DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
CHENNAI - 600036

Optimization of Floating Point Modules for SHAKTI RISC-V Processors

A Thesis Submitted by

JOYANTA MONDAL

(EE19M058)

For the award of the degree Of

MASTER OF TECHNOLOGY

28 June 2021

QUOTATIONS

Some say the world will end in fire, Some say
in ice.
From what I've tasted of desire I hold
with those who favor fire. But if it had
to perish twice,
I think I know enough of hate To say
that for destruction ice Is also great
And would suffice.

ROBERT FROST

DEDICATION

To all of my beloved

THESIS CERTIFICATE

This is to undertake that the Thesis (or Project report) titled **OPTIMIZATION OF FLOATING POINT MODULES FOR SHAKTI RISC-V PROCESSORS**, submitted by me to the Indian Institute of Technology Madras, for the award of M.Tech., is a bonafide record of the research work done by me under the supervision of Dr. Nitya Ranganathan, as my project mentor. The contents of this Thesis (or Project report), in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

In order to effectively convey the idea presented in this Thesis, the existing work of the current version of floating point modules has been used with the permission of IIT,Madras.

Place: Chennai 600 036

Date: 28th Jun 2021

Joyanta Mondal
M.Tech(EE) Student

Prof. Kamakoti V.
Research Guide

Prof. Boby George
Research Co-Guide

TABLE OF CONTENTS

ACKNOWLEDGMENTS

Thanks to my mentor Dr. Nitya Ranganathan and all those who made this possible with their valuable thoughts .

ABSTRACT

SHAKTI is an open-source initiative by the Re-configurable Intelligent Systems Engineering (RISE) group at **Indian Institute of Technology, Madras** to develop the first indigenous Indian industrial-grade and commercial-grade micro-processor based on RISC-V architecture. Currently SHAKTI Team is working on development of C-Class , E- Class and I-Class, these 3 variants of processors for different applications like IoT,Desktop Applications, Cloud-computing etc. The C-Class and I-Class core supports the computation of IEEE 754 floating point formats, with pipelined floating-point modules, for different computation-intensive applications.

The existing floating-point modules in C-Class and I-Class cores, based on Bluespec library, are parameterized for flexibility and convenience of instantiation of any bit-widths. This modules support the computation of all of the IEEE 754 standard formats. In the other hand, the implementation in C-Class and I-Class still use only 32 and 64-bit floating point but using these parameterized modules gives us reduced performance since we cannot do effective optimizations that would typically be done if we know the bit-width in advance. In the existing modules, some of the algorithms were subjected to generate long critical path due to parameterized bit width , therefore making the overall performance slower. Those algorithms have been optimized with the help of known bit-width of the inputs, for C and I-Class floating point modules.

The existing floating-point modules in SHAKTI processors, were in the form of combinational circuits followed by the placement of several dummy registers, and the software based optimization result was not up to the mark due to high burden on register-retiming. The dedicated single precision and double precision version of floating point modules have been re-designed with manual-pipeline of internal registers, to reduce the register placement overheads on the software based register-retiming process. Manual register placement has significantly reduced the number of arbitrarily placed dummy registers and improved the efficiency of optimal-pipeline process with the help of internally placed registers inside the design. As a result of manual pipeline and algorithm optimization for fixed bit width, the new dedicated 32 and 64 bit floating-point multiplier, converter and adder modules has shown significant improvement on their performance. These modules are being currently used with the I-Class core, but also suitable to be plugged in to the C-Class cores.

LIST OF TABLES AND FIGURES

GLOSSARY

The following are some of the commonly used tools and programs in the project:

BSC ***Bluespec*** compiler, provides maximum compatibility with any synthesis toolchain and comes with an included simulator ("Bluesim"). *Bluespec* provides RISC-V processor IP and tools for developing RISC-V cores and subsystems.

Design-Vision Software used for synthesis of RISC-V subsystems in SHAKTI modules, developed by Synopsis.

Testfloat Testcase generation tool for floating point numbers for different operations, developed by University of California at Berkeley.

ABBREVIATIONS

IITM	Indian Institute of Technology Madras
GHz	Gigahertz
FMA	Floating-point Multiplication and Addition
IEEE	Institute of Electrical and Electronics Engineers
MSB	Most Significant Bit
LSB	Least Significant Bit
NaN	Not a Number
RISC	Reduced Instruction Set Computer
ISA	Instruction Set Architecture
I.D	Input Delay
O.D	Output Delay
FinFET	Fin Field-Effect Transistor
RCA	Ripple Carry Adder
CLA	Carry LookAhead Adder

CHAPTER 1: INTRODUCTION AND BACKGROUND

SHAKTI is the First Indian micro-processor, designed and developed by RISE group at IIT-Madras, with an aim of building an open-source ecosystem of production and commercial grade processors, SoCs' and peripheral IPs', development boards and SHAKTI based software platforms. There are currently 3 variants of SHAKTI Processors based on **RISC-V** architecture, aimed at Internet of Things (IoT), Embedded and Desktop markets, such as,

A) **E-Class** : the E-Class are 32/64 bit micro-controllers capable of supporting all extensions of the RISC-V ISA, aimed at low-power and low-computation applications. The E-Class is an in-order 3 stage pipeline having an operational frequency of less than 200 MHz on silicon. It is positioned against ARM's M-Class (Cortex-M series) cores. It is capable of running real-time operating systems like FreeRTOS, Zephyr and eChronos. Market segments of E-Class processor support smart-cards, IoT devices, motor controls and robotic platforms.

B) **C-Class** : The C-Class is a 64 bit controller class of processor, aimed at mid-range embedded applications. The core is highly optimized, 5-stage in-order design with MMU support and the capability to run operating systems like Linux and Sel4. It is extremely configurable with the support of the standard RV64GC ISA extensions. It targets mid-range compute systems running over 200-800 MHz. It can also be customized up to 2 GHz. It is positioned against ARM's Cortex A35/A55. The application domain of this class ranges from embedded systems, motor-control , IoT, storage, industrial applications such as networking, gateways etc.

C) **I-Class** : The I-Class is a super-scalar 4-wide out-of-order (OoO) processor with potential applications in general-purpose computing and high-end embedded markets. It's features also include multi-threading, aggressive branch-prediction, non-blocking caches and 12-stage deep pipeline system. The operational clock frequency of this processor is 1.5 to 2.5 GHz. The team is currently working on implementing atomics, memory dependence prediction, instruction window/scheduler optimizations, implementation of some functional units, performance analysis/projections and optimizations to meet the first target frequency on 1 GHz on 22 nm processor.

The C-Class and I-Class core contains pipelined floating-point modules for IEEE single precision and double precision formats. The main advantage of floating point is that can represent any data within its range, using fixed width of bits. IEEE Standard 754 floating point is the most common representation today for real numbers on computers, including Intel-based PC's, Macs, and most Unix platforms.

IEEE 754 Format has 3 basic components:

1. **The sign of Mantissa** : This is as simple as the name. 0 represents a positive number, while 1 represents a negative number.
2. **The biased exponent** : The exponent field needs to represent both positive and negative exponents, A bias is added to the actual exponent in order to get the stored exponent.
3. **The normalized Mantissa** : The mantissa is part of a number in scientific notation or a floating point number, consisting of it's significant digit. Here we only have two digits, i.e. 0 and 1. So a normalized mantissa is one with only one 1 to the left of the decimal.

The floating point format used in the SHAKTI cores are only the single precision (32 Bit) and double precision (64 Bit) type, and dedicated to be used with the C-Class and I-Class cores. The single precision floating point number is of 32 bit size, and the MSB(31st Bit) denotes the sign, the next 8 bit (Bit [30:23]) is reserved for exponent and the rest 23 bits (Bit [22:0]) indicates the mantissa part without the normalization bit. The double precision format has an width of 64 bits, with MSB (63rd Bit) as sign, next 11 bits (Bit [62:52]) as exponent and the rest 52 bits (Bit [51:0]) as the mantissa, without the normalization bit at the left. The exponent bias used, is 127 for single-precision and 1023 for double-precision format and added to the actual exponent, to represent the respective numbers.

IEEE has reserved some values that can cause ambiguity inside the operation, and these values are considered as special values with an indication of raising special flags, if generated by any floating-point operation. This values are,

a) **Zero :** exponent and mantissa bits are 0. -0 and +0 are distinct values, though they both are equal.

b)**Denormalized :** If the exponent is all zeros, but the mantissa is not, then

the value is a denormalized number. This means this number does not have an assumed leading one before the binary point.

c)**Infinity :** The values +infinity and -infinity are denoted with an exponent of all ones and a mantissa of all zeros. The sign bit distinguishes between negative infinity and positive infinity. Operations with infinite values are well defined in IEEE.

d) **Not A Number (NAN) :** The value NAN is used to represent a value that is an error or does not hold any value.. This is represented when exponent field is all ones with a zero sign bit and some non-zero number in the significand field. This is a special value that might be used to denote a variable that doesn't yet hold a value. There are two types of NaN representation in IEEE format, one with the MSB Bits of mantissa (23^{rd} Bit) to be zero, is called Signaling NaN or sNaN, generates an error while passing through any function, indicating an invalid operation. The NaN format with 1 as MSB Bit of mantissa (23^{rd} Bit) , is called Quiet NaN, and it can be passed through most of the operations, without generating any exception.

The range of positive floating point numbers can be split into normalized numbers, and denormalized numbers which use only a portion of the fractions' precision. Since every floating-point number has a corresponding, negated value, the ranges above are symmetric around zero. The typical dynamic range of floating point format is defined as $[-(2 - 2^{-(\text{mantissa size})}) \times 2^{\text{bias}}$ to $+(2 - 2^{-(\text{mantissa size})}) \times 2^{\text{bias}}]$, considered to be highest among all binary number representation formats.

There are five distinct numerical ranges that floating-point numbers are not able to represent with the scheme presented so far, and generates exception on following cases:

1. Negative numbers less than $-(2 - 2^{-(\text{mantissa size})}) \times 2^{\text{bias}}$ (negative overflow)
2. Negative numbers greater than $-2^{-(\text{bias}+\text{mantissa size} -1)}$ (negative underflow)
3. Zero
4. Positive numbers less than $+2^{-(\text{bias}+\text{mantissa size} -1)}$ (positive underflow)
5. Positive numbers greater than $+(2 - 2^{-(\text{mantissa size})}) \times 2^{\text{bias}}$ (positive overflow)

It is possible to express very large and small numbers (1.17exp-38 to 3.4exp+38 for single precision and 2.22exp-308 to 1.8exp+308 for double precision) with fixed bit size, in IEEE 754 floating point formats and therefore very convenient to

be used for almost all scientific and computation intensive operations in most of the computer platforms. The existing floating point modules in the SHAKTI cores were designed to instantiate for any bit width, along with the large combinational pattern to be optimized by software-based register-retiming, that eventually failed to achieve the stipulated performance due to complex constructions of the algorithm. On the contrary, floating point modules dedicated for 32 bit and 64 bit input size, were required for C-Class and I-Class with faster performance and therefore the existing design needed further optimizations to implement parallelism in some the algorithms to shorten the overall critical path and manual pipeline to reduce the burden for register-retiming on the hardware synthesis tool. The goal of this paper is to discuss about the optimization procedure for floating-point multiplier,converter and adder-subtractor modules for C-Class and I-Class, to improve their performance.

CHAPTER 2: ALGORITHM OPTIMIZATION

The existing design of Floating Point modules in SHAKTI Processors, derived from standard Bluespec Library, gives instantiation support for all of the standard IEEE floating point formats, i.e, Half-precision,Single-precision, double-precision,quad-precision etc., but that leads to a complex design of variable bit-width and imposes high burden on software-based register-retiming process due to it's combinational type construction pattern. Eventually the performance of the existing design fails to meet the target frequency of 1 GHz in I-Class core, for several floating point units, such as Floating-Point Multiplier, Converter, Adder-Subtractor and FMA units. Several cases, combinational algorithms used in the old design, due to their construction complexity to combine all the supported formats, affected the performance of the overall module with their extremely large critical path, and were not optimized during the register-retiming process. The other hand, the current version of SHAKTI C-Class and I-Class cores contains support for only IEEE 754 single and double precision formats of floating point representation, and the older design algorithms needed optimizations in their critical path, with the development of two separate 32 bit and 64 bit fixed input size modules, to perform under target frequency. Here, the prior knowledge of input size helps to reconstruct some of the algorithms with parallel operating segments to reduce the overall critical path, and this was not possible for the parameterized bit-width cases. For example, the normalization operation used in all of the floating-point modules, employs the MSB zero counting algorithm and that was carried out by a "for – loop " with inter-dependent iterations, generating long critical path for higher input size in parameterized cases. If the input size was fixed, the MSB zero counting algorithm could be divided into several parts those can work parallel, and lastly choosing one of them would be easier to reduce the overall critical path. The rounding operation has also been improved to achieve better accuracy with the inclusion of sticky bits, in the rounding algorithm. Several cases, it has been inspected that keeping only the 32-bit and 64-bit formats will lead to less design space compared to the older complex designs along with improvement of performance. The optimization description for several algorithms is being discussed in detail, in the later sub-topics.

2.1 Optimization in MSB Zero Counting:

All of the modules make use of an "MSB zeros counting" algorithm, in their normalization operations. The previous design of this algorithm was generating long chain due to inter-dependent ‘for-loop’ for counting zeros from MSB side, in case of parameterized bit-width instantiation. There was no way to split the algorithm to parallel segments for this case as we could not know the number of parallel segments beforehand. The new dedicated 32 bit and 64 bit modules to be used with I-Class and C-Class, the fixed size input is split into pre-calculated numbers of segments, then MSB zeros are counted from each of the segments in parallel, with ‘for-loop’s with much less size, and lastly final number of MSB zeros can be obtained by choosing or adding MSB zero-counts of the segments, based on the result of ‘OR’ing all of the bits of a particular segment. If the ‘OR’-result is 0 then the bit-width of that segment is added to the next segment count, otherwise the count goes to the actual output. The segmentation helps to count zeros parallel from different segments of the input, that can reduce the overall computation time to computation time without segmentation/n + multiplexing time , where n is the number of segmented partitions, and multiplexing time is << zero-computation time of one segment. This modification has shown a significant impact to reduce the overall latency for all of the modules.

Flow-Chart of Parallel MSB Zero Counting Algorithm :
(32 Bit example)

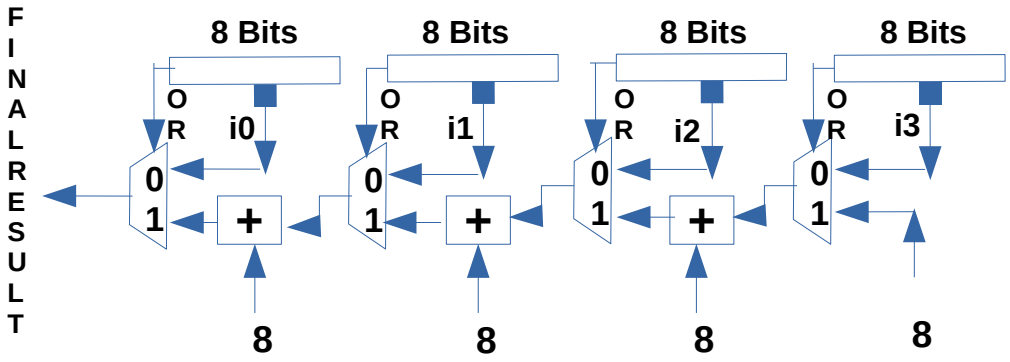


Fig 1: Modified MSB Zero Counting Unit

2.2 Algorithm Optimization for Multiplier Module:

The existing version for floating point multiplier module was using normal multiplication algorithm that failed to operate under stipulated frequency. The multiplication algorithm in the new modules has been modified with modified Booth's Algorithm for unsigned numbers(as there as separate sign bit for floating point formats).The internal multiplications to generate the partial products has been generated with shift and add procedure.

A typical floating point multiplication is as follows,

a) Sign = XOR (sign of multiplicand,sign of multiplier)

b) Exponent = Sum of the exponents of multiplier and multiplicands

c) Mantissa = Product of the mantissa parts of multiplier and multiplicand, rounded to mantissa size.

Evidently, the multiplication operation is carried out only on mantissa parts of the floating point input operands, 23 bits for single-precision and 52 bits for double-precision numbers. The standard multiplication algorithm used in the existing multiplier module gives us long critical path due to sequential addition of 23 Nos. of 23 bit vectors and 52 Nos. of 52 bit vectors, for single and double precision numbers. This process can be illustrated by the pseudo-code :

```
product = 0;
for ( integer i=0 ; i < size_of_multiplier_mantissa; i=i+1)
{
        if (multiplier[i]== 1)
                { product = product + multiplicand << i ; }
}
```

This standard multiplication algorithm generates long adder chain for higher multiplier and multiplicand size. The size of adder chain has been reduced in the new multiplier module by employing the Modified Booth's Algorithm to the mantissa multiplication part. The modified Booth’s multiplication algorithm provides us an way to reduce the number of partial products and generating them in parallel. According to the new algorithm, the mantissa parts of both multiplier and multiplicand has been extended to left by 1 bit for single-precision (result = 24 bit) and 2 bits for double-precision (result = 54 bit)to make them multiple of 6. Now the multiplier vectors are divided into group

of 6 bits and we obtain 4 groups in case of single, and 9 groups in case of double precision numbers.

Now each of the group is individually multiplied with the multiplicand vector using shift-and-add process, to generate the partial products of the size = multiplicand size + size of multiplier segments . Therefore, it generates 4 Nos. of 24+6 = 30 bit partial product vectors for single-precision and 9 Nos. of 54+6 = 60 bit partial product vectors for double-precision numbers. All of this partial products are then shifted and added to obtain the final product vector. The following pseudo – code describes the partial product addition part of the new algorithm, such as,

```
actual_product = 0;
for ( integer i=0 ; i < No._of_partial_products i=i+1)
{
        actual_product = actual_product + partial_product [i] << (i*6) ;
}
```

The typical no. of partial products will be 4 and 9, for single and double precision numbers, respectively. On the comparison, the standard multiplication algorithm using ‘ * ‘, employed in the existing module has to do 22 sequential addition of 23 Nos. of 23 bit vectors and 51 sequential addition of 52 nos. of 52 bit vectors for single and double precision versions respectively, but the new multiplier using the Modified Booth’s multiplication algorithm performs 9 sequential addition (5 sequential addition of multiplicand of 24 bits, for 6 vectors + 3 sequential addition of partial-product of 30 bits each, for 4 vectors) for single-precision and 14 sequential addition (5 sequential addition of multiplicand of 54 bits, for 6 vectors + 8 sequential addition of partial-product of 60 bits each, for 9 vectors) for double-precision versions, with the same type of adder algorithm (The standard RCA version from Bluespec library). The new multiplication algorithm has reduced the number of partial product addition in a great way, and helped to improve the performance of the overall unit.

Flow-Chart of Modified Booth's Algorithm : (24 Bit * 24 Bit example)

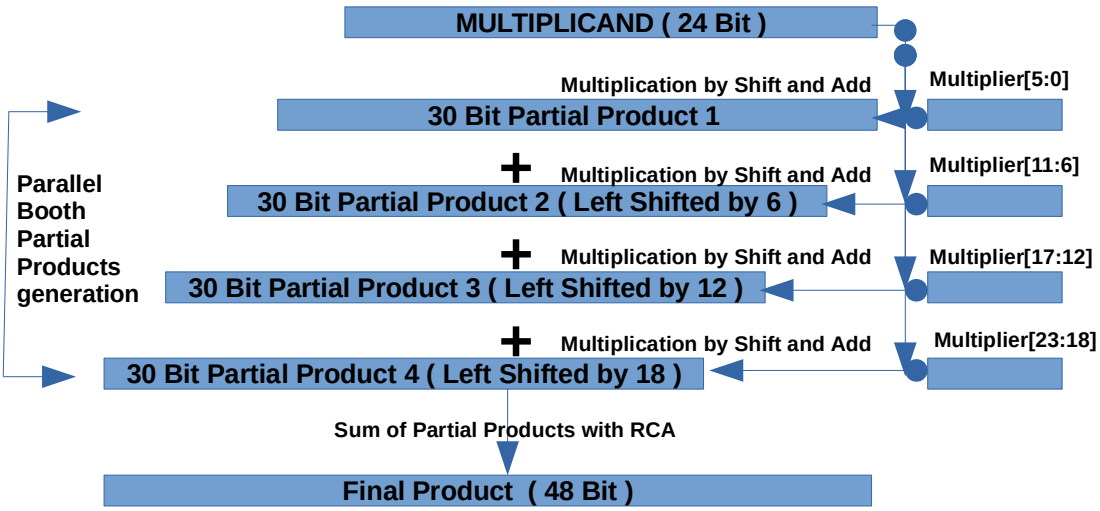


Fig 2: Booth Multiplication Process Diagram

2.3 Algorithm Optimization for Adder Module:

A typical floating point addition takes the following steps :

a) sign bit = sign bit of Max(Operand 1, Operand 2)

b) exponent = exponent of Max(Operand 1, Operand 2)

c) unrounded mantissa or significand = (mantissa of Max(operand1,operand2)) + (mantissa of Min(operand1,operand2))>>exponent difference of two operands. The result is the rounded to obtain final mantissa of the result.

Only the extended mantissa part of two operands is required for addition operation and for the previous version of floating point adder,normal ripple carry adder from the standard Bluespec library, was used for the mantissa-addition part. The ripple carry addition process is inherently slow and the delays also increases with the size of the adder itself. As a result,the previous floating point adder module failed to meet the target frequency after the software-pipelining. To overcome this problem,The new dedicated 32 bit and 64 bit floating point addition module contains 28 bit (mantissa extended by 2 bits to left and 3 bits to right)and 57 bit (mantissa extended by 2 bits to left and 3 bits to right) version of Carry-Lookahead-Adder to speed up the addition process.

The normal addition process using Ripple-Carry-Adder involves a large critical path to calculate the MSB of the result ,as addition with full-adder block for each bit pair depends on the carry from the addition of previous bit pair. A typical Ripple-Carry addition process for 8-bit example has been presented here:

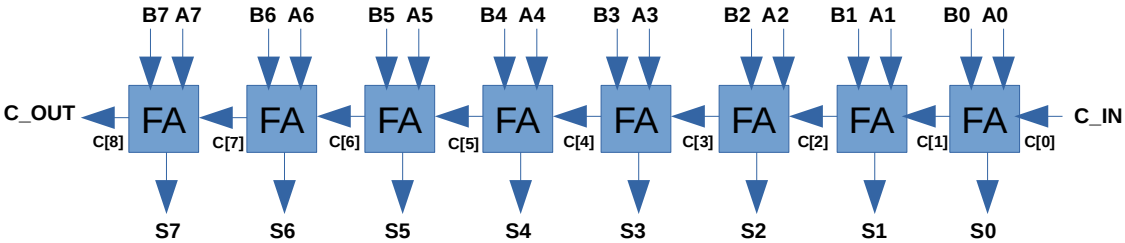


Fig 3.1 Schematic of 8 bit ripple-carry adder

The sum and carry bits generated by any full-adder block, is dependent of the carry generated at the previous full-adder block, as C[i+1] = (A[i] & B[i]) | (A[i] & C_in[i]) | (B[i] & C_in[i]) and S[i] = A[i] ^ B[i] ^ C_in[i]. There is no way to calculate the carry bits generated from each bit pairs before proceeding to actual addition, so the addition of one pair of bits, has to wait for the carry generated from the previous bit. In this way, the RCA version of adder introduces tremendous delay to generate the final sum and carry-out for higher adder size.

This bottleneck has been overcome by The Carry-Lookahead-Adder version with two N bit vectors A and B in the following way,

i) The partial sum vector(Z) is generated by Z= A **XOR** B;

ii) The carry vector(C) is generated by a combinational circuit beforehand ,

C[i] = G[i] **OR (** P[i] **AND** C[i] **)**

where G[i] = A[i] AND B[i], P[i] = A[i] OR B[i], C[0] = C_in.

iii) The final sum vector(S) is generated by S= Z **XOR** C;

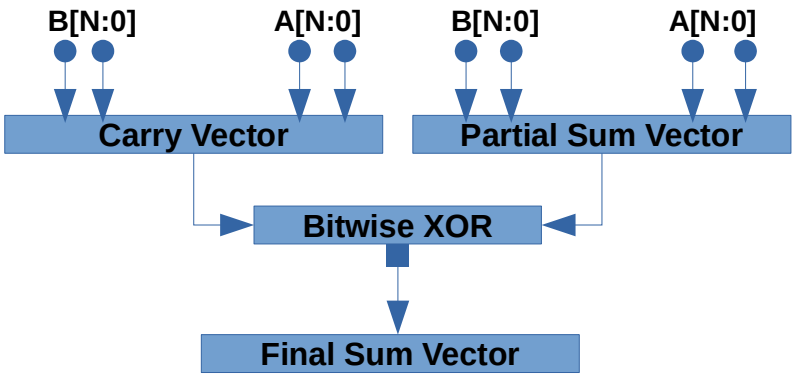


Fig 3.1 Schematic of N bit carry-lookahead adder

For carry look-ahead addition, it is a 3 stage process irrespective of the adder size (**N**). All of the carry bits are either generated (G[i] == 1) or the previous carries are propagated(P[i] ==1) depending on the status of input bits, with a special carry-generation circuit to obtain a carry-vector comprising of all the actual carry_in bits for every bit pair. Thus all the carries are available for every pair of bits before the actual addition part. This eliminates the long critical path delay issues in MSB Bits due to the previous carry dependency,as in the RCA version of adders. The CLA adders are considered as one of the fastest type of adders, and that helped improving the performance of new floating point adder module compared to the previous version. The typical carry-generation circuit to generate 8 bit carry-vector, for a 8 bit carry-lookahead adder has been shown in the next diagram.

Special Carry-Evaluation Circuit for CLA Adder (8 bit Example) **:**

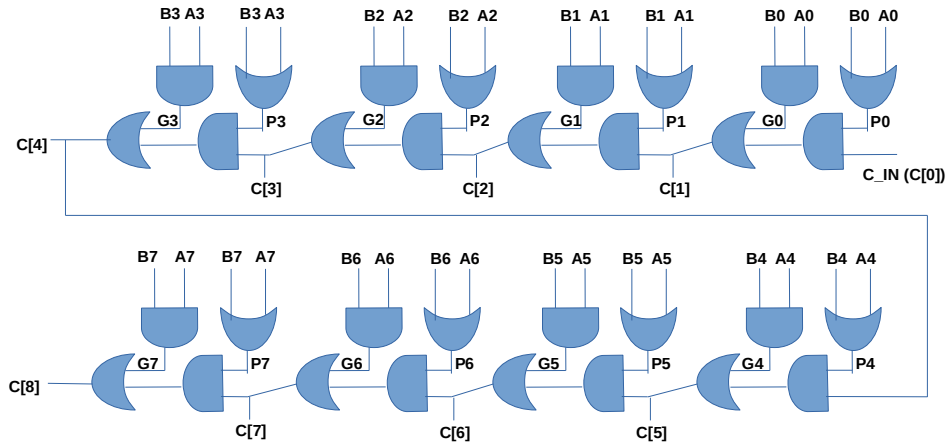


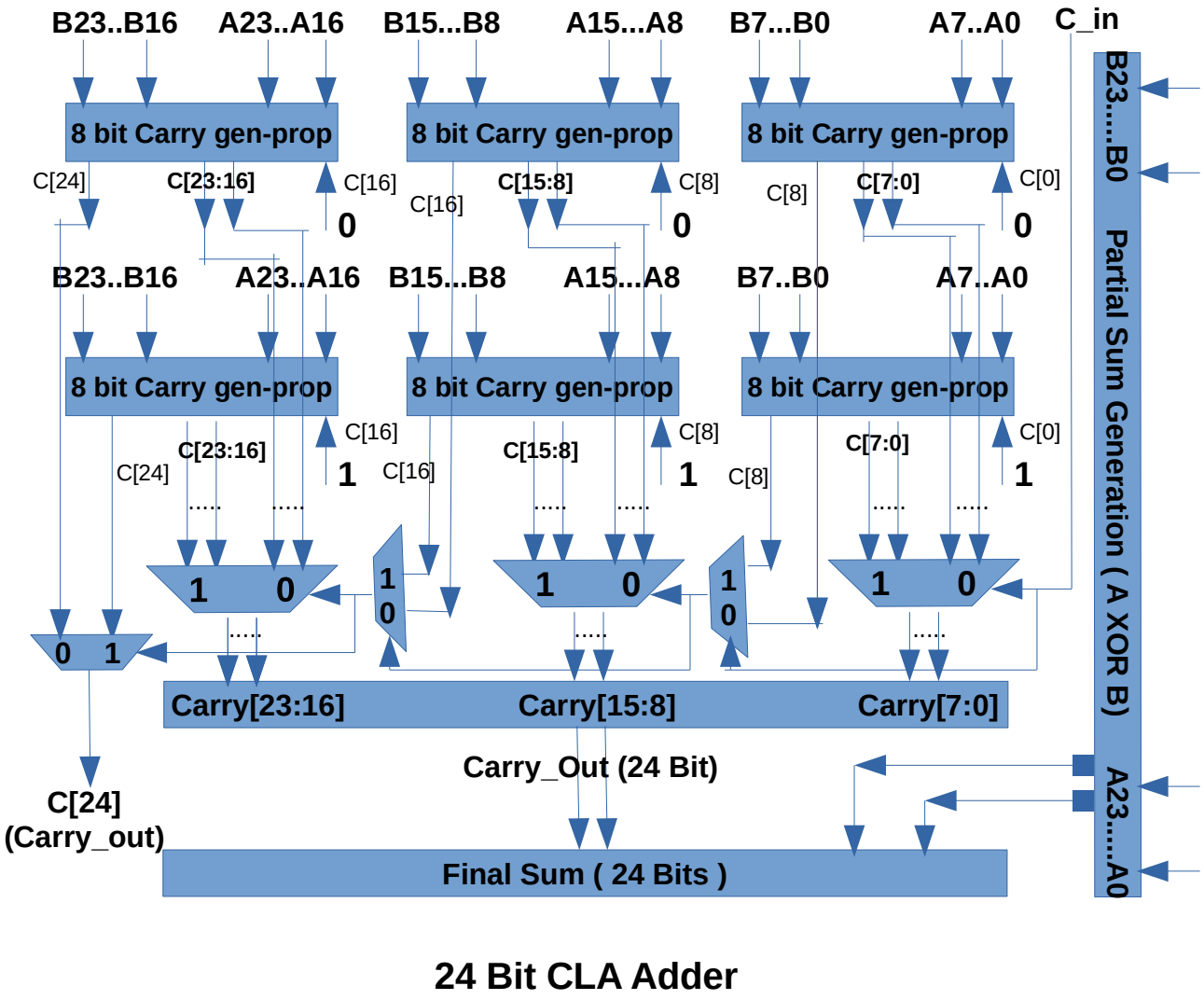
Fig 3.3 Schematic of special carry-evaluation circuit in carry-lookahead adder

The partial sum generation (Z= A^B), carry-generate (G = A&B) and carry-propagate (P = A|B) bits can be generated from the input bits directly and in parallel, at the first stage of CLA Adder. The actual carry evaluation process with the generate and propagate bits, needs a complex circuit for carry generation and propagation part,and as the each carry propagation stage depends on the previous carry generation, and increases the critical path length of the system for large bit-width. The floating-point adder modules in C-Class and I-Class incorporates addition of two 28 bits mantissa for single-precision and two 54 bits mantissa, for double precision numbers, therefore the above construction of carry-evaluation circuit introduces large delay to obtain the final carry-vector for large size. To further optimize this design, the operand vectors are segmented in several equal parts (4 Nos. of 7 bit segments in single-precision and 7 Nos. of 7 bit segments, 1 No. of 8 bit segment in double-precision), and each of the segment pairs of operands, are fed to individual pairs of carry-evaluation blocks(for both carry_in =0 and carry_in = 1) to concurrently generate the carry-vectors of the associated segments. Two carry segments in the carry-evaluation blocks are being evaluated for a particular input pair, for both carry_in as 0 and 1, and being chosen as actual carry segments at the final section, depending on the carry_out bit of each previous section . The carry_out bit and carry segment of the first section is chosen from two segments, by the actual carry_in bit to the adder module itself and this process repeats sequentially for all the rest of the segments. This type of construction eliminates the long carry generation chain for larger bit widths and the critical

path for carry generation gets reduced as a multiple of almost 1/n, where n is the number of segments of the actual carry vector .Meanwhile, the partial sum vector is generated by bitwise-XOR of operand mantissa-vectors,and the partial sum is again bit-wise XOR-ed with the generated carry vector to obtain the final sum as the result.

A typical example for 24 Bit modified CLA Adder with 3 segments of 8 bits, has been illustrated in the next diagram.

Schematic of Modified Carry-generation circuit:
(24 Bit example of Carry-Look-Ahead Adder)



24 Bit CLA Adder

Fig 3.4 : 24 Bit Carry-Look-ahead Adder

2.4 Improvement in Rounding Operation:

Rounding operation is very essential for all of the multiplier,adder and converter modules, as the mantissa results from actual operations are often generated in unrounded manner. The rounding operation is the heart for any operand to single-precision floating-point conversion function, and also plays an unavoidable role to generate the final results of the other converters,multiplier and adder modules. The accuracy of rounding operation inherently depends on the accuracy of rounding modes. There are 3 significant bits in the rounding specification of IEEE 754 standard : a) the bit just after the significand part is the “guard” bit, b) the bit after guard bit is the “round” bit, and the OR of rest of the bits in the unrounded portion is considered as the “sticky” bit. The existing floating point modules employed the rounding of mantissa part with guard and round bits only. The other hand,the new modules uses the sticky bits along with the guard and round bits, to improve the accuracy of rounding operations further. This modification has a significant impact on the rounding of numbers, where the guard and round bits are zero, but enough non-zero bits, in the rest of the portion in the unrounded version.

2.5 Special Operations and Exception Generation:

The input data of all of the floating point units may contain infinity, zero or NaN, also this type of data can be generated from the operation itself. The result should be reported as specified formats according to IEEE standards for representation of ambiguous results. This situation leads to special operations and has been implemented as a parallel path to the main computation path, inside all of the new floating point modules. The exceptions generated from the results due to the NaN operations,overflow,underflow,and inexact condition of output due to rounding,are taken care of according to SHAKTI-RISCV specified standards.

CHAPTER 3: DEVELOPMENT OF MANUAL PIPELINING

As mentioned earlier,The previous version for floating point modules were designed to support all of the standard formats of IEEE 754 floating point specification and all of the modules were created as series of combinational circuits, and performance analysis was based on software-based register-retiming process. Dummy registers were added in overall critical path of the modules,after the series of the combinational logic blocks. The dummy registers were optimally placed in the combinational blocks to create pipeline stages by the synthesis tool, and the number of dummy registers were varied to accommodate the design with no or least negative slack in the critical path. Eventually the synthesis failed to create an effective register placement to accommodate the design to operate under target frequency of 1 GHz in I-Class, because of high computational loads in software-based register-retiming process.

The current version of SHAKTI I-Class and C-Class processors use IEEE 754 single precision and double precision formats only,so there would be no requirement of a module supporting all the standard versions . The new floating point modules uses dedicated counterparts for 32 bit and 64 bit floating point operations and each of the module has their own pipelined design,and have unique internal pipeline register placements. Much less number of dummy registers has been added to critical path after the internal pipelining registers for each module,to reduce the computational burden on software-based register-retiming. As the number of dummy registers reduces,the placement of the dummy registers plays an important role to the overall performance of the modules,which has been inspected and reported with the most effective placement towards improving the latency.

Important computational blocks from the existing modules for floating point modules in C-class and I-class cores, have been identified and optimized to reduce the overall critical path, as the first step of optimization process. Next, each of the optimized computational blocks has been placed within the sets of internal registers to create a pipelined design. Also dummy registers have been placed in suitable position inside the pipeline stages, to accommodate the design with no timing violation. The register-retiming process performed by the synthesis tool, now has to optimize the register placements inside the local pipeline stages, instead of optimal register placement on the whole design at a time. Thus, the manually constructed pipeline stages in the new modules, has a great impact on reducing the burden on register-retiming and the register-retiming process is becomes more effective for the new modules, accommodating the overall design of 32 bit and 64 bit versions into 1 and 2 cycle less latency, compared to the existing version of the multiplier and converter modules. The same optimization on floating

point adder module, improves the timing violation at the same latency.

The dummy registers placed after the main combinational circuit in the existing modules, were not optimally placed in the software-based register retiming process and the overall module failed to achieve the target latency for floating-point multiplier, converters and adder modules. The new modules have dedicated internal register placement sandwiched between the blocks of the combinational part, to convert to a manually pipelined system. The overall combinational path of the older modules were consisting of mainly 4 steps to find the output result, such as,

a) evaluation of exponent,sign, and special results.

b) operation on input mantissa parts

c) normalization operation on the result mantissa.

d) rounding operation on the normalized but unrounded mantissa.

The multiplier and adder modules in the new cases, have separate combinational blocks to execute each of this steps, in a pipelined manner. Each of the blocks are supported by set of registers at the back and the front. Dummy registers has been placed in between two register-set, in some of the cases, to accommodate the design without any timing-violation after hardware-synthesis. The placement of dummy registers affects the overall performance of the system and the best one is mainly decided by trial and error method. Normally, the combinational blocks having larger critical path can be supported by one dummy registers at the back, to split the overall critical path while register-retiming but that is never the ultimate solution. The dummy register were placed in different position in the overall design, and the best placement has been reported for all of the modules. We also depend on the software based register-retiming process for the new modules to analyze the performance, but in this case the internal registers optimally placed in the design itself, reduces the load on the software based register-retiming process, and the optimal pipeline gets a lot way better than it was in case of older modules. We have seen that the number of dummy registers+internal pipeline registers reduces in the multiplier modules after the register-retiming with no violation, compared to the number of dummy registers in the older modules. Also the converter modules require the evaluation of sign,exponent and special results along with normalization and rounding, with no extra operation on the input mantissa parts. We have successfully accommodated the design into two clock cycle with the help reduced internal input and output delay, that also saves a lot of design area, compared to the older modules. Overall, all of the new modules with internal registers have shown the improvement of the register-retiming for the synthesis tool, due the

manually placed pipeline registers in the design.

The details of the internal pipeline stages for different modules is being discussed in the next sections.

3.1 Manual Pipeline for Multiplier Module:

Overall computation path of a floating point multiplication can be thought of a series of following operations,

(a)sign,exponent evaluation and partial product generation using **Booth's Algorithm**,

(b)evaluating actual product by shift-add of partial products, and normalize the result,

(c)rounding operation of the normalized result to obtain the final mantissa.

The input and outputs of each block is connected to a set of pipeline registers to propagate the data and dummy registers has been used at the back to avoid any negative slack in any of the input_to_flop, flop_to_flop and flop_to_output path of the design, with the use of software-based register-retiming, on the synthesis tool. In this case, the retiming load on software program becomes less and it only takes 2 dummy registers for both single and double precision module compared to 6 and 7 dummy registers(latency 7 and 8 cycles) for single and double precision versions in the previous design. The effective dummy register placement for single and double precision modules are,

32 bit multiplier module :

stage1 (dummy) > stage2 > stage3 > stage4 > stage5 > output

64 bit multiplier module :

stage1 (dummy) > stage2 > stage3 > stage4 (dummy) > stage5 > stage6 > output

Thus latency of the previous design has been improved by one cycle in both single precision and double precision modules, in the new design.

The stages are illustrated as follows,

a) Single-precision Multiplier :

Thus latency of the previous design has been improved by one cycle in both single precision **Stage 1:** Input is being taken and passed to the next register set, It is a dummy path with no computation.

Stage 2: Computational path with six parallel blocks, operating for exponent evaluation,special input checking and generation of 4 Booth partial products, Special operation flags are updated and carried throughout the later stages.

Stage 3: Computation path with normalization of the sum of Booth Partial Products, that

generates the unrounded final mantissa of 48 bits, with update of overflow and underflow flags.

Stage 4: Computation path including the rounding operation with guard,round and sticky bits, to generate final 23 bit mantissa of final result as well as updating the inexact flag.

Stage 5: Final Result of the multiplication (32 bit floating point number) is delivered to a dummy path and made ready for output register, output register delivers the final output in IEEE 754 specified standards.

b) Double-precision Multiplier :

Stage 1: Input is being taken and passed to the next register set, It is a dummy path with no computation.

Stage 2: Computational path with eleven parallel blocks, operating for exponent evaluation,special input checking and generation of 9 Booth partial products of 108 bits, Special operation flags are updated and carried throughout the later stages.

Stage 3: The Booth partial products that has been generated from previous stage, is added to get the actual product mantissa.

Stage 4: Dummy path with no computation, the actual product mantissa, exponent and other flags are carried to the next set of registers.

Stage 5: Computation path with normalization of actual product, that generates the unrounded final mantissa of 108 bits, with update of overflow and underflow flags. The generated exponent is carried to the next stage.

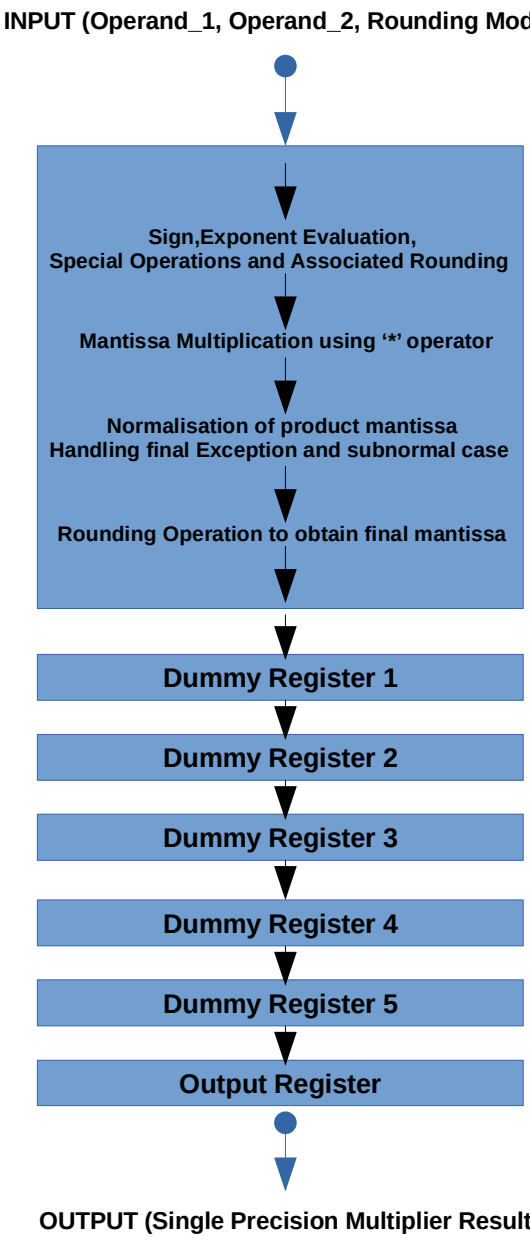
Stage 6: Computation path including the rounding operation with guard,round and sticky bits, to generate final 23 bit mantissa of final result as well as updating the inexact flag. Final Result of the multiplication (64 bit floating point number) is delivered to a dummy path and made ready for output register, output register delivers the final output in IEEE 754 specified standards.

Performance analysis of all of the multiplier modules has been done under the input delay of 60% and the output delay of 20% of the clock period (600 ps). The parallel segmentation of MSB zero counting algorithm and the modified Booth's multiplication algorithm has helped to shorten the critical path of the design,and the latency has been improved by 1 cycle over the existing design.

PIPELINE COMPARISON OF MULTIPLIER MODULES:

OLD MULTIPLIER MODULES

Single Precision Multiplier Module:
(Stage :6 ;Latency:7 ;I.D:60% ;O.D:20%)



Double Precision Multiplier Module:
(Stage :7 ;Latency:8 ;I.D:60% ;O.D:20%)

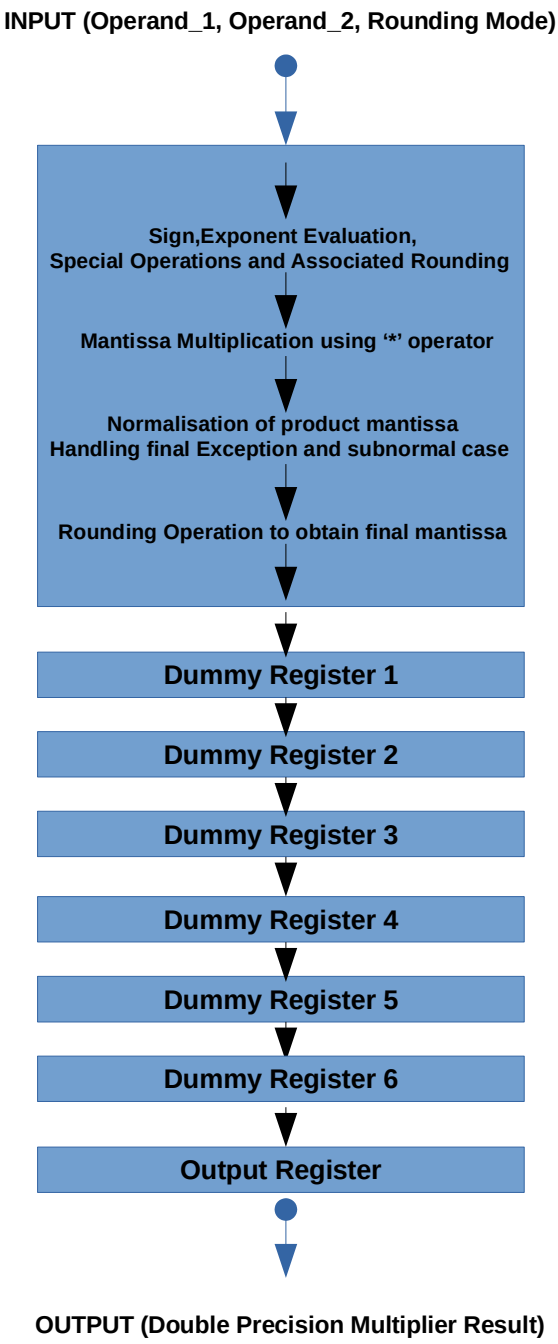
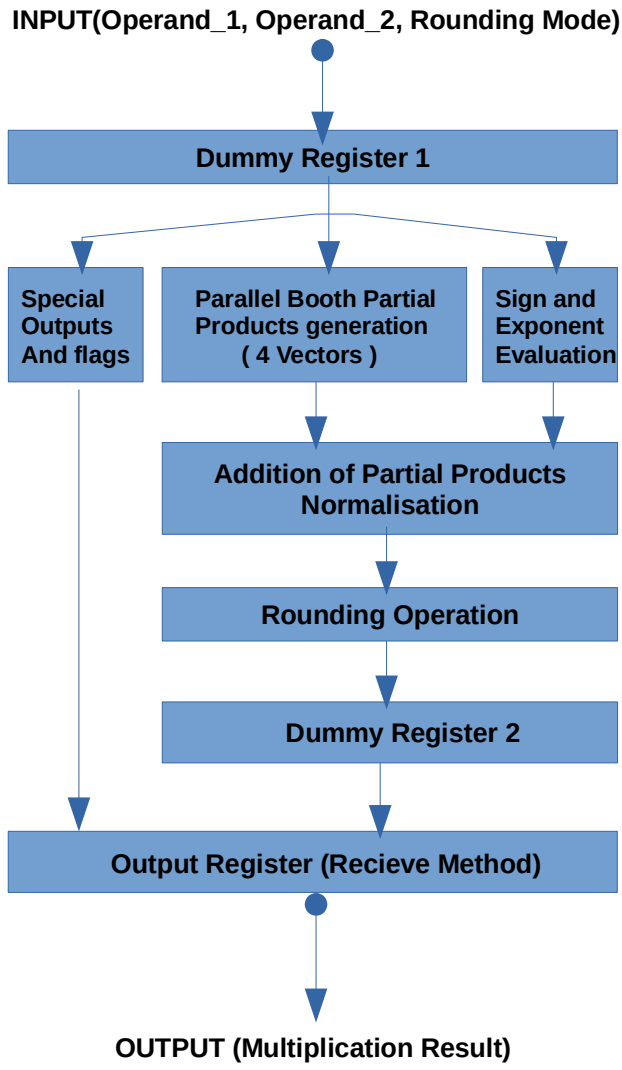


Fig 4 : Pipeline Diagram of Old Multiplier Modules

NEW MULTIPLIER MODULES:

Single Precision Multiplier Module:
(Stage :5 ;Latency:6 ;I.D:60% ;O.D:20%)



Double Precision Multiplier Module:
(Stage :6 ;Latency:7 ;I.D:60% ;O.D:20%)

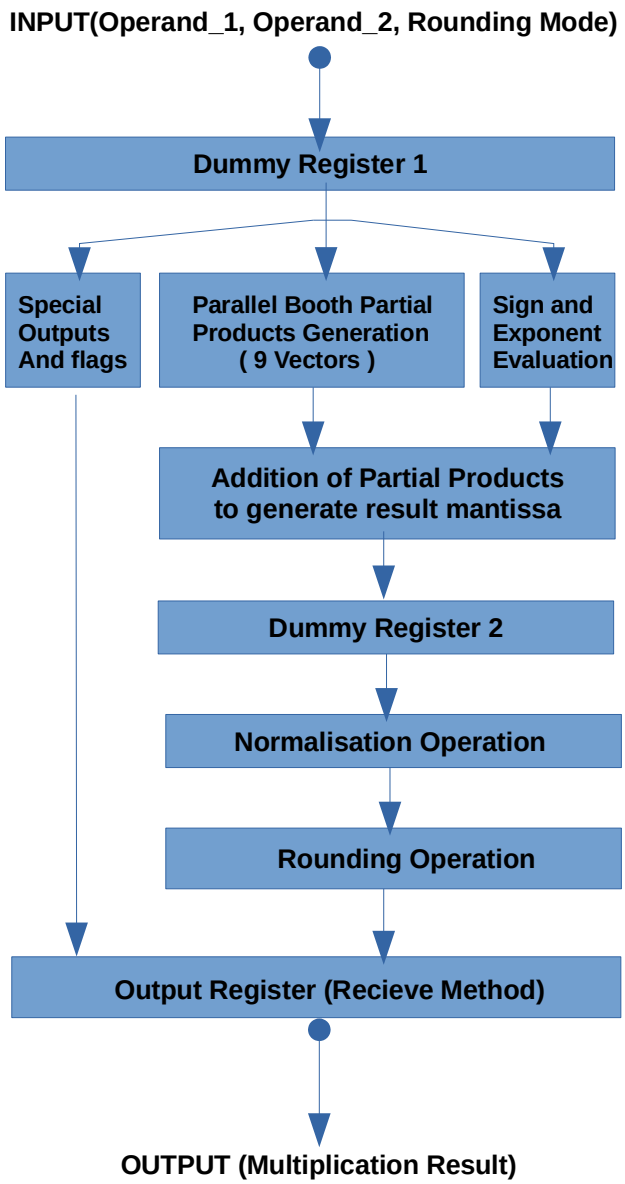


Fig 4 : Pipeline Diagram of New Multiplier Modules

3.2 Pipeline for Converter Modules:

shorten the critical path of the design,and the latency has been improved by 1 cycle over the Single to double precision converter,double to single precision converter,integer to single and double precision converters have been optimized to reduce their latency from the previous design. The previous converter design supporting all of the standard floating point formats, with latency =4 cycle, requires 3 dummy registers to have zero or positive slack in the critical path, with software-based register-retiming. The new converter modules has been designed for dedicated 32 bit and 64 bit versions and the operation have been eventually sped up due to elimination of the version supporting all the standard formats. The modification of MSB zero counting unit has helped to improve the latency of the converter modules. The computation path has been segmented and set of internal pipeline registers have been placed between the segments in the new design, thus reducing the number of dummy registers leading to less burden on register-retiming. The single to double precision conversion operation is mainly zero-extending the LSB part of mantissa along with exponent calculation. This has very less size of critical path and enough to accommodate itself in one clock cycle. The all other conversions consists of exponent evaluation,exception generation and rounding the mantissa, they have been accommodated to have 2 cycles of latency for each module in the new design.

Illustration of various stages in the floating point converter modules :

a) Single precision to double precision converter :

Single to double precision conversion of floating point numbers includes the bias modification of exponents and LSB zero extension of mantissa. These two operations have short critical path and enough to be accommodated in one clock cycle, along with the output ready in the output register.

b) Double precision to single precision converter :

Double to single precision conversion of floating point numbers includes the bias modification of exponents, update of overflow and underflow flags, and rounding operation of 52 bit mantissa. To obtain a 23 bit single precision mantissa. These operations together do not have a short critical path to accommodate in one cycle, so all of the computation operation is built in 1st stage and that delivers the final result to the output register in the next clock cycle.

c) Integer to floating point converter :

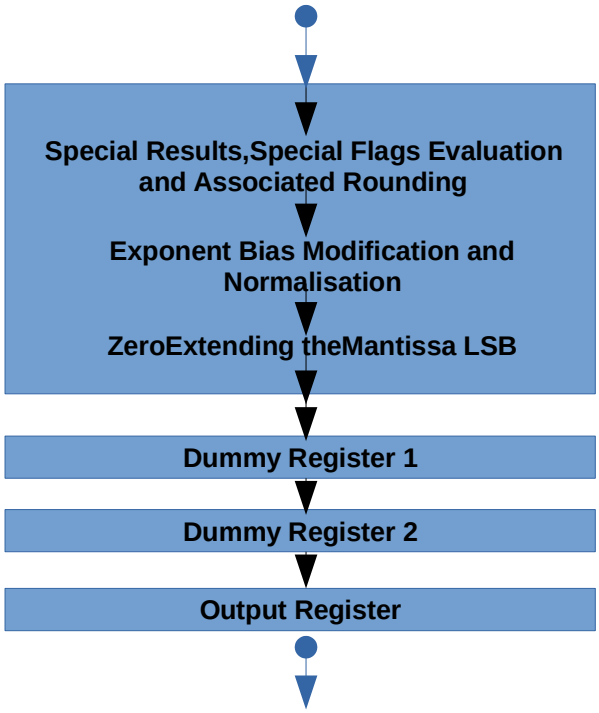
Integer to single and double precision conversion of floating point numbers starts with the MSB zero counting of the short or long,signed or unsigned integer input, thus the exponent and mantissa part of the floating point output is generated at the first stage. The MSB zero counting algorithm has been modified from previous design with the implementation of parallel operating segments, to make the operations faster. The exponent and unrounded mantissa is generated by decimal point shifting from the integer input and the final floating point result is generated after rounding operation of the unrounded mantissa, in the 1st stage. Then this result is being stored in the output register in the next clock cycle. The rounding operation converts the unrounded mantissa to the 23 bit mantissa for integer to S.P conversion and to 52 bit mantissa for D.P conversion process. Unrounded mantissa is zero extended on LSB, where target floating point mantissa size is greater than the integer width.

Performance analysis of all of the converter modules has been done under input and output delay of 20% of clock period, for both. The parallel segmentation of MSB zero counting algorithm has helped to eliminate the generation of long chain in case of higher size inputs, in case of Integer to floating point converters, thus improved the performance in a great way.

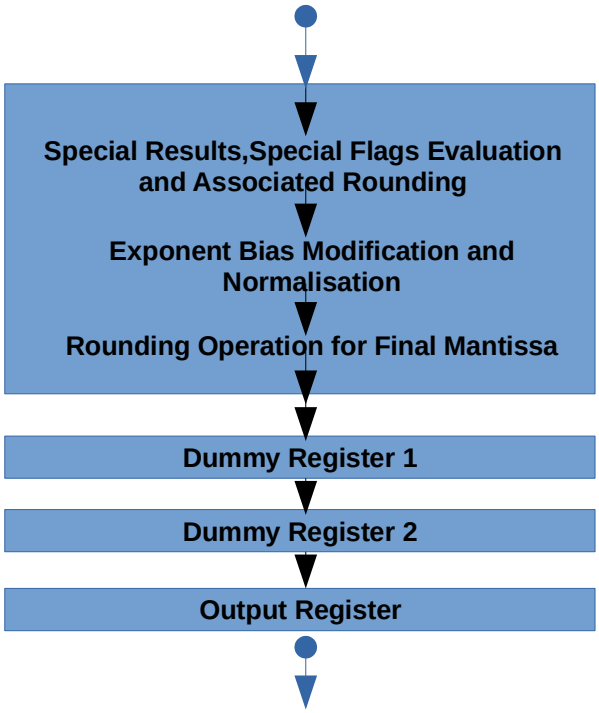
2.PIPELINE COMPARISON OF CONVERTER MODULES:

OLD CONVERTER MODULES:

S.P to D.P Converter Module :
(Stage : 3; Latency : 4;I.D:60% ;O.D:20%)



D.P to S.P Converter Module :
(Stage : 3 ; Latency : 4;I.D:60% ;O.D:20%)



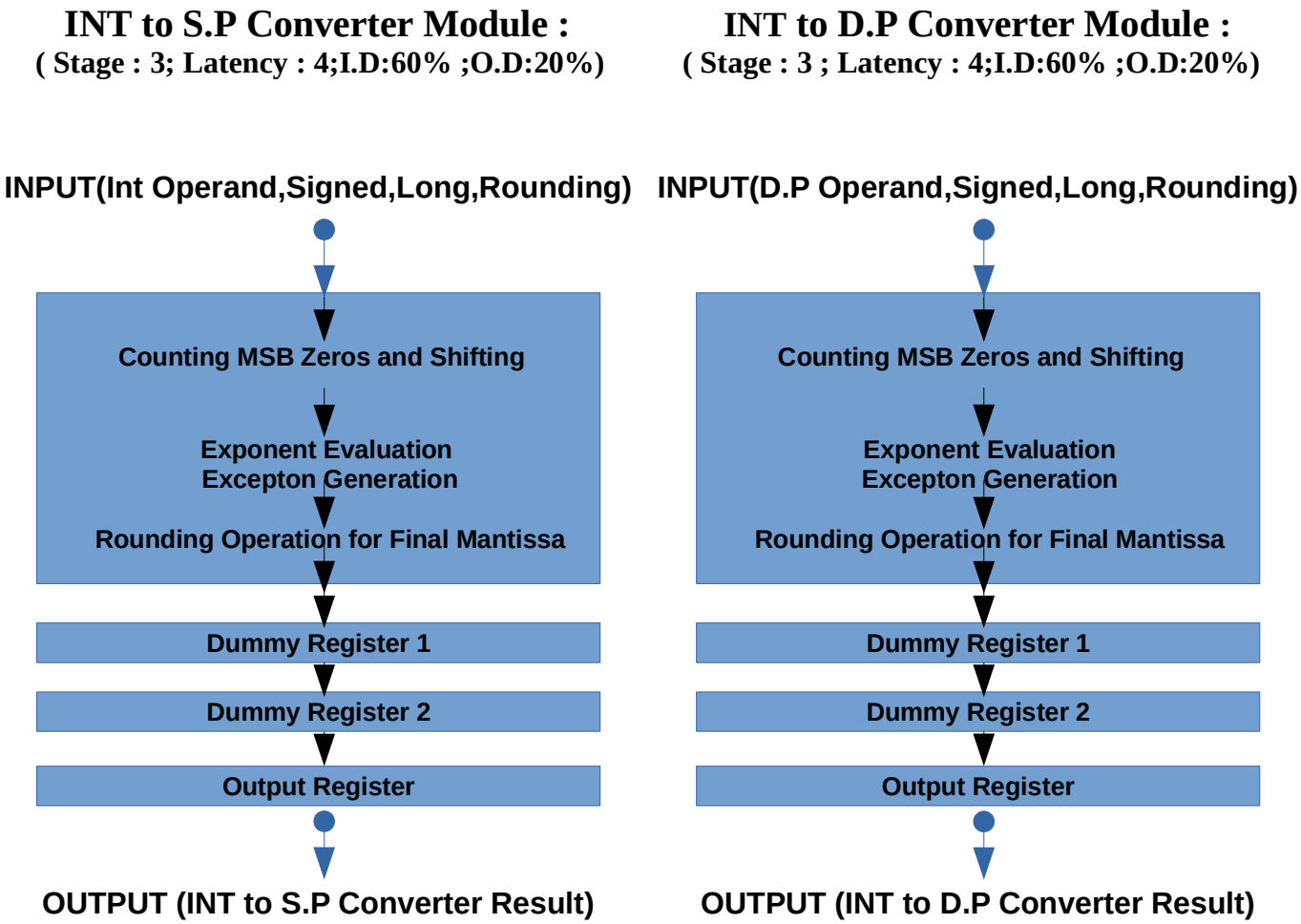
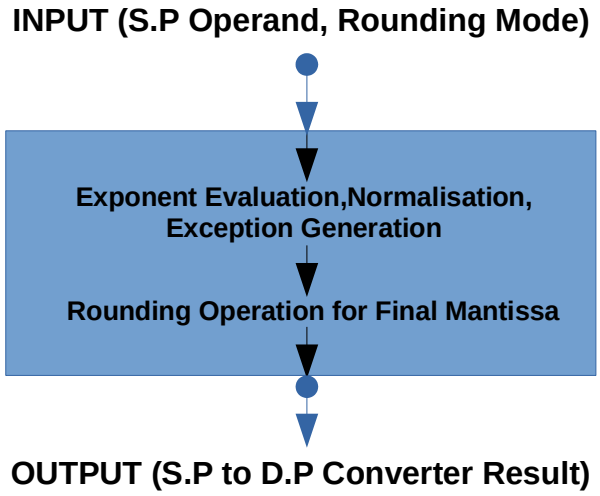


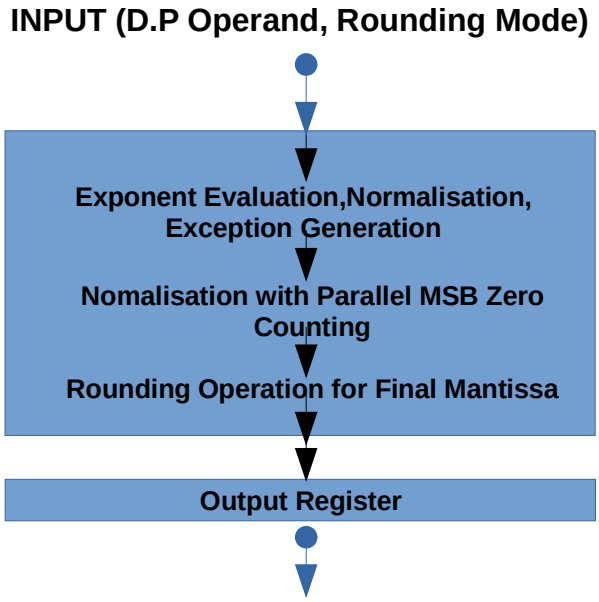
Fig 5 : Pipeline Diagram of Old Converter Modules

NEW CONVERTER MODULES:

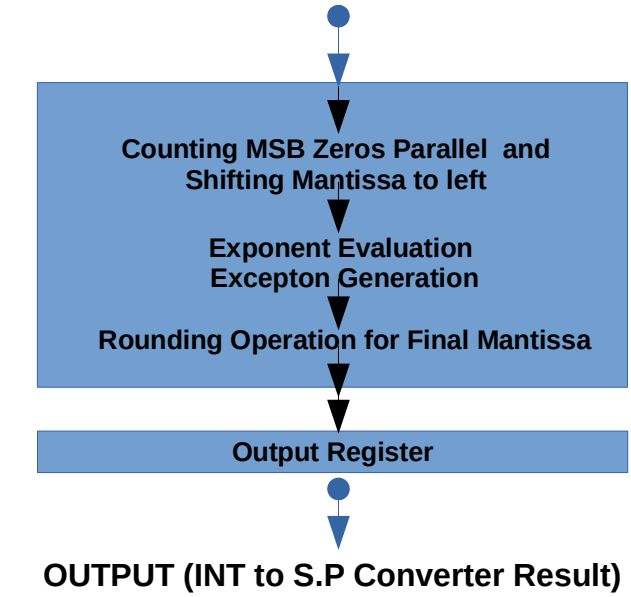
S.P to D.P Converter Module :
(Stage: 0; Latency: 1; I.D: 20%; O.D: 20%)



D.P to S.P Converter Module :
(Stage: 1; Latency: 2; I.D: 20%; O.D: 20%)



Int to S.P Converter Module :
(Stage: 1; Latency: 2; I.D: 20%; O.D: 20%)



Int to D.P Converter Module :
(Stage: 1; Latency: 2; I.D: 20%; O.D: 20%)

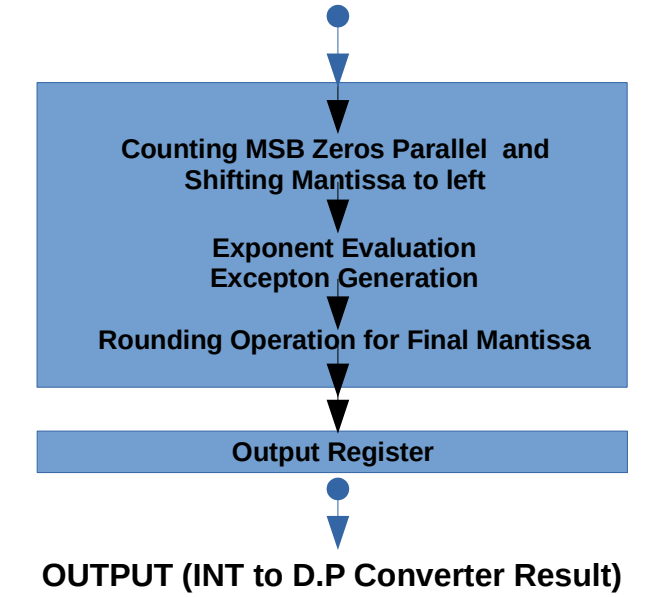


Fig 6 : Pipeline Diagram of New Converter Modules

3.3 Manual Pipeline for Adder Module:

Overall computation path of a floating point multiplication can be thought of a series of following operations,

(a)sign,exponent and exponent difference evaluation,shifting the mantissa

(b)partial sum generation,evaluating actual sum by adding partial sums with Carry-Lookahead-Adder and normalization operation on the result,

(c) rounding operation on extended mantissa to obtain final mantissa.

Similar to the multiplier module, data propagation between each of this computational blocks are supported by internal pipeline registers placed between each pair of blocks. The dummy registers are added in the output side of critical path,and register-retiming was performed on this design and the number of dummy registers required, is found out to be reduced to 1 for single precision adder, and 2 for double precision adders, compared to the numbers 4 and 5 respectively, in case of previous design. The effective placement of dummy registers for new adder module is as follows,

32 bit adder module :

stage1 (dummy) > stage2 > stage3 > stage4 > stage5 > output

64 bit adder module :

stage1 (dummy) > stage2 > stage3 > stage4 (dummy) > stage5 > stage6 > output

The stages are illustrated as follows:

a) Single-precision Adder :

Stage 1: Input is being taken inside in this stage, exponent differences are calculated,special inputs are identified, and mantissa of one operand is shifted right according to the exponent difference.

Stage 2: Dummy path with no computation, transferring the results of stage 1 to the stage 3.

Stage 3: Computation path with modified Carry Lookahead adder-subtractor for actual sum generation, (the unrounded final mantissa of 28 bits), normalization of the result using the segmented counting MSB zero algorithm, along with the update of overflow and underflow flags. Addition or subtraction operation is triggered by the XOR of the sign bits (0 for addition, 1 for subtraction) of two operands.

Stage 4: Computation path including the rounding operation with guard,round and sticky

bits, to generate final 23 bit mantissa of final result as well as updating the inexact flag. Final Result of the addition (32 bit floating point number) is transferred to the output register, output register delivers the final output in IEEE 754 specified standards.

b) Double-precision Adder :

Stage 1: Input is being taken and transferred to the next register set, it is a dummy stage with no computation.

Stage 2: Computation path with two parallel blocks, computing sign,exponent differences and right shifting of mantissa in one block, while determining the special inputs and updating the special flags with the other block.

Stage 3: Computation path with modified Carry Lookahead adder-subtractor for actual sum generation, (the unrounded final mantissa of 57 bits). Addition or subtraction operation is triggered by the XOR of the sign bits (0 for addition, 1 for subtraction) of two operands.

Stage 4: Computation path for normalization of the result mantissa and updating the overflow and underflow flags.

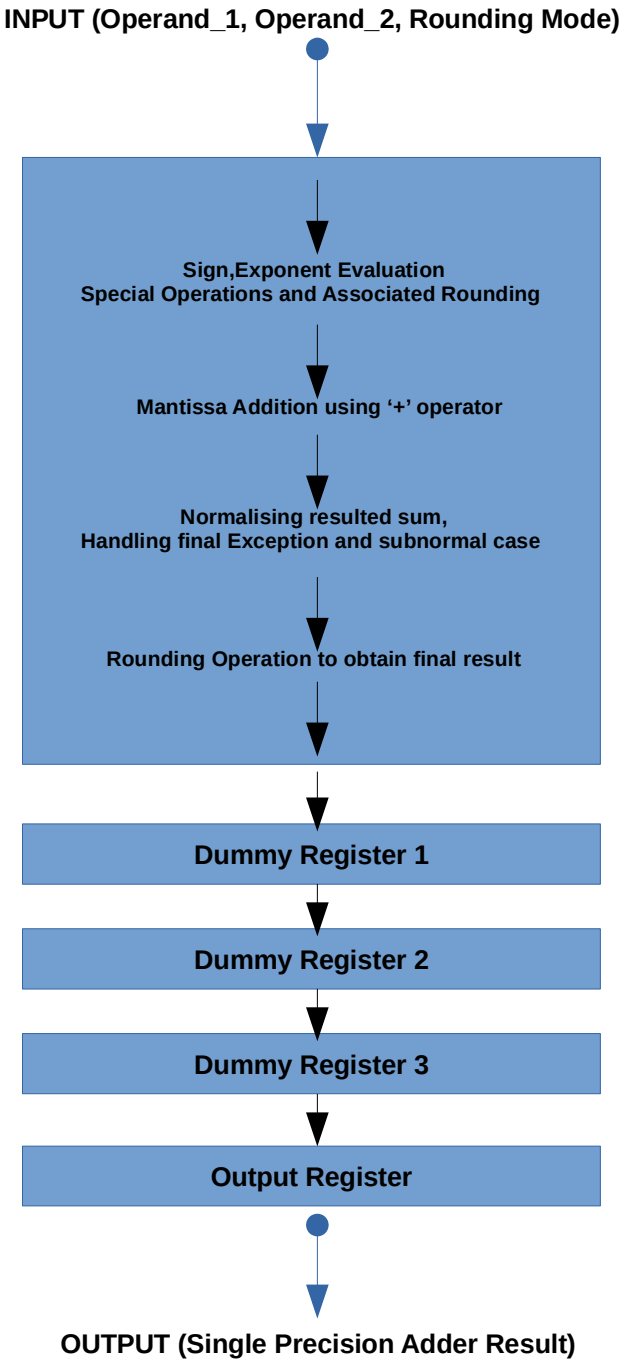
Stage 5: Computation path including the rounding operation with guard,round and sticky bits according to IEEE specification, to generate final 23 bit mantissa of final result as well as updating the inexact flag. Final Result of the addition (32 bit floating point number) is transferred to the output register, output register delivers the final output in IEEE 754 specified standards.

Performance analysis of all of the adder modules has been done under the input delay of 60% and the output delay of 20% of the clock period (600 ps). The parallel segmentation of MSB zero counting algorithm and the modified CLA adder has helped to shorten the critical path of the design,and all the slack got improvement over the existing design.

3. PIPELINE COMPARISON OF ADDER MODULES:

OLD ADDER MODULES:

Single Precision Adder Module:
(Stage :4 ;Latency:5 ;I.D:60% ;O.D:20%)



Double Precision Adder Module:
(Stage :5 ;Latency :6 ;I.D: 60%; O.D:20%)

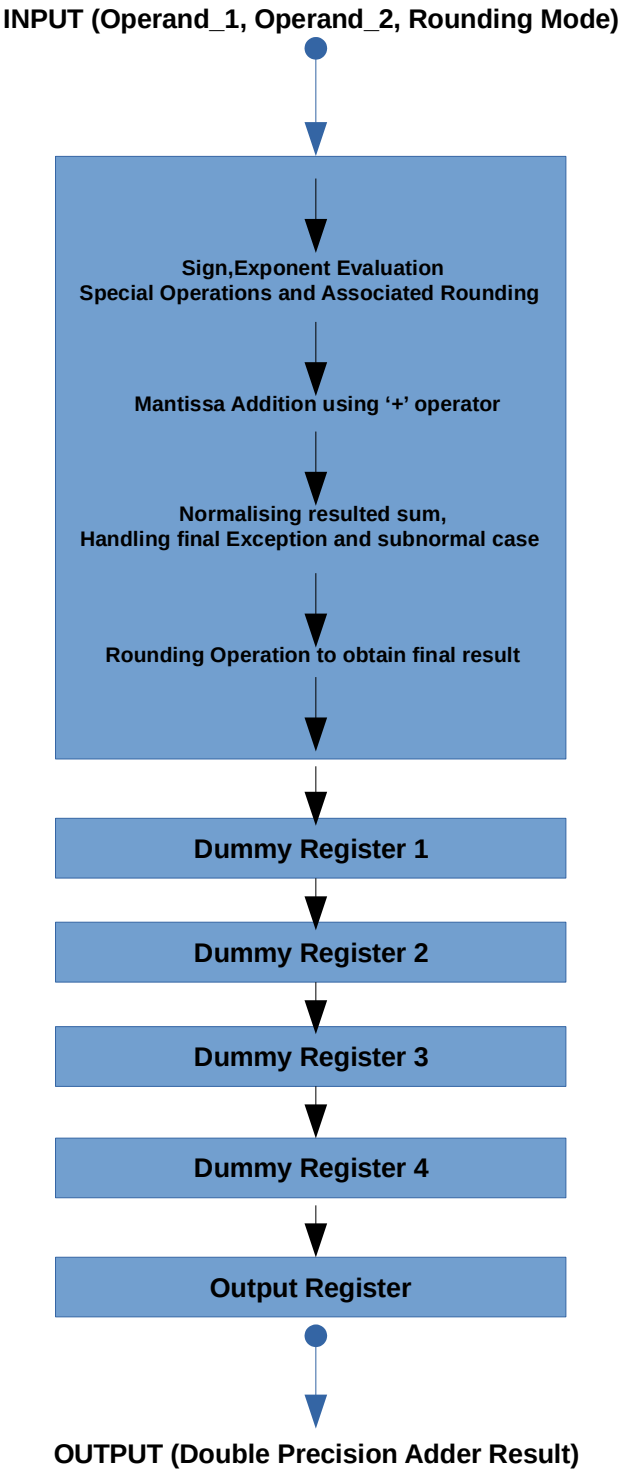


Fig 7 : Pipeline Diagram of Old Adder Modules

NEW ADDER MODULES:

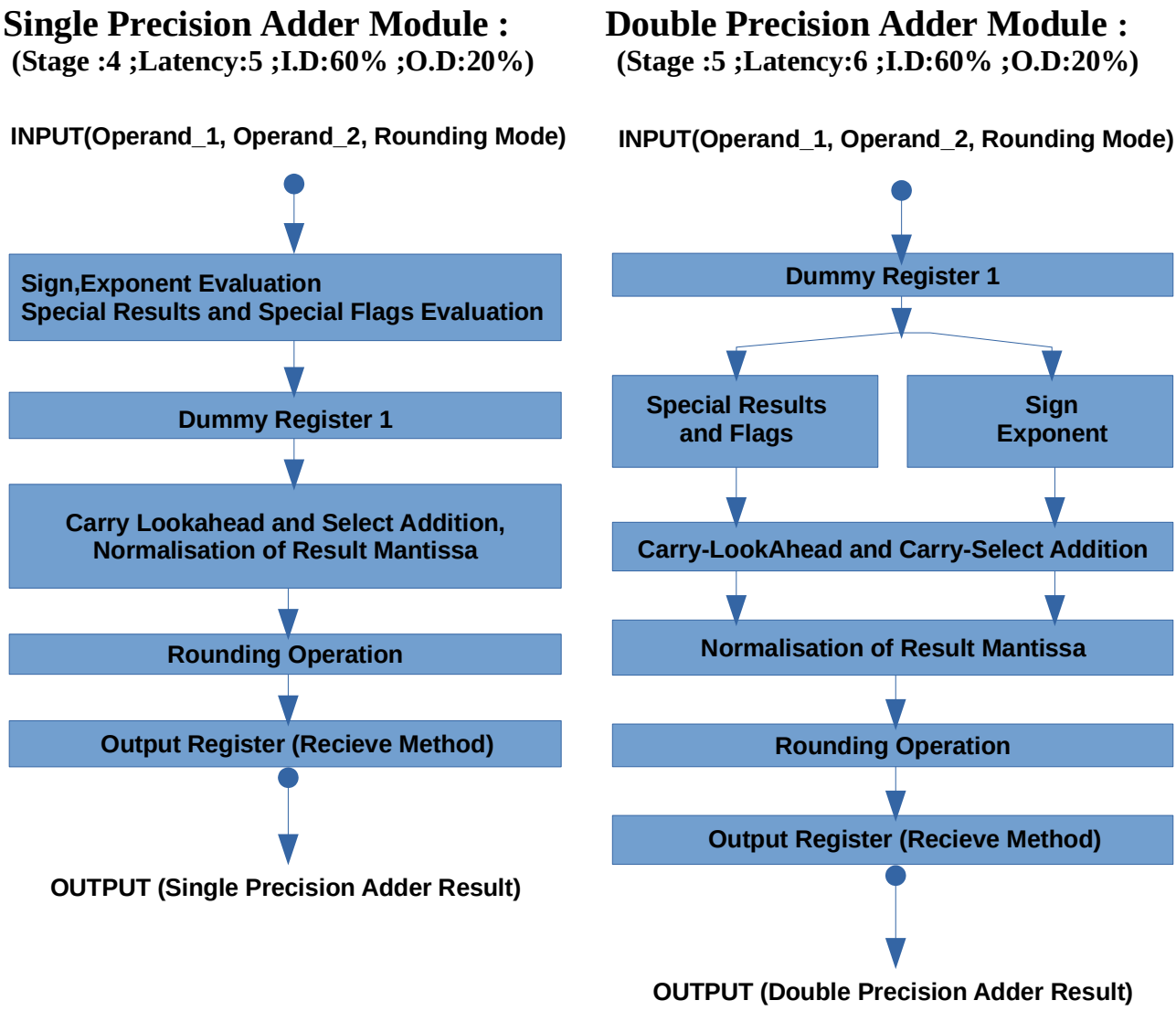


Fig 8 : Pipeline Diagram of New Adder Modules

CHAPTER 4: TESTING AND PERFORMANCE ANALYSIS

Each of the newly designed floating point units has been successfully tested for Five Million values generated by TestFloat, for each of the 5 available rounding modes in SHAKTI class of micro-processors, i.e, Nearest_Even, Nearest_Away_Zero, Plus_Infinity, Minus_Infinity and Towards_Zero. The performance analysis and register-retiming has been done based on 600 ps. clock period and 22 nm. , ln,0.75V FinFET technology, with Design-Vision software by Synopsys. The performance of Multiplier and Adder modules are analyzed with external Input-Delay-Ratio of 0.6 and Output-Delay_Ratio of 0.2, where all of the converter modules works on Delay-ratio of 0.2 clock period, same for both input and output. The values related to different slack and area, are reported under most effective placement of internal dummy registers inside the modules, as discussed in the previous sections.

Table 1 :

PERFORMANCE COMPARISON OF OLD AND NEW F.P MODULES:

MAIN_CLOCK_PERIOD : 600 ps ; Technology: 22 nm, ln ,0.75V

Modules	New Modules Performance					Old Modules Performance				
	I-to-F Slack	F-to-F Slack	F-to-O Slack	Latency	Area	I-to-F Slack	F-to-F Slack	F-to-O Slack	Latency	Area
Single prec. Multiplier	0.06	0.02	5.10	6	3865.42	0.99	-54.23	217.21	7	3075.34
Double prec. Multiplier	0.11	0.00	0.12	7	13594.80	2.880	-124.8	133.90	8	9370.33
S.P to D.P Converter	0.24	-	-	1	217.73	0.06	-	2.32	2	282.64
D.P to S.P Converter	0.34	-	0.02	2	437.10	-134.6	-	-55.58	2	1132.32
INT to S.P Converter	-38.43	-	-74.38	2	1387.47	-90.55	-20.15	0.27	3	1546.96
INT to D.P Converter	-51.73	-	-47.17	2	1349.37	-115.3	-51.73	0.04	3	1665.62
Single precision Adder	-11.63	0.05	5.61	5	1676.23	-168.6	0.10	166.19	5	1369.16
Double precision Adder	-6.23	0.14	0.68	6	3633.61	-16.85	0.42	0.81	6	3346.07

CHAPTER 5

CONCLUSION

The new floating-point modules for C-Class and I-Class core of SHAKTI RISC-V processors, has been re-designed for dedicated 32 bit and 64 bit IEEE 754 formats to improve their performance. Removal of parameterized bit width scheme from the existing version, helped to optimize some of the algorithms (counting MSB zero, multiplication, addition) with the implementation of parallel operating segments, to reduce the overall critical path, compared to the long critical path due to the unfolding of inter-dependent “for loops” in some operations with the existing designs, as the parallel segmentation is not possible without the prior knowledge of input size. Manual pipeline has helped to reduce the burden of software-based register-retiming, thus the software based optimization of the modules, were more effective and as a result, all of the modules have shown significant improvement towards the reduction of their operating time. The design performance of the new modules is highly sensitive to the placement of internal registers/flops because of manual pipeline, and the modules are constructed with most effective placements of dummy registers along with the internal register-set. Optimized algorithms with fixed bit-width of the operands in the new modules also have shown a reduction of overall design area along with the reduction in their latency, compared to the existing counter-parts in the cases of floating-point converter and adder modules, proving the better cost-effectiveness compared to the previous parameterized bit-width based design with same size of inputs.

The combinational pattern in the existing modules has been changed to fully pipelined block based design, to improve the overall performance in the new floating point modules. The algorithms for multiplier and adder modules have been replaced with their faster counterparts, using segmentation for parallel operation where necessary, to reduce their overall critical path. The dedicated 32 and 64 bit floating point converter modules only depends on the exponent evaluation and mantissa-rounding, these two operations are carefully accommodated in two clock cycles for the new modules, with the exclusion of unnecessary logic generated from the parameterized input size in the previous modules, helping towards the reduction of both latency and area. The exception generation part has

been separated from the main computation path for most of the modules, that reduces the overall operating time with the parallel execution of some part of the internal logic. The rounding operation has also been modified with the sticky bits according to the IEEE 754 rounding specification, to improve the accuracy of the result.

There is a lot of scope in future improvement for latency and area optimization with the new floating point single and double precision multiplier, floating-point to floating-point converter, and integer to floating-point converters and floating-point adder-subtractor modules. For example, Booth's partial product generation for the multiplier module was carried out by shift and add process of 6 vectors of the size of mantissa, and that incorporates the use of RCA Adder for addition, and can be replaced by Wallace or Dada multiplication algorithms, to make the operation faster. The optimized floating point single-to-double and double-to-single precision converter modules are very compact with less area and less latency, compared to the previous versions, and they can be effectively used along with the multiplier or adder modules, and therefore eliminating the need for separate versions for both single and double precision operands. This will have a great impact towards the area and cost optimization of the SHAKTI floating point cores. Overall, the new floating point modules with the optimized algorithm for dedicated input size and manual pipeline, shows an effective way for better performance optimization with less burden on software-based register-retiming process, and can be employed for any other SHAKTI Modules according to their optimization requirement.

REFERENCES

1 **E-Class repository : https://gitlab.com/shaktiproject/cores/e-class**

2 **C-Class repository : https://gitlab.com/shaktiproject/cores/c-class**

3 **FBox repository :** https://gitlab.com/shaktiproject/cores/fbox/

4 **Rounding Specification for IEEE 754 specified standards.**

CURRICULUM VITAE

2.1 NAME : Joyanta Mondal

2.2 DATE OF BIRTH : 15 Sep 1994

2.3 EDUCATIONAL QUALIFICATIONS

Bachelor of Technology (B. Tech.)

2016 Institution : Indian Institute of Engineering,Science and Technology, Shibpur
Specialization : Electrical Engineering

Master of Technology (M. Tech.)

Institution :Indian Institute of Technology Madras

Specialization :Electrical Engineering

Submission Date : 28 June 2021

COMMITTEE

CHAIRPERSON : Dr. David Kolipillai

Professor and Head

Department of Electrical Engineering

GUIDE(S) : Dr. Kamakoti V

Professor

Department of Computer Science and Engineering, IIT, Madras

Dr. Nitya Ranganathan

Department of Computer Science and Engineering, IIT, Madras

Dr. Boby George

Professor

Department of Electrical Engineering, IIT, Madras

MEMBERS : Dr. Kamakoti V

Professor

Department of Computer Science and Engineering, IIT, Madras

Dr. Ayon Chakrabarty

Professor

Department of Computer Science and Engineering, IIT, Madras