



# Multiplication by rational constants

Florent de Dinechin

## ► To cite this version:

Florent de Dinechin. Multiplication by rational constants: LIP research report 2011-3. 2011. ensl-00610328v1

**HAL Id: ensl-00610328**

**<https://ens-lyon.hal.science/ensl-00610328v1>**

Submitted on 21 Jul 2011 (v1), last revised 15 Nov 2011 (v3)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Multiplication by rational constants

*LIP research report 2011-3*

Florent de Dinechin

LIP, Université de Lyon (ENS-Lyon/CNRS/INRIA/UCBL)

46, allée d'Italie, 69364 Lyon Cedex 07

{Florent.de.Dinechin}@ens-lyon.fr

**Abstract**—Multiplications by simple rational constants often appear in fixed-point or floating-point application code, for instance in the form of division by an integer constant. The hardware implementation of such operations is of practical interest to FPGA-accelerated computing. It is well known that the binary representation of rational constants is eventually periodic. This article shows how this feature can be exploited to implement floating-point multiplication by a rational constant in a number of additions that is logarithmic in the precision. An open-source implementation of this technique is provided, and is shown to be practically relevant for constants with small numerators and denominators, where it provides improvements of 20 to 40% in area with respect to the state of the art. It is also shown that for such constants, the additional cost for a correctly rounded result is very small, and that correct rounding very often comes for free in practice.

**Index Terms**—multiplication by a constant, floating-point, reconfigurable computing

## I. INTRODUCTION

Multiplication by constants has received much attention in the literature, especially as many digital signal processing algorithms can be expressed as products by constant matrices. In such cases the constants are typically irrational (e.g. the square roots of unity in a Fast Fourier Transform).

The context of the present work is quite different. We are interested in porting applications into optimized architectures for FPGAs, and in this process we want to optimize the arithmetic operators when possible. Such applications often involve multiplications by constants, and this article studies how to implement them as efficiently as possible. Our initial motivation was floating-point divisions by 3 and by 9 appearing in stencil applications, but this work covers multiplications by arbitrary rational constants  $a/b$ . However, we should already point out that, at least in the hardware context, this study is relevant mostly for small values of  $a$  and  $b$ , “small” meaning here that they fit on few bits.

There are two contributions in this article. The first is a systematic exploitation of an empirical observation made by Gustafsson and Qureshi in [1]. We present an algorithm that builds multipliers by rational constants based on their periodic representation. These multipliers use a number of additions that is asymptotically logarithmic in the precision of the constant. This logarithmic complexity (which is optimal [2]) for a large class of constants is interesting in its own right, in a field where little is known in terms of theoretical complexity [3]. In practice, however, exploiting

periodicity is useful only when the period is short. We show in Section II how to compute this period for an arbitrary rational, and in Section III how to build optimal adder trees out of it.

The second contribution is to show in Section IV that correct rounding of the result (that is, obtaining the correctly rounded product of  $X$  by  $a/b$ , and not of  $X$  by some finite fixed-point or floating-point approximation to  $a/b$ ) comes at very little overhead for our “small” rationals: it requires to consider only a few more bits of the binary representation of the constant. Actually, these bits come often for free in the optimal architectures of Section III.

This technique has been implemented as the `FPCConstMultRational` operator in the open-source `FloPoCo` arithmetic core generator [4]. Section V provides some data for correctly rounded multipliers by  $1/3$ ,  $1/9$  and  $7/5$ , showing area savings up to 40% with respect to the previous state of the art.

## II. ON THE PERIODICITY OF THE BINARY REPRESENTATION OF RATIONAL NUMBERS

The most usual system of representing numbers is the position system, where a number is represented by a sequence of digits, and each digit is weighted by powers of some radix, usually 10 (decimal system) or 2 (binary system).

In such a system, any rational number  $a/b$  has an eventually periodic representation. This is true in decimal ( $1/3 = 0.3333\ldots$ ,  $1/9 = 0.1111\ldots$ ) but also in binary ( $1/3 = 0.010101\ldots$ ,  $1/9 = 0.1111000011110000\ldots$ ). Numbers with a finite decimal representation can be viewed as a special case where the periodic pattern is composed of zeroes, for instance  $0.5 = 0.5000\ldots$ . A number with a finite decimal representation may have an infinite binary one, for instance  $1/5 = 0.2_{10} = 0.001100110011\ldots$ . The opposite is not true, due to the fact that two divides ten.

The following lemma tells us which numbers have a purely periodic binary representation:

**Lemma 1.** *Let us consider an irreducible fraction  $c/d$ , where 2 divides neither  $c$  nor  $d$ . If  $c < d$ , then the binary representation of  $c/d$  is purely periodic, i.e. it starts with an occurrence of the periodic pattern.*

The condition that 2 divides neither  $c$  nor  $d$  is related to the fact that powers of two, in binary, correspond to shifts which, for the purpose of multiplication by  $a/b$ , may be handled separately in a trivial way.

If  $c > d$  the Euclidean division of  $c$  by  $d$  gives us  $c = hd + c'$ , and we may rewrite  $c/d = h + c'/d$ . For the purpose of multiplying an input  $x$  by the constant  $c/d$ , we therefore have  $cx/d = hx + c'x/d$ . Existing literature addresses the multiplication by the finite integer constant  $h$ , so we may focus on the multiplication by the purely periodic constant  $c'/d$ .

The following lemma (which seems to have very old decimal roots [5]) allows us to compute the periodic pattern:

**Lemma 2.** *Let  $c/d$  be an irreducible fraction, where  $c < d$ , and 2 divides neither  $c$  nor  $d$ . The size  $s$  of its period is the multiplicative order of 2 modulo  $d$ , i.e. the smallest integer such that  $2^s \bmod d = 1$ . The periodic pattern is the integer  $p = \lfloor 2^s c/d \rfloor$ .*

*Proof:* By definition of  $s$  we have  $2^s = kd + 1$  for some integer  $k$ . Therefore  $p = \lfloor 2^s c/d \rfloor = \lfloor \frac{(kd+1)c}{d} \rfloor = \lfloor kc + c/d \rfloor = kc$  since  $c/d < 1$ . We deduce that

$$2^s c/d = p + c/d,$$

where the recursive occurrence of  $c/d$  exactly expresses the periodicity of the fraction  $c/d$ . ■

**Examples:**

- $1/3$  has period size  $s = 2$  because  $2^2 \bmod 3 = 1$ . The pattern is  $\lfloor 1 \times 2^2/3 \rfloor = 1$ , which we write on 2 bits 01, and we obtain that

$$1/3 = 0.(01)_2^\infty ;$$

- $5/9$  has period size  $s = 6$  because  $2^6 \bmod 9 = 1$ . The pattern is  $\lfloor 5 \times 2^6/9 \rfloor = 35$ , which we write on 6 bits 100011, and we obtain that

$$5/9 = 0.(100011)_2^\infty.$$

Thanks to these lemmas, algorithm 1 determines the periodic binary representation of a fractional number  $a/b$ . This representation consists of 4 integers:

- $s$  the period size in bits, a positive integer,
- $p$  the periodic pattern, a positive integer that we will usually write in binary,
- $h$  the header, a positive integer, also typically written in binary,
- $e$  the scaling factor exponent, or shift.

such as

$$a/b = 2^e \left( h + \sum_{i=1}^{+\infty} \frac{p}{2^{si}} \right).$$

Let us now exploit this representation to build a multiplier of a variable  $x$  by a fraction  $a/b$ .

### III. PERIODICAL SHIFT-AND-ADD TREES

In this section, we input the periodic representation of a rational constant  $a/b$ , and also a precision  $w_0$ , which is the number of bits of  $a/b$  that have to be considered for the multiplication. The value of  $w_0$  typically expresses the accuracy requirements of the floating-point or fixed-point context. For instance, section IV will define the value of

**Algorithm 1** Computing the periodic representation of a rational  $a/b$  as a tuple of integers  $(e, h, p, s)$ .

---

```

1: procedure PERIODICREPRESENTATION( $a, b$ )
2:    $(c, d) \leftarrow \text{SIMPLIFY}(a, b)$ 
3:    $e \leftarrow 0$ 
4:   while  $c \bmod 2 = 0$  do
5:      $c \leftarrow c/2$ 
6:      $e \leftarrow e + 1$ 
7:   end while
8:   while  $d \bmod 2 = 0$  do
9:      $d \leftarrow d/2$ 
10:     $e \leftarrow e - 1$ 
11:  end while
12:  (Now  $a/b = 2^e c/d$  with both  $c$  and  $d$  odd)
13:   $h \leftarrow c/d$  (header)
14:   $c \leftarrow c \bmod d$  (periodic part)
15:
16:  (Now  $a/b = h + c/d$  with  $c < d$ )
17:
18:   $s \leftarrow 1$  (period size)
19:  if  $d = 1$  then ( $a/b$  has a finite binary representation)
20:     $p \leftarrow 0$  (periodic pattern)
21:  else ( $a/b$  has an infinite binary representation)
22:     $t \leftarrow 2$  (Invariant of the loop below:  $t = 2^s$ )
23:    while  $t \bmod d \neq 1$  do
24:       $s \leftarrow s + 1$ 
25:       $t \leftarrow 2t$ 
26:    end while
27:     $p \leftarrow ct/d$  (periodic pattern)
28:  end if
29:  return  $(e, h, p, s)$ 
30: end procedure

```

---

$w_0$  that ensures correct rounding for a given floating-point format.

In [1], Gustafsson and Qureshi suggested trying to represent a real constant on more than  $w_0$  bits if it leads to a shift-and-add architecture with fewer additions. They indeed mention the fact that, due to their periodic representation, rational constants are good candidates for exploiting this idea. However, they do not try to exploit this idea systematically, and do not consider the issue of correct rounding. With the notations of previous section, let us define

$$\pi_0 = 2^{-s} p x .$$

The  $2^{-s}$  factor simply scales the integer  $p$  to an approximation of  $c/d < 1$ , so  $\pi_0$  is an approximation of  $cx/d$ . We may then compute increasingly accurate approximations of  $cx/d$  as

$$\pi_1 = \pi_0 + 2^{-s} \pi_0 ,$$

$$\pi_2 = \pi_1 + 2^{-2s} \pi_1 ,$$

and in general

$$\pi_{i+1} = \pi_i + 2^{-2^i s} \pi_i$$

so that  $\pi_i$  is the product of  $x$  by an approximation of  $c/d$  of size  $2^i s$  bits.

Therefore, a constant corresponding to  $2^i$  repetitions of the period may be built in  $i$  additions, and this is optimal [2]. Let us now consider the details, including further optimizations. We note  $w_h$  the size in bits of the header  $h$ . We need to compute  $hx + cx/d$ , where  $cx/d$  is the periodic part. If  $w_0$  is the precision to which  $h + c/d$  must be represented, then  $c/d$  must be represented at least on  $w_0 - w_h$  bits.

First, one of the existing methods is used to build  $px$  and, if  $h \neq 0$ ,  $hx$ . As these two constants should be small for the method to be relevant, exhaustive exploration techniques [6], [7] may be used to perform this step optimally. Such methods lead to less than 4 additions for any  $h$  or  $p$  of size smaller than 12 bits, and less than 5 additions for sizes smaller than 19 bits. The FloPoCo implementation currently uses the simpler heuristic presented in [8] which, in our experiments, also consistently computes the minimal architecture, due to the fact that both  $h$  and  $p$  are very small integers for the simple rationals that have motivated this work.

Then we may compute the  $\pi_i$ . In this process, we may stop as soon as  $2^i s \geq w_0 - w_h$ . However, it is usually possible to implement a smaller last addition. Let  $i$  be such that  $2^i s < w_0 \leq 2^{i+1} s$ : we must compute the  $\pi_j$  for  $0 \leq j \leq i$ . Let  $j$  be the smallest integer such that  $(2^i + 2^j)s \geq w_0 - w_h$ . As  $j \leq i$ ,  $\pi_j$  is already computed, and the last stage may compute

$$f = \pi_j + 2^{-2^i s} \pi_i$$

(another option would be to compute  $f = \pi_i + 2^{-2^j s} \pi_j$ , but this would lead to a larger adder [8]).

If  $h = 0$ , this is all. If  $h \neq 0$ , we still have to add  $hx$ . This product is itself computed using a classical constant multiplier, in parallel to the computation of the fractional product. There are two possible parenthesing of the two final additions:  $r = (hx + \pi_j) + 2^{-2^i s} \pi_i$  or  $r = hx + (\pi_j + 2^{-2^i s} \pi_i)$ . We may assume that the computation of  $hx$  has a depth strictly smaller than that of  $\pi_i$  (which should be the case for “small” rationals). With this assumption, as soon as  $j < i$ , the first parenthesing leads to a shallower tree, and is therefore preferred. If  $i = j$ , the second parenthesing will be preferred when it leads to a smaller overall number of full adders.

Figure 1 illustrates the resulting architecture on the example of  $a/b = 7/5$  for single precision, with a target precision is  $w_0 = 28$  (providing correct rounding according to Section IV). The smallest value of  $i$  such that  $2^i s \geq 28$  is  $i = 3$ . We don’t find in this case a smaller  $j$  such that  $(2^{i-1} + 2^j)s \geq 28$ . For this simple example the product  $px$  is computed in one addition only, while the product  $hx$  is computed in zero additions.

This figure also illustrates a small additional optimization: we trim, whenever possible, leading and trailing zeroes from the various sub-constants to minimize datapath width. For instance, for  $a/b = 7/5$ , the period is  $p = 0110_2$ , but  $\pi_0$  is actually computed as  $x \times 11_2$  and the two zeroes are added only when performing the shifts. The final result is actually one bit more accurate than it seems, since there is one more trailing zero to the truncated constant. These

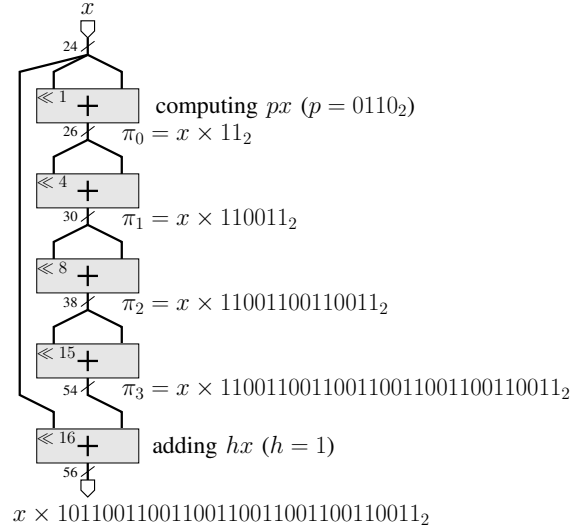


Fig. 1. Multiplication of a 24-bit mantissa by  $7/5$  (period  $p = 0110_2$ , header  $h = 1$ )

technical details are taken into account by the generator in FloPoCo.

#### IV. CORRECT ROUNDING

It turns out that for the small rational constants for which this method is of interest (“small” meaning small  $a$  and small  $b$ ), obtaining a correctly rounded result is also fairly cheap. More specifically, the following theorem holds. It is, in essence, a generalization of the “exclusion lemma” used to prove that some division algorithms are correctly rounded [9].

**Theorem 1.** *Let  $p$  be the precision of the input  $X$  and  $q$  be the precision of the result  $R$ , and assume  $q \geq p$ . If  $C$  is a constant obtained by truncating the binary representation of  $a/b$  to at least  $q + 1 + \lceil \log_2 b \rceil$  bits, then rounding the product  $CX$  to the nearest floating-point number of precision  $q$  is equivalent to rounding the exact product  $\frac{a}{b}X$ .*

Note that this theorem covers the most useful case when the input and output precisions are identical.

*Proof:* For reasons already invoked, we may assume without loss of generality that  $X \in [1/2, 1)$ , and that both  $a$  and  $b$  are odd. Let us use the integral significand representation of the input  $X$ :  $X = \frac{I}{2^p}$  where  $I$  is an integer. We want to show that  $\frac{a}{b}X$  cannot be too close to the mid-point between two floating-point numbers in the result format. Such a mid-point  $M$  can be written  $M = \frac{2J+1}{2^{q+1}}$ .

The distance between  $\frac{a}{b}X$  and  $M$  is therefore written:

$$\begin{aligned} \left| \frac{a}{b}X - M \right| &= \left| \frac{a}{b} \frac{I}{2^p} - \frac{2J+1}{2^{q+1}} \right| \\ &= \left| \frac{2^{q+1-p}aI - (2J+1)b}{2^{q+1}b} \right| \end{aligned}$$

Here,  $2^{q+1-p}aI$  is an even integer since  $q \geq p$ . On the other hand,  $(2J+1)b$  is an odd integer, as the product of two

constant	$p$	This work		using [8]		depth
		$p_c$	#FA	$p_c$	#FA	
<b>1/3</b> $h = 0$ $p = 01_2$	24	32	118	27	190	4
	53	64	317	56	368	5
	113	128	792	116	1026	6
<b>1/9</b> $h = 0$ $p = 000111_2$	24	30	132	29	131	5
	53	60	356	58	408	6
	113	120	885	118	1116	7
<b>7/5</b> $h = 1$ $p = 0110_2$	24	33	139	28	193	5
	53	65	366	57	595	6
	113	129	900	117	1507	7

TABLE I  
SIZE (IN FULL-ADDERS) AND DEPTH (ALSO NUMBER OF ADDERS) OF  
THE SIGNIFICAND MULTIPLIERS FOR SOME CORRECTLY ROUNDED  
RATIONAL CONSTANT MULTIPLIERS. THE PRECISIONS CHOSEN ARE  
THOSE OF THE IEEE754-2008 FORMATS.

odd integers. We deduce that their difference is at least one, hence

$$\left| \frac{a}{b}X - M \right| \geq \frac{1}{2^{q+1}b}$$

This defines an “exclusion zone” around mid-points. Following a classical argument [8], if we compute  $R \approx \frac{a}{b}X$  such that  $|R - \frac{a}{b}X| < \frac{1}{2^{q+1}b}$ , rounding  $R$  to  $q$  bits is then equivalent to rounding  $\frac{a}{b}X$ . Truncating the infinite representation of  $\frac{a}{b}$  to the precision  $2^{-q-1-\lceil \log 2b \rceil}$  provides this accuracy. ■

## V. RESULTS

Table I provides some Full Adder (FA) counts for the proposed multipliers obtained thanks to the FloPoCo implementation, compared to the previous implementation in [8]. Incidentally, FloPoCo generates test-benches along with the operators [4], and the correct rounding claim has been verified through exhaustive simulation for several constants and several precision.

In all the cases of Table I, the previous approach from [8] already builds a tree of optimal depth, but not of optimal size. Specifically, it does build sub-constants which are repetitions of the period, but the numbers of repetition are not always powers of two, which prevents reusing them optimally.

We should point out again that these operators, with one single rounding error, are more accurate than rounding  $a/b$  to a floating-point number  $f$  of the same format as the result, then multiplying by  $f$ . For multiplier by  $1/b$ , they are functionally equivalent to a correctly rounded division by  $b$ , which would consume much more resources.

## VI. CONCLUSION

The periodic binary representation of rational constants can be usefully exploited to build efficient hardware for the multiplication by such constants. An implementation of this idea in the open-source FloPoCo core generator is demonstrated. This technique is mostly relevant for constants  $a/b$  where both  $a$  and  $b$  are small integers, and in this case correct rounding of the multiplication by the

infinitely accurate rational constant comes at a minor overhead: On most of the examples studied, correct rounding is for free. An important application of this technique is the implementation of divisions by small integers.

It could be interesting to study if variations of this technique could not be used to implement division by small integers in multiple-precision packages such as MPFR.

## Acknowledgements

Thanks to B. Pasca and A. Plesco for bringing up this question, and to N. Brisebarre for his lights on number theory.

## REFERENCES

- [1] O. Gustafsson and F. Qureshi, “Addition aware quantization for low complexity and high precision constant multiplication,” *IEEE Signal Processing Letters*, vol. 17, no. 2, pp. 173–176, 2010.
- [2] O. Gustafsson, “Lower bounds for constant multiplication problems,” *IEEE Transactions On Circuits And Systems II: Express Briefs*, vol. 54, no. 11, pp. 974 – 978, Nov. 2007.
- [3] V. Dimitrov, L. Imbert, and A. Zakaluzny, “Multiplication by a constant is sublinear,” in *18th Symposium on Computer Arithmetic*. IEEE, 2007, pp. 261–268.
- [4] F. de Dinechin and B. Pasca, “Designing custom arithmetic data paths with FloPoCo,” *IEEE Design & Test of Computers*, Aug. 2011.
- [5] J. W. L. Glaisher, “Periods of reciprocals of integers prime to 10,” *Proc. Cambridge Philos. Soc.*, vol. 3, pp. 185–206, 1878.
- [6] A. Dempster and M. Macleod, “Constant integer multiplication using minimum adders,” *Circuits, Devices and Systems, IEE Proceedings*, vol. 141, no. 5, pp. 407–413, 1994.
- [7] O. Gustafsson, A. G. Dempster, K. Johansson, and M. D. Macleod, “Simplified design of constant coefficient multipliers,” *Circuits, Systems, and Signal Processing*, vol. 25, no. 2, pp. 225–251, 2006.
- [8] N. Brisebarre, F. de Dinechin, and J.-M. Muller, “Integer and floating-point constant multipliers for FPGAs,” in *Application-specific Systems, Architectures and Processors*. IEEE, 2008, pp. 239–244.
- [9] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhauser Boston, 2009.