Synthesis Time Reconfigurable Floating Point Unit for Transprecision Computing

Giulia Stazi, Federica Silvestri, Antonio Mastrandrea, Mauro Olivieri, and Francesco Menichelli

Dept. of Information Engineering, Electronics and Telecommunications (DIET)
Sapienza University of Rome, Via Eudossiana, 18, 00184 Roma, Italy
`{stazi,mastrandrea,olivieri,menichelli}@diet.uniroma1.it`

Abstract. This paper presents the design and the implementation of a fully combinatorial floating point unit (FPU). The FPU can be reconfigured at implementation time in order to use an arbitrary number of bits for the mantissa and exponent, and it can be synthesized in order to support all IEEE-754 compliant FP formats but also non-standard FP formats, exploring the trade-off between precision (mantissa field), dynamic range (exponent field) and physical resources.
This work is inspired by the consideration that, in modern low power embedded systems, the execution of floating point operations represents a significant contribution to energy consumption (up to 50% of the energy consumed by the CPU). In this scenario, the adoption of multiple FP formats, with a tunable number of bits for the mantissa and the exponent fields, is very interesting for reducing energy consumption and, simplifying the circuit, area and propagation delay. Adopting multiple FP formats on the same platform complies with the concept of *transprecision computing*, since it allows fine-grained control of approximation while meeting the required constraints on the precision of output results. The designed FPU has been tested in order to evaluate the correctness of all supported operations, and implemented on a Kintex-7 FPGA. Experimental results are provided, illustrating the impact and the benefits derived by the use of non-standard precision formats at circuit level.

Keywords: Floating Point Unit, Low Power Consumption, Approximate computing, Transprecision Computing

1 Introduction

The execution of FP operations in most embedded applications emerges as a major contributor to system energy consumption. In [1] experimental results show that 30% of the energy consumption is due to FP operations and an additional 20% is caused by moving FP operands from memory and registers and viceversa. Approximate computing [2–5] is an emerging technique which proposes to relax the specifications on precise computation allowing digital systems to introduce errors implied by imprecise hw/sw, and trading off quality, in terms of computational accuracy, for energy consumption or speed. According to another paradigm, transprecision computing [6], low power embedded systems should be designed to deliver the required precision for computation. In this scenario several works try to overcome the limitations of fixed-format FP types: for example in [7, 8] multi-precision arithmetic software libraries for performing calculations on number with arbitrary precision are proposed. In [1] the authors present a transprecision FPU capable of handling 8-bit and 16-bit operations in addition to the IEEE-754 compliant formats.

In this paper, we describe the design and the implementation of a fully combinatorial and reconfigurable FPU, supporting IEEE-754 and also capable of working with reduced precision formats, characterized by an arbitrary number of bits for the mantissa and the exponent.

2 Floating Point representation, IEEE-754 standard

The IEEE floating point standard (IEEE 754) [9] is a technical standard for floating-point computation, established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE).

$$X = (-1^s) \times 1.m_b \times b^e \tag{1}$$

According to Equation 1 a floating point number consists of three fields:a sign bit (s), a biased exponent(e) and a mantissa (m).

The IEEE standard for *half precision* floating-point numbers has 1 sign bit, 5-bit exponent, and 11-bit mantissa, for *single precision* has 1 sign bit, 8-bit exponent, and 23-bit mantissa and finally for *double precision* has 1 sign bit, 11-bit exponent, and 52-bit mantissa. The advantage of floating-point over fixed-point is the range of numbers that can be represented with the same number of bits. In addition, the floating point IEEE-754 standard has defined representations for zero, negative and positive infinity, and NaN (Not a Number).

3 Reconfigurable Floating Point Unit

The primary goal of this work is the design and implementation of a fully combinatorial and reconfigurable FPU using VHDL hardware description language at Register Transfer Level (RTL). The unit allows to perform operations on floating point numbers in any format with a maximum length of 64 bits and with an arbitrary number of bits dedicated to the exponent and to the mantissa fields. In this way, it is possible to support floating point operations fully compliant with those defined by the IEEE-754 standard (double-precision, single-precision and half-precision) and operations with reduced precision formats. The FPU has been made reconfigurable at synthesis time by declaring the length of all signals and variables as function of two generic types, m and e, used respectively for defining the length of the mantissa and the exponent.

The hardware architecture has been designed trying to satisfy the following targets: (a) reduced area occupation; (b) low power consumption; (c) maximum propagation speed. The FPU is fully combinatorial, in order to be inserted in a CPU 1-cycle pipeline stage. The implemented operations are: sum, subtraction, multiplication, conversion from floating point to integer and from integer to floating point.

3.1 Floating_Point_Unit_core

Fig. 1 and Fig. 2 represent, respectively, the *Floating_Point_Unit_core* internal diagram and external interface. The FPU does not support operations between denormalized numbers because they would led to a larger use of logic and therefore of area occupation in exchange for a marginal increase in accuracy. The approach is indeed the same one used in the VFP of ARM processors: all denormalized numbers in input to the FPU are directly approximated to 0. The other FPU input signals are:

- *operation*, which encodes the type of operation to be executed;
- *control*, which configures the unit before performing computational tasks; in particular it assigns the sign during integer to float conversion and it determines whether exceptions are enabled on the status port .

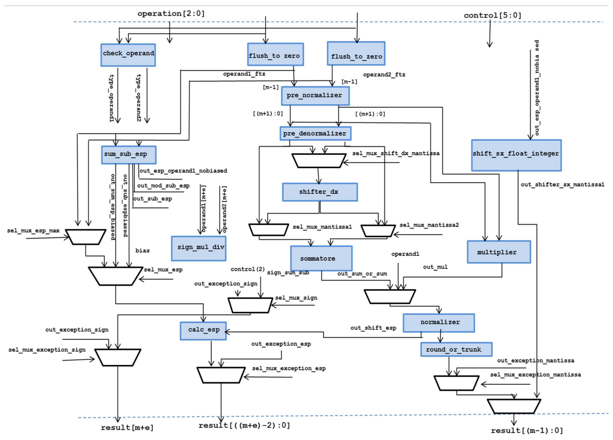

Fig. 1. Floating Point Unit Core architecture

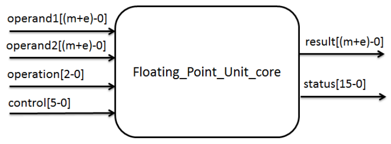

Fig. 2. External interface of FPU core

The output signals are:

- *result*, which provides the result of the performed operation. Again, its length depends on the number of digits for the exponent and the mantissa (e and m parameters).

Table 1. List of analyzed formats

#bit sign	#bit exponent	#bit mantissa	Total bits
1	5	10	16 (IEEE half precision)
1	6	11	18
1	5	12	18
1	6	13	20
1	5	14	20
1	7	16	24
1	6	17	24
1	8	23	32 (IEEE single precision)
1	10	37	48
1	9	38	48
1	11	52	64 (IEEE double precision)

– *status*, which contains information regarding exceptions that have occurred and that are appropriately flagged.

The FPU supports the following exceptions: inexact result, invalid operation, underflow and overflow. When a case of underflow is detected, the result is approximated to 0, when instead an overflow occurs, the result is approximated to infinity. As far as the exception of an invalid operation is concerned, the result is set to *NaN*.

4 Experimental Results

4.1 Testing

The testing phase, performed immediately after the FPU design, represented an important step in order to make sure that the computational unit operates according to its design specifications and produces the correct results. To verify the behavior of the designed core for a variety of inputs, a testbench has been built. The testbench inserts input test vectors, automatically generated by a C program, into the FPU and then compares the results processed by the computational core with the output produced by the C program (using hardware FP). All validation was performed simulating the FPU at behavioral level using *Modelsim SE 10.1c*[10].

Considering a FP number of predefined length, the partition of bits between the mantissa and the exponent fields has an impact on the represented numbers, enforcing a trade-off between dynamic range and precision; in particular the number of bits in the exponent field affects the range of numbers that can be represented while in the mantissa field modifies the precision of the represented number. We centered our analysis on reduced precision formats from a minimum of 16 bits up to 64 bits; this choice is supported by the consideration that single precision IEEE format is often not necessary for applications in embedded domains while IEEE half precision can be affected by underflow/overflow problems. In particular, Table 1 shows the list of standard and non-standard formats analyzed in this section.

4.2 Synthesis and implementation

The next step was the synthesis and the implementation of the FPU core on Kintex 7 FPGA through*Xilinx Vivado Design Suite*[11]. Since the FPU is fully combinatorial, it does not have an input clock signal. As timing constraint for the project, a *virtual clock*, which is not connected to any design object, was used. To obtain data that reflect the effective gate area occupation, we run synthesis with hardware FPGA DSP block disabled, leaving the synthesizer with the possibility of using only the remaining logic blocks. This approach was required in order to compare the area occupied by different FP formats, which otherwise would have been implemented using DSP block using fixed FP formats. Table 2 shows the results gathered from the *Utilization report.* This report has been produced after the implementation of the FPU core at 40 MHz clock speed (25 ns is the minimum period obtained for the single precision FP format) and it collects data regarding number of LUTs and slices used in the unit. It can be seen that the reduced precision formats allow to significantly limit hardware resources. Propagation speed results, extrapolated from the *Timing Summary Report* produced after each implementation, are illustrated in Table 3. For half precision and for precisions between 18 and 20 bits, propagation speed remains constant at 20 ns, 25% better than single precision. Single precision presents a propagation speed of 25ns, decreasing the operating frequency to 40 MHz; finally double precision introduces an increase in propagation time of about 45% with respect to the best case.

Table 2. Resources @ 40MHz clock with DSP disabled

Precision	**#slice**	**#LUT**	**% resources rel. to single precision**
half	191	624	37.9%
m11_e6	198	664	39.2%
m12_e5	209	730	41.5%
m13_e6	238	796	47.2%
m14_e5	243	807	48.2%
m16_e7	299	1037	59.3%
m17_e6	238	1114	67.0%
single	504	1787	100%

Table 3. Propagation delay for reduced precision formats

Precision	**Propagation delay [ns]**
half	20
m11_e6	20
m12_e5	20
m13_e6	20
m14_e5	20
m16_e7	21
m17_e6	21
single	25
m37_e10	25
m38_e9	25
double	29

5 Conclusions

In this paper, the design of a FPU, synthesizable with arbitrary precision formats, was presented. We showed that a FPU with reduced precision is a good solution for low-power and low-cost microprocessor systems. The savings in terms of resource occupation, for the analyzed formats, range from about 38% for m17_e6 format and reach about 63% for m11_e6 format with respect to single precision. Moreover, reducing precision has also considerably decreased propagation delay. In particular, the propagation delay on reduced-precision implementations was about 20 ns, with a gain of about 25% and 45% with respect to single and double precision formats.

In future works, synthesizing the FPU on ASIC will allow more accurate estimate of area occupation and speed gain and will also add an estimate and a comparison on power consumption, which was revealed not reliable using FPGA as target.

References

1. G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini, "A transprecision floating-point platform for ultra-low power computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018.* IEEE, 2018, pp. 1051–1056.
2. J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS).* IEEE, 2013, pp. 1–6.
3. G. Stazi, F. Menichelli, A. Mastrandrea, and M. Olivieri, "Introducing approximate memory support in linux kernel," in *Ph. D. Research in Microelectronics and Electronics (PRIME), 2017 13th Conference on.* IEEE, 2017, pp. 97–100.
4. F. Menichelli, G. Stazi, A. Mastrandrea, and M. Olivieri, "An emulator for approximate memory platforms based on qemu," in *International Conference on Applications in Electronics Pervading Industry, Environment and Society.* Springer, 2016, pp. 153–159.
5. G. Stazi, L. Adani, A. Mastrandrea, M. Olivieri, and F. Menichelli, "Impact of approximate memory data allocation on a h.264 software video encoder," in *International workshop on Approximate and Transprecision Computing on Emerging Technologies (ATCET).* Springer, 2018.
6. A. C. I. Malossi, M. Schaffner, A. Molnos, L. Gammaitoni, G. Tagliavini, A. Emerson, A. Tomás, D. S. Nikolopoulos, E. Flamand, and N. Wehn, "The transprecision computing paradigm: Concept, design, and applications," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018.* IEEE, 2018, pp. 1105–1110.
7. D. H. Bailey, H. Yozo, X. S. Li, and B. Thompson, "Arprec: An arbitrary precision computation package," 2002.
8. L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "Mpfr: A multiple-precision binary floating-point library with correct rounding," *ACM Transactions on Mathematical Software (TOMS)*, vol. 33, no. 2, p. 13, 2007.
9. W. Kahan, "Ieee standard 754 for binary floating-point arithmetic," *Lecture Notes on the Status of IEEE*, vol. 754, no. 94720-1776, p. 11, 1996.
10. M. Graphics, "Modelsim-advanced simulation and debugging," 2012.
11. T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.