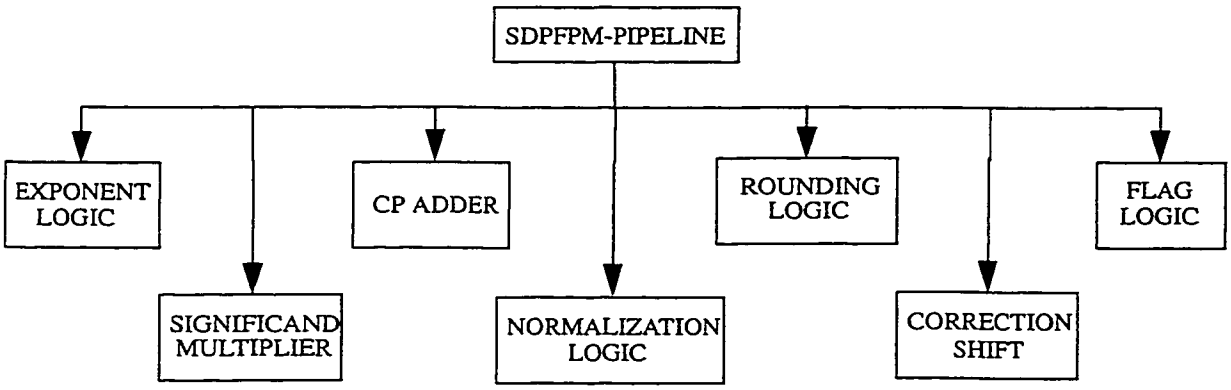


Fig B.2 - Structure of VHDL code of the single data path FPM

Appendix C
VHDL Code of Floating Point Adders

VHDL code of the transition activity scaled triple data path floating point adder (TDP-FADD) and the single data path floating point adder (FADD) for both pipelined and low latency architectures, along with test benches, simulation results, dc-shell scripts to run Design Compiler and Power Compiler and reports generated after synthesis to 0.35 micron CMOS technology and FPGA technology (containing results which are summarized in Section 4.5) are in the discs attached.

Structure of VHDL code of the low latency TDPFADD is shown in Fig C.1. Low latency architecture of TDPFADD has input output data alignment registers and a register stage between Control logic and three data paths to address transition activity scaling as shown in Fig 4.2. Pipelined structure of TDPFADD has the same sub-entities shown in Fig C.1 but some extra register stages are included to get five stages of pipelining as shown in Fig 4.13. *STICKY5* is the entity containing 1's complement logic, shown in Fig 3.8, for the evaluation of Sticky bit.

Structure of VHDL code of the low latency single data path FADD is shown in Fig C.2. Entity *SELECTORS* contains MUXs to select significand for pre-alignment shift and tentative exponent of the result. Pipelined single data path FADD has register stages as shown in Fig 4.16.

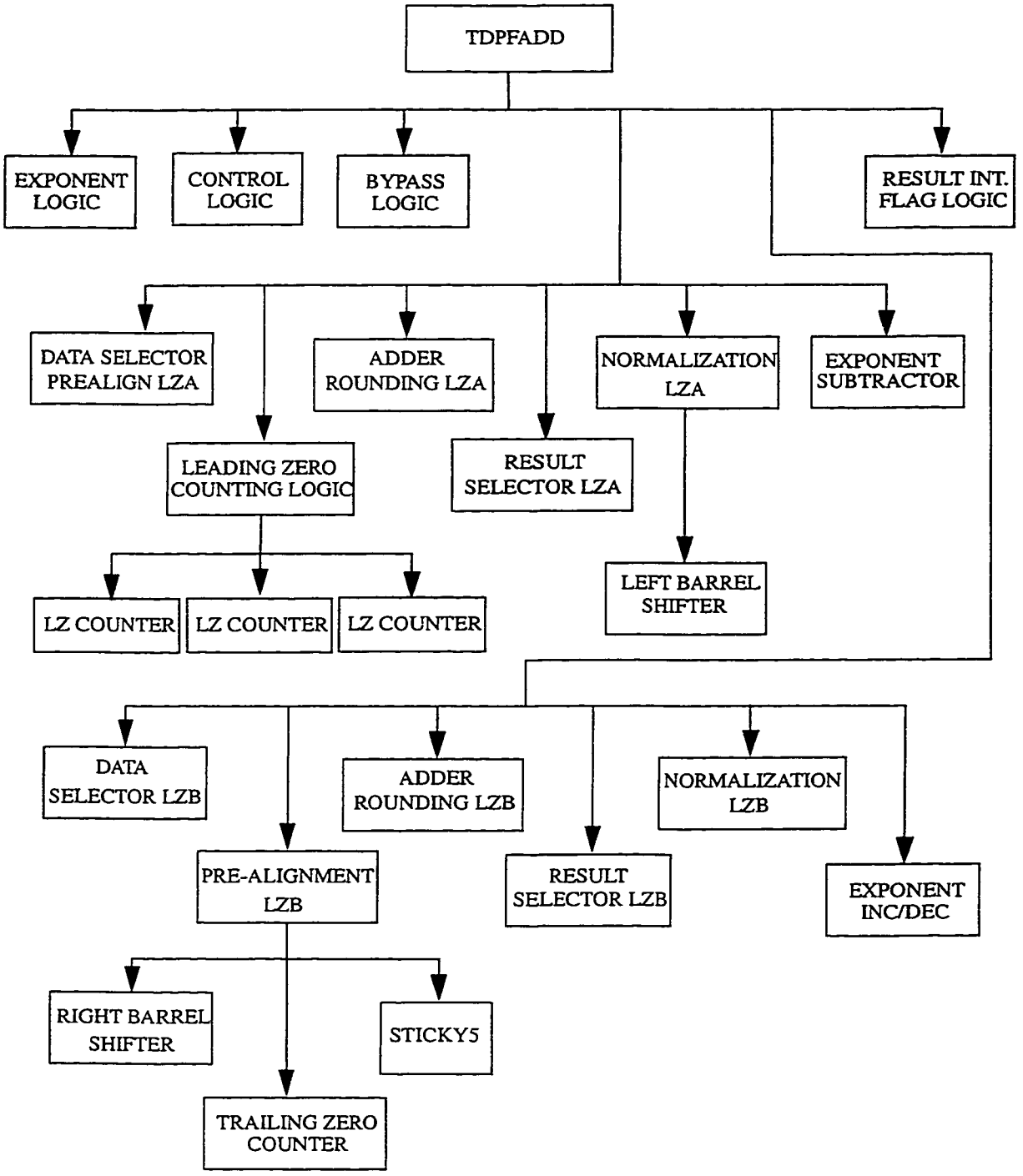


Fig C.1 - Structure of VHDL code of the TDPFADD

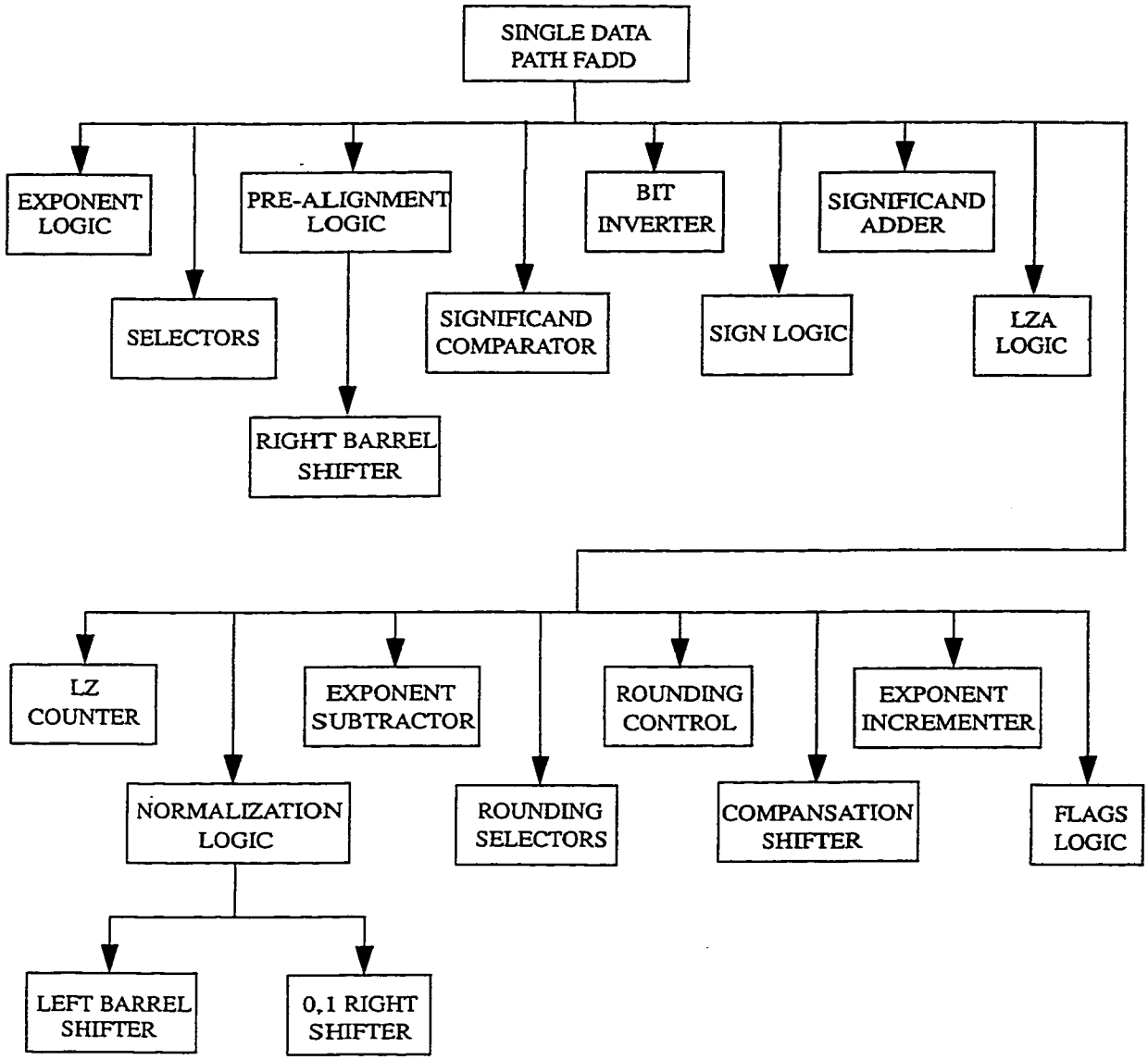


Fig C.2 - Structure of VHDL code of the single data path FADD

Appendix D

VHDL Code of Floating Point Multiply - Accumulate Fused Units

VHDL code of IBM compatible MAF, IEEE compliant MAF and IBM MAF, along with test benches, simulation results, dc-shell scripts to run Design Compiler and Power Compiler and reports generated after synthesis to 0.35 micron CMOS technology and FPGA technology (containing results which are summarized in Section 5.6) are in the discs attached.

Structure of VHDL code of the IBM compatible MAF is shown in Fig D.1. Pipelined registers are inserted in the top level entity *IBM-COMPATIBLE-MAF* as shown in Fig 5.9. *STICKY5* and *STICKY6* are the entities containing 1's complement logic, shown in Fig 3.8 and Fig 5.11 respectively.

VHDL code of IEEE compliant MAF is structured as shown in Fig D.2. *HA-ARRAY* entity contains the arrays of half adders. Pipelined registers are inserted in the top level entity *IEEE-COMPLIANT-MAF* as shown in Fig 5.12.

Structure of VHDL code of IBM MAF is shown in Fig D.3. Registers are inserted for pipelining as shown in Fig 5.15.

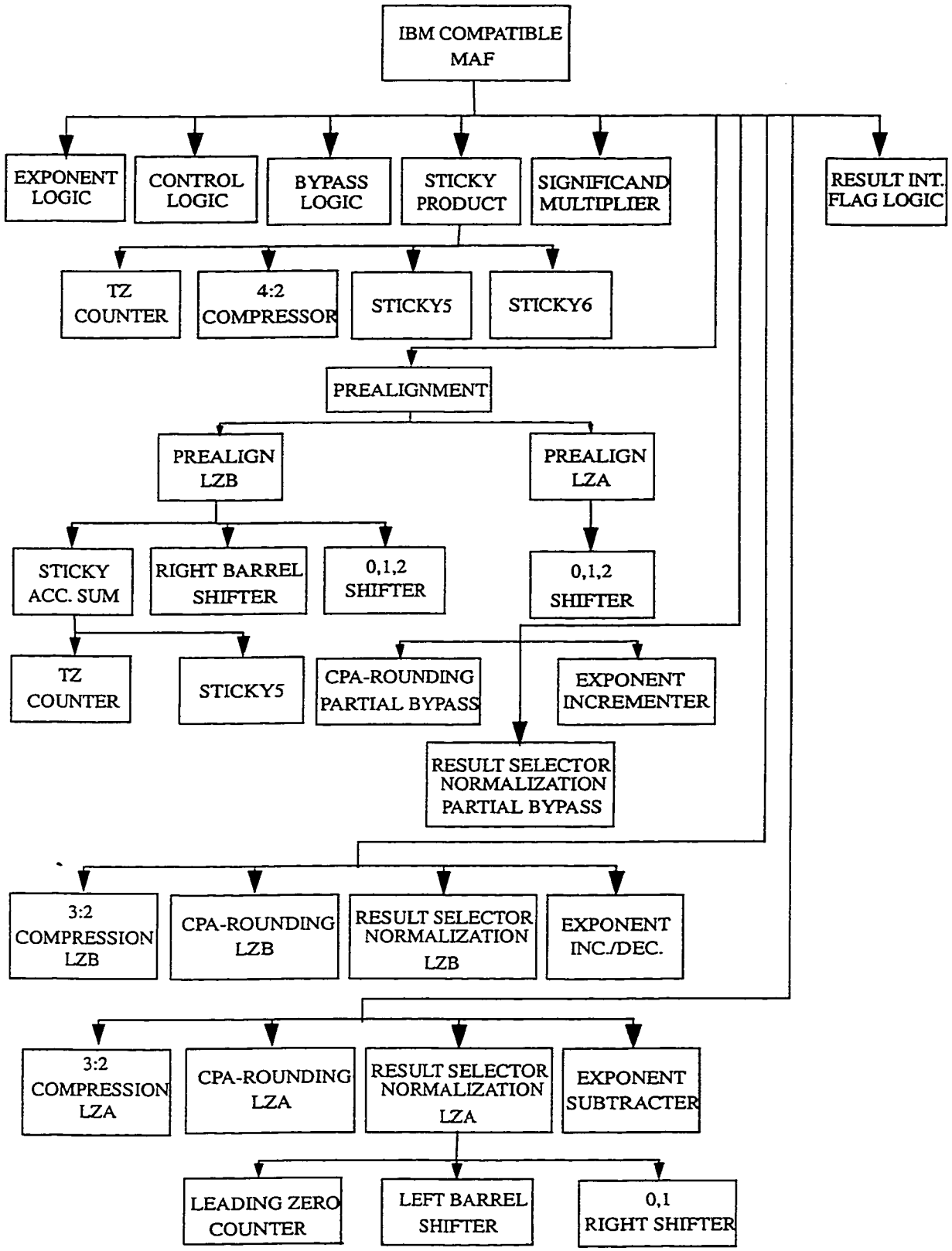


Fig D.1 - Structure of VHDL code of the IBM compatible MAF

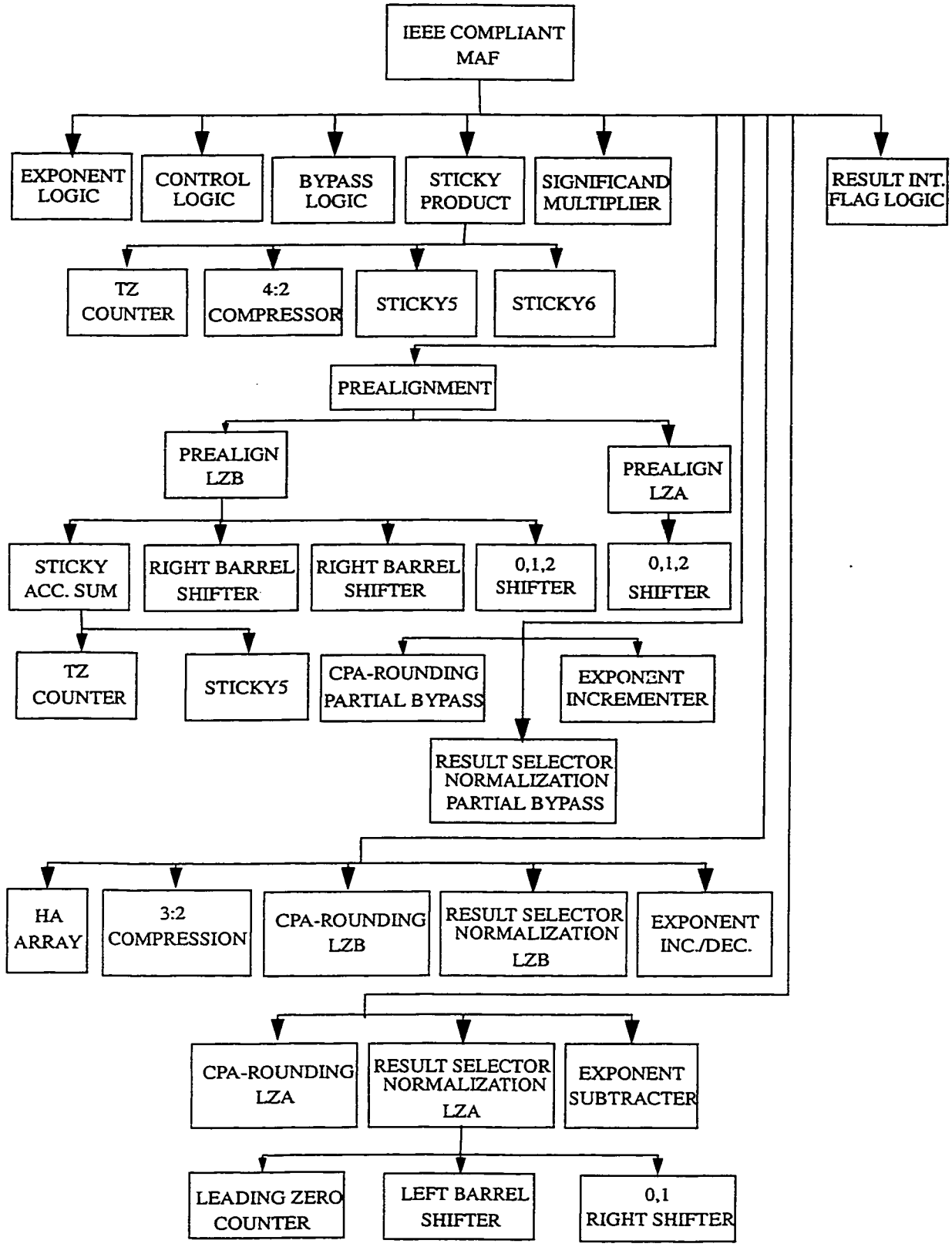


Fig D.2 - Structure of VHDL code of the IEEE MAF

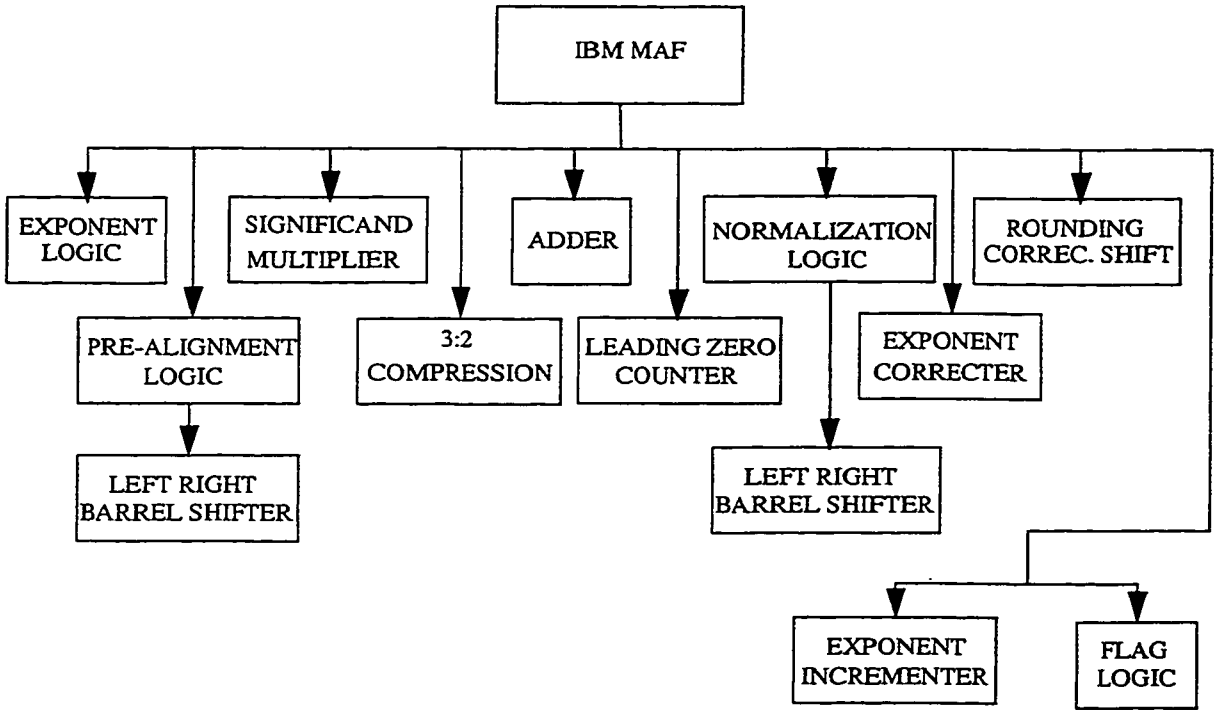


Fig D.3 - Structure of VHDL code of the IBM MAF