New Results on Non-normalized Floating-point Formats

Sonia González-Navarro and Javier Hormigo

Abstract—Compulsory normalization of the represented numbers is a key requirement of the floating-point standard. This requirement contributes to fundamental characteristics of the standard, such as taking the most of the precision available, reproducibility and facilitation of comparison and other operations. However, it also imposes a high restriction in effectiveness of basic arithmetic operation implementation. In many embedded applications may be worth to sacrifice the benefits of normalization for gaining in implementation metrics. This paper analyzes and measures the effect of removing the normalization requirement in terms of precision and implementation savings for embedded applications. We propose several adder and multiplier architectures to deal with non-normalized floating-point numbers, and quantify the accuracy loss and the improvements in hardware implementation. Our experiments show that it is possible to reduce the area and power consumption up to 78% in ASIC and 50% in FPGA implementations with a reasonable accuracy loss.

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

1 Introduction

EMBEDDED systems are becoming a key issue in computing system development. The growth in new applications and services based on Internet of Things (IoT), communications and automation has accentuated its interest. These systems typically have severe restrictions, such as power or energy consumption limitations, low computing capabilities, reduced budget, etc. For that reason, fixed-point formats have been traditionally used for these applications. However, the increasing complexity of new embedded applications means that floating-point (FP) computation is required by many of these applications to reach the minimum acceptable performance.

When floating-point is required usually the IEEE-754 standard [1] or a very similar format is used because of its reliability, compatibility and familiarity. However, the IEEE-754 standard was designed for general purpose processor applications and the requirements for embedded applications are not usually 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).

Going back to the fifties, far before the IEEE floating-point standard was approved, several studies were carried out to establish an unnormalized floating-point arithmetic [2], [3]. Although the aim was a floating-point format which allowed a more easily identification of their degree of precision, the authors claimed that their method had also advantages from the point of view of operation speed.

Other more recent works have investigated unnormalized floating-point arithmetic, which could be applied to specific-application data-paths [4], [5]. In these approaches, the bit-width is internally increased to keep the same precision. In contrast, in [6] and in this paper, we study the consequences of having a FP format without compulsory normalization to be applied to software-base processors which requires constant-size storage memory (such as DSPs or microcontrollers). Therefore, keeping the bit-width constant is fundamental. We showed that, since normalization allows taking advantage of all the bits included in the significand, removing normalization causes some accuracy loss, along with a lack of reproducibility [6]. However, at the same time, it opens the possibility of having important area and power savings. Removing the normalization condition allows designers to trade off between implementation cost and accuracy, and to tune efficiently the target requirements. Hence, this is a kind of approximate computing [7].

In this paper, we extend the study presented in [6], where the analysis of the proposed non-normalized FP arithmetic units was conducted assuming a non-normalized FP format similar to the binary32 IEEE-754. The details of that format can be found in [6] but the most important aspects about it are: it keeps most of the characteristics of the binary IEEE-754 standard format, except for the necessity of all numbers being normalized, and therefore, the leading one could not be implicitly stored; the denormal numbers are naturally

• *S. González-Navarro and J. Hormigo are with the Departamento de Arquitectura de Computadores, Campus de Teatinos s/n, 29071, Málaga, Universidad de Málaga, Andalucia Tech, España.*
E-mail: sonia@ac.uma.es, fjhormigo@uma.es

Manuscript received October 15, 2018; revised XXX XX, 2018. This work was supported in part by the following Spanish projects: TIN2016-80920-R, JA2012 P12-TIC-1692, JA2012 P12-TIC-1470.

handled without needing a special treatment. For rounding, truncation, and rounding to nearest (through Half-Unit Biased (HUB) approach [8]) were considered in [6]. However, truncation showed a poor accuracy. Hence, in this paper, we only consider round-to-nearest implemented by HUB approach. HUB 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 [9].

The new contributions presented in this paper include:

- the proposal of a new adder architecture, which improves significantly the accuracy of the previous ones,
- converters between non-normalized and IEEE-754 binary format,
- formal error analysis for non-normalized arithmetic units and the evaluation of an additional DSP algorithm for the experimental analysis,
- the implementation results for new sizes and frequencies,
- the analysis of the pipelined architectures,
- the implementation results for FPGA.

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.

The rest of the paper is organized as follows: Section 2 summarizes other proposals of changes of the floating-point standard found in the literature; in Section 3, we present different architectures to operate with non-normalized FP numbers for addition, multiplication and conversion from/to IEEE-754 standard; Section 4 provides a formal error analysis for non-normalized arithmetic units and experimentally measures how these errors affect some basic DSP algorithms; in Section 5, we provide, analyze and compare the implementation results of the proposed architectures; finally, Section 6 summarizes the conclusions of this work.

2 Related work

There have been different attempts to reduce the cost of implementing floating-point numbers for embedded applications. Some of them deal with the precision by using inexact operators, like in [10] and [11], while others deal with the selection of the FP format used. The latter usually proposed either slight or deeper modifications of the IEEE-754 standard.

The proposals with the slightest modifications are probably Xilinx FP implementations [12] and Flopoco [13]. These ones keep most of the characteristics of the standard, but adapting them to FPGA requirements. The main difference is the fact that the bit-width of the significand and the exponent can be specified in a given range. This allows users to adapt range and precision to the specific application in order to adjust the implementation cost. Another optimization comes from eliminating features which are scarcely used in these applications, such as subnormal-number support and rounding modes different to "round-to-nearest even". Besides those, Flopoco uses two additional bits to encode special cases [13], which simplifies the logic for their detection but increasing the cost of transmission or storage. In the same direction, since barrel-shifters map very poorly on FPGA, several academic works propose the use of high-radix digit for the significand in order to reduce the cost of the shifting [14] [15]. In return, these approaches require larger significand to keep the same accuracy as the standard.

Other approaches take advantage of knowing a priori the specific operations required to build a dedicated and optimized datapath where several FP operations are chained for the specific application. In [16], and [4] a fused FP data-path is proposed. 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. Nowadays, these savings may have even increased, since the tool has been probably improved over these ten years, but new information has not been published related to that.

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 FP standard [5]. The main characteristics of FFP are: the bit-width of both significand and exponent are flexible; moreover, they are both represented using two's complement 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 FP standard. As a consequence, most of them expand the number of bits internally used to represent the numbers. This increase may not be a problem when considering a cheap memory available (as in FPGAs) or a specific-application deep data-path with no intermediate results write back to memory. However, in this work we address software-base systems, i.e., specific-application processors, such as DSPs, microcontrollers, GPUs or other more specific ones (implemented on either ASIC or FPGAs). In these systems storage of intermediate results and communication is fundamental. Thus, our goal will be to improve hardware implementations, but keeping the same bit-width as that of the IEEE-754 standard.

3 Arithmetic units for non-normalized FP numbers

Being normalization not compulsory, designers have to take into account that input numbers may be not normalized, and, for the output results, they could trade accuracy off for implementation cost. In [6] we presented some architectures to simplify the implementation of addition and multiplication for non-normalized FP numbers. In this section, we review the more effective ones, propose a new adder, and study the conversion with IEEE-754. To facilitate the reading, we use the same notation as in [6] for the adder and multiplier architectures. As mentioned in Section 1, in this paper, we only consider the architectures implementing round-to-nearest by HUB approach.

3.1 Adders for non-normalized FP numbers

First, let us present some general ideas about addition in this context. Due to the fact that input numbers may be non-normalized, it is complicated to calculate the sign of the result of the fixed-point addition in advance for effective subtraction. Since the difference between significands may be very significant, it could happen that the operand with the highest exponent were smaller than the other operand. Therefore, in our proposal, when there is an effective subtraction, the operand with the lowest exponent is always two's complemented and an absolute value operation may be required after the fixed-point addition.

On the other hand, there are two options to control the overflow of the fixed-point addition: as regular FP adders, selectively performing a one-position right-shifting and an increment of the exponent if the result produces an overflow; or consider that an overflow always occurs, and hard-wiring the one-position right-shifting of the fixed-point addition result. In the latter case, implementation is simpler but the LSB of the result is lost, if no real overflow occurs. Thus, we recommend the first option since fixed-point overflow in non-normalized numbers is much less likely.

Two different adders were proposed in [6] (and their HUB versions which perform round-to-nearest rounding): A1 on which normalization and rounding circuit are completely eliminated except for the fixed-point overflow control; and A2 on which partial normalization is performed. The A2 design has a special leading zero detector, which detects up to two leading zeros. 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 increase the area and the delay of the critical path, but it greatly improves the error figures as is shown in [6]. In this architecture the exponent has to be decremented when left-shifting is performed, and therefore underflow could happen. If underflow is detected, the result flushes to zero.

The error performance of A2 could be improved by taking into account that normalization may be required even although right-shifting alignment were performed at the input. Based on that, a new architecture, A3, is proposed in this paper and its HUB version A3H is shown in Fig. 1. When normalizing, rather than introducing zeros at the LSBs, the two MSBs of the discarded bits for alignment are used instead. This small change has a significant impact on accuracy as we will see in Section 4. Note that due to using the HUB approach the implicit least significant bit (ILSB) of each input number is appended at the shifter and at the adder, respectively.

3.2 Multipliers for non-normalized FP numbers

For multiplication, the problem of having not normalized inputs is double. The number of leading zeros of the result is the addition of the number of leading zeros of each input operand. At first, any normalization at the output is compulsory and these could be implemented reading the leading zeros directly at the input. In [6], two different approaches are explored to partially normalizing the output result: high-radix normalization and limited normalization (similar to A2 philosophy). However, [6] shows that the effect of this normalization at the multiplier is minimal if partial normalization is implemented at the addition (A2 or A3) and the implementation cost increases very significantly. Consequently, we discard using normalization at the multiplier in this paper.

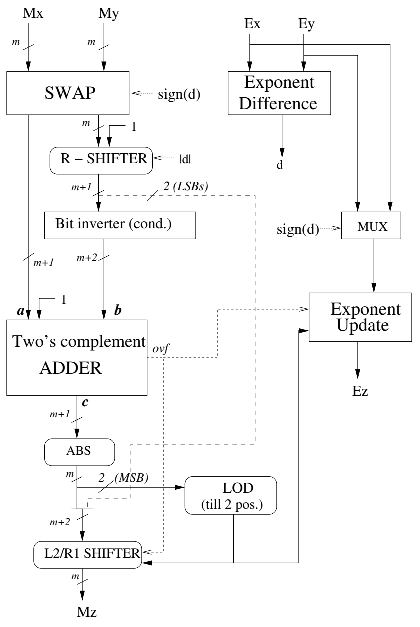


Fig. 1. Architecture of the optimized non-normalized A3H adder.

Again, fixed-point overflow could be managed in two different ways. First, considering that fixed-point overflow always occurs and hard-wiring the right-shifting. Second, checking the fixed-point overflow and using a multiplexer to act accordingly. As for addition, the first option is simpler but it produces loss of precision when overflow does not occur, which is very likely. The second has higher implementation cost but it is more accurate. The architectures, M and M2, presented in [6] implement the first and second options respectively (and their HUB versions MH and M2H).

3.3 Converters between non-normalized and IEEE-754 binary format

Conversion from IEEE-754 to non-normalized format, only requires including the implicit leading one of the significand and removing the least significant bit (LSB) to fit the bit-width. The implementation of this converter is simple and could be hard-wired. In contrast, converting from non-normalized numbers into IEEE-754 binary ones requires the hardware needed to perform normalization of the result in any IEEE compliant arithmetic unit (leading-one detector, left-shifter and exponent update). In case of having non-normalized HUB numbers, it is necessary to append the ILSB into the left-shifter. Fig. 2 depicts this converter, where grey box represents the hardware added to handle HUB numbers.

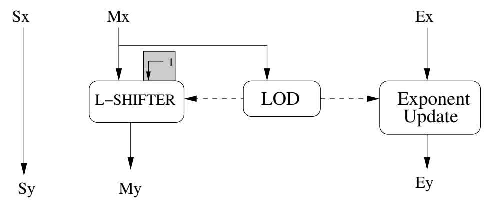


Fig. 2. Converter from non-normalized to IEEE-754 binary format.

4 Accuracy of non-normalized FP numbers

Using non-normalized numbers implies loss of accuracy compared to IEEE-754 standard due to several factors. In this section, we investigate the different sources of error through a formal error analysis, and how these errors impact the quality of the results of several basic DSP algorithms.

4.1 Formal error analysis

To develop the formal error analysis, let us consider that the non-normalized numbers have an implicit leading zero, similarly to the implicit leading one of IEEE FP numbers. This assumption does not modify the error but it allows us to unify the formulation for both FP representations. Let X^{Z_X} be a FP number with a m-bit significand M_X, a sign bit S_X, and an exponent E_X, which represents a real value with the form

$$X^{Z_X} \rightarrow (-1)^{S_X} \cdot M_X \cdot 2^{E_X} + e_X, \tag{1}$$

where e_X is the error of the representation, and Z_X is the number of leading zeros of the significand, such that

$$2^{-Z_X} \leq M_X < 2^{-Z_X+1}. \tag{2}$$

For an IEEE FP normalized number, Z_X is always zero, whereas for a non-normalized number $1 \leq Z_X < m$ ($1 \leq Z_X$, because there is not implicit leading one, and $Z_X < m$, because $Z_X = m$ would mean a zero value which is a special case).

Let us call e'_X the error of the significand itself, such that

$$e_X = e'_X \cdot 2^{E_X}, \tag{3}$$

and let e''_X be the error of the significand expressed in terms of units round-off,

$$e'_X = e''_X u \tag{4}$$

with u being the unit round-off, and $u = 2^{-m}$ [17].

Let us call R_X the relative error of the number X^{Z_X}. Taking into account Eq. 3 and Eq. 4, it fulfills

$$R_X = \left|\frac{e_X}{X^{Z_X}}\right| = \left|\frac{e_X}{M_X \cdot 2^{E_X}}\right| = \left|\frac{e'_X}{M_X}\right| = \left|\frac{e''_X}{M_X}\right| u. \tag{5}$$

After these definitions, below we will compare the relative errors produced by the proposed designs. Moreover, for each arithmetic unit, we will provide the bounds for the significand error of the results (e''_X) as a function of the corresponding error of the inputs operand. This will allow the reader to compute the bounds of the error for any sequence of operations.

Let us study first the error due to the representation itself by considering a FP number, A^{Z_A}, whose error comes only from rounding the significand ($e''_A = 1$), and then, it fulfills

$$|e_A| \leq 2^{E_A} \cdot u. \tag{6}$$

Thus, the relative error, R_A, fulfills

$$R_A = \left|\frac{e'_A}{M_A}\right| \leq \frac{u}{2^{-Z_A}} = 2^{Z_A} \cdot u. \tag{7}$$

Therefore, taking into account that $Z_A = 0$ for normalized numbers whereas $Z_A > 0$ for non-normalized numbers, the bound of the relative error for a non-normalized number is at least double than the one for a normalized number.

Secondly, let us study the addition operation of two arbitrary FP numbers, $A^{Z_A} + B^{Z_B} = C^{Z_C}$, for different adders, including the IEEE one for comparison purpose. Let us suppose there is no overflow, underflow, or special cases, and $E_A \geq E_B$. In this case, the intermediate result of the addition (without sign) is:

$$(M_A \pm M_B \cdot 2^{-d} + e'_A \pm e'_B \cdot 2^{-d}) \cdot 2^{E_A}, \tag{8}$$

where "$\pm$" represents an effective addition or subtraction, respectively, and d, the exponent difference ($d = E_A - E_B$). Then, after normalization (if necessary) and rounding, and considering Eq. 2 to Eq. 5,

$$\begin{aligned} |e_C| &\leq \left|u \cdot 2^{E_C} + (e'_A \pm 2^{-d} \cdot e'_B) \cdot 2^{E_A}\right| \\ |e'_C| &\leq \left|u + (e'_A \pm 2^{-d} \cdot e'_B) \cdot 2^{E_A - E_C}\right| \\ |e''_C| &\leq \left|1 + (e''_A \pm 2^{-d} \cdot e''_B) \cdot 2^{E_A - E_C}\right| \\ R_C &\leq \frac{\left|1 + (e''_A \pm 2^{-d} \cdot e''_B) \cdot 2^{E_A - E_C}\right|}{2^{-Z_C}} u \end{aligned} \tag{9}$$

Let us define δ as the number of new leading zeros in the significand due to the fixed-point addition. It is fulfilled that $-1 \leq \delta < m$, being $\delta = -1$ a significand overflow or a reduction of one leading zero. For the IEEE case, $Z_C = Z_A = Z_B = 0$ and $E_C = E_A - \delta$, and consequently

$$\begin{aligned} |e''_C| &\leq \left|1 + (e''_A \pm 2^{-d} e''_B) \cdot 2^{\delta}\right| \\ R_C &\leq \left|1 + (e''_A \pm 2^{-d} e''_B) \cdot 2^{\delta}\right| u \end{aligned} \tag{10}$$

As it is known, the errors carried by the input operands are attenuated if a significand overflow occurs ($\delta = -1$) and highly reinforced for a catastrophic cancellation ($\delta >> 1$).

For the non-normalized adders (A1H, A2H, and A3H), if overflow of the significand occurs, $E_C = E_A + 1$ and $Z_C = 1$. Then,

$$\begin{aligned} |e''_C| &\leq \left|1 + (e''_A \pm 2^{-d} e''_B) \cdot 2^{-1}\right| \\ R_C &\leq \left|2 + e''_A \pm 2^{-d} e''_B\right| u. \end{aligned} \tag{11}$$

Thus, the relative error is double that for the IEEE case (substituting $\delta = -1$ in Eq. 10).

On the other hand, if no significand overflow occurs, for A1H, $E_C = E_A$ and then

$$\begin{aligned} |e''_C| &\leq \left|1 + e''_A \pm 2^{-d} e''_B\right| \\ Z_C &= \min(Z_A, Z_B + d) + \delta \\ R_C &\leq \left|1 + (e''_A \pm 2^{-d} \cdot e''_B)\right| \cdot 2^{Z_C} u. \end{aligned} \tag{12}$$

For A2H, $E_C = E_A - \min(2, \min(Z_A, Z_B + d) + \delta) = E_A - \Delta$ where $0 \leq \Delta \leq 2$ is the number of position left-shifted due to the partial normalization, then

$$\begin{aligned}
|e''_C| &\le \left|1 + (e''_A \pm 2^{-d} e''_B) \cdot 2^{\Delta}\right| \\
Z_C &= \min(Z_A, Z_B + d) + \delta - \Delta \\
R_C &\le \left|1 + (e''_A \pm 2^{-d} \cdot e''_B) \cdot 2^{\Delta}\right| \cdot 2^{Z_C} u = \\
&= \left|2^{-\Delta} + (e''_A \pm 2^{-d} \cdot e''_B)\right| \cdot 2^{\min(Z_A, Z_B + d) + \delta} u
\end{aligned} \tag{13}$$

For A3H, Z_C is the same as for A2H but the error due to rounding is $|e_{rd}| \le u \cdot 2^{E_C - \Delta}$, and then

$$\begin{aligned}
|e''_C| &\le \left|2^{-\Delta} + (e''_A \pm 2^{-d} e''_B) \cdot 2^{\Delta}\right| \\
R_C &\le \left|2^{-2\Delta} + (e''_A \pm 2^{-d} \cdot e''_B)\right| \cdot 2^{\min(Z_A, Z_B + d) + \delta} u
\end{aligned} \tag{14}$$

Comparing Eq. 12, Eq. 13, and Eq. 14, the relative errors among the non-normalized adders only differ in the first addend of the equations. In contrast to IEEE one, on which δ only affects the part of the error coming from the input operands, the number of leading zeros of the result increases both components of the error. For A2H, and A3H, the first addend is reduced when $\Delta > 0$ (partial normalization), specially for A3H. Moreover, the partial normalization reduces Z_C which will reduce the error in future operations.

Finally, let us study the multiplication operation $A^{Z_A} \cdot B^{Z_B} = C^{Z_C}$ for different multipliers, including IEEE one for comparison purpose. Let us suppose there is no overflow, underflow, or special cases. In this case, the intermediate result of the multiplication (without sign) is:

$$(M_A \cdot M_B + M_A \cdot e'_B + M_B \cdot e'_A + e'_A \cdot e'_B) \cdot 2^{E_A + E_B} \tag{15}$$

after normalization (if necessary) and rounding, and considering Eq. 2 to Eq. 5,

$$\begin{aligned}
|e_C| &\le |u \cdot 2^{E_C} + (e'_B \cdot 2^{-Z_A+1} + \ldots \\
&\ldots + e'_A \cdot 2^{-Z_B+1} + e'_A \cdot e'_B) \cdot 2^{E_A+E_B}| \\
|e'_C| &\le |u + (e'_B \cdot 2^{-Z_A+1} + \ldots \\
&\ldots + e'_A \cdot 2^{-Z_B+1} + e'_A \cdot e'_B) \cdot 2^{E_A+E_B-E_C}| \\
|e''_C| &\le |1 + (e''_B \cdot 2^{-Z_A+1} + \ldots \\
&\ldots e''_A \cdot 2^{-Z_B+1} + e''_A \cdot e''_B u) \cdot 2^{E_A+E_B-E_C}| \qquad (16) \\
R_C &\le |u + (e''_B \cdot 2^{-Z_A+1} u + \ldots \\
&\ldots e''_A \cdot 2^{-Z_B+1} u + e''_A \cdot e''_B u^2) \cdot 2^{E_A+E_B-E_C}| \cdot 2^{Z_C}.
\end{aligned}$$

Taking into account that $u = 2^{-m}$, in a typical case, the addend $(e''_A \cdot e''_B u^2)$ can be neglected and consequently

$$\begin{aligned}
|e''_C| &\le \left|1 + (e''_B \cdot 2^{-Z_A+1} + e''_A \cdot 2^{-Z_B+1}) \cdot 2^{E_A+E_B-E_C}\right| \\
R_C &\le \frac{\left|1 + (e''_B \cdot 2^{-Z_A+1} + e''_A \cdot 2^{-Z_B+1}) \cdot 2^{E_A+E_B-E_C}\right|}{2^{-Z_C}} u.
\end{aligned} \tag{17}$$

For the IEEE case, $Z_x = 0$ and taking into account that $E_C = E_A + E_B + \phi$ where $\phi = 1$, if there is an overflow in the fixed-point multiplication, and $\phi = 0$ otherwise,

$$\begin{aligned}
|e''_C| &\le \left|1 + (e''_B + e''_A) \cdot 2^{1-\phi}\right| \\
R_C &\le \left|1 + (e''_B + e''_A) \cdot 2^{1-\phi}\right| u.
\end{aligned} \tag{18}$$

Most likely $\phi = 0$ and consequently the errors of the input operands have an important impact on the final error.

For MH, the significand overflow is not checked, and then,

$$\begin{aligned}
E_C &= E_A + E_B + 1 \\
Z_C &= Z_A + Z_B - \phi \\
|e''_C| &\le \left|1 + (e''_B \cdot 2^{-Z_A} + e''_A \cdot 2^{-Z_B})\right| \\
R_C &\le \left|1 + (e''_B \cdot 2^{-Z_A+1} + e''_A \cdot 2^{-Z_B+1}) \cdot 2^{-1}\right| \cdot 2^{Z_A+Z_B-\phi} u \\
&= \left|2^{Z_A+Z_B} + (e''_B \cdot 2^{Z_B} + e''_A \cdot 2^{Z_A})\right| \cdot 2^{-\phi} u.
\end{aligned} \tag{19}$$

For M2H, the overflow is detected and consequently

$$\begin{aligned}
E_C &= E_A + E_B + \phi \\
Z_C &= Z_A + Z_B - 1 \\
|e''_C| &\le \left|1 + (e''_B \cdot 2^{-Z_A} + e''_A \cdot 2^{-Z_B}) \cdot 2^{-\phi+1}\right| \\
R_C &\le \left|1 + (e''_B \cdot 2^{-Z_A+1} + e''_A \cdot 2^{-Z_B+1}) \cdot 2^{-\phi}\right| \cdot 2^{Z_A+Z_B-1} u \\
&= \left|2^{Z_A+Z_B-1} + (e''_B \cdot 2^{Z_B} + e''_A \cdot 2^{Z_A}) \cdot 2^{-\phi}\right| u
\end{aligned} \tag{20}$$

Comparing both non-normalized multipliers (Eq. 19 and Eq. 20), as expected, the relative error is identical if a significand overflow occurs ($\phi = 1$). However, the addend corresponding to the rounding error for MH will be double than that of M2H when $\phi = 0$. Comparing with the IEEE, the number of leading zeros of the input operands has a strong impact on the final error. Considering the minimum possible, $Z_A = Z_B = 1$, only the first addend would be different, being double and quadruple than that of IEEE for M2H and MH, respectively. However, if the number of leading zeros increases, the relative error would increase exponentially.

4.2 Experimental error results

In this section, we analyze how using non-normalized FP impacts on the final accuracy when implementing several basic DSP algorithms. To do this, we followed the same procedure that we described in [6]. We designed the different non-normalized FP architectures 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 a 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. In this way, the main program runs in the processor, but the input data are converted to non-normalized format, and all additions and multiplications are performed in the coprocessor. When any other arithmetic operation is required, the non-normalized numbers are converted to conventional format, the operation is performed in the processor, and then, the results is converted back to non-normalized format.

We proceed as follows: the target algorithm is implemented using C programming language for double precision FP numbers (double) to use it as the reference implementation; moreover, the same function is implemented for both the IEEE 32-bit precision (float), using the regular processor, and the 32-bit non-normalized format, using the coprocessor; the results of both 32-bit implementations are

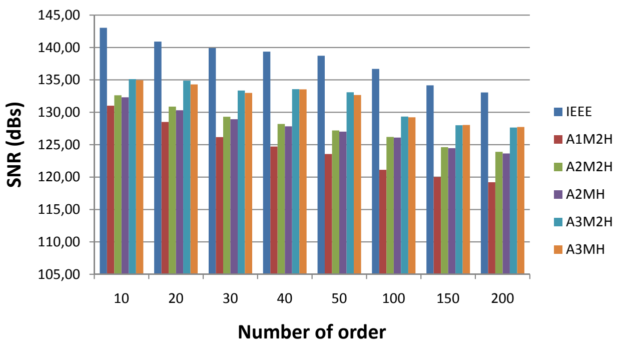


Fig. 3. SNR of FIR filters for different arithmetic architectures.

compared with the double precision results, calculating the error in terms of signal-to-noise ratio:

$$\mathrm{SNR}_{dB} = 10 \cdot \log_{10}\left(\frac{\sum y^2}{\sum (y-y')^2}\right), \tag{21}$$

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. Since the analysis that we provided in [6] clearly shows that truncation of the results is not enough to guarantee a minimum accuracy level, in this new study we only consider rounding to nearest through HUB approach [8]. Moreover, taking into account that partial normalization in the multiplier does not improve the error when the adder A2H is utilized [6], the multipliers MRx, which includes left shifting of the output significand using x-bit digits, and MLx, which performs left shifting of the output significand up to x-bit positions, are dismissed in this study. Hence, in this study, we tested the three HUB adders: the basic adder A1H, which only implements one-bit right shifting of the output significand, A2H which implements one-bit right shifting for the overflow of the significand and also up to 2-bit left shifting for partial normalization; and A3H which is very similar to A2H but the partial normalization uses the bits discarded in the significand alignment for filling the right bit positions. These adders are combined with two HUB multipliers: MH, the simplest multiplier without any shifting of the output; and M2H which only implements 1-bit right-shifting when an significand overflow occurs. Each coprocessor is named using the name of the adder utilized (without the H) followed by the name of the multiplier. For instance, the coprocessor A2MH uses adder A2H and multiplier MH. The variations of the SNR when using all these approaches are 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. 3 shows some results obtained for the same cut-off frequency and different numbers of orders. The main conclusion observing the results is that A3H improves very significantly the results obtained by A2H. On the other hand, as was reported in [6], M2H only improves very slightly the results of MH and A1H get much worst results than A2H. The mean of the difference respect the standard implementation is 13.95 dBs for A1M2H, 10.3 dBs for A2M2H and only 6.3 dBs for the A3M2H. We should note that this is almost the lost of precision expected for one bit reduction due to the lost of the implicit leading one (6 dBs).

TABLE 1
SNR obtained for Kalman filter implementations

IEEE	HUB		
146.2	A1H	A2H	A3H
MH	0.0	135.5	140.0
M2H	133.8	135.9	**140.3**

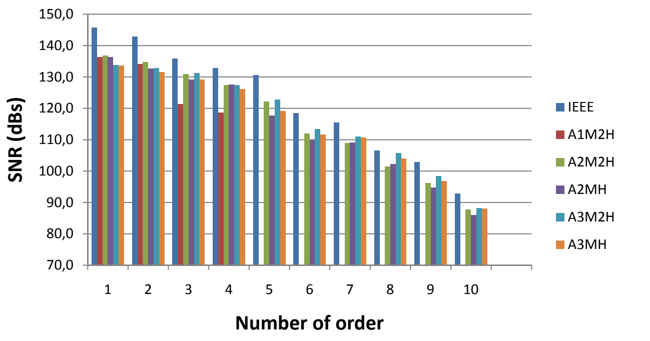


Fig. 4. SNR of IIR filters for different arithmetic architectures.

In another experiment, we use a very simple Kalman filter that approximates a constant value through a set of noisy measurements. As input signal is used a random number ranging between $[-0.1, 0.1]$ plus 0.5. For this algorithm, any kind of normalization in either the adder or multiplier is compulsory in order to obtain acceptable results [6]. Table 1 shows the SNR obtained for the architectures with the best results. Again the best result is achieved by A3M2H. It is also observed that the improvement of using A3H instead of A2H is very significant.

In another experiment, 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. As an example, Fig. 4 shows the SNR obtained for IIR direct-form I filters for the same cut-off frequency and different numbers of orders (direct-form II or other cut-off frequencies show a similar behavior). Similarly to previous examples, A3M2H obtains the best results, about 6 dBs below the IEEE results. However, in this case, A2M2H gets the second best results, showing that the improvement using M2H instead of MH is more relevant for this algorithm. We should also highlight that the performance of A1M2H collapse from order 5, which confirms that A1H adder only could be used for very specific applications.

Finally, we implemented a Cholesky decomposition algorithm for different matrix sizes. We used a thousand input matrices generated randomly on each experiment. Fig. 5 shows the mean of the SNR obtained for each matrix size. As in previous experiments, except FIR filters, the accuracy of A1M2H degrades very rapidly. This result confirms that adders should include at least a partial normalization. The

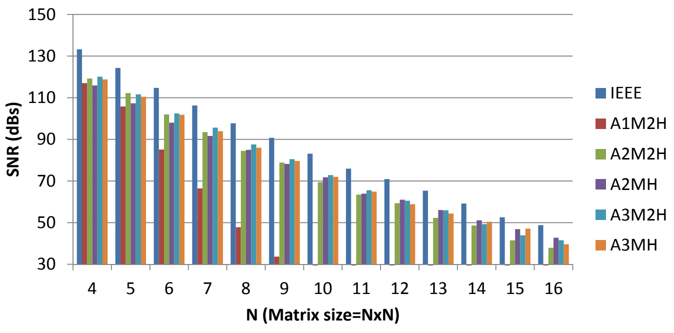


Fig. 5. SNR of Cholesky decomp. for different arithmetic architectures.

differences among the other non-normalized architectures are more subtle than in previous algorithms. Again A3M2H obtains the best results, followed very closely by A3MH. The difference of A3M2H respect to IEEE decreases with the matrix size, ranging from 13 dBs to 7 dBs, with a mean of 10 dBs.

5 Analysis of Implementation Results

5.1 ASIC implementation

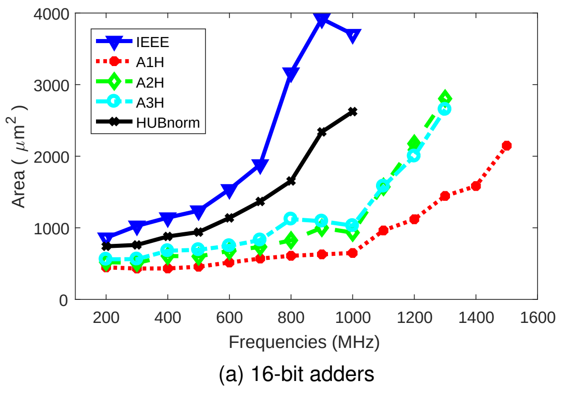
The proposed non-normalized architectures, along with the IEEE-754 and the normalized HUB [9] ones (denoted as IEEE and HUBnorm, respectively) have been designed using VHDL and synthesized using Synopsys Design Compiler Ultra (version H-2013.03-SP2) and the TSMC 65nm library. We considered default cell activity and "typical-case" operating conditions in which the temperature is $25°C$ and voltage $V_{dd} = 1.0V$. All architectures, including IEEE and HUBnorm, only implement rounding to nearest, and no special cases or denormals. The implementations have been conducted for both combinational and pipelined architectures except the converter, that was implemented only as combinational. From here on, we mean as non-normalized adders and multipliers to A1H, A2H, A3H and MH, M2H architectures.

5.1.1 Analysis of the combinational architectures

This subsection is intended to confirm the trend of the figures of merit of the combinational IEEE and non-normalized adders and multipliers presented in [6] with the new size binary16. Furthermore, the new adder A3H is compared to the previous ones. We carried out implementations increasing the target frequency in steps of 100MHZ, and present area and power consumption of each architecture.

Fig. 6 and Fig. 7 show the area and power consumption, respectively, of 16-bit and 32-bit adders. As expected, A2H and A3H have very similar area and power consumption since their architectures only differ in the bits introduced in partial normalization. A1H is the one with better implementation results whatever the width of the adder because no normalization is performed. It can be seen also that the IEEE and HUBnorm adders are not able to reach the same high frequencies as the non-normalized adders, since the normalization circuit is in the critical path.

Comparing the non-normalized adders with the IEEE one, the former have between 35%-84% less area and consume between 30%-87% less power for the 16-bit adders,



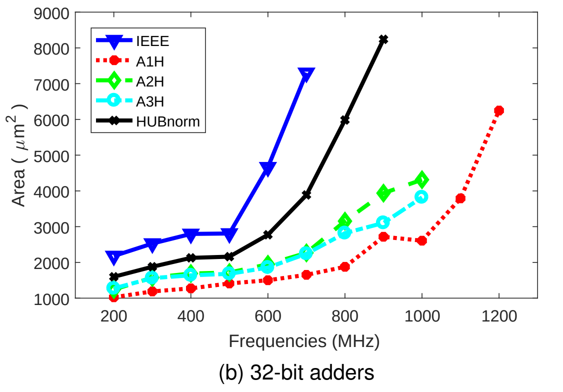
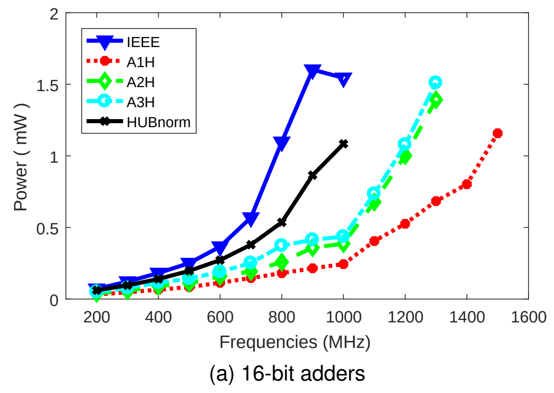


Fig. 6. Area of combinational adders of different bit-width.



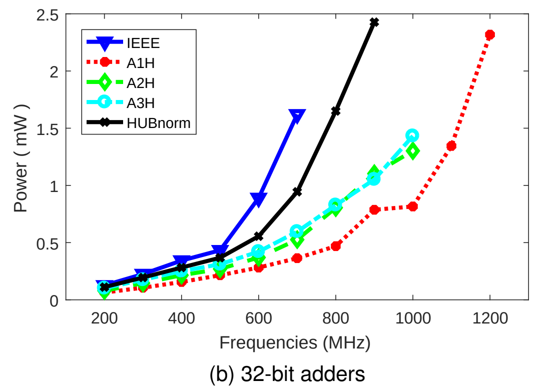


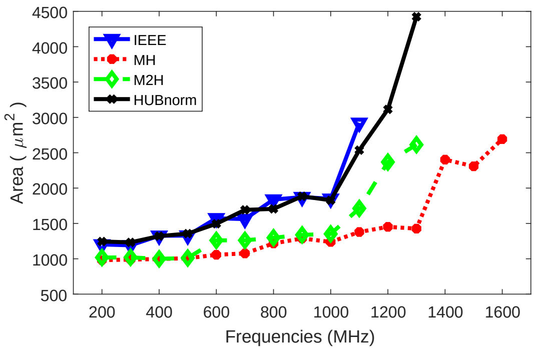
Fig. 7. Power consump. of combinational adders of different bit-width.

and 38%-78% less area, and 19%-78% less power for 32-bit adders. That confirms that non-normalized implementation achieves important savings for all bit-width. Furthermore, they reach much higher frequencies. On the other hand, comparing them with HUBnorm, the improvements are lower, but still very significant. For example, for 32-bit, on average A3H needs 34% less area and 27% less power consumption than the HUBnorm adder. That allows us to verify that the improvements obtained for the non-normalized adders are not only a merit of using HUB representation.

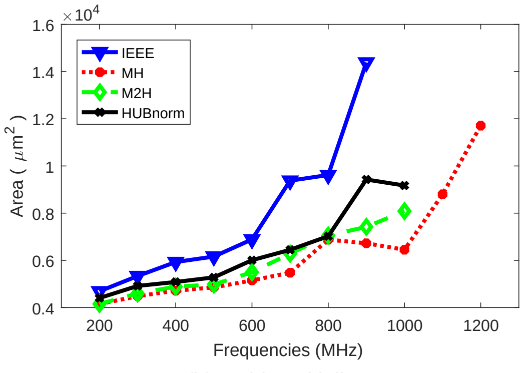
Similarly, Fig. 8 and Fig. 9 represent the area and power consumption of the combinational IEEE, MH, M2H and HUBnorm [9] multipliers. It can be seen that the improvement of non-normalized multipliers is not as great as for the adder case, because IEEE multipliers do not have normalization logic, and consequently the improvement comes mainly from the simplification of the rounding. For 32-bit designs, compared to the IEEE multiplier, MH reduces on average around 30% of area and 35% of power consumption, and M2H, 25% of area and 30% of power. Regarding the HUBnorm multiplier, note that it has practically the same architecture as M2H, except for the logic due to the implicit leading one. The improvement is only remarkable in the case of 16-bits (around 35% less area and power for MH, and 25% for M2H).

5.1.2 *Analysis of the pipelined architectures*

In order to study how the savings of the proposed architectures are affected when pipelining, we proceeded as follows: several registers were added at the output of the combinational datapath of each architecture and the tool Synopsys Design Compiler was directed to perform automatic retiming of the registers. We targeted several clock frequencies in the range of embedded applications and applied pipeline depths from two to five stages. Given a

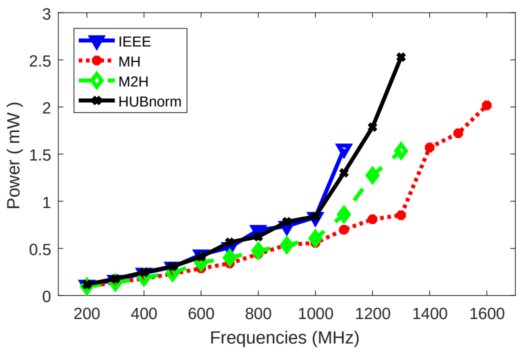


(a) 16-bit multipliers

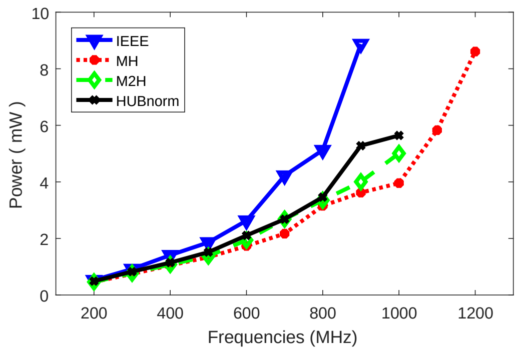


(b) 32-bit multipliers

Fig. 8. Area of combinational multipliers of different bit-width.



(a) Power of 16-bit multipliers



(b) Power of 32-bit multipliers

Fig. 9. Power consump. of combinational multipliers of different bit-width.

target frequency, and for each architecture, we selected the implementation with the minimum area. For these selected implementations, we provide their area, power consumption and number of pipeline stages, one stage meaning a combinational implementation.

Table 2 and Table 3 show the results corresponding to the 32-bit and 16-bit adders, respectively. It is observed that the area and power savings for non-normalized adders are still very significant when including pipelining and the number of pipeline stages are generally less than or equal to that of the others. The area reduction is very stable when varying the frequency, with a mean around 50% for A1H, and around 40% for A2H and A3H, for 32-bit width, and around 65% for A1H, and 47% for AH2 and AH3, for 16-bit width. The area reduction of HUBnorm is around 20% in both cases, which means that this reduction is due to the simplification of the rounding, but the rest is due to avoiding normalization. Regarding the power, the savings varies from one frequency to another. For instance, considering 32-bit architectures, they range from 18% to 73% for A1H, from 14% to 58% for A2H, and from 18% to 52% for A3H, with an average of 55%, 39%, and 35%, respectively. For 16-bit architectures, this power reduction is even higher with a mean of 71%, 54%, and 43% for A1H, A2H, and A3H, respectively.

Similarly, Table 4 and Table 5 show the results corresponding to the 32-bit and 16-bit multipliers, respectively. As expected, the improvements are lower than for the adders, but they are still significant. In all cases, non-normalized multipliers need the same number or fewer pipeline stages than the others. Compared to IEEE, the area reduction remains very similar for all frequencies, being on

TABLE 2
Number of stages (in parenthesis), area (μm^2) and power consumption (mW) for 32-bit adders for several frequencies

		400MHz	600MHz	800MHz	1GHz	1.2GHz
IEEE	area (stages)	2797 (1)	3194 (2)	4977 (2)	4979 (2)	5333 (2)
	power	0.34	0.88	1.71	2.19	2.88
A1H	area (stages)	1274 (1)	1502 (1)	1877 (1)	2608 (1)	3509 (3)
	power	0.16	0.28	0.47	0.82	2.37
A2H	area (stages)	1690 (1)	1948 (1)	3154 (1)	3217 (3)	3153 (2)
	power	0.22	0.37	0.81	1.88	1.94
A3H	area (stages)	1639 (1)	1857 (1)	2819 (1)	3118 (2)	3039 (2)
	power	0.25	0.42	0.83	1.79	2.10
HUBnorm	area (stages)	2128 (1)	2774 (1)	3824 (2)	3998 (2)	4498 (2)
	power	0.28	0.56	1.47	1.80	2.45

TABLE 3
Number of stages (in parenthesis), area (μm^2) and power consumption (mW) for 16-bit adders for several frequencies

		400MHz	600MHz	800MHz	1GHz	1.2GHz
IEEE	area (stages)	1142 (1)	1537 (1)	1890 (4)	2081 (2)	2500 (2)
	power	0.18	0.37	0.96	1.11	1.52
A1H	area (stages)	433 (1)	516 (1)	610 (1)	649 (1)	1120 (1)
	power	0.07	0.11	0.18	0.24	0.53
A2H	area (stages)	607 (1)	678 (1)	826 (1)	933 (1)	1358 (3)
	power	0.09	0.16	0.26	0.39	1.12
A3H	area (stages)	680 (1)	749 (1)	1061 (2)	1030 (1)	1381 (2)
	power	0.12	0.19	0.54	0.44	1.06
HUBnorm	area (stages)	880 (1)	1139 (1)	1462 (2)	1867 (2)	2036 (2)
	power	0.14	0.27	0.67	1.0	1.34

TABLE 4
Number of stages (in parenthesis), area (μm^2) and power consumption (mW) for 32-bit multipliers for several frequencies

		400MHz	600MHz	800MHz	1GHz	1.2GHz
IEEE	area (stages)	5930 (1)	6807 (2)	8196 (4)	8322 (2)	8726 (2)
	power	1.41	2.77	4.72	5.29	6.87
MH	area (stages)	4714 (1)	5151 (1)	6454 (2)	6348 (2)	6913 (2)
	power	1.05	1.73	3.21	4.01	5.23
M2H	area (stages)	4886 (1)	5499 (1)	6534 (2)	6569 (2)	7300 (2)
	power	1.08	1.94	3.29	4.11	5.44
HUBnorm	area (stages)	5084 (1)	6007 (1)	7023 (1)	7428 (3)	7984 (3)
	power	1.14	2.11	3.46	4.90	6.24

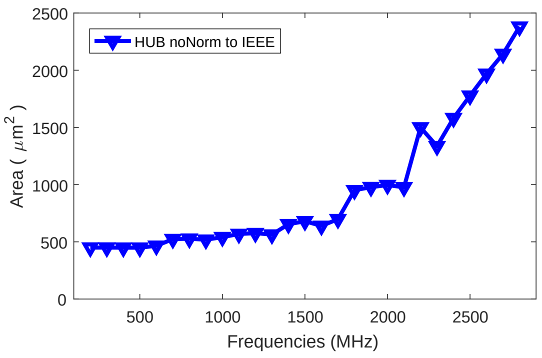
TABLE 5
Number of stages (in parenthesis), area (μm^2) and power consumption (mW) for 16-bit multipliers for several frequencies

		400MHz	600MHz	800MHz	1GHz	1.2GHz
IEEE	area (stages)	1326 (1)	1573 (1)	1838 (1)	1850 (1)	2408 (3)
	power	0.24	0.43	0.69	0.83	1.80
MH	area (stages)	997 (1)	1057 (1)	1218 (1)	1239 (1)	1452 (1)
	power	0.18	0.29	0.44	0.56	0.81
M2H	area (stages)	1000 (1)	1261 (1)	1298 (1)	1353 (1)	1822 (2)
	power	0.19	0.35	0.47	0.61	1.34
HUBnorm	area (stages)	1321 (1)	1496 (1)	1706 (1)	1828 (1)	2491 (2)
	power	0.25	0.41	0.62	0.84	1.74

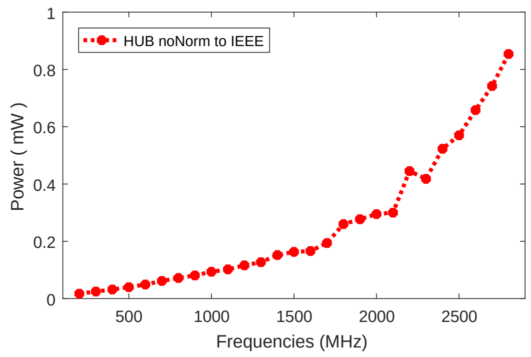
average 22% and 33% for MH, and 19% and 25% for M2H, for 32-bit and 16-bit multipliers, respectively. These figures are more variable and slightly higher for the reduction in power consumption, on average, 29% and 36% for MH, and 25% and 25% for M2H, for 32-bit and 16-bit multipliers, respectively. These savings are very similar when comparing with HUBnorm for 16-bit units. However, considering 32-bit multipliers, the mean area reduction compared to HUBnorm is 12% for MH, and 9% for M2H, and practically the same values for the reduction in power consumption.

5.1.3 Converter from non-normalized format to IEEE-754 binary format

In Section 3.3 we presented a converter from HUB non-normalized format to IEEE-754 one. Fig. 10 shows the implementation results of that converter for a 32-bit word length. As can be seen, the area and power are steadily low till 1.7GHz. As example, compared to the combinational A1H (the less consuming adder) the converter area is between 9% and 44% (about a 28% on average) of the A1H area and the power is between 5% and 27% (about a 16% on average) of the adder power.

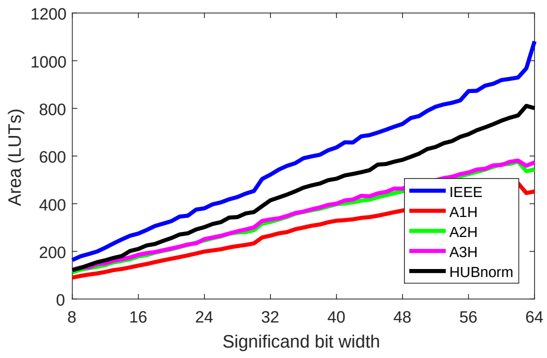


(a) Area of 32-bit converters

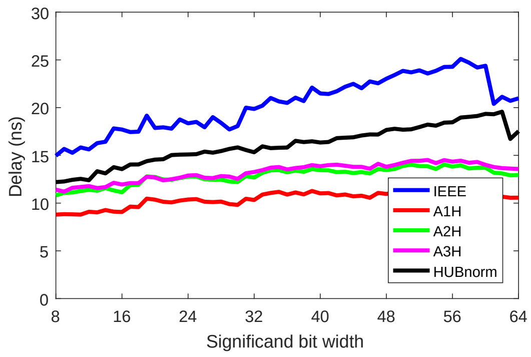


(b) Power of 32-bit converters

Fig. 10. Non-normalized to IEEE converters area and power consump.



(a) Area



(b) Delay

Fig. 11. Mean of adder results when varying the significand bit-width.

5.2 FPGA implementation

Nowadays, implementations using FPGA are found in many application-specific systems, specially if ad hoc hardware implementation is required. As showed in Section 2, the FP formats for FPGA, even the IEEE-like ones, allows free selection of exponent and significand bit-width. In this subsection, we analyze the implementation results for the proposed architectures considering this fact. Then, all the architectures have been synthesized using Xilinx ISE 14.3 targeting Xilinx Virtex-6 FPGA xc6vlx240t-1 for a significand bit-width ranging from 8 to 64 bits and an exponent bit-width from 5 to 12 bits. Next, the main results are shown.

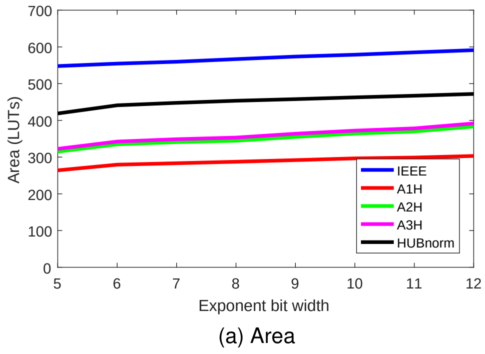
5.2.1 Adders

Although implementation results vary in each case (exponent-significand combination), the general behavior is similar when varying the exponent bit-width. In order to condense as much information as possible, Fig.11 represents, for each significand bit-width, the mean of the results of all utilized exponent bit-widths. As expected, A1H obtains the best results, since it removes completely the normalization and rounding logic, while A2H and A3H get similar figures, being A2H slightly better than A3H. Remember that A2H and A3H perform partial normalization up to 2 bits and they only differs in the bits introduced at the right when the significand is left shifted. In all cases the improvement compared to IEEE and HUBnorm architectures is very significant, due to the fully normalization logic that these units need.

Regarding the area, except for significand widths of around 8 and 64 bits, A1H achieves about a 50% reduction compared to IEEE-like adders, while the area reduction goes from about 35% to near 40% for both A2H and A3H when the significand bit-width increases, being always slightly higher for A2H. Similarly, the delay reduction goes up from 40% to 55% for A1H and from 25% to 45% for both A2H and A3H, although, in general, a little higher for A2H. Compared to HUBnorm, the improvement goes down, since HUBnorm also removes the rounding logic, but it is still very significant, being on average, around 35% for A1H and around 20% for A3H and A2H, for both area and delay.

It is also observed that close to 64-bit significand, the area goes up abruptly for the IEEE adders and, consequently, the delay falls as a result. This is automatically decided by the implementation tool, and it is not followed by the non-normalized architectures. As a consequence, area reduction increases up to 58%, 50% and 47% for A1H, A2H and A3H, respectively, while delay reduction goes down to 50%, 38% and 35%, respectively.

Similarly to previous figure, Fig. 12 shows, for each exponent bit-width, the mean of the results of all significand bit-widths. Whereas the relative area reduction (compared to IEEE) for A1H is not affected by the exponent bit-width (almost 50%), there is a decrease in this reduction for both, A2H and A3H, when the exponent bit-width increases, such as it goes down from 45% to 35%. We think that this is caused by the logic needed to update the exponent due to partial normalization. That behavior is similar when compared with HUBnorm. Analyzing the delay reduction, we have that, except for 5-bit and 12-bit exponents, where the reduction is higher, the relative delay reduction is only affected slightly for the exponent bit-width, being about



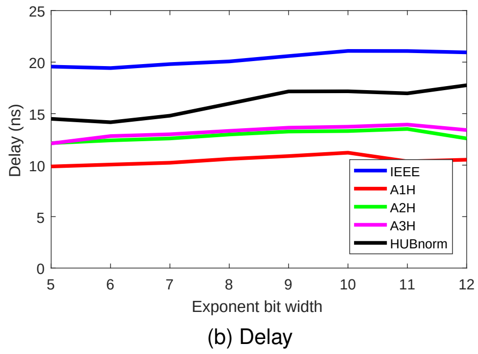
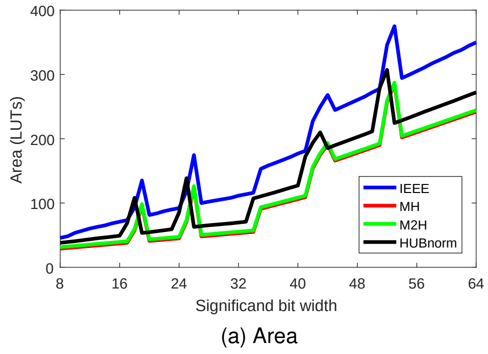


Fig. 12. Mean of adder results when varying the exponent bit-width.



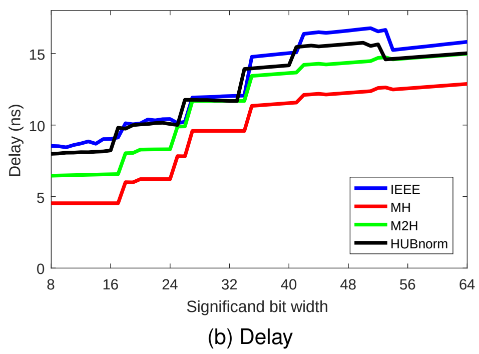


Fig. 13. Mean of multiplier results when varying the significand bit-width.

49%, 36% and 34% for A1H, A2H and A3H, respectively.

5.2.2 Multipliers

For the implementation of multipliers, we study the regular case where a mix of embedded multipliers and logic (LUTs) is automatically selected by the synthesizer software in each case. In the text below, we present only the area in terms of logic, since the number of embedded multipliers coincides for the four architectures except for several significand bit widths (specifically, for 19-,26-, 34-, 41-, 51-, 53-bit significand). In those cases, the tool adds new embedded multipliers, and since HUBnorm approach needs a bit more in the multiplier, the transition in the number of embedded multipliers for HUBnorm architectures occurs a bit earlier than for IEEE and non-normalized approaches. That is the reason for having different number of embedded multipliers at those bit-widths. Note that, although, at first, MH and M2H would also require a bit more due to using the HUB approach, they also need one bit less due to the lack of the implicit leading one, and consequently the number of bits of their multipliers is the same as for IEEE approach.

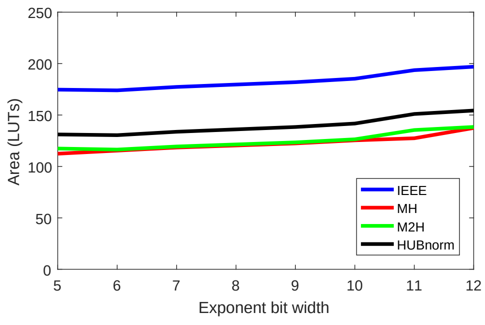
As for the adders, we present the average results to condense as much information as possible, but the specific gain should be studied in each case. For each significand bit-width, Fig.13 shows the mean of the results of all utilized exponent bit-widths. Similarly, Fig. 14 presents, for each exponent bit-width, the mean of the results of all significand bit-widths. Regarding the area, MH and M2H utilize practically the same area, because the LUTs used for shifting the significand in M2H are the same as the ones used for implementing the flush to zero when underflow occurs in MH and M2H. The area of non-normalized multipliers is lower than the area of IEEE in all cases. The area increases steadily with the exponent bit-width but depends more strongly on the significand bit-width. We have to clarify that the peaks in the lines of Fig. 13a corresponds to transitions in the number of embedded multipliers added by the tool. If we do not consider those transitions, the area reduction compared to HUBnorm goes from around 20% to 10%. Regarding IEEE multipliers, the area reduction when using non-normalized ones moves between 30% and 50%, being higher in the medium range of significand bit-width.

Regarding the delay, M2H is significantly slower than MH due to the shifting when there is a significand overflow, which is in the critical path. Again, the delay depends strongly on the significand bit-width but not as much on the exponent bit-width. In this case, there is not big difference between IEEE and HUBnorm. Higher relative speed improvements of non-normalized multipliers are achieved for significand below 24 bit (a mean of about 45% for MH and 23% for M2H). The average delay reduction compared to IEEE is about 29% and 12% for MH and M2H, respectively.

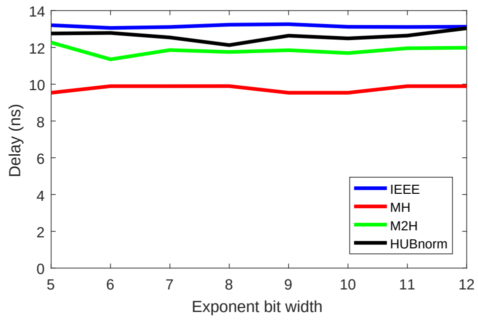
At this point, we should note that although area and speed improvement may seem lower for FPGA than for ASIC implementation, for the former, both reductions (in area and delay) are simultaneous. That is, the same architecture produces savings in area and increases the speed.

6 Conclusions

In this paper, we have broadened the study carried out in [6] about the impact of eliminating the normalization requirement in FP formats. Using non-normalized format allows designers to trade precision off for reducing hardware cost. We have proposed several non-normalized architectures for addition, multiplication and conversion. The effectiveness of these architectures has been studied in terms of precision and hardware implementation. The implementation



(a) Area



(b) Delay

Fig. 14. Mean of multiplier results when varying the exponent bit-width.

results have been analyzed for a wide range of sizes and frequencies, for both combinational and pipeline versions, and targeting both ASIC and FPGA implementation. In most of the cases, the savings are very significant with a moderate precision loss. From the results, we can conclude that although adder A1H obtains better implementation figures, it suffers the most from accuracy loss compared with the other adders. In contrast, it is with adder A3H that, in most of the cases, there is less accuracy loss. Furthermore, the cost of the implementation of A3H is similar to A2H for the combinational case, and a little better for the pipelined case. On the other hand, regarding the multipliers, MH is the one with the best implementation results, and, although M2H may improve the accuracy, this improvement is not very significant. Therefore, although it should be studied for each case, in general, the ideal combination of arithmetic units would be A3MH. Nonetheless, if the specific application has severe implementation restrictions and can afford greater loss of accuracy, then the ideal combination would be A1MH. We should highlight that using a non-normalized format has others important drawbacks besides loss 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. We hope these results encourage other researchers to investigate new options to cover the embedded system characteristics and open a debate on the necessity to define a new extension of the standard to cover embedded-system applications.

References

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

[2] N. Metropolis and R. L. Ashenhurst, "Significant digit computer arithmetic," *IRE Transactions on Electronic Computers*, no. 4, pp. 265–267, 1958.

[3] R. L. Ashenhurst and N. Metropolis, "Unnormalized floating point arithmetic," *Journal of the ACM (JACM)*, vol. 6, no. 3, pp. 415–428, 1959.

[4] 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.

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

[6] S. Gonzalez-Navarro and J. Hormigo, "Normalizing or not normalizing? an open question for floating-point arithmetic in embedded systems," in *Computer Arithmetic (ARITH), 2017 IEEE 24th Symposium on*. IEEE, 2017, pp. 188–195.

[7] 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

[8] 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.

[9] ——, "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.

[10] W. Liu, L. Chen, C. Wang, M. ONeill, and F. Lombardi, "Design and analysis of inexact floating-point adders," *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 308–314, Jan 2016.

[11] H. Zhang, W. Zhang, and J. Lach, "A low-power accuracy-configurable floating point multiplier," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, Oct 2014, pp. 48–54.

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

[13] 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.

[14] 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.

[15] 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.

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

[17] J. Muller, N. Brisebarre, F. de Dinechin, C. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.

Sonia Gonzalez-Navarro received the B.S. and M.S. degrees in Mathematics in 2000 and the Ph.D. degree in Computer Science in 2006, both from the University of Malaga, Spain. She is currently an assistant professor in the Computer Architecture Department at the University of Malaga. Her research interests include computer arithmetic, floating point number computation, high-performance architectures for data-intensive applications and near-data processing.

Javier Hormigo received the M.Sc. and Ph.D. degrees in telecommunication engineering from the Universidad de Malaga, Spain, in 1996 and 2000, respectively. He joined the Universidad de Malaga in 1997, where he is currently an Associate Professor with the Computer Architecture Department. His current research interests include computer arithmetic, specific application architectures, and field-programmable gate array.