Normalizing or Not Normalizing? An Open Question for Floating-Point Arithmetic in Embedded Systems

Sonia Gonzalez-Navarro, Javier Hormigo
Universidad de Malaga, Andalucia Tech,
Departamento de Arquitectura de Computadores, Campus de Teatinos s/n, 29071 Malaga, España
Email: sonia@ac.uma.es, fjhormigo@uma.es

***Abstract*—Emerging embedded applications lack of a specific standard when they require floating-point arithmetic. In this situation they use the IEEE-754 standard or ad hoc variations of it. However, this standard was not designed for this purpose. This paper aims to open a debate to define a new extension of the standard to cover embedded applications. In this work, we only focus on the impact of not performing normalization. We show how eliminating the condition of normalized numbers, implementation costs can be dramatically reduced, at the expense of a moderate loss of accuracy. Several architectures to implement addition and multiplication for non-normalized numbers are proposed and analyzed. We show that a combined architecture (adder-multiplier) can halve the area and power consumption of its counterpart IEEE-754 architecture. This saving comes at the cost of reducing an average of about 10 dBs the Signal-to-Noise Ratio for the tested algorithms. We think these results should encourage researchers to perform further investigation in this issue.**

***Index Terms*—Normalization, embedded systems, DSP, floating-point, standard.**

I. Introduction

After 30 years the IEEE-754 standard [1] is at last generalized and almost compulsory for any Floating-Point (FP) design. On the other hand, until recently, FP formats had been considered too expensive to implement for digital signal processing (DSP) and other embedded applications. For this reason, fixed-point formats had been used instead. However, the increase in the complexity of the algorithms and new embedded applications have led engineers to start utilizing FP formats in the implementation of these applications. Due to the fact that IEEE-754 standard is completely generalized, this one has been usually used, sometimes with slight modifications, to implement those applications. However, the IEEE-754 standard was designed for general purpose processor applications and the requirements for embedded applications not always are the same. In fact, there are many different variations of the IEEE-754 standard defined by different companies to support these new requisites. As a consequence, there are many similar formats close to IEEE-754 but incompatible among them. We believe it is time to open a debate to define a new extension of the standard to cover embedded applications. With this paper, we aim to provide a step on this direction.

One of the main characteristics of the IEEE-754 standard for binary numbers is the representation of the *normal* numbers. To make the encoding of *normal* numbers unique, the value of the significand D is maximized by increasing/decreasing the exponent until $1 \leq D < 2$. This operation is commonly known as normalization, and it has to be carried out after every arithmetic operation (if necessary).

However, normalizing the results of each operation implies a significant overhead in hardware implementation. In fact, there are some companies that have already proposed special FP format which do not always perform normalization to improve hardware costs. As examples, Altera (Intel FPGA) and Synopsys which have developed fused FP data-path [2] and Flexible Floating-Point [3], respectively. In this paper we also study a FP format without compulsory normalization, but from a different perspective. Specifically, we keep the same number of bits instead of keeping the same accuracy. Hence, since normalization allows taking advantage of all the bits included in the significand, removing normalization will cause some accuracy loss.

Accuracy vs reproducibility trade-off is not new in embedded applications. There are different approaches to do that, but most of them have been subsumed under the term *approximate computing* [4]. Thus, the avoidance of normalization could be considered as approximate computing since an important area and power consumption savings could be obtained, but at the cost of some accuracy loss and lack of reproducibility. Nevertheless, neither the reduction in hardware cost nor the loss of accuracy have been thoroughly studied (or measured) in embedded systems yet.

Consequently, the main goal of this paper is twofold: to study how normalization affects the quality of the results in embedded system applications and to measure the improvement in hardware implementation when using non-normalized FP formats. We want to clarify that this paper does not pretend to propose a new FP format, but just to study whether it is beneficial to include normalization in the definition of a new FP format specially designed for embedded systems. Thus, this work has to be seen as a small contribution to define this format, but not a complete new format proposal.

This paper is organized as follows. Section II summarizes the related work found in the literature and Section III exposes the characteristics of the evaluated format. In Section IV we propose several adders and multipliers to operate with this

format, which are validated through software in Section V and through synthesis in Section VI. The conclusions of this work are shown in Section VII.

II. Related work

Several approaches have been proposed to reduce the cost of implementing FP numbers on application specific design. Most of them have been proposed in the context of FPGAs, since they allow low development cost and flexibility to explore different designs. They are usually modifications of the IEEE-754 standard.

The proposals with the slightest modifications are probably Xilinx FP implementations [5] and Flopoco [6]. These keep most of the characteristics of the standard, but adapting them to FPGA requirements. The main differences are flexible bit-width of the significand and exponent, no subnormal-number support, and only "round-to-nearest even" rounding mode. Besides those, Flopoco uses two additional bits to encode special cases [6].

Since barrel-shifters map poorly on FPGA, several academic works propose the use of high-radix digit for the significand in order to reduce the cost of the shifter implementation [7][8]. These approaches require larger significand to keep the same accuracy as the standard.

A more aggressive approach is the proposal of Altera by mean of fused FP data-path [2][9]. This solution aims the implementation of dedicated FP data-path where several FP operations are chained. A compiler automatically reduces the number of alignments and normalizations required by consecutive operations and adjusts the significand bit-width to accommodate the bit growths. Therefore, this intermediate format is not normalized. An average of 50% saving in both area and latency is reported, and the results are overall more accurate than the IEEE FP standard implementation.

More recently an even more radical approach has been proposed by Synopsys, but targeting ASIC implementation instead of FPGA. In this case, the proposed FP format, called Flexible Floating-Point (FFP) is radically different to the IEEE standard [3]. The main characteristics of FFP are: the bit-width of both significand and exponent are flexible; moreover, they are represented using two's complements instead of Sign-Magnitude (SM) and excess, respectively; seven status flags are appended to the number to indicate special cases and other circumstances; and, one's complement is also allowed to represent the significand. Besides those, significand does not need to be in normalized form, and rounding is not usually applied since significand size could be accommodated to avoid loss of accuracy.

In all these approaches the main goal is to achieve more optimized circuits but providing the same as, or more accuracy than, that of the IEEE standard. As a consequence, most of them expand the number of bits used to represent the numbers. This increase may not be a problem when considering cheap memory available (as in FPGAs) or a specific-application data-path design with no intermediate results saved to mass memory storage. However, in this work we address a different situation where micro-controllers or specialized processing units are used, but intermediate storage and communication is also fundamental. Thus, our goal will be to improve hardware implementation but keeping the same bit-width as that of the IEEE-754 standard.

III. Characteristics of a non-normalized FP format

To study how normalization affects accuracy and hardware implementation, we evaluate a format that keeps most of the characteristics of the binary IEEE-754 standard format, except for the necessity of all numbers being normalized. But this change forces us to consider also other ones.

First, if numbers are not always normalized, the leading one could not be implicitly stored. This change directly produces a loss of precision, because the significand of our format actually has one bit less. On the other hand, due to this explicit representation of all bits, denormal numbers are naturally handled without needing a special treatment and the same happens with the zero value.

Another important change compared to the IEEE standard is the rounding mode. We consider only a fixed rounding mode. We have studied two cases: round-toward-zero (truncation) and round-to-nearest, but the latter implemented by using Half-Unit Biased (HUB) numbers [10]. Those were selected in order to keep rounding as simple as possible, since none of them require any additional operations, but simply discarding the Least Significant Bits (LSBs). HUB FP approach appends an Implicit Least Significant Bit (ILSB) set to one, allowing rounding-to-nearest by truncation and two's complement by bit-wise inversion [11].

Besides these main changes to the standard, it seemed sensible to not consider the other special values, i.e., NaN and infinity. First, this facilitates the implementation of this non-normalized format, and mainly, we do not think they are very useful for the target applications (although if these special values were required, they could be included in the format). Then, they are not implemented in the architectures for either non-normalized or IEEE numbers (for the latter we also omit denormal numbers). Consequently, we do not think the inclusion of these special cases would change significantly the conclusions derived from this work.

Summarizing, in this paper we use a format similar to binary32 [1], that we evaluate in two versions. In the regular version the format includes: one sign bit (S), 8 bits of exponent (E) and 23 bits of significand (D) with 22 fractional bits, which value is calculated always following this equation $(-1)^s \cdot D \cdot 2^{(E-127)}$ (there are not special cases). To calculate the value of the HUB version, the previous equation is used but appending the ILSB to D (i.e. the significand has the form $\{D, 1\}$ where $\{,\}$ denotes a concatenation operator). The evaluated format could be straightforwardly extended to other bit-width, but in this paper we will focus on 32-bit size.

IV. Arithmetic units for non-normalized FP numbers

In this section we present different architectures to simplify the implementation of the main arithmetic operations, addition and multiplication, for non-normalized FP numbers. Since normalization is not compulsory (but could be used), designers could trade accuracy off for implementation cost. We have investigated several architectures that are relatively straightforward although more complex ones could be figured out.

A. Adders for non-normalized FP numbers

Fig. 1 (eliminating grey boxes) shows a basic adder architecture, that we denote as A1, to manage non-normalized numbers. Comparing it with an IEEE compliant adder, basically, the normalization circuit (leading-one detector and left-shifter) and rounding circuit are removed in this approach. However, a one-position right-shifting and an increment of the exponent are still performed if the result of the fixed-point addition produces an overflow. Another option to remove this selective shifting could be to consider that an overflow always occurs at the fixed-point adder and to perform always a right-shifting (hardwired) of the addition result. In this case, we lose the LSB of the result if actually no overflow occurs.

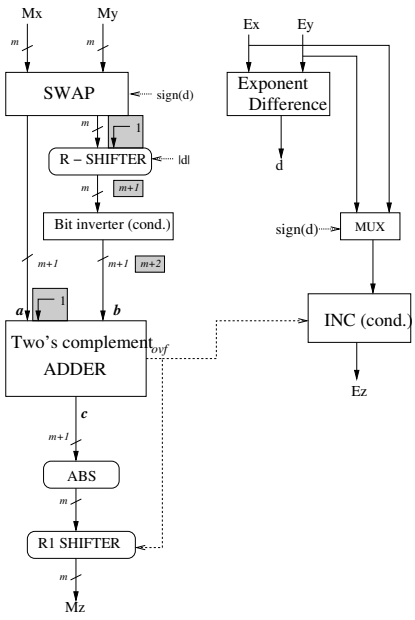

Fig. 1. Architecture of the non-normalized A1 adder. HW added for HUB adder version in grey area

Due to the fact that numbers may be non-normalized, it is complicated to calculate the sign of the result of the fixed-point addition in advance for input operands with different sign. If only exponents are compared, even if one operand has an exponent clearly higher than the other, it could happen that the operand with the highest exponent were smaller than the other operand because the difference between significands could also be very significant. Therefore, an absolute value operation has to be performed after the fixed-point addition.

Consequently, in our basic adder design, the operand with the lowest exponent is the one that is always two's complemented when it is necessary. A sign bit is incorporated to the significands ($m + 1$ bits), before adding them. For the two's complement operation a conditional bit inverter is used to perform the one's complement and a sticky bit calculation after alignment to decide whether the input carry at the fixed-point adder should be set to one or zero. If the result of the addition is negative, a complete two's complement operation is carried out at the output of the fixed-point adder. Next, the result is right-shifted one position and the exponent is updated if an overflow has been detected.

We have to note that there is no detection of the underflow exception. Normalization is not performed, so no subtraction to the exponent is taken. Thus, the result of the FP operation could be zero, but no underflow could occur.

A possible optimization to this design is to incorporate the rounding to nearest operation, taking the advantage of the two's complement operation in the absolute value module. So we could have the better rounding mode simply by fusing both operations. However, we prefer to explore the HUB format which allows to change the two's complement circuits for a conditional bit inverter and at the same time produce round-to-nearest operation at not additional cost. In Fig. 1 grey boxes represent the hardware added to implement the HUB version of A1, denoted as A1H. Using the HUB approach, the ILSB has to be appended to the input significands and the absolute value operation is simplified, because bit inversion produces two's complement.

We also test intermediate solutions, where left-shifting is performed but only for a few bits. Fig. 2 shows this new adder that we denote as A2 (A2H for HUB version). Comparing it with A1, the A2 design has a special leading zero detector, which detects up to two leading zeros at the output of the absolute value circuit. Furthermore, it has a barrel shifter that can perform a one-position right-shifting (in case of detecting overflow) and left-shifting up to 2-bit positions. This will increase the area and the delay of the critical path, but it will improve the error figures as we will see in section V. In this architecture the exponent has to be decremented when left-shifting is performed, and therefore underflow could happen. Although it is not depicted in Fig. 2, this situation is detected in the design and the result flushes to zero.

B. Multipliers for non-normalized FP numbers

Fig. 3 (without grey boxes) shows the simplest multipier where the significand of the result is obtained taken the m Most Significant Bits (MSBs) of the fixed-point multiplication (performing truncation) and no normalization is carried out. We denote this multiplier as M. Grey boxes in Fig. 3 represents the hardware added to implement the HUB version of the M multiplier that we call MH. The ILSBs are appended to each input significand and consequently the fixed-point multiplier is one-bit wider. Excluding this, the circuit remains the same, but the result is a HUB number rounded to nearest.

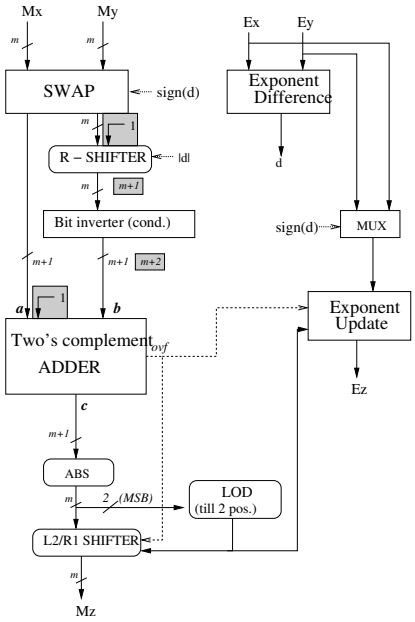

Fig. 2. Architecture of the optimized non-normalized A2 adder. HW added for HUB adder version in grey

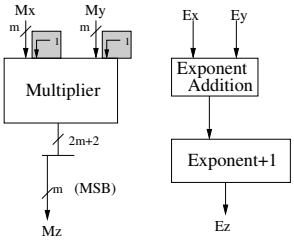

Fig. 3. Architecture of the non-normalized M multiplier. HW added for HUB multiplier version in grey

In these architectures (M and MH), it is assumed that the MSB of the result of the fixed-point multiplication is always one. From here on, we denote to this situation as that an overflow of the multiplication has occurred. Under this assumption, the exponent is always incremented and the result of the multiplication is right-shifted one position (simply by taking the m MSB of the result). This assumption simplifies hardware but at the cost of losing one bit of precision when overflow does not occur (which is the most likely situation). Thus, we have implemented another multiplier, shown in Fig. 4, that we denote as M2. In M2 the right-shifting and the increment of the exponent is carried out only when an overflow of the multiplication is really detected. Again, grey boxes represent the additional circuit for the HUB version, that we denote as M2H.

Multiplication of non-normalized numbers may degrades strongly the accuracy of the results, since the number of leading zeros of the result is the addition of the number of leading zeros of each input operand. Therefore, we may think of normalizing (at least partially) the output of the multiplier to keep the accuracy of the computation within

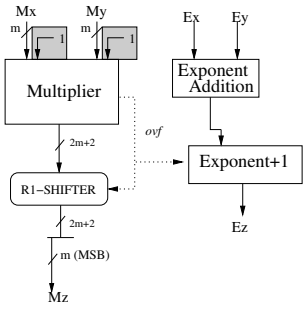

Fig. 4. Architecture of the non-normalized M2 multiplier. HW added for HUB multiplier version in grey

reasonable limits. As we said above, the number of leading zeros at the multiplier output could be calculated by counting them at the input operands. Fig. 5 depicts a general design of a multiplier performing normalization using this scheme (grey boxes are only used in the HUB version). Two Leading Zero Detectors (LODs) are used to count the number of leading zeros of each input significand in parallel with the multiplication. In addition, a barrel left-shifter is added at the output of multiplier.

We explored different options for this scheme. Concretely, besides the complete normalization, we studied two different approaches for partial normalization to reduce the impact of the normalization circuit. Consequently, although the architecture remains the same, both the LOD circuit and the left-shifter circuit are different in each approach. These two approaches always consider overflow of the multiplication as in M.

In one approach, we implement a more roughly normalization using higher radix than two (digits of x-bits), to count zeros and to perform the shifting. This is accomplished by introducing x-input OR-gates before counting the zeros and eliminating the multiplexors with the least amount of shifting at the barrel-shifter. This approach produces lighter circuits but with less accuracy. Changing the number of bits per digits, we can balance implementation cost and accuracy, being 1-bit per digit, a complete normalization circuit. We denote these multipliers as MRx (MRxH, for HUB version), where x denotes the number of bits per digit.

A different strategy is to use radix-2 but limiting the maximum number of left-shifting positions, similarly to A2 adder. As we will see in the next section, even allowing very few shifting positions, this partial normalization has a great impact on accuracy. In this case, the number of zeros counted in each input significand is limited to x bits and, as a consequence, the number of left-shifting positions is bounded to 2x bits. We call these multipliers as MLx (MLxH, for HUB version), where x denotes the maximum number of shifting positions allowed.

We have to note that multiplication of small operands may produce underflow when the exponent of the result is calculated. This situation is detected and the result is flushed to zero, which is a desirable behavior for the target applications. Although this detection is not shown in the previous figures,

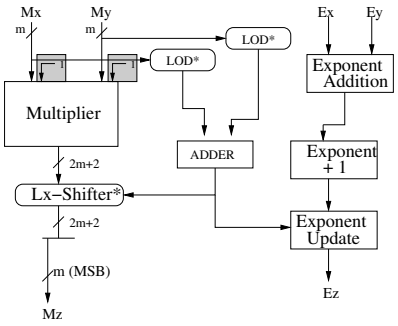

Fig. 5. Architecture of the non-normalized MRx and MLx multipliers. Symbol * denotes specific function of the circuits. HW added for HUB multipliers in grey

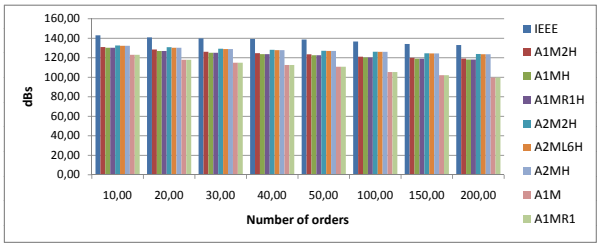

Fig. 6. SNR of FIR filters for different arithmetic architectures

it has been implemented in all multipliers.

V. Quality of the results when operating with non-normalized numbers

Using non-normalized numbers implies loss of accuracy compared to IEEE standard due to several factors. First, one bit of precision is directly lost due to the lack of the implicit leading one. Second, when adding two numbers using the basic architecture, it is lost as many significant bits as the minimum between the positions shifted for alignment and the number of leading zeros in the significand with the highest exponent. Similarly, when multiplying, the sum of leading zeros of both significands equals the number of bits lost.

In this section we analyze the impact on final accuracy when implementing several basic DSP algorithms. To do this, we have designed the architectures presented in the previous section using VHDL. Using these arithmetic units we have designed an embedded system in a Xilinx Zynq-7010 FPGA which contains an ARM dual-core Cortex^{TM}-A9 processor, such that the designed 32-bit arithmetic unit for non-normalized numbers works as a coprocessor. Specific functions have been designed to allow mapping addition and multiplication into the coprocessor, along with conversions from/to IEEE FP standard.

We proceed as follows: the target algorithm is implemented using C programming language for double precision FP numbers (double) to use it as reference implementation; moreover, the same function is implemented for both IEEE 32-bit precision (float) using the regular processor and 32-bit non-normalized format using the coprocessor; the results of both 32-bit implementations are compared with the double precision results, calculating the error in terms of signal-to-noise ratio ($\text{SNR}_{dB} = 10 \cdot \log_{10}\left(\sum y^2 / \sum (y - y')^2\right)$, where y is the reference signal, and y' is the signal to be evaluated).

This experiment has been carried out for several coprocessors with different combination of adders and multipliers. Specifically, we tested two adders: A1 which only implement one-bit right shifting of the output significand, and A2 which also performs up to 2-bit left shifting for partial normalization. These adders are combined with several multipliers: M, the simplest multiplier without any shifting of the output; MRx which includes left shifting of the output significand using x-bit digits; MLx which performs left shifting of the output significand up to x-bit positions; and M2 which only implements 1-bit right-shift when an overflow of the multiplication occurs. Moreover, all these architectures could be regular or using HUB numbers (H). For instance the approach A1MR4H uses the HUB versions of adder A1 and multiplier M with 4-bit digit normalization. The decrement of the SNR when using all these approaches is studied below.

First, we start with FIR filter implementations because they are extensively used in DSP applications. We have run several experiments using low-pass filters with different cut-off frequencies and a wide range of orders. A linear chirp signal ranging between $[-1, 1]$ plus a random signal ranging between $[-0.1, 0.1]$ is used as input signal, which is a typical input signal to test low-pass filters. Fig. 6 shows some results obtained for the same cut-off frequency and different numbers of orders. Although, we have tested all approaches, this figure only represents the most significant ones in order to explain the following conclusions. First, HUB approaches (rounding-to-nearest) perform much better than regular ones (truncation). The latter has significantly lower SNR from the beginning and this difference clearly goes up with the number of orders. In contrast, the difference of SNR between HUB and IEEE approaches remains similar. It seems that the higher accuracy reduction is mainly caused by the truncation. Second, normalization of the multiplier output does not affect the results. That may be explained by the fact that all multiplications are performed over normalized operands, because these are the signal input and the filter coefficient and they come directly from a conversion of an IEEE number. Therefore no improvement could be expected by normalizing the multiplier result. Finally, utilizing A2 greatly improves the results whereas M2 improves them slightly, and as a consequence, A2M2H provides the better results being on average about 10 dBs lower than the IEEE ones.

In another experiment we use a very simple Kalman filter that approximates a constant value through a set of noisy measurements. As input signal, a random number ranging between $[-0.1, 0.1]$ plus 0.5 is used. We should note this algorithm requires a division which has been implemented by conversion to IEEE standard. Table I shows the SNR obtained for all architectures tested. It can be observed that, in this case, any kind of normalization in either the adder or

TABLE I
SNR OBTIANED FOR KALMAN FILTER IMPLEMENTATIONS

IEEE	HUB		no HUB
146.2	A1	A2	A1
M	0.0	135.5	0.0
M2	133.8	**135.9**	124.0
MR1	133.9	135.5	123.0
MR4	132.0	135.5	123.0
MR8	1.3	135.5	1.3
ML4	133.9	135.5	123.4
ML6	133.9	135.5	123.2

multiplier is compulsory in order to obtain acceptable results. This normalization could be very limited as in A2 adder, but if there is no normalization or this one is not fine enough, the results may be catastrophic like happens to A1M or A1MR8. Again, the best results are achieved by A2M2H, but any architecture using A2 adder obtain very close results. Similarly to FIR filters, HUB architectures perform much better than regular ones, which indicates that using rounding-to-nearest seems crucial for these applications.

Finally, we implemented IIR low-pass filters using both direct-form I and II implementations for different cut-off frequencies and orders. The same input signal as that of the FIR filters is used. Fig. 7 shows the SNR obtained for IIR filter implementations using direct-form II. Only these results are shown because, although the results of direct-form I are different, they are similar compared in relative terms of accuracy.

This new experiment confirms several conclusions extracted from the previous ones, mainly that HUB approaches provide much better results and that the approaches using A2 adder give significantly higher SNR than the ones using A1, specially A2M2H. Although the influence of normalization on multipliers is negligible if A2 adder is used, it is very significant in architectures using A1. To study in details this, Fig. 8 shows the SNR obtained for these latter architectures. As expected, MR1 obtains the best results which gradually and quickly goes down when x (the number of bits per digit) goes up. For the case of MLx, accuracy remains very similar to MR1 until the moment that the complexity of the designs exceeds a threshold and the accuracy falls off dramatically.

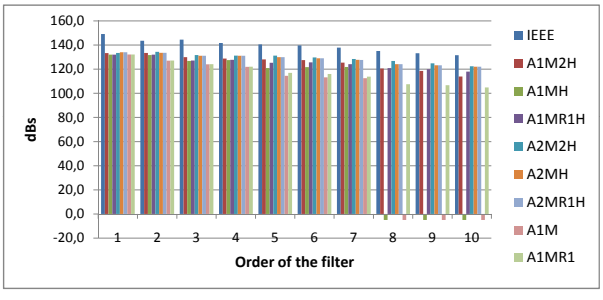

Fig. 7. SNR of IIR filters for different arithmetic architectures

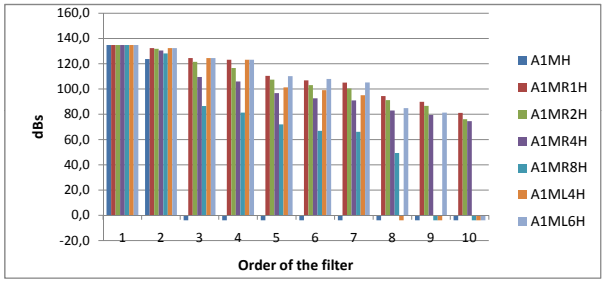

Fig. 8. SNR of IIR filters for A1 and different multipliers architectures

VI. IMPLEMENTATION RESULTS AND COMPARISON

All adder and multiplier designs presented in section III have been synthesized using Synopsys Design Compiler Ultra H-2013.03-SP2 and the TSMC 65nm library with default cell activity and "typical-case" operating conditions in which the temperature is $25°C$ and voltage $V_{dd} = 1.0V$. The area and power estimations provided by this tool is presented in this section. In this paper we only analyze combinational implementations, but different results may be expected for pipelining ones, which will be study in future works.

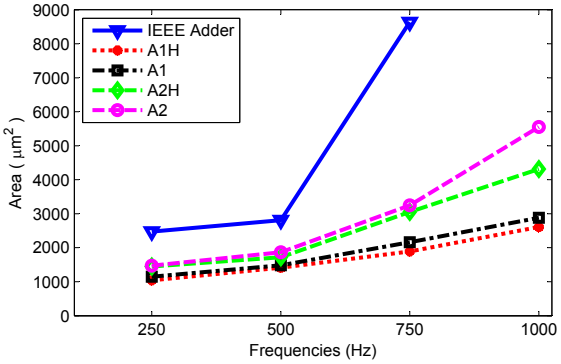

Fig. 9. Area of 32-bit adders

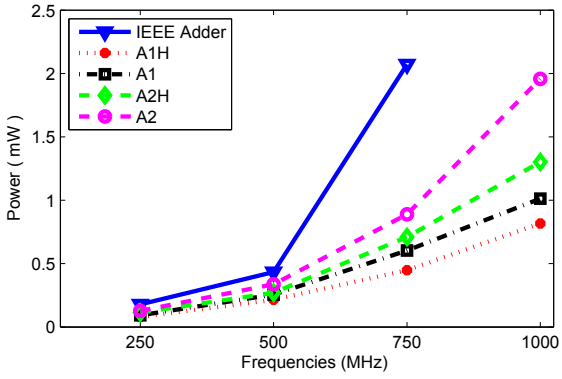

Fig. 10. Power consumption of 32-bit adders

The area and power consumption for the 32-bit adders are shown in Fig. 9 and Fig. 10, respectively. From these results, we observe that the HUB adders have less area and consume less power than their counterpart regular ones. This could be

caused by the simplification of the two's complement operation for HUB approaches. We should remind HUB solutions also has better accuracy (see section V). Comparing them with the IEEE one, all architectures significantly reduce area, and this reduction increases with the frequency, ranging from 40% to 58% at 250 MHz and from 62% to 78% at 750 MHz. Similar reduction is observed for the power consumption although in a slightly less amount. This area and power reduction is due to the elimination of normalization and rounding circuits. Being a single-path architecture, these logic elements are in the critical path which explains the high area growth at 750 MHz (the maximum frequency for IEEE architecture). However, thanks to the reduction in the critical path, all proposed architectures can work at 1GHz. Even working at this frequency, they reduce the area and power up to 70% and 60%, respectively, compared to the IEEE architecture at 750 MHz.

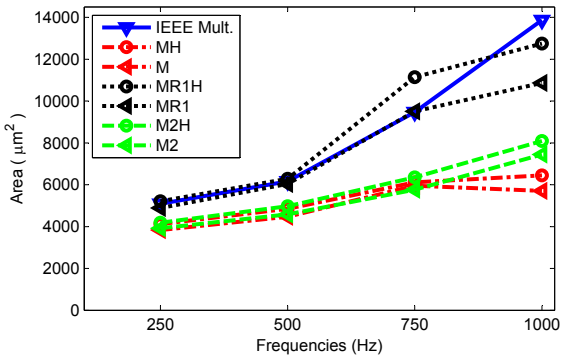

Fig. 11. Area of 32-bit multipliers

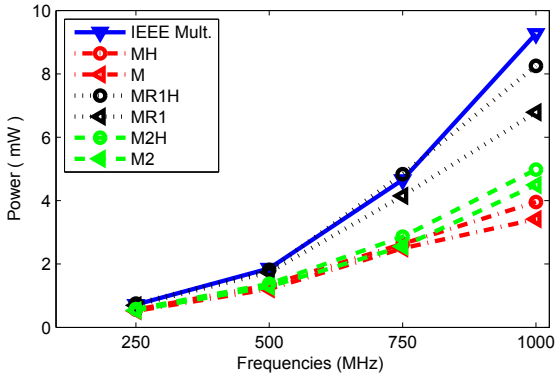

Fig. 12. Power consumption of 32-bit multipliers

In the case of the multipliers, from Fig. 11 and 12 we can observe that the improvement in this case is not as greater as the one obtained for the adders. In fact, the area and power consumption of MR1 (which carries out full normalization) are significantly greater than the IEEE multiplier for some frequencies. That is because the IEEE multiplier does not have a normalization circuit whereas MR1 has a complete shifter and two LODs. However, the rest of architectures achieve an area and power reduction up to 25% at 250, and up to 60% at 1GHz. This reduction mostly comes from the elimination of the rounding logic including the sticky bit computation. In contrast to the adders, HUB multiplier architectures consume significantly more area and power than their regular counterparts, due to the inclusion of the ILSB in the fixed-point multiplier.

For clarity, not all architectures with limited normalization (MRx and MLx) are represented in the previous figures, but their numbers are between the ones of M and MR1. The area and power consumption of the HUB ones are presented in Fig. 13 and Fig. 14 for further study. It can be seen that MLx architectures have much less implementation cost than MRx. The LODs of MLx only check the x MSBs whereas the ones of MRx need the previous step of OR gates and a more complicated barrel shifter, depending on x. Therefore, taking into account that they have similar accuracy (see section V), MLxs seem better options than MRxs.

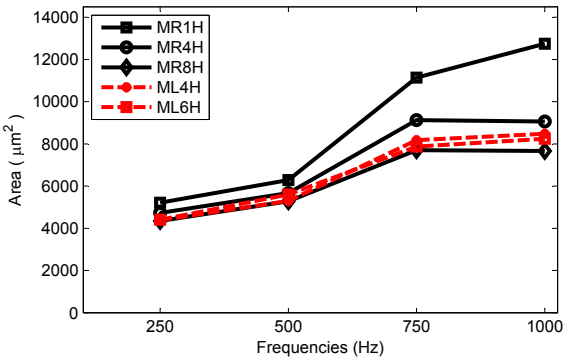

Fig. 13. Area of some HUB multipliers with partial normalization (MRx and MLx)

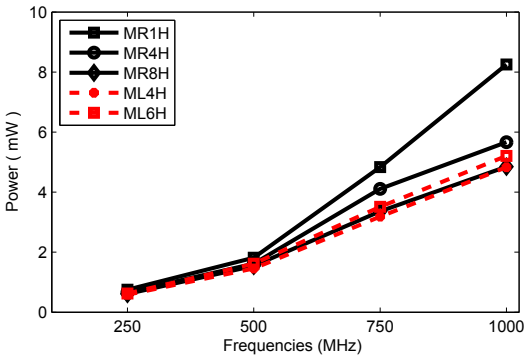

Fig. 14. Power consumption of some HUB multipliers with partial normalization (MRx and MLx)

To estimate the implementation results in a complete system, we also compare the combined results, i.e. having one adder and one multiplier. This estimate will vary if there is a different adders/multipliers ratio. Fig. 15 shows the total area required for implementing several significant combinations. Similarly, Fig. 16 shows the power consumption for these combinations. As expected the regular approaches require a little less area than HUB ones. However, we focus on the latter due to the poor error performance of the former.

All combinations achieve an important area reduction compared to the IEEE implementation, and this reduction rises when the frequency increases. This reduction ranges from 12% to 32% at 250MHz and from 22% to 56% at 750MHz. Considering the power reduction, it is slightly lower, ranging from 4% to 27% at 250 MHz and from 18% to 51% at 750MHz. As expected, among the proposed architectures, A2MR1H has the worst implementation results. In contrast, A1MH requires the minimum area and power, but its error performance is the poorest. The combination with the lowest error, A2M2H, achieves a very significant reduction ranging from 25% to 48% for the area and from 23% to 47% for the power. Similarly, A2MH gets the highest area and power reduction among the set of combinations that include A2 and have the same error performance. A2MH provides slightly less accuracy (an average of 1dB less) and also a slight reduction of area and power compared to A2M2H, ranging from 1.3% and 13% for the area and from 2.3% to 6.3% for the power.

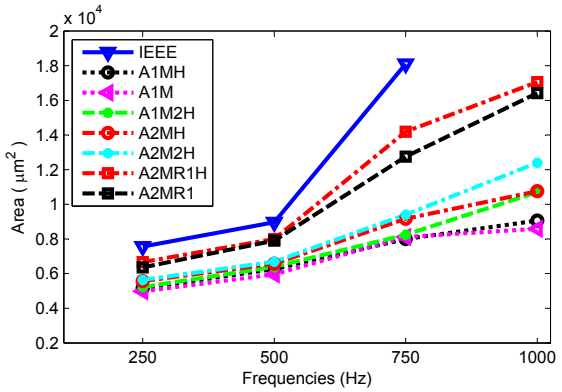

Fig. 15. Combined adder+multiplier area

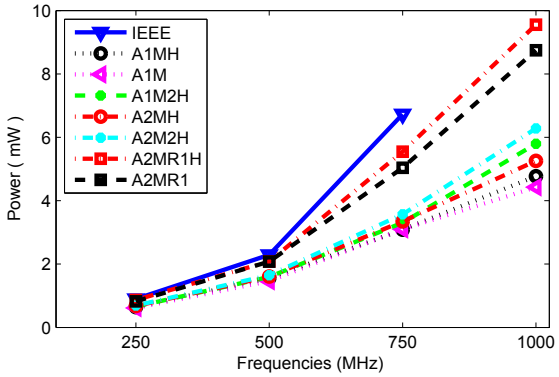

Fig. 16. Combined adder+multiplier power consumption

VII. Conclusions and Future Work

In this paper, we have studied the impact in both implementation results and accuracy when eliminating the requirement of normalization in FP formats. We have provided several new architectures to add and multiply non-normalized numbers and their implementation results. We have shown the accuracy degradation produced by implementing a few basic DSP algorithm using these architectures. Using round-to-nearest, in this case through HUB approach, is fundamental to keep errors bounded. Having some normalization at the adder output improves results significantly, even if this normalization were very limited. Similarly, detection of significand overflow in adders and multipliers is also fundamental.

We should highlight that using a non-normalized format has others important drawbacks besides lost of accuracy. First, it makes reproducibility among different architectures almost impossible. Second, comparison of numbers costs much more because it needs a previous normalization. However, these drawbacks are not a problem for many embedded applications.

This work should be seen as an encouragement for performing further investigation on this area. We plan to measure accuracy on more complex algorithms. This study should be also carried out for different bit-width (half or double precision), pipelined architectures, and different implementation technology like FPGA. As a further step to define a new extension of the FP standard, other aspects of the format should be studied like sign representation of the significand and exponent.

Acknowledgment

This work has been supported by the following Spanish projects: TIN2013-42253-P, TIN2016-80920-R, JA2012_P12-TIC-1470 and TIC-1692.

References

[1] *IEEE Std 754-2008 (Revision of IEEE Std 754-1985), IEEE Standard for Floating-Point Arithmetic*, 2008.

[2] M. Langhammer, "Floating point datapath synthesis for FPGAs," in *Proceedings - 2008 International Conference on Field Programmable Logic and Applications, FPL*, 2008, pp. 355–360.

[3] Synopsys, "DWFC Flexible Floating Point Overview," no. August, pp. 1–6, 2016.

[4] S. Mittal and Sparsh, "A Survey of Techniques for Approximate Computing," *ACM Computing Surveys*, vol. 48, no. 4,62, pp. 1–33, mar 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2891449.2893356

[5] Xilinx, "LogiCORE IP floating-point operator v7.0, product guide, PG060," *www.xilinx.com/support/documentation*, 2014.

[6] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *Design Test of Computers, IEEE*, vol. 28, no. 4, pp. 18–27, July 2011.

[7] A. Ehliar, "Area efficient floating-point adder and multiplier with IEEE-754 compatible semantics," in *Field-Programmable Technology (FPT), 2014 International Conference on*, Dec 2014, pp. 131–138.

[8] J. Villalba, J. Hormigo, F. Corbera, M. Gonzalez, and E. Zapata, "Efficient floating-point representation for balanced codes for FPGA devices," in *Computer Design (ICCD), 2013 IEEE 31st Int. Conf. on*, Oct 2013, pp. 272–277.

[9] M. Langhammer and T. VanCourt, "FPGA floating point datapath compiler," in *Proceedings - IEEE Symposium on Field Programmable Custom Computing Machines, FCCM 2009*, 2009, pp. 259–262.

[10] J. Hormigo and J. Villalba, "New formats for computing with real-numbers under round-to-nearest," *IEEE Transactions on Computers*, vol. 65, no. 7, pp. 2158–2168, 2016.

[11] ——, "Measuring Improvement When Using HUB Formats to Implement Floating-Point Systems under Round-to-Nearest," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2369–2377, 2016.