OCAPI: Architecture of a VLSI Coprocessor for the GCD and the Extended GCD of Large Numbers

Alain GUYOT

Laboratoire TIM3/IMAG
46 Avenue Félix Viallet F38031 Grenoble FRANCE

Abstract

In this paper, various algorithms for finding the greatest common divisor (GCD for short) and extended GCD of very large integers are explored. In particular the trade-off between computation time and area is examined. Two of the algorithms, from which the method to derive variants is straightforward, are detailed. Then the architecture of a VLSI processor dedicated to GCD as well as multiply, divide, square root, ... of very large numbers (> 600 decimal digits), using an internal radix 2 redundant representation and supporting multiple precision, is devised.

Index terms : GCD, extended GCD, redundant number system, most significant digit first algorithm, very large integers

1. Introduction

Computation of the GCD of two integers is used in integer arithmetic (e.g. normal form) as well as rational arithmetic [1] either to get the canonic form or perform rounding. Extended GCD is necessary for multiple precision GCD [2] and to work over $\mathbb{Z}/p\mathbb{Z}$. Computer algebra programs (reduce, macsyma, ...) reportedly spend 85% of their time in GCD [3] for some applications because of the frequency and the comparatively high cost of this operation. So, while all the algorithms studied in this paper have a complexity of O(n), it is important to find a good one.

2. Background

The "binary algorithm" proposed by Stein [4] relies on shift, subtraction, exchange and comparison; all operations except comparison suitable for large integers. For convenience, right shift n position(s) is noted "$/2^n$" and left shift "$*2^n$".

This work was supported by the GCIS

function GCD (A,B) ; { assume A odd }
begin
  while B ≠ 0 **do**
  if B mod 2 = 0 **then** $\begin{pmatrix} A \\ B \end{pmatrix} := \begin{pmatrix} A \\ B/2 \end{pmatrix}$ { B is even }
  else if A ≥ B **then** $\begin{pmatrix} A \\ B \end{pmatrix} := \begin{pmatrix} B \\ (A-B)/2 \end{pmatrix}$ {A and B are odd}
  else $\begin{pmatrix} A \\ B \end{pmatrix} := \begin{pmatrix} A \\ (B-A)/2 \end{pmatrix}$;
  GCD := A ;
end { GCD } .

This algorithm gives the GCD because 1) every transformation preserves the GCD(A,B), and 2) GCD(A,0) = A.

It is not very far from Euclid's algorithm [5] when binary non restoring division is developed:

function GCD (A,B); {assume A or B odd, $A \geq B$, $2^n \leq A < 2^{n+1}$}
begin
  while B ≠ 0 **do**
  if $B < 2^n$ **then** $\begin{pmatrix} A \\ B \end{pmatrix} := \begin{pmatrix} A \\ B*2 \end{pmatrix}$
  else if A≥B **then** $\begin{pmatrix} A \\ B \end{pmatrix} := \begin{pmatrix} B \\ (A-B)*2 \end{pmatrix}$ {$2^n \leq B \leq A < 2^{n+1}$}
  else $\begin{pmatrix} A \\ B \end{pmatrix} := \begin{pmatrix} A \\ (B-A)*2 \end{pmatrix}$;
  while A mod 2 = 0 **do** A := A / 2; GCD := A ;
end { GCD } .

Basically, Stein's algorithm tests and forces to zero the LSB of B while Euclid's does the same with the MSB.

Brent and Kung [6] have built a systolic algorithm based on Stein's one that, instead of comparing actual numbers, compares an estimate of their number of bits, and uses serial binary addition/subtraction. Due to the estimates, addition becomes necessary to ensure stability [6]. Kornerup and Matula [1] use signed binary digits for a

carry propagation free addition/subtraction and only the sign, as in the SRT division. Purdy [7] proposed use the same notation [1] and the same evaluator as Brent's [8]. Yun and Zhang [9] later improved their algorithm to make it about twice as fast and to include as well the extended GCD.

Those algorithms rely on the affectation $B := (B - q*A)* 2^p$, and it is easy to build new algorithms by changing the ranges of q and p. Since we want one hardwired addition/subtraction per cycle but we can afford several shifts we will limit $|q|$ to powers of 2. In this paper, depending on the realization, the transistor cost of another shift varies from 2 (parallel) to 10 (serial) transistors, the cost of an adder is from 20 (2's complement) to 40 (redundant) transistors per digit.

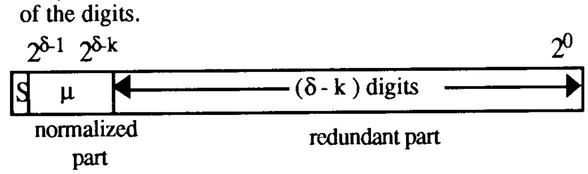
3. LSB first approach

Let us give in pseudo-Pascal, as a first example, an algorithm that divides the greatest of the two numbers by 4 at each iteration by forcing to two of its LSBs 0 whenever necessary, where $q \in \{-1,0,1,2\}$ and $p = -2$. This algorithm simplifies when conventional binary notation is used since then $a_0+b_0=0$ never happens and absolute value is no longer needed as an operation and of course the adder is simpler in term of gates as we will see later.

```
function GCD (A, B, δa ,δb) ;
{assume A odd, δa=⌈log2A⌉ δb=⌈log2B⌉, # of bits of A and B}
begin
while δb > 0 do    { B≠0 }
begin
    if b0 = 0 then
    begin
      if b1=0 then  B := (B+0) / 4 { q = 0 }
      else  B := (B+ 2*A) / 4      { q = 2 }
    end
    else if  δa ≥ δb  then
    begin swap ( δa , δb ) ;  { exch }
      if ( a0+b0=0 and  a1+b1=0) or
      (a0+b0≠0 and |a1+b1| = 1) then { q = ± 1 }
      ( A )    (     B     )       ( A )    (     B     )
      (   ) := (           ) else  (   ) := (           )
      ( B )    ( (A+B) / 4 )       ( B )    ( (B-A) / 4 )
    end  else
      if ( a0+b0=0 and a1+b1=0) or
      (a0+b0≠0 and |a1+b1| = 1) then
      ( A )    (     A     )       ( A )    (     A     )
      (   ) := (           ) else  (   ) := (           ) ;
      ( B )    ( (A+B) / 4 )       ( B )    ( (B-A) / 4 )
    δb := δb - 2 ;
  end ;
  GCD := A ;
end.
```

4. MSB first approach

The second example forces the MSB of B to 0 and tries to predict if the 2nd MSB will also turn to be 0. To test the MSB, redundant notation must be used whether the addition/subtraction is done in serial or in parallel. Let us note $\bar{1}$, 0 or 1 the values of a signed binary digit. Due to the many ways of writing integers in redundant notation, the numbers of digits δ_a and δ_b of A and B alone are useless for comparison. For example $A=1 1 \bar{1} \bar{1} \bar{1}$ is in fact smaller than B=10 despite $\delta_a=5$ and $\delta_b=2$. So besides δ_a and δ_b we use also the first k digits μ_a and μ_b of A and B. A normalization forces the 1st digit of μ_a and μ_b to be non-zero, and all the k digits to have the same sign (0 has both signs), μ_a and $\mu_b \in [-2^k+1..-2^{k-1}] \cup [2^{k-1}..2^k-1]$. They are stored as the common sign and the absolute value of the digits.



Of course the larger k is, the more digits in μ_a and μ_b , the more accurate the predictors and the quicker the convergence of the GCD algorithm, in number of cycles. But since there is a carry propagate to normalize μ, each cycle would also last longer. So in the present paper we propagate the carry on 3 bits because it is the minimum allowing us to perform another operation of interest: division [10,11], and we try also propagation on 4 bits. When μ_a and μ_b are normalized A and B are seminormalized, that is we always have:

$$(|\mu_a|-1)2^{\delta_a-k}<|A|<(|\mu_a|+1)2^{\delta_a-k} \& (|\mu_b|-1)2^{\delta_b-k}<|B|<(|\mu_b|+1)2^{\delta_b-k}.$$

Since A and B are only seminormalized, for the following GCD algorithm, values $|\mu_b| = 2^{k-1}$ and $|\mu_a| = 2^k-1$ deserves special attention. Let us take as an example k = 3, $B = 100\bar{1}0$ ($\delta_b=5$, $\mu_b=4$) and A= 1110 ($\delta_a=4$, $\mu_a=7$). If $B:=B-A*2^{\delta_b-\delta_a}$ is performed, the result will be $B=\bar{1}0010$, that also is seminormalized and negatif, so the next operation will be $B:=B+A*2^{\delta_b-\delta_a}$ that gives back the previous value, and the algorithm <u>never ends</u>. By rewriting B when necessary before swapping A and B, the algorithm prevents μ_a from taking the values 7 or -7. A similar loop forever is reported in [1] and treated by forcing a shift in the loop. The table on next page summarizes the new range of μ_b from the value μ_a and the previous value of μ_b after B:=B-A, in bold the values that should not be taken, and in italics the column $\mu_a=7$ that is avoided. A symmetrical table can be drawn for B:=B+A.

$\mu_b \setminus \mu_a$	4	5	6	7
1	-1,+1	-2,0	-3,-1	*-4,-2*
5	0,+2	-1,+1	-2,0	*-3,-1*
6	+1,+3	0,+2	-1,+1	*-2,0*
7	+2,+4	+1,+3	0,+2	*-1,+1*

The following pseudo-Pascal program is the behavioral description of a chip for the GCD with $q \in \{-1,0,1\}$ and $p \in \{0,1,2\}$. The program manipulates four integers: δ_a and δ_b with $\log_2(\log_2$ (max. value of A or B)) bits, μ_a and μ_b with k bits (unsigned) plus one bit sign.
Let n be δ_b-k-1, b_n be the most significant signed digit of B and $b_{n..0}$ denote the string $b_n,b_{n-1}, \ldots,b_0$.

```
function GCD (A, B, k, δa ,δb) ;
{ δa=⌈log2A⌉ δb=⌈log2B⌉, number of bits of A and B }
begin
while δb > 0 do          {B ≠ 0 }
begin
   if |μb| < 2^(k-1) then
   begin
   if ((δa>δb) or ((δa = δb) and |μa| > 2^(k-1) )))
   and ( bn * μb) = 2^(k-1)-1 then
         begin μb := μb + bn ; bn := - bn end
   else if |μb| < 2^(k-2) then
   begin μb:=4*μb+2*bn+bn-1; bn..2:=bn-2..0; δb:= δb -2 end
   else begin μb:=μb+μb+bn; bn..1:=bn-1..0; δb := δb -1 end
   end else begin
   if (δa > δb) or ( (δa = δb) and (|μa| > |μb| ) ) then
         begin swap ( δa , δb ) ; swap (A , B ) end ;
   if | μa + μb| < 2^(k-2) then
         begin B := (B + A) * 4 ; δb := δb - 2 end
   else if | μa - μb | < 2^(k-2) then
         begin B:= (B - A) * 4; δb := δb - 2 end
   else if | μa + μb | < 2^(k-1) then
         begin B:= (B + A) * 2; δb := δb - 1 end
   else  begin B:= (B - A) * 2 ; δb := δb - 1 end ;
   end;
end ;
 GCD := A ;
end.
```

5. Generalization

One addition/subtraction can force to 0 at least either the 2 least significant positions or the most significant position of B, and sometimes more positions. To take advantage of the 0s in more positions, that is to shift them out, is more costly in term of transistors. On the contrary forcing just one LSB position makes the circuit slower but the operation part and the control simpler. In a systolic approach, with a distributed control, the control part becomes preponderant [6]. Thus the time/area of different approaches has been plotted, where the time is approximated by the number of cycles and the area by the number of transistors. Distributivity of addition/subtraction and shift as well as serialization of less frequent operations were used to keep the number of transistors low.

Approaches with examination of the LS digits

##	q (quotient)	p (position)	bits or digits examined	
L1	-1 , +1	-1	b_1	
L2	-1, 0 , +1	-1	a_1,b_1	note 1
L3	-1, 0 , +1	-1 , -2	a_0,a_1,b_0,b_1	note 2
L4	-1,0,+1,+2	-2	a_0,a_1,b_0,b_1	note 3
L5	-2,-1,0,+1,+2,+4	-2 , -3	a_0,a_1,b_0,b_1,b_2	

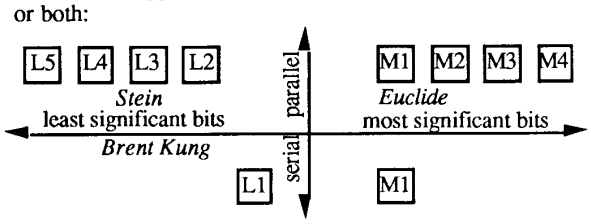
note 1: This is the Brent-Kung [6] variant
note 2: This is the Yun-Zhang [9] variant
note 3: This is the first example detailed in this paper

Approaches with examination of the MS digits

##	q (quotient)	p (position)	k	
M1	-1 , 0 , +1	1	3	
M2	-1 , 0 , +1	1 , 2	3	note 4
M3	-1 , 0 , +1	1	4	
M4	-1 , 0 , +1	1 , 2	4	

note 4: This is the second example detailed in this paper

Some of the approaches were realized in serial, in parallel or both:



Pseudo Pascal description of the variants, as well as the corresponding hardware are detailed in [12] available through the IMAG librarian.

6. Extended GCD

Now we want the time/area trade off of a circuit that computes a pair of integers R and S that follows the Bezout's equation, A*R - B*S = H, H denotes the GCD(A,B). This is done by executing in the reverse order a sequence of operations corresponding to the ones that led to H (1st column of the tables of next page) while preserving Bezout's identity (3th column). For that phase, the initial values of (A,B,R,S) are (H,0,1,1). Unfortunately, for the LSB approach, that leads to R and S that may not be integer, so for this approach we first preserve $A*R-B*S=H*2^n$ (4th column) and then simplify R and S while keeping them integers.

Let us give the table for the LSB example

compute H =GCD(A,B)	restore A and B	Preserves A*R-B*S=H	Preserves A*R-B*S =H*2^n
B := B / 4	B := B * 4	S := S / 4	R := R * 4
B := B/4+A/2	B:=B*4-A*2	R := R - S/2 S := S / 4	R := R*4-S*2
$\begin{pmatrix} A \\ B \end{pmatrix} := \begin{pmatrix} B \\ A \end{pmatrix}$	$\begin{pmatrix} A \\ B \end{pmatrix} := \begin{pmatrix} B \\ A \end{pmatrix}$	$\begin{pmatrix} R \\ S \end{pmatrix} := \begin{pmatrix} -S \\ -R \end{pmatrix}$	$\begin{pmatrix} R \\ S \end{pmatrix} := \begin{pmatrix} -S \\ -R \end{pmatrix}$
B := (B+A)/4	B:=B * 4 - A	R := R - S/2 S := S / 4	R := R * 4 - S
B := (B-A)/4	B:=B*4 + A	R := R - S/2 S := S / 4	R := R * 4 + S

and the table for the MSB example

compute H = GCD(A,B)	restore A and B	preserves A*R - B*S = H when restoring A and B
B := B * 2	B := B / 2	S := S * 2
B := B * 4	B := B / 4	S := S * 4
$\begin{pmatrix} A \\ B \end{pmatrix} := \begin{pmatrix} B \\ A \end{pmatrix}$	$\begin{pmatrix} A \\ B \end{pmatrix} := \begin{pmatrix} B \\ A \end{pmatrix}$	$\begin{pmatrix} R \\ S \end{pmatrix} := \begin{pmatrix} -S \\ -R \end{pmatrix}$
B := (B+A)*2	B:=B/2 - A	R := R - S * 2 ; S := S * 2
B := (B-A)*2	B:=B/2 + A	R := R + S * 2 ; S := S * 2
B := (B+A)*4	B:=B/4 - A	R := R - S * 4 ; S := S * 4
B := (B-A)*4	B:=B/4 + A	R := R + S * 4 ; S := S * 4

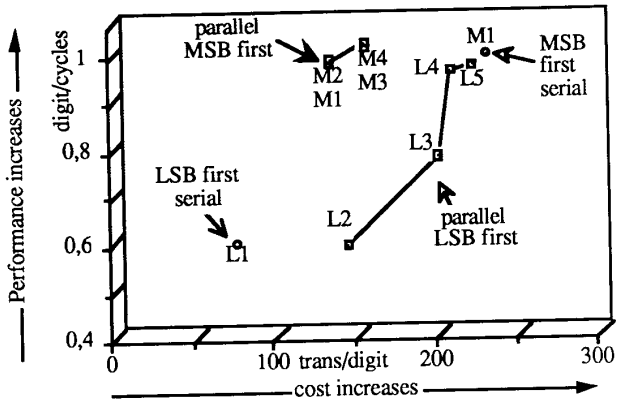
Simplification of R and S is straightforward, but adds cycles and transistors, since it is necessary to keep A and B in registers, to the extended GCD computation in the LSB approach. Since A,B and H are odd, R and S exibit the same parity.

while n > 0 **do**
begin n := n-2;
  if $r_0 = 0$ **then begin if** $r_1 = 0$
  then $\begin{pmatrix} R \\ S \end{pmatrix} := \begin{pmatrix} R/4 \\ S/4 \end{pmatrix}$ **else** $\begin{pmatrix} R \\ S \end{pmatrix} := \begin{pmatrix} R/4+B/2 \\ S/4+A/2 \end{pmatrix}$ **end**
    else begin if $(a_0+r_0 = 0$ **and** $a_1+r_1 = 0)$ **or**
    $(a_0+r_0 \neq 0$ **and** $|a_1+r_1| = 1)$
  then $\begin{pmatrix} R \\ S \end{pmatrix} := \begin{pmatrix} (R+B)/4 \\ (S+A)/4 \end{pmatrix}$ **else** $\begin{pmatrix} R \\ S \end{pmatrix} := \begin{pmatrix} (R-B)/4 \\ (S-A)/4 \end{pmatrix}$ **end** ;
end .

If we have only one adder, the two additions/subtraction that are presented here in parallel must be done serially.

7. Measures

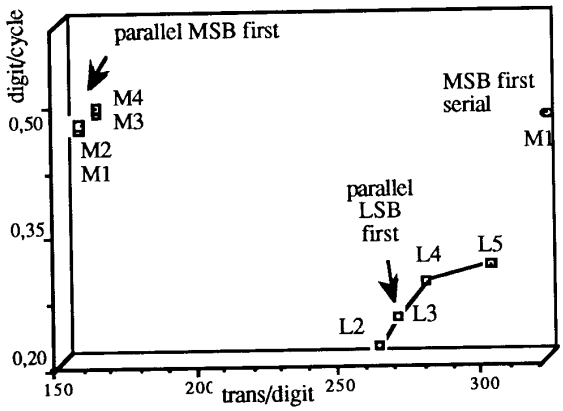
The first plot below shows the performance-cost ratio of the different approaches and variants for the GCD computation. The cost was obtained by carefully designing an operation part for each algorithm [12], and the performance by running a simulation of each of the operation parts with the same set of 200 samples of randomly generated 2,000-bit integers whose first and last bit was forced to be non-zero. The trade off of each variant has been locally optimized. The time to load in the operands was not counted.



Algorithm L5 that removes an average of 2.5 digits from the least significant position of B at each cycle happens to be slower than algorithm M1 that removes 1 digit at the most significant position when it is not stalling to rewrite B. The size, or number of significant digits, of the result of an addition/subtraction is not deductible from the size of the operands, and errors tend to accumulate. After several trials, we settled on no size adjustment, the size of the result being the size of the biggest operand.

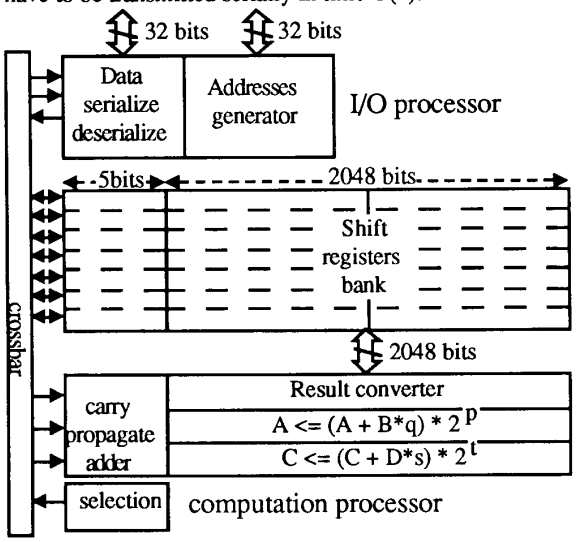
Having computed the GCD in n cycles, the number of additional cycles required for the extended GCD is n minus the cycles when the algorithm was stalling for the MSB approaches, and n plus the number of cycles to simplify the result in the LSB one.

The LSB first serial approach is very cheap for the GCD, and the GCD only. Otherwise the MSB approaches are far more interesting for the extended GCD, as shown on the following plotting the cost of extended GCD.



8. Coprocessor for high precision

This VLSI circuit is designed to be placed close to a microprocessor and plugged into the memory bus, in order to significantly reduce the time required for high precision software computation. Due to the limited word size of the memory and pad limitation of the circuit, long numbers have to be transmitted serially in time O(n).



The goals of the architecture are:

1- To overlap computation and communication, so on line computation is desirable, but also pipelining of load in, execution and store back [13].

2- To minimize external bus occupancy by keeping intermediate results or local variables in an on-chip register bank

3- To benefit from a very large operator to avoid multiple precision operations. Internal redundant notation keeps the carry propagation path short [14,15]. The operator must take advantage of the parallel transmission of internal variables with the register bank and of the serial transmission of external variables.

4- To offer add, subtract, multiply, divide, GCD, extended GCD, square root [16,17] and also provide support for rational numbers [1] and matrices.

5- To efficiently use the silicon. So the datapath and the operator are configurable to perform several operations in parallel on short operands and/or on operands transmitted in parallel.

The I/O processor acts as a DMA. It spies on the buses for the address and the value of the first word of the operand, and then requests the bus and generate the addresses sequence for the next words [13]. Whenever available it uses the burst mode to speed up the exchange. It can interleave two loads and one store.

Each register of the bank is accessible in parallel or in serial mode. The most significant bit occupies the rightmost position. One parallel and six serial accesses can take place simultaneously. A crossbar allows to serially load or store registers while computing on others in the pipeline mode, or to bypass the bank in on line mode. The crossbar includes a serial adder. For the extended GCD, three of the registers are used as a stack to record the sequence of operations.

Two adder/subtractor-shifters, each with two registers for redundant numbers, compose the computation processor. For multiply, divide [14] or square root [16,], they are used together in the on line mode or independently in the parallel/serial mode to perform two different operations. On the figure, A,B,C,D represent registers for large numbers, $q,s \in \{-1,0,1\}$ and $r,t \in \{0,1,2\}$. Registers B and D can be loaded serially, from the left to the right position[14]. A pair of register converts on the fly serial signed binary result into conventional representation (-1 or 0 in the most significant position, 0 or 1 elsewhere) [18] and into a representation with no 0 (only -1 and 1). This late representation is useful for matrix triangularization by Givens' rotation method since it does not change the scale factor [19].

9. Conclusion

By simulation we have found that, in the GCD computation, algorithms that decimate the most significant positions are faster than the ones that decimate the least significant positions and are even faster for the extended GCD, while at the same time demanding a lower transistor count.

This has led to the architecture of a VLSI coprocessor for high precision arithmetic that support the GCD and the extended GCD on top of standard operations. The overall complexity is expected to be about 700 K transisors with a density of 5 Kt/mm^2 in 1.5µ and a very high regularity. It will be linked to a software package [20,21] which is now used to tune the circuit.

Acknowledgements

We are indebted to the referees specially for suggestions of new applications on rational arithmetic.

Many thanks to Dr. J.L. Roch for his introduction to computer algebra, to Dr. J.M. Muller for his introduction to on line operations, to S. Chebbi who carried out all the simulations with an equal temper and to R. Bouraoui who has designed a prototype chip for the extended GCD.

References

[1] *P. Kornerup & D. W. Matula* "Finite Precision Rational Arithmetic: An Arithmetic Unit" IEEE Trans. on Computers, Vol.C34-4 1983

[2] *D.H. Lehmer* "Euclid's Algorithm for Large Numbers" AMMMM 45, pp. 227,233, 1938

[3] *J. Davenport & Y. Robert* "VLSI and Computer Algebra: The GCD Example" Dynamical Systems and Cellular Automata Academic Press, London 1985

[4] *J. Stein* "Computing GCD without division" J. Comp. Phys. Vol. 1, pp 397-405 1967

[5] *D.W. Knuth* "Seminumerical Algorithms" (The Art of Computer Programming) pp.317-336,Addison-Wesley, Reading, Massachusetts 1981

[6] *R.P. Brent & H.T. Kung* "Systolic VLSI Array for linear-time GCD computation" proc. VLSI'83, Elsevier Science Publishing, North Holland-IFIP 1983

[7] *G. B. Purdy* "A carry-free algorithm for finding the greatest common divisor of two integers" Comp. & Maths. with Appls, Vol.9 pp311-316, 1983

[8] *R.P. Brent & H.T. Kung* "A systolic algorithm for integer GCD computation" Proc. 7th Symposium on Computer Arithmetic 1985

[9] *D.Y. Yun & C.H. Zhang* "A fast carry-free algorithm and hardware design for extended integer GCD computation" Proc. ACM Symposium on Symbolic Algebraic Computing pp 82-84 1986

[10] *V.C Hamacher & J. William* "A linear time divider array" Canadian Electr. Engineering Journal, Vol6, N° 4, 1981

[11] *M. Ercegovac & K. Trivedi* "On line algorithms for division and multiplication" IEEE Trans. on Computers, Vol.C26-7 pp 681-687 1977

[12] *R. Bouraoui, A. Guyot & J.L. Roch* " The best way to build a circuit for the extended GCD of large integers" IMAG research report 1990

[12] *R. Bouraoui, A. Guyot & J.L. Roch* " The best way to build a circuit for the extended GCD of large integers" IMAG research report 1990

[13] *M. Cosnard, A. Guyot, B. Hochet, J.M. Muller, H. Ouaouicha, Ph. Paul & E. Zysman* "The FELIN arithmetic coprocessor" Proc. 8th Symposium on Computer Arithmetic, 1987

[14] *A. Guyot, Y. Herreros & J.M. Muller* "JANUS, an on-line multiplier/divider for manipulating large numbers" Proc. 9th Symposium on Computer Arithmetic 1989

[15] *G. Metze & J.E. Robertson* "Elimination of carry propagation in digital computer" proc. IFIP conference, 1959 reprinted in IEEE Computer Arithmetic Vol 1, E. E. Swartzanlander ed.

[16] *V. Oklobdzija & M. Ercegovac* "An on-line square root algorithm" IEEE Trans. on Computers, Vol.C31-1 1982

[17] *A. Guyot & Y. Kusumaputri* "OCAPI: A circuit for on line radix 2 high precision arithmetic" submitted to ESSCIRC91, Milano, September 1991

[18] *M. Ercegovac & T. Lang* "On-the-fly conversion of Redundant into Conventional Representation" IEEE Trans. on Computers, Vol.C36-7 1987

[19] *M. Ercegovac & T. Lang* "Redundant and On-Line CORDIC: Matrix Triangularisation and SVD" IEEE Trans. on Computers, Vol.C39-6 1990

[20] *J.L. Roch* "L'architecture du Système PAC et son Arithmétique Rationnelle" Ph.D. diss., IMAG/INPG 1989

[21] *J.L. Roch* "The PAC System, general presentation" Proc. CAP90. E. Kaltofeln ed. North Holland. 1990