Microprocessors and Microsystems 35 (2011) 535–546

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Customizing floating-point units for FPGAs: Area-performance-standard trade-offs

Pedro Echeverría *, Marisa López-Vallejo

Departamento de Ingeniería Electrónica, Universidad Politécnica de Madrid, Spain

ARTICLE INFO

Article history:
Available online 7 May 2011

Keywords:
Floating-point arithmetic
FPGAs
Library of operators
High performance

ABSTRACT

The high integration density of current nanometer technologies allows the implementation of complex floating-point applications in a single FPGA. In this work the intrinsic complexity of floating-point operators is addressed targeting configurable devices and making design decisions providing the most suitable performance-standard compliance trade-offs. A set of floating-point libraries composed of adder/subtracter, multiplier, divisor, square root, exponential, logarithm and power function are presented. Each library has been designed taking into account special characteristics of current FPGAs, and with this purpose we have adapted the IEEE floating-point standard (software-oriented) to a custom FPGA-oriented format. Extended experimental results validate the design decisions made and prove the usefulness of reducing the format complexity.

1. Introduction

Current deep sub-micron technologies allow manufacturing of FPGAs with extraordinary logic density and speed. The initial challenges related to FPGAs programmability and large interconnection capacitances (poor performance, low logic density and high power dissipation) have been overcome while providing attractive low cost and flexibility [1].

Subsequently, use of FPGAs in the implementation of complex applications is increasingly common but relatively new when dealing with floating-point applications ranging from scientific computing to financial or physics simulations [2–4]. This is a field of increasing research activity due to the performance and efficiency that FPGAs can achieve. The peak FPGA floating-point performance is growing significantly faster than the CPU counterpart [5] while their energy efficiency outperforms CPUs or GPUs [6]. Additionally, FPGA flexibility and inherent fine-grain parallelism make them ideal candidates for hardware acceleration improving GPUs capabilities for a particular set of problems with complex datapaths or control and data inter-dependencies [7]. FPGAs flexibility also allows the use of tailored precision, what can significantly improve certain applications. Furthermore, new FPGA architectures have embedded resources which can simplify the implementation of floating-point operators.

However, the large and deeply pipelined floating-point units require careful design to take advantage of the specific FPGA features. Designing this kind of application from scratch is almost impossible or makes the design cycle extremely long. Thus, the availability of complete and fully characterized floating-point libraries targeting FPGAs has become a must.

The IEEE standard for binary floating-point arithmetic was conceived to be implemented through custom VLSI units developed for microprocessors. However, if the target hardware is an FPGA, the internal architecture of these operators must be highly optimized to take advantage of the FPGA architecture [8]. Furthermore, implementations with slight deviations from the standard could be of great interest, since many applications can afore some accuracy reduction [3,9], given the important savings that can be achieved: reduced hardware resources and increased performance.

Several approaches have addressed the hardware implementation of a set of floating-point operators [10–12], but none of them includes the wide and general analysis carried out here. Some works only include the basic operators (adder/subtracter, multiplier, divider and square root) [10]. Other works focus on the implementation of particular floating-point operator implementations [11–13]. Regarding advanced operators (exponential, logarithm and power functions) few works can be found, standing out [14–16], with implementations of the exponential and logarithm functions.

In [8], the potential of FPGAs for floating-point implementations is exploited focusing on the use of internal fixed formats and error analysis what requires specific analysis for every application and supporting tools. Therefore, floating-point operators are mainly replicated in hardware without tailoring the format to the applications. In this scenario is where we have focused on, improving the performance of the floating-point units taking advantage of FPGA flexibility. We have tuned the architecture of floating-point operators to get the best performance-cost trade-off with slight deviations from the

* Corresponding author.
E-mail addresses: petxebe@die.upm.es (P. Echeverría), marisa@die.upm.es (M. López-Vallejo).

0141-9331/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2011.04.004

standard. This approach was also discussed in [17] but it is extended here in several ways:

- We have included advanced operators.
- A more complete set of deviations is studied.
- We perform an in-depth analysis of the implications of the deviations.
- We study the replicability of the operators.
- We provide a set of recommendations to achieve the resolution and accuracy of the standard with high performance.

Our proposed libraries include both conventional and advanced operators. Starting by an almost fully-compliant library[1] (`Std`), we have made several design decisions that allow clear improvements in terms of area and performance. These design decisions include the substitution of denormalized numbers by zero, the use of truncation rounding or the definition of specific hardware flags that allow the use of extended bit width internally.

The interest of this work is focused on the impact of those decisions over floating-point operators and not in presenting new architectures. Thus, the main contributions of this work are the following:

- A thorough analysis on the implications of the proposed design decisions has been carried out focusing on the performance-accuracy trade-offs. This is the base of a set of recommendations that can be considered when a complex floating-point application is implemented in configurable hardware.
- A complete set of varying accuracy floating-point libraries has been developed including both conventional (adder/subtracter, multiplier, divider and square root) and advanced (exponential, logarithm and power functions) operators.
- A systematic approach based on specific interfaces has been adopted allowing the use of extended bit widths. It simplifies the implementation of complex applications and reduces the resources needed for chained operators fitting in a single FPGA with better performance.
- Two final FPGA oriented libraries with significant hardware improvements have been implemented, taking advantage of the proposed design decisions.

The paper structure is as follows: Section 2 summarizes the floating-point format. Section 3 presents the key design decisions that are proposed while Section 4 describes the particular architecture of each operator. Experimental results are thoroughly discussed in Section 5, paying special attention to the influence of the proposed design decisions. Finally, Section 6 introduces a hardware library specially designed for FPGAs and Section 7 draws some conclusions.

2. Floating-point format IEEE 754

The IEEE Standard [18] is mainly designed for software architectures, usually using 32 bit words (single precision) or 64 bit words (double precision). Each word (Fig. 1) is composed of a sign (s, 1 bit), a mantissa (mnt, m_b bits) and an exponent (exp, e_b bits), being the value of a number:

$$s \times mnt' \times 2^{exp'} = s \times h \cdot mnt \times 2^{exp-bias} \quad (1)$$

where h is an implicit bit known as the hidden bit and the *bias* is a constant that depends on e_b being its value $2^{e_b-1}-1$. With this number representation the floating-point format can represent

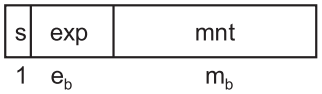

Fig. 1. Floating-point word.

zeros, infinities, exceptions (Not a Number, NaN) and two number types, normal ones (normalized) and numbers very close to zero (denormalized). The differentiation among these five types is based on the exponent and mantissa values. Table 1 depicts all possible combinations of exponent and mantissa values.

The standard is specifically designed to handle these five number types sharing a common format while maximizing the total set of numbers that are represented. Combining these two facts increases the complexity of the arithmetic units because, in addition to the calculation unit itself, it is needed a preprocessing (also known as prenormalization) of the inputs numbers and a postprocessing (also known as postnormalization) of the output numbers, see Fig. 2.

Therefore, when implementing a floating-point operator, the hardware required is not only devoted to the calculation unit itself, additional logic is needed just to handle the complexity of the format. This logic represents a significant fraction of the area of a floating-point unit, a 48% of logic in average for the studied operators, as will be shown in Section 5. In a general way preprocessing logic includes:

- Analysis of the number type of the inputs, which includes exponent and mantissa analysis.
- Determination of operation exceptions due to the number type or sign of the inputs (square root, logarithm).
- Normalization of inputs.
- Conversion of inputs to the format required by the calculation unit.

Table 1
Types of floating-point numbers.

Type	Exponent	Mantissa	h	value
Zero	0	0	–	±0
Denormalized	0	≠0	0	Eq. (1)
Normalized	1 to $2^{e_b}-2$	–	1	Eq. (1)
Infinities	2^{e_b-1}	0	–	±∞
NaN	2^{e_b-1}	≠0	–	–

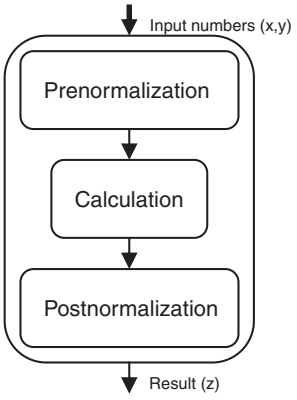

Fig. 2. Floating-point operator.

[1] Only some software issues as exception handing with additional flags and signaling NaNs (not a number) are not implemented.