# Design of a Systolic Coprocessor for Rational Addition

Tudor Jebelean
RISC – Linz
A-4040 Linz, Austria (Europe)
tudor@risc.uni-linz.ac.at

**Abstract**

We design a systolic coprocessor for the addition of signed normalized rational numbers. This is the most complicated rational operation: it involves GCD, exact division, multiplication and addition/subtraction. In particular, the implementation of GCD and exact division improve significantly (2 to 4 times) previously known solutions. In contrast to the traditional approach, all operations are performed *least-significant digits first*. This allows bit-pipelining between partial operations at reduced area-cost. An Atmel FPGA design for 8-bit operands consumes 730 cells (3,500 equivalent gates) and runs at 25 MHz (5 MHz after layout). For 32-bit operands this would be in the same timing range as the software solutions, however, a significant speed-up can be expected for longer operands because the linear time-complexity of the hardware algorithms.

## 1 Introduction

The operations with rational numbers (also referred as *arbitrary precision arithmetic*) are the basic building block for algebraic computations [5]. In fact, in many typical algebraic calculations (e. g. solving systems of polynomial equations) the underlying arithmetic consumes sometimes more than 80% of the computing time [16, 6, 2]. During the last decades, research in computer arithmetic focused on fixed precision (floating point) computations, which are used in numerical calculations [18]. However, we are currently witnessing a fast increase of interest in exact computations in various areas of mathematics and engineering [4, 7]. Developing coprocessors for arbitrary rational arithmetic is therefore important both from the practical and the theoretical point of view. Although parallel [systolic] algorithms for the various integer operations have been studied for a long time, only recently some attempts at building specialized hardware for rational arithmetic have been performed [9, 10, 17]. We approach the problem of *addition* because it is the most complicated: it contains all the integer operations. We consider *normalized* inputs (the numerator and the denominator are relatively prime). This contrasts with the approach in [14], were the inputs are not considered normalized. Also, the design will work for *signed* rational numbers (i. e. the numerator is signed and the denominator is positive).

The approach is *bit serial, least-significant digits first* (LSF) and systolic. This contrasts with the previous mentioned attempts, which work MSF, using redundant representation. LSF computations need simpler algorithms, because the classical complement representation is used. This leads to improvements both in area and in speed. We employ *purely* systolic algorithms, that is all the communications are local. Moreover, communication with the host takes place only at the lower end of the systolic array. This leads to better performance in area and speed when large circuits are implemented. This performance improvement is quite significant especially when one uses FPGAs. Also,

*Supported by Austrian Forschungsförderungsfonds (FWF), project P10002-PHY.

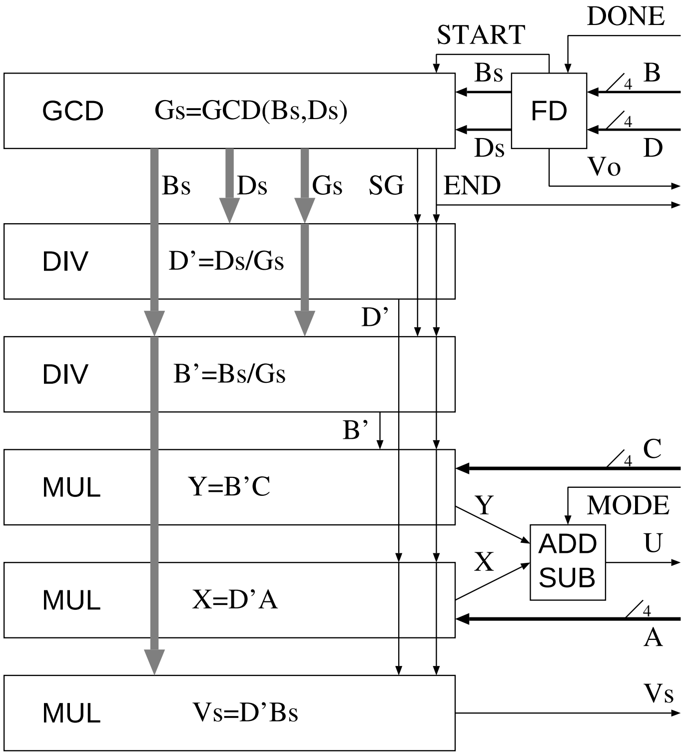


Figure 1: Structure of the systolic device.

this makes the design fully scalable by simple tiling of identical components, and without modification of the clock cycle. These characteristics are essential for arbitrary precision arithmetic.

The next section describes the **overall structure** of the design: during the serial input of the denominators the computation of their greatest common divisor (GCD) starts. After that two parallel exact division are performed, while their results are piped to three parallel multiplication units, whose results are piped through an add/subtract cell to the output. The numerators are serially fed-in during multiplication. In time, the input of the denominators overlaps with the first half (in average) of GCD computation, while the exact divisions, numerators input, multiplications, and results output overlap completely. The **GCD array** is an improvement of [13] (25% area reduction, 75% time reduction), using a special processor-clustering technique. Also, in contrast with the previous approach, the input is serially fed-in during computation. The **exact division array** is a significant improvement of [12] (area reduction 75%). In contrast with the parallel/serial division array of [15], which has similar area consumption and speed, our design is parallel/parallel, taking advantage of the fact that the operands are already in the array. The **multiplication algorithm** is an adaptation of the parallel/serial one presented in [15], but we add one additional pipe for serial feeding of the parallel operand. This gives a serial/serial multiplier which has the same functionality as [1], but with a simpler design. Because the products are obtained is LSF bit serial form, the **addition/subtraction** can be performed by a full-adder with carry feedback. We use a cell which operates as adder or subtracter depending on a selection signal.

## 2 Rational addition

Let $A/B$ and $C/D$ be the normalized operands, and suppose one wants to compute $U/V = A/B + C/D$. Due to characteristics of our systolic algorithms we will consider shifted operands. Let $k$ be the largest integer such that $2^k$ divides both $B$ and $C$ (in other words, $k$ is the number of common trailing zero bits), and let $B_s = B/2^k, D_s = D/2^k, G_s = GCD(B_s, D_s), V_s = V/2^k$ be the shifted integers. Based on [11] we consider the following algorithm:

[1] $G_s \leftarrow GCD(B_s, D_s)$
[2] $B' \leftarrow B_s/G_s, \quad D' \leftarrow D_s/G_s$
[3] $X \leftarrow AD', Y \leftarrow CB'$
[4] $U \leftarrow X + Y$
[5] $V_s \leftarrow B_s D'$

Note that the divisions in [2] and [3] are *exact*, i. e. the remainder is known in advance to be null. Also, the subtraction can be performed by the same algorithm, by changing only the operation [4]. If the rational inputs are signed, that means $A$ and $C$ are signed, than only operations [3] and [4] are signed. The structure of the device is presented in fig. 1, and can be seen as a *sandwich* of systolic arrays. The first array computes the GCD $G_s$ of $B_s$ and $D_s$. The inputs are fed in bit-serial and LSF manner to the lower end of the array, 4 bits at a time. These inputs are computed by a pass-through constant-area feeder FD, which skips the common least-significant zero bits. These bits also constitute the least significant part of $V$. Through the wire $V_0$, the coprocessor informs the host how long this part is. Information about the length of the inputs is given by the host by setting the DONE signal at the last bit of each input. If the inputs have $n$ bits, than the GCD array will need $2n$ steps (in average) to deliver the result, the first $n$ steps are overlapping in time with the input operation. The result $G_s$ is delivered in bit parallel manner, together with the values of $B_s$ and $D_s$. The GCD array signals the end of the computation on the wire END. This triggers the beginning of the subsequent computations. Also, a bit SG is used to indicate the sign of $G_s$. (Due to characteristics of the GCD algorithm, the result could appear in its negative form, and SG signals the need to complement it.) $G_s, B_s$ and $D_s$ are fed in parallel into the 2 division arrays DIV. In fact, the 2 arrays overlap in the handling of $G_s$, which gives some area reduction. The signals END and SG are piped through the division arrays, and not sent globally like in [12]. The division begins to deliver the results $B', D'$ right away, in bit-serial LSF manner, one bit at a time. The results $B', D'$ of the exact divisions are than fed in serial manner, one bit at a time, into the 3 multiplication arrays MUL. $A$ and $C$ are also fed in serially during this multiplications, LSF 4 bits at a time. $A$ and $C$ are signed, represented in the usual 2's complement form. The start of the input operation is triggered by the array using END. The multiplication $B_s D'$ is done in parallel/serial manner. All the products are available right away in serial LSF manner, one bit at each cycle. The denominator $V_s$ goes directly to output, while $X, Y$ are piped through the adder/subtracter ADD/SUB, whose operation mode is selected by the host using MODE.

# 3 Greatest Common Divisor

The algorithm for the computation of the GCD is a slight modification of the *plus-minus* Brent-Kung algorithm [3]. Let us denote by $A$ and $B$ the inputs to the GCD algorithm an by $a_1, a_0, b_1, b_0$ their least-significant bits. At the beginning the common trailing null bits of $A, B$ are shifted out, then if $a_0 = 0$ the operands are interchanged. After that, a loop continues until $B = 0$:

- if $b_0 = 0$ then $B$ is shifted one position;
- otherwise $(A, B)$ is replaced by $(B, (A \pm B)/2)$ – the operation $\pm$ is $+$ iff $a_1 \neq b_1$.

The systolic design is different from the one presented in [13] in the following respects:

- Both inputs are fed-in serially. This requires a 4-bits pipeline for each input.
- The initial operations (shift both, interchange) are realized in a separate unit, which has constant-area and introduces a constant delay. This simplifies the design of the array and leads to both area and clock-time reduction.

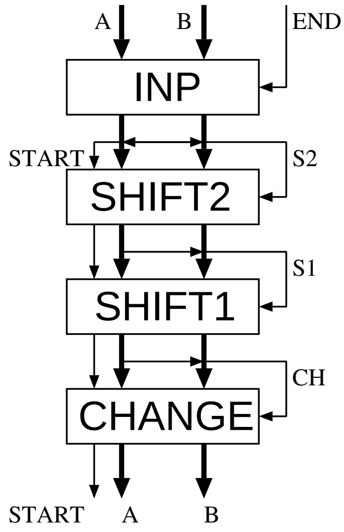


Figure 2: Structure of the feeder for the GCD array.

- A novel clustering technique allows halving of the number of steps.

Due to the characteristics of the GCD systolic array, each input has to be 4-bit wide. **The feeder** FD receives the inputs (denoted $A$ and $B$ in this section) on 4 bits each and delivers the results also on 4 bits each. The structure of the feeder is presented in fig. 2. The unit INP performs an AND between the inverted DONE and each input bit. Therefore, INP will continue to produce null bits after the host signals end of input through DONE. An OR of the 8 wires generates the signal START, which signals the beginning of computation. The common least significant null bits of A and B are ignored. Also, a negated OR of $a_0, a_1, b_0, b_1$ generates the signal SH2, which commands the next unit. The unit SHIFT2 shifts the operands by 2 bits if SH2 is 1. First each signal is delayed through a 1-clock latch, then 8 multiplexors choose the desired values. The signal SH1 is computed as negated OR of $a_0$ and $b_0$ and drives the 1-bit shifter SHIFT1, which is also composed of 8 latches and 8 multiplexors. Finally CHANGE interchanges $A$ and $B$ if the signal CH (inverted $a_0$) is 1. Note that the delay through the feeder is constant (3 clocks), while the area (experimentally 109 Atmel cells) is also independent of the length of the input.

**The systolic array** performs the loop of the GCD algorithm. This is simpler than the operations performed by the previous version of the systolic array [13], however the basic principles remain the same:

- the numbers $A, B$ are represented in 2's complement;
- the rightmost cell decides upon the next instruction to be performed, by examining $b_0, a_1, b_1$;
- the instruction signal generated by cell 0 is pipelined right-to-left, together with the carry-borrow produced at each plus-minus operation.

A space-time diagram of the atomic operations is presented in fig. 3. Due to the bidirectional communication, a loss of efficiency occurs: each processor can be active only the second step. In [13] we presented a space-clustering technique for improving the area consumption, by preserving the average running time (number of steps). In the present design we use a space-and-time clustering compacting 2 cells and two steps in one processor-step, as shown in fig. 3 (grey cells are active). The clustering technique is similar to the one presented in [8]. The structure of the new processor is shown in fig. 4. A signal LOAD (initially null) is taken from START of the feeder and regulates the loading of the initial values of $A, B$ through the multiplexors M. The initial values are fed using a 4-bit pipeline for each operand, which features a set of latches each 2 processors (4 cells). The latches L (fig.4) regulate

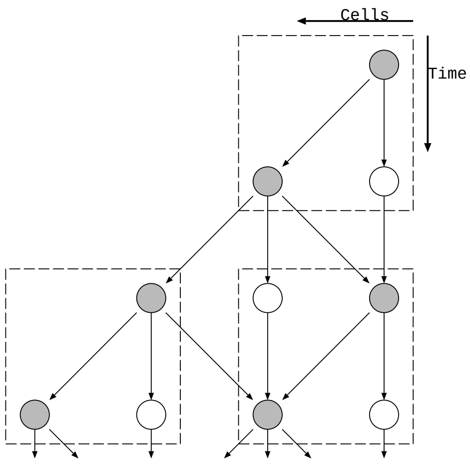


Figure 3: Time-space clustering in the GCD array.

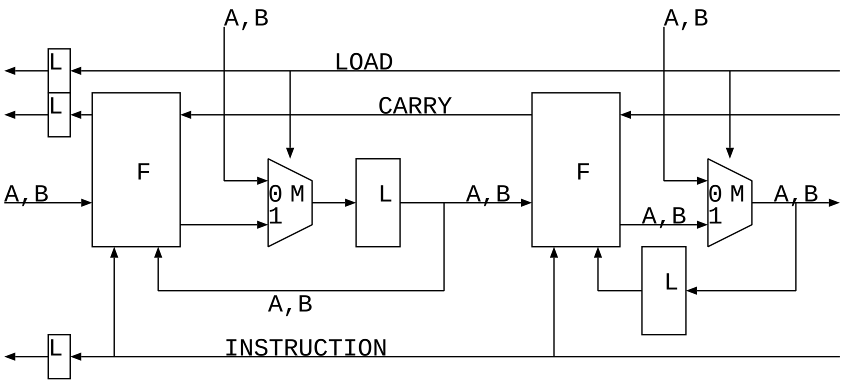


Figure 4: Double-cell processor in the GCD array.

the data-flow in the array. In particular, the different positions of the A,B latches in the left and in the right cell ensure correct operation according to the clustering shown in fig. 3. The instruction signal has two bits (SH for *shift B* and PM for *plus* or *minus*) and selects the operation to be performed by the functional units F. The functional unit performs the shift and the plus-minus operation using a full adder, an XOR, an AND and a multiplexor.

Additionally, the GCD array uses two tags TA, TB in order to keep track of the lengths of the operands (significant bits are tagged), and a bit SA which pipes rightward the sign of operand A. The structure of the double cell processing these signals is similar to the one presented in fig. 4, but the functional units are different. Each unit is composed of an OR gate and 3 multiplexors.

In the Atmel FPGA implementation, a GCD array for 8-bit inputs consumes 224 cells (1,129 equivalent gates) and has a clock cycle of 39.6 ns. Together with the feeder it takes 333 cells (1,598.5 equivalent gates). This is 72% of the area of the design in [13] and 62% of the clock time. Moreover, the present array needs two times less steps. After layout, the cell count is 953 (60% of the old design) and the clock time is 205 ns (47%), hence the new array runs 4 times faster.

# 4 Exact division

By *exact division* we understand the division without remainder, i. e. when it is known in advance that the remainder is null. We present here an improvement of the algorithm and implementation in [12]. The basic idea of *exact division* is the following: Let $A, G$ be such that $G \mid A$ and $G$ is odd. We want to find the exact quotient $X = A/G$. Let $a_0, g_0, x_0$ be the least-significant binary digits of $A, G, X$. Then from $a_0 = x_0 * g_0$ and $g_0 = 1$ one gets $x_0 = a_0$. Now if we denote by $X'$ the shifted $X$: $X = x_0 + 2 * X'$, then one has: $(A - G * x_0)/2 = G * X'$, hence the next digit of $X$ can be computed by applying the same scheme to shifted $(A - G * x_0)$ and $G$. The data-flow in the systolic algorithm is shown in fig.5 and the structure of the systolic processors is shown in fig. 6. G1 and G0 are two successive bits of $G$ received from the GCD array. They are inverted by the XOR gates if the sign of $G$ is negative (SG produced by the GCD array). X is the current bit of the quotient, which is computed in the rightmost cell. Two successive values of X (separated by the latch L) are multiplied with the two successive bits of $G$ and then subtracted from the current bit of $A_s$ in the subtracters SUB (equipped with carry feed-back). A1 and A0 are two successive bits of $A_s$ as received from the GCD array. These initial values are loaded into the division array through the two multiplexors as directed by the signal START (initially null). This signal is triggered by the output signal END of the GCD array. The systolic array is similar to the serial-parallel division design described by [15], however our design is obtained in a different manner, receives both inputs in parallel, and treats also the sign of the divisor $G_s$. Implemented on Atmel FPGA, an 8-bit array for 2 divisions consumes 160 cells (534 equivalent gates). This is 24% of the array presented in [12] (535 cells, 2,183.5 equivalent gates).

# 5 Experiments and conclusions

We do not describe here the logic design of the multiplication array, which is the one presented by [15], enhanced with a 4-bit pipeline for one of the operands. This gives a serial-serial multiplier with the same functionality as Atrubin's design [1], but with a simpler structure. The implementation of one array for 8-bit operands consumes 82 cells (273 equivalent gates). Three such arrays are needed in the arithmetic unit - the last one, however, does not need the pipeline because the input is taken in parallel from the GCD array.

The area consumption of the components in an Atmel implementation for 8-bit operands is as follows: feeder: 97 Atmel cells, GCD: 224 cells, divider: 160 cells, multiplier: 82 cells, and full rational adder: 729 cells (3,428 equivalent gates). The clock-time is 40 ns (like the GCD array), hence a speed of 25 MHz is possible (before layout). Automatic layout was not successful on a 6005 chip, but most probably the new 6010 chip will be able to accommodate an 8-bit implementation. (Due to the fully

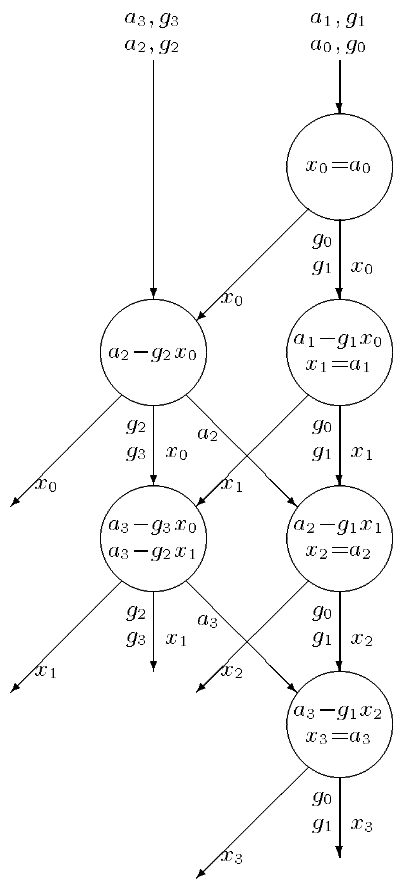


Figure 5: Data flow in systolic exact division (borrows not shown).

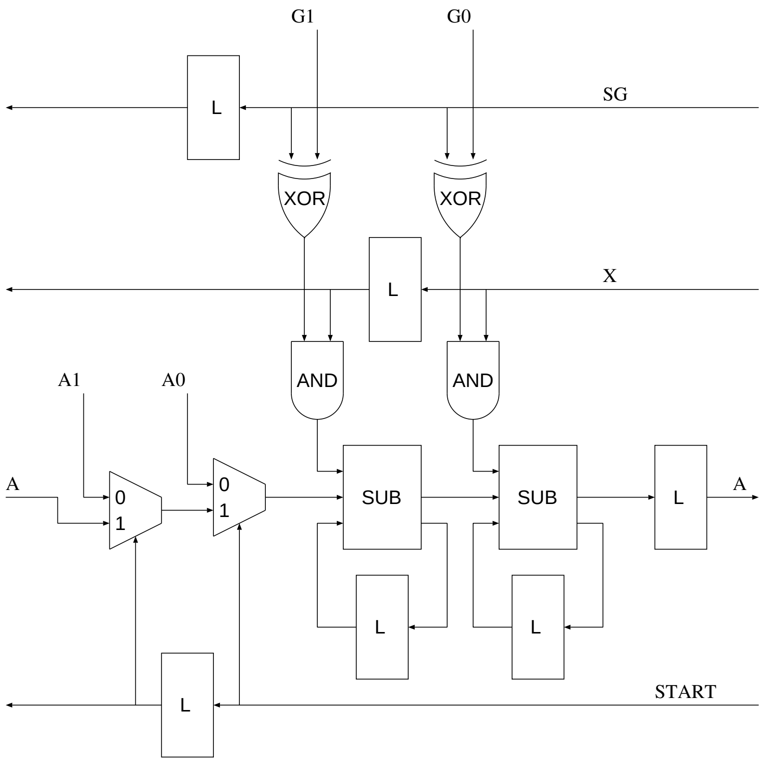


Figure 6: Systolic processor for exact division.

systolic characteristics of the algorithms, longer operands can be treated by simply tiling several chips.) A slowdown of 4 to 5 is to be expected after layout, hence a speed of 5 to 6 MHz. Since $3n$ steps are necessary (in average) for adding $n$-bit operands, 32-bit operands would need about 100 steps by 200 ns, that is 20 $\mu$s. An optimized C program on a DECstation 5000/240[1] solves the problem in 14.3 $\mu$s, hence the timings are in the same range. However, for longer operands the speed-up will increase linearly, because the time complexity of the systolic array is linear, while the timing of the software algorithms increases quadratically.

# References


[1] A. J. Atrubin. A one–dimensional iterative multiplier. *IEEE Trans. on Computers*, C-14:394–399, 1965.

[2] G. Attardi and T. Flagella. Memory management in PoSSo solver. *Journal of Symbolic Computation*, 1994. To appear.

[3] R. P. Brent and H. T. Kung. A systolic algorithm for integer GCD computation. In K. Hwang, editor, *Procs. of the 7th Symp. on Computer Arithmetic*, pages 118–125. IEEE Computer Society, June 1985.

[4] M. Bronstein, editor. *ISSAC'93: International Symposium on Symbolic and Algebraic Computation*, Kiev, Ukraine, July 1993. ACM Press.

[5] B. Buchberger, G. E. Collins, and R. G. K. Loos (eds). *Computer Algebra, Symbolic and Algebraic Computation.* Springer Verlag, Wien-New York, 1982.

[6] B. Buchberger and T. Jebelean. Parallel rational arithmetic for Computer Algebra Systems: Motivating experiments. In *3rd Scientific Workshop of the Austrian Center for Parallel Computation*, Weinberg, Austria, March 1992. Report ACPC/TR 93-3, February 1993.


[1] The speed of single precision integer arithmetic is: addition: 0.026 $\mu$s, multiplication: 0.353 $\mu$s, division: 0.905 $\mu$s.

[7] A. M. Cohen and L. J. van Gastel, editors. *SCAFI'92: Studies in Computer Algebra for Industry II*, Amsterdam, 1992. Report Series of the Computer Algebra Netherlands Expertise Center.

[8] Sh. Even and A. Litman. On the capabilities of systolic systems. *Math. Sys. Theory*, 27:3–28, 1994.

[9] A. Guyot, Y. Herreros, and J.-M. Muller. JANUS, an on-line multiplier/divider for manipulating large numbers. In M. J. Irwin and R. Stefanelli, editors, *ARITH-8: 8th IEEE Symposium on Computer Arithmetic*, pages 106–111, Como, Italy, May 1987. IEEE Computer Society Press.

[10] A. Guyot and Y. Kusumaputri. OCAPI: A prototype for high precision arithmetic. In A. Halaas and P. B. Denyer, editors, *VLSI'91*, pages 11–18. IFIP, North Holland, 1991.

[11] P. Henrici. A subroutine for computations with rational numbers. *Journal of the ACM*, 3:6–9, 1956.

[12] T. Jebelean. Systolic normalization of rational numbers. In L. Dadda and B. Wah, editors, *ASAP'93: International Conference on Application-Specific Array Processors*, pages 502–513, Venice, Italy, October 1993. IEEE Computer Society Press.

[13] T. Jebelean. Designing systolic arrays for integer GCD computation. In P. Capello, R. M. Owens, E. E. Swartzlander, and B. W. Wah, editors, *ASAP '94, San Francisco, August*, pages 295–301. IEEE Computer Society Press, 1994.

[14] T. Jebelean. Rational arithmetic using FPGA. In W. Luk and W. Moore, editors, *More FPGAs*, pages 262–273. Abingdon EE&CS Books, Oxford, 1994. Proceedings of FPLA'93: International Workshop on Field Programmable Logic and Applications, Oxford, UK, September 1993.

[15] P. Kornerup. A systolic, linear-array multiplier for a class of right-shift algorithms. *IEEE Trans. on Computers*, 43:892–898, 1994.

[16] W. Neun and H. Melenk. Very large Gröbner basis calculations. In Zippel, editor, *Computer algebra and parallelism. Proceedings of the second International Workshop on Parallel Algebraic Computation*, pages 89–100, Ithaca, May 1990. LNCS 584, Springer Verlag.

[17] C. Riem, J. König, and L. Thiele. A Case Study in Algorithm–Architecture Codesign: Hardware Accelerator for Long Integer Arithmetic. In *Proc 3rd International Workshop on Algorithms and Parallel VLSI Architecures*, Katholieke Universiteit Leuven, Belgium, 1994.

[18] E. E. Swartzlander, editor. *Computer Arithmetic*, volume 2. IEEE Computer Society Press, 1990.