FPGA IMPLEMENTATION OF FLOATING POINT ADDER AND MULTIPLIER UNDER ROUND TO NEAREST

SAKTHIVEL
Assistant Professor, Department of ECE,
Coimbatore Institute of Engineering and Technology

RATHI.B
PG Scholar, M.E –VLSI Design, Department of ECE,
Coimbatore Institute of Engineering and Technology

Abstract- FPGA is increasingly being used to design high-end computationally intense microprocessors capable of handling both fixed and floating point mathematical operations. Addition and multipliers is the most complex operation in a floating point unit and also it offers a major delay while taking significant area. Over the years VLSI has developed many floating point adder and multiplier algorithm mainly aimed to reduce the overall latency. An efficient design of floating point adder and multiplier on to an FPGA offers major area and performance overheads with the recent advancements in FPGA architecture area, density and speed has been the main focus of attention in order to improve performance. This paper analysis the benefits of using pipeline format to implement floating point (FP) arithmetic under a round to nearest mode from a quantitative point of view. Using the pipelining format to represent numbers allows the removal of rounding logic of arithmetic units, including sticky bit computation. This is shown for FP adder and multiplier. Experimental analysis demonstrates that the pipeline format and the corresponding arithmetic units maintain the same accuracy as the conventional ones. On the other hand, the implementation of these units, based on the basic architectures, shows that the pipeline format simultaneously improve area and delay.

***Key Points:*, Field Programmable Gate Array (FPGA), Floating Point (FP), Pipelining**

I.INTRODUCTION

The rounding operation is performed in almost all arithmetic operations involving real numbers. There are several ways to perform this operation, although unbiased rounding-to-nearest (RN) has the best characteristics. It provides the closest possible number to the original exact value, but if the exact value is exactly halfway between two numbers, then it is selected randomly.

The most commonly used approach is the tie-to-even method, which is the default mode of the floating-point (FP) IEEE-754 standard. However, the implementation of this rounding mode is relatively complex, and the area and the delay introduced for rounding circuits may be very large, since they normally lead in the critical path. For this reason, it is generally used in the FP circuits. Many researchers have proposed different architectures to reduce the impact of this delay by merging rounding with other operations or removing it from the critical path. For instance, an FP adder was proposed in, wherein if the result of an addition is the input to another one, the addition required for rounding up is postponed until the next operation. In, a dedicated circuit to compute the sticky bit in parallel with the main path was proposed with the aim of accelerating the implementation of multiplication.

A compound adder (a circuit that, having a carry-save input, delivers the results and the result plus one) was proposed in to generate the rounded result of any operation. In three different methods were compared for multipliers that simplify rounding decisions and merge the rounding up with the computation of the operation. Similarly, Burgess proposed combining rounding with the final addition to convert the carry-save solution into the conventional representation. In, a rounding scheme was presented for high-speed multipliers based on a rounding table and prediction. A totally different approach would be to use a new real-number encoding, in order to simplify the implementation of RN. Thus, the problem would change from optimizing the rounding operation to dealing with arithmetic operations under the new number representation. This proposal is found in with RN representations and with half-unit-biased (HUB) using pipelining formats.

Together with other advantages, these new formats allow performing RN simply by truncation. On the other hand, these new formats are based on the simple modifications of the conventional formats and so could be applied to practically any conventional format. In this paper, we focus on the HUB using PIPELINING FP formats. The efficiency of using the HUB formats for a fixed point representation has been demonstrated in and. By reducing the bitwidth while maintaining the same accuracy, the area cost and delay of finite impulse response filter implementations have been dramatically reduced. In this paper, we perform a quantitative estimation of the benefit obtained using the PIPELINING formats to implement the FP computation systems under RN. Some preliminary results for half-precision FP adders and multipliers were presented. The work in shows that the area and power consumption of a basic FP adder could be improved by up to

70% for high frequencies when using HUB formats, whereas they remain the same for the basic FP multiplier. We provide an experimental error analysis to confirm the viability of using the HUB formats instead of classic ones, and also compare the results of the application-specific integrated circuit (ASIC) implementation of a basic adder, a basic multiplier.

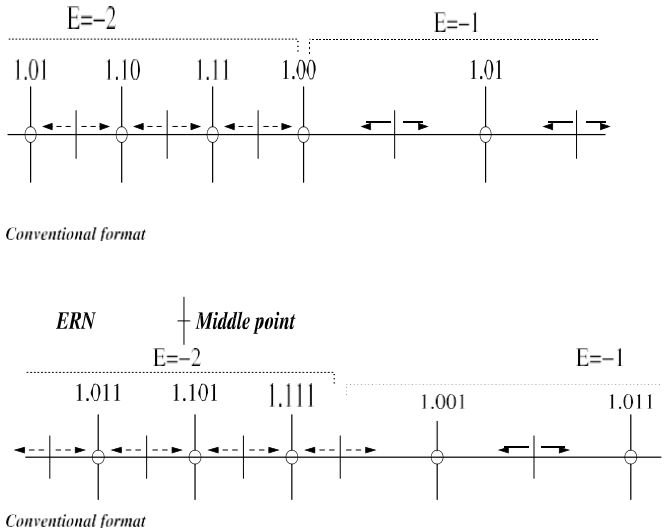

Fig. 1. Example of ERNs for a conventional FP format and its HUB version.

II. BASIC OPERATIONS UNDER HUB FORMATS USING PIPELINING

The general procedure to operate with the HUB numbers involves the following steps. First, the ILSB is explicitly appended to the significand of input operands. Second, the operation is performed in a classic way, such that all the bits of the significand result before rounding are obtained. Finally, the significand is rounded simply by truncation. However, since the ILSB is a constant value, the datapath could be further optimized depending on the specific operation. Next, we develop several architectures to support the HUB numbers. These architectures are adapted from the basic architectures. We are aware that many optimizations to these architectures have been proposed in the literature. However, none can be selected as the best, since this selection depends on many different factors. Even if the more relevant ones were selected, it would not be possible to review all of them in this paper. Thus, we simply describe the adaptation of these basic architectures with the aim of their being used as examples to investigate the implementation of other much more sophisticated approaches.

Next, we study in detail the architectures to implement two basic operations, addition and multiplication, and different conversions. For this purpose, let us consider an *m*-bit significand including the leading one, which is explicitly represented to facilitate the explanation. In contrast, the ILSB of the HUB version is not included in *m*. We focus on the implementation of the significand path, because the exponent path remains practically unchanged. We will draw the attention to any modification required.

A pipeline can be defined as a set of data processing elements connected in series, so that the output of one element is the input of the next one. In most of the cases we create a pipeline by dividing a complex operation into simpler operations. We can also say that instead of taking a bulk thing and processing it at once, we break it into smaller pieces and process it one after another.

In microprocessors for executing an instruction there are many intermediate stages like getting instruction from memory, decode the instruction, get any other required data from memory, process the data and finally write the result back to memory. Without a pipeline a single instruction has to fully go through all these stages before the next instruction is fetched from the memory. But if we apply the concept of pipelining in this case, when an instruction is fetched from memory, the previous instruction must have already decoded.

The pipeline design technique decomposes a sequential process into several sub processes, called stages or segments. A stage will perform a particular function and produces an intermediate result. It consists of an input latch, also called a register or buffer, followed by a processing circuit. (A processing circuit can be a combinational or sequential circuit.) The processing circuit of a given stage is connected to the input latch of the next stage. A clock signal is connected to each input latch. At each clock pulse, every stage transfers its intermediate result to the input latch of the next stage. In this way, the final result is produced after the input data have passed through the entire pipeline, completing one stage per clock pulse.

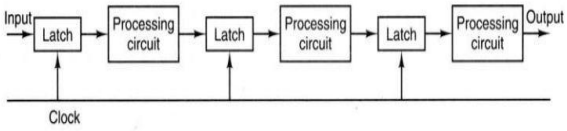

Fig. 2. Standard Pipeline Structure

A. FP Addition for HUB Numbers using pipelining format

A basic FP addition for conventional numbers requires several steps, which could be implemented using the significand datapath . First, the operand exponents (d = Ex − Ey) are compared and the significands are aligned

accordingly. The latter is usually performed by right shifting (|d| bits) the significand corresponding to the number with the lowest exponent, which is selected using the swap module and the sign of d.This is done under stage 1 of the pipelining concept. The computation of the sticky bit corresponding to the bits shifted beyond the precision of the significand is also performed.

The sticky bit is required for the computation of two's complement and rounding. Second, either the effective addition or the subtraction of the aligned significands is performed for the m + 3 MSBs (the significand plus guard, round, and sticky bits). In general, in order to perform subtraction, the significand corresponding to the lower exponent is previously one's complemented using the significand comparator and the conditional bit inverters.

Moreover, the sticky bit is introduced in the adder to complete the two's complement transformation. The result of addition has to be normalized by shifting 1 bit to the right, if an overflow is produced. Otherwise, it is shifted to the left if there are leading zeros whose number is computed in the leading one detector.

Besides normalization, the result has to be rounded based on the two LSBs of the result and the sticky bit. However, no roundup is required when the result has at least two leading zeros. Thus, left shifting and rounding are performed in parallel paths. On the other hand, the new exponent is also generated in a parallel path. If the result of addition is rounded up, an overflow may be produced, which requires a new correction of the exponent. The same basic architecture could be used for the HUB numbers, although the significands are 1 bit larger and the rounding circuit is removed. However, knowing that the ILSB always equals 1, the significand datapath is further optimized.

The first difference is in the right shifter used to align the significands, because the ILSB has to be included at the input to obtain a correct result if no shifting is performed. Furthermore, the sticky-bit computation logic has been removed. Given that the ILSBs of both the significands equals 1, the sticky bit is always one for nonaligned significands. Moreover, the sticky bit is not required for aligned significands because shifting is not performed. In this HUB architecture, we should note that the conditional bit inverters directly perform the two's complement, as explained in Section II, and no carry input is required in the fixed-point adder.

The conditional inverter at the output of the right shifter has to be modified to control when shifting is not performed. In this case, the ILSB is explicitly represented by the LSB of the output. It then has to be set to 1 after the inversion to complete the two's complement operation (see Section II). On the other hand, shows that the ILSB of the second operand is appended at the corresponding input of the fixed-point adder. Despite this, the fixed-point adder is slightly shorter than the one shown.

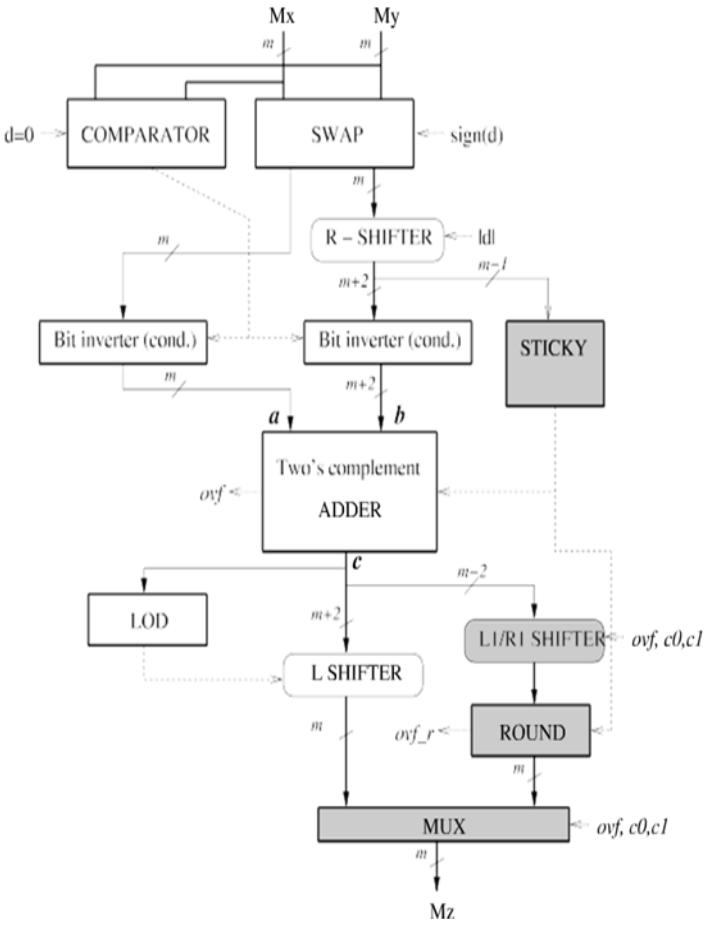

Fig.3. Basic FP adder architecture

The latter requires two guard bits and the carry input for the sticky bit (i.e., m+2 bits) dueto the rounding operation. However, in the HUB approach. No guard bits are required because rounding is performed by truncation. Thus, the fixed point adder has only m+1 bits (one additional bit to support the ILSB). In fact, the ILSB is shown in the architecture to simplify the explanation, although, as presented. This fixed-point addition can be implemented using an m-bit adder and an inverter.

Finally, the rounding path is removed, because rounding is simply performed by truncation. This is done under 2nd stage of the pipelining concept. Consequently, given that explicit rounding up is not performed for the HUB architecture, overflow could not occur after rounding. Thus, the additional correction of the exponent required is eliminated, which also simplifies the exponent datapath.

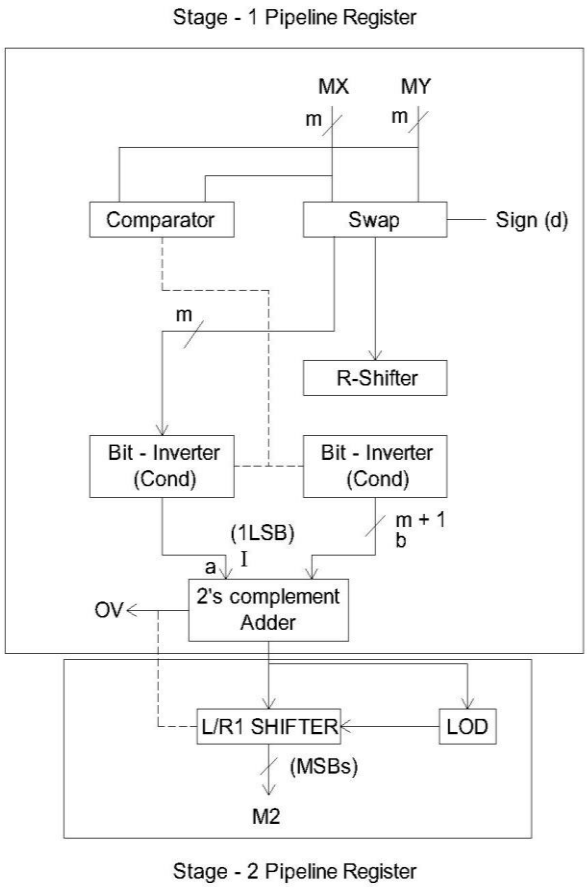

Fig.4. FP Adder using pipelining register

B. FP Multiplication for HUB Numbers using pipelining register

A classic FP multiplication is simpler than FP addition. In this multiplier format, three stage pipelining are used to implement the architecture.

For the first stage of pipelining multiplier, the new exponent is computed by adding the exponents of the input operands, whereas the significands are multiplied and the exponents are added to rounding the value, thus obtaining a value that is double the size. In the second stage of pipelining concept, the result of multiplication is normalized by shifting it 1 bit to the right, if it is required. Finally, the resulting number is rounded to fit the size of the significand.

The rounding requires computing the sticky of the m −2 LSBs of the result of the fixed-point multiplication, and the addition of one ULP for rounding it up. This is obtained in third stage of pipeling format. When the HUB numbers are used, the significands again have to be appended with the hidden ILSB that equals one.

Thus, the fixed-point multiplier is 1 bit larger, which increases the size of the fixed-point multiplier. In contrast, the computation of the sticky bit is again prevented [11]. To simplify the explanation, let us assume that the significand is scaled such that the ILSB is the only fractional bit. Let us call the integer part of both the input significands A and B. The product of both the significands is then,

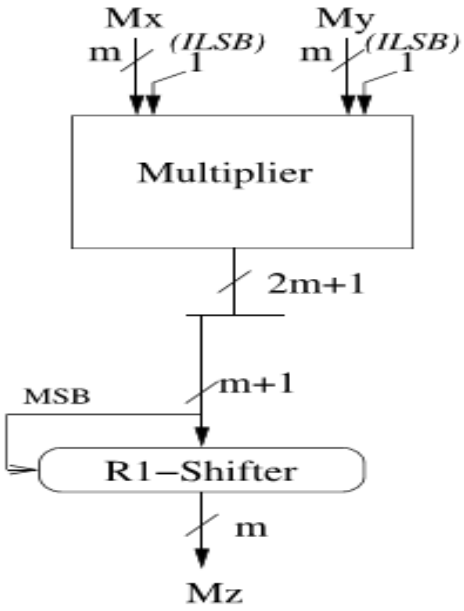

Fig.5. Basic FP multiplier architecture

Thus, the fixed-point multiplier is 1 bit larger, which increases the size of the fixed-point multiplier. In contrast, the computation of the sticky bit is again prevented [11]. To simplify the explanation, let us assume that the significand is scaled such that the ILSB is the only fractional bit. Let us call the integer part of both the input significands A and B. The product of both the significands is then,

$$(A+1/2)\cdot(B+1/2)= A\cdot B+ 1\ 2\ (A+B)+1/4.$$

Given the last addend of this result, we conclude that the LSB of the result of multiplication is always 1. Thus, the sticky bit is always 1 and no logic is required to compute it. Moreover, given that the rounding is simply performed by truncation, the final incrementer is not required. Therefore, as shown in Section, although the area of the fixed-point multiplier increases, the overall area decreases.

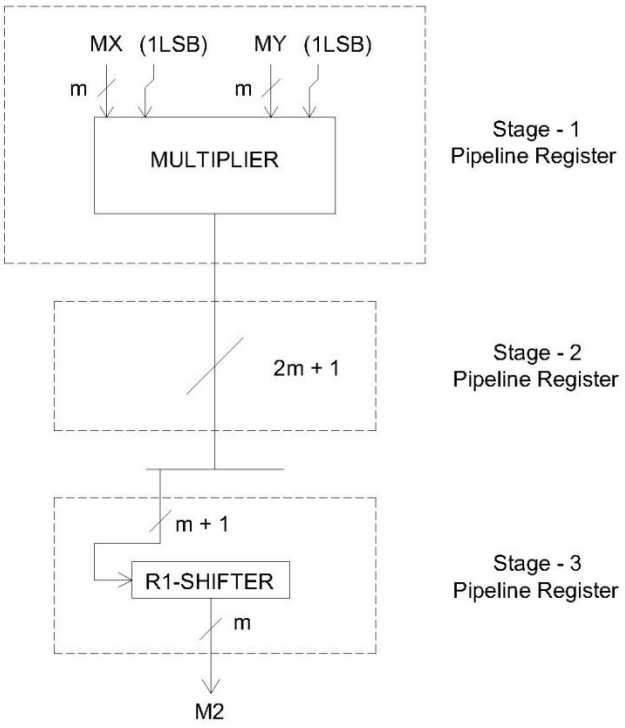

Fig.6. FP multiplier using pipelining register

IV. IMPLEMENTATION RESULT AND COMPARISON

In this section, we experimentally study the convenience of using the proposed HUB FP formats using pipelining to implement the real-number computation. First, we provide an experimental error analysis to confirm that the use of the PIPELINING formats does not damage the accuracy of the computation. Second, we analyze the main results of the hardware implementation of the proposed HUB FP circuits compared with the classic implementation.

A. Experimental analysis

An empirical error study is provided to demonstrate that the pipelining formats could be used instead of the IEEE standard in FP-specific applications, while guaranteeing the same level of precision. In a first experiment, we tested the arithmetic operations. In these experiments, we utilized 32-bit two's complement fixed-point numbers within the range (−11) (i.e., 1 sign bit and 31 fractional bits) as exact real input numbers. They were converted into an internal 32-bit FP format with only 24 bits for the significand. Thus, a rounding was required for the said conversion. Two different internal formats were tested: 1) the standard IEEE-754 single precision and 2) its corresponding HUB using pipelining format.Similarly, we also studied the circuits used to convert between single precision and double precision. The area required for converting from double precision into single precision. It can be observed that the area of the HUB implementation is practically constant for the ranges of the frequencies tested. This is because the critical path is very short for this circuit. In this case, the elimination of the rounding logic dramatically reduces the area. The circuit for the standard format requires between 2.5 and 4 .4 as much area as the HUB approach.

Fig.7:Floating point adder without pipelining unit

Fig.8:Floating point adder with 2 stage pipelining unit

Regarding power consumption, although this reduction is slightly less, it is still very high. The power consumption of the standard converter is between 2 and 3.6 times greater than the one for the HUB circuit. However, we should note that the relative impact of these circuits on the overall area or power is much lower than that on the adder or the multiplier.

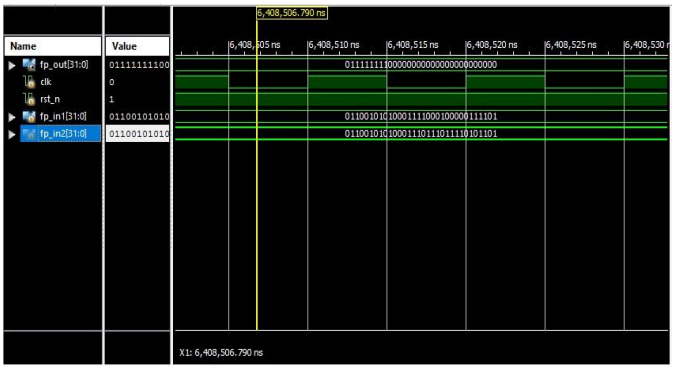

Fig.9:Floating point multiplier with 3 stage pipelining unit

Performance Parameter	Modified FP Multiplier
Area	671+74
Delay	6.199

Table - 1: FP MULTIPLIER

Performance Parameter	Modified FP Adder Without Pipeline	Modified FP Adder With Two Stage Pipeline
Area (Slices+Registers)	681+98	671+74
Delay	19.700ns (Fmax:50.761mhz)	8.852ns (Fmax:112.969mhz)

TABLE – 2: FP ADDER

CONCLUSION

This paper presents a way to simplify the FP systems using the pipelining formats. When the preferred rounding mode is RN, the implementation of the arithmetic unit for dealing with the pipelining FP number is much simpler than when using classic units. There are many floating-point adder implementations available for FPGAs but to the best of our knowledge, no work has been done to design and compare different implementations for each sub component used in the floating-point addition and multiplication for a FPGA device. The main objective of our work was to obtain best overall latency to improve custom designs by implementing these components.

REFERENCES

[1] P.Vivek Karthick, N.Renith, G.Srinidhi, S.Priyadharshini, M.Vidhya, "FPGA based Real Time Communication for Tele Health using Android phone ," International Journal of Scientific Research in Computer Science, Engineering and Information Technology(IJSRCSEIT), ISSN:2456-3307, Vol 2, Issue 2, pp.258-260, March-April.2017.

[2] Javier Hormigo and Julio villalba, "Measuring Improvement Using HUB Format To Implement Floating-Point systems Under Round-To-Nearest," IEEE Transactions on very large integeration systems(vlsi), Vol 24,No 6,june 2016.

[3] R.S.Keote, P.T.Karule, "VLSI Design of 64bit × 64bit High Performance Multiplier with Redundant Binary Encoding" IEEE 2016.

[4] Y. He and C. Chang, "A new redundant binary Booth encoding for fast 2-bit multiplier design," IEEE Trans. Circuits Syst.I,Reg. Papers,vol. 56, pp. 1192–1199, 2009.

[5] J.-Y. Kang and J.-L. Gaudiot, "A Simple High-Speed Multiplier Design," IEEE Trans. Computers, vol. 55, no. 10, pp. 1253-1258, Oct. 2006.

[6] P. M. Seidel, G. Even, "Delay-Optimization Implementation of IEEE Floating-Point Addition," IEEE Transactions on computers, pp. 97-113, February 2004, vol. 53, no. 2.

[7] J. D. Bruguera and T. Lang, "Leading-One Prediction with Concurrent Position Correction," IEEE Transactions on Computers, pp. 1083–1097, 1999, vol. 48, no.10.

[8] S. F. Oberman, H. Al-Twaijry and M. J. Flynn, "The SNAP Project:
Design of Floating-Point Arithmetic Units" Proc. 13th IEEE Symp. on Computer Arithmetic, pp. 156-165, 1997.

[9] V.G. Oklobdzija, D. Villeger, and S.S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," IEEE Trans. Computers, vol. 45, no. 3, pp. 294-306, Mar. 1996.