JANUS, an On-line Multiplier/divider for manipulating large numbers.

Alain GUYOT¹, Yvan HERREROS¹ and Jean-Michel MULLER¹,².

¹Laboratoire TIM3-IMAG, INPG, 46 Avenue Félix Viallet, 38031 Grenoble Cedex, FRANCE.
²CNRS, Laboratoire LIP-IMAG, Ecole Normale Supérieure de Lyon, 46 Allée d’Italie, 69364 Lyon Cedex 07.

Abstract.
This paper deals with the detailed VLSI implementation of a fast bit-serial operator designed to perform very high precision (600 decimal digits) additions, multiplications and divisions, and some of the applications of the circuit.

Introduction.
Our aim is to present a VLSI implementation of an on-line multiplier/divider unit able to manipulate quite large numbers (up to 600 decimal digits). On line arithmetic, presented by Ercegovac and Trivedi in 1977 [6] is a digit serial arithmetic where the digits circulate from the most significant to the least significant. Digit serial arithmetic allows digit-level pipelining (which enable fast computation) and circulation of data most significant digit first is necessary in order to perform some computations like division or maximum of two numbers.

Since in classical number systems the carries propagate from the least significant position to the most significant one, on-line arithmetic needs the use of carry-free redundant number systems, like Avizienis’s signed-digits systems [1] or carry-save notation. Frequently, the radix chosen is different from 2 since a carry-free addition algorithm, due to Avizienis [1], may be used in radix r ≠ 2. For instance, the paste-up system, presented by Irwin and Owens in [15] uses radix 4. In radix 2, carry-free addition is possible, but with two inconveniences: the algorithm seems more complicated, and the delay is larger (see [2] for proof). We shall show here that the first inconvenience vanishes if we choose a good binary representation of the digits in radix-2 signed digit notation, and we shall chose radix 2 for our circuit.

The multiplier uses a three-input parallel adder, a three-input serial adder and a one-digit multiplier which are described first. The divider uses the multiplier to compute the least significant digits of the partial remainder, plus four slices to compute its four most significant digits in a nonredundant form. For clarity reasons, the multiplier is drawn with three digits only, the generalization to any number of digits is straightforward. Most significant digits are at the left. Multiplication and Division are performed following Ercegovac and Trivedi’s algorithm [6].

A. Number representation.
Notation. To avoid the carry propagation delay in addition, a signed binary digit (SBD) notation has been adopted. Each SBD \(x\), which is \(\overline{1}\), 0 or 1 is represented by a couple (a,b) of negative and positive bits referred to as (-/+) such that \(x = a - b\). So \(\overline{1}\) is (1,0), +1 is (0,1) and 0 is either (0,0) or (1,1). To change the sign of a SBD, one can either permute its two bits or logically complement them. This remark is worth a lot of transistors since most of the elementary combinatorial operations on SBD, like digit multiply or add, are increasing logic functions and the simple restoring gates provided by the technology are decreasing. This remark about change of sign is also useful to change an adder into a subtractor without changing any gate.

B. The operators
B.1 The PPM operator.
This operator computes \(d\) and \(e\) from its three-input \(a, b\) and \(c\), that gives either \((a + b - c)\) or \((c - a - b)\), depending on the sign affected to the inputs and the outputs. It is used to build a carry-free parallel (2 gives 1) addition, a carry-save parallel addition (CSA) (3 gives 2) as well as a serial addition.
Most significant bits first serial addition.

A and B are fed in serially, boxes labeled D are one cycle delays (master-slave flip-flops). By combining this circuit with a one SBD CSA and two boxes D, the three-input serial adder used in the multiplier is built.

**B.2 SBD multiplier.**

The choice of radix 2 allows the product of two SBD to be one SBD.

**On-line multiplier.**

In this picture, double wires carry SBD, while the boxes labeled L are latches to hold digits of A and B. Before computation, there are all cleared (set to zero). During computation a control bit (token) runs backward to load serially the incoming SBD in the A and B latches. The detailed timing of the serial computation of A times B is explained on 3 bits in the table below.
B.3 SBD Divider.

We implement the algorithm of Eregovac and Trivedi [6]. As shown on fig. 7 the divider consists of two parts. The right part is the same multiplier as described in fig. 6. It multiplies the partial quotient (signed) by the partial dividend, adding one bit to both of them at each clock cycle, and add the signed result to the partial remainder PR (without carry propagation on up to 2048 positions). The right part of the circuit propagates a carry on the 4 most significant positions of the partial remainder in order to simplify comparisons, and from its value predicts the next SBD of the quotient. The division algorithm is listed below. Let us assume that there is a radix point between the two parts of the circuit, i.e., the right part works on the fractional part of the numbers while the left part works with the integer parts. This is purely conventional since the divider does not have to know the real weight of the digits. In the algorithm the divider is assumed to be positive, each of its SBD as well as the result has to be complemented if it turns out to be negative.

Algorithm [6]

While $D < 8$ do

1. Start with the four bits known part $D$ of the divisor is big enough to ensure the convergence of the division algorithm, i.e., $7 < \text{divisor} < 16$ or $8 \leq D \leq 15$.
2. Start multiplying.
3. Loop until dividend is exhausted.

   1. Get next SBD from dividend into multiplier.
   2. $PP := PR \cdot 2 + 0$; 
      $PN := PR \cdot 2 + 0$; 
      Next SBD of quotient := $PP - PN$;
      [PN and PP are the 2 bits of the SBD]
      [next SBD of quotient goes to the multiplier and is output from the circuit]
      $PR := PR + \text{incoming digits from multiplier} + \text{next digit from divisor}$;
      $\text{if PR then } PR - D\text{ else if PN then } PR + D\text{ else PR }$;
      $\text{if } PR < 2 \cdot D \text{ then }$.
5. End.

Eventually, the product is computed as follows:

\[
\begin{align*}
(a_2 b_2 a_1 b_1 a_0 b_0) \cdot 2 & = (a_2 b_2 a_1 b_1 a_0 b_0) + (a_2 b_2 a_1 b_1 a_0 b_0) \cdot 2 \\
& = (a_2 b_2 a_1 b_1 a_0 b_0) + (a_2 b_2 a_1 b_1 a_0 b_0)
\end{align*}
\]

Implementation considerations.

The circuit fig. 6 is assembled from three instances of the identical SBD slice. By adding more slices a circuit for any precision can be obtained, no resizing of the transistors being necessary, since each slice communicates only with the next ones, so the size of the slices can be kept small. Besides the non recoding SBD addition is far less demanding for transistors than the previous solutions known to the authors [10] and few costly master-slave flip-flops are used. Altogether, each slice contains 32 gates and 152 transistors, so the 600 decimal digit (2048 SBDs) multiplier would cost a little over 300k transistors, with a very high regularity.
C. Synchronization
From the on-line adder of fig. 4, the multiplier of fig. 6 and the divider of fig. 7 and a few switches, it is easy to assemble a programmable operator which is slightly more complex than the divider and can perform any of the operations. This circuit is designed to be controlled by a Transputer via the links. As in the Transputer, the flow of data is regulated by a token that goes backward to request for data.

![Fig. 7 On line divider.](image)

![Fig. 9 Example of data sharing.](image)

The circuit is fitted with two fifo for the input operands to delay one operand when the other is not yet available or when the weights have to be aligned, for addition and subtraction. This feature allows sharing of data.

**Example.** Let us compute \((x+b)x\) (x is shared).

It is easy to see that if the fifo is longer than the delay of the addition, regulation for \(x\) in the multiplier is performed through the adder. Duplication of data is allowed provided that it is used to compute the same result.

D. Implementation and performances.
We have already designed (as a test circuit) a 64 digit SBD multiplier. The floorplan and the layout are given in Fig. 10 and 11. For each cycle, less than 10 gates are passed through. With a delay of 3ns per gate, a frequency of 30MHz can be expected that is 30 million digits of result per second. Provided that enough circuits are pipelined and that numbers fit in up to 600 decimal digit, this figure is almost independent of the complexity of the expression.
Fig. 11 Layout.

Fig. 10 Organization of the circuit.
References


