A SINGLE CHIP 1024 BITS RSA PROCESSOR

André Vandemeulebroecke[1], Etienne Vanzieleghem[2], Tony Denayer[1] and Paul G. A. Jespers[3]

This work has been supported by the Région Wallonne of Belgium and by the FNRS (National Fund for Scientific Research).

[1] Mietec N.V., Raketstraat 62, B-1030 Bruxelles, Belgium - Phone (32) - 2 - 242.50.10.

[2] Alcatel Bell Telephone, Francis Wellesplein n°1, B-2018 Antwerpen, Belgium - Phone (32) - 3 - 240.40.11.

[3] Microelectronics Laboratory, Université Catholique de Louvain, Place du Levant 3, B-1348 Louvain-la-Neuve, Belgium - Phone (32) - 10 - 47.25.40.

ABSTRACT

A new carry-free division algorithm will be described; it is based on the properties of RSD arithmetic to avoid carry propagation and uses the minimum hardware per bit i.e. one full-adder. Its application to a 1024 bits RSA cryptographic chip will be presented. Thanks to the features of this new algorithm, high performance (8 kbits/s for 1024 bits words) was obtained for relatively small area and power consumption (80 mm2 in a 2 µm CMOS process and 500 mW at 25 MHz).

1. INTRODUCTION

With the constant growth of data communications, security is becoming an increasingly important feature. Cryptography provides two ways to solve some aspects of information protection: secret or public key cryptosystems. DES (Data Encryption Standard) is the most popular system of the first category. Integrated versions can attain high processing speeds (20 Mbits/s, [1]) but the key unicity entails the problem of key management. In order to solve this problem, public key cryptosystems use different keys for enciphering and deciphering. The RSA cryptosystem (Rivest, Shamir and Adelman [2]) has emerged among all proposed solutions during the last decade.

The basic RSA operation is a modular exponentiation involving large numbers (typically 200 to 1000 bits). Such a size is mandatory to protect the system against cracking. Modular exponentiation can be split into successive modular multiplications. While powerful multiplication algorithms are well-known, carry-free modular computation (in fact a division which in remainder is kept) algorithms only begin to emerge [3]. As the size of the processed numbers is very large, it is mandatory to

J.J. Quisquater and J. Vandewalle (Eds.): Advances in Cryptology - EUROCRYPT '89, LNCS 434, pp. 219-236, 1990.

overcome the carry propagation problem, in order to keep the addition time as short as possible relative to the clock frequency. Solutions to this problem were already proposed [4] but the required hardware per bit was too big (more than two equivalent full-adders) and consequently not suitable for a 1024 bits RSA chip.

In this paper, we present a new carry-free division scheme based on the Redundant Signed Digit (RSD) representation and using the most reduced hardware (one full-adder by bit). This technique has been applied to the design of a 1024 bits RSA single chip whose performances are best compared to all known solutions published up to now [5, 6, 7, 8, 9]. In sections II to IV, basic concepts of RSD arithmetic are presented and the carry-free division method is described. In sections V to X, the 1024 bits RSA chip is presented; it is compared to other currently available implementations and some specific features of its architecture are detailed.

2. CARRY SAVE AND RSD ARITHMETIC

The conventional addition of two numbers in classical binary form is usually the bottleneck for speed improvement in digital integrated circuits. While other digital operations may be performed in parallel (shift, data storage...), addition requires a carry propagation.

The most significant digit of the sum of two n-digit numbers written in conventional form actually depends on the 2.n data. This case occurs with a full carry propagation from the LSB to the MSB.The classical addition of two binary numbers is sketched in fig 1.

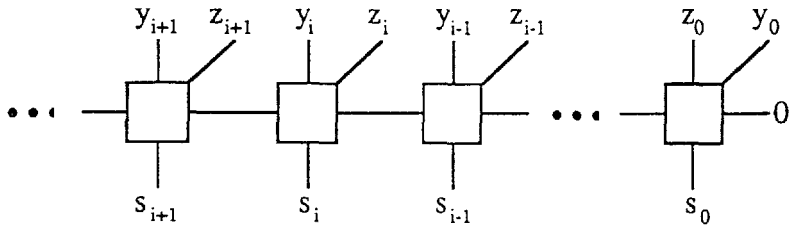


Fig. 1. Classical addition with carry propagation

As the global delay of the adder depends on the carry delay, the basic full adder cell is optimized regarding this criteria. Different solutions proposed to solve the carry propagation problem use redundant representations for the operands and the result. The most well known method is the use of carry save arithmetic.

Carry save adders are widely used in fast arithmetic processors, because of their performance in terms of speed and silicon area. The basic principle of the carry save addition is to reduce the sum of three binary numbers to the sum of two binary numbers without carry propagation. Let us consider the two operands Y (in redundant form) and Z (in classical form). Y is represented as the sum $Y^* + Y^{**}$ and y_i^* , y_i^{**} , $z_i \in \{0, 1\}$. With one level of classical full adders, we can reduce the sum of the three binary numbers: $Y^* + Y^{**} + Z$ to a sum of two binary numbers: $S^* + S^{**}$ (Fig. 2).

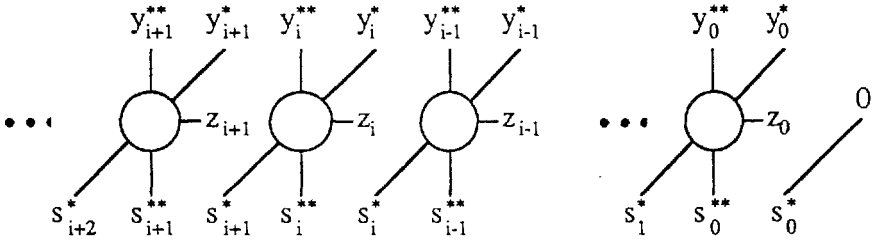


Fig. 2. Carry save addition scheme

The addition process is fully parallel, hence the basic cell has to be optimized for both carry and sum delays. Such a cell is sketched by a ring full adder, to distinguish it from the above square full adder (fig. 1). For each of them, one can write :

$$y_i^{*} + y_i^{**} + z_i = 2 \cdot s_{i+1}^{*} + s_i^{**}$$

Defining S as $S^{*} + S^{**}$, we obtain Y + Z = S with Y and S in redundant carry save form and Z in classical binary form.This scheme is extensively used in parallel multipliers.

The Redundant Signed Digit arithmetic (RSD) has been introduced in the late 50's by Avizienis [10] and is quite similar to the carry save one. In the RSD representation, a number X can be viewed as the difference between two positive binary numbers X^{*} and X^{**}. We have:

$$X = \sum_{i=0}^{n} x_i.2^i = \sum_{i=0}^{n} (x_i^{*} - x_i^{**}).2^i \quad \text{with } x_i^{*}, x_i^{**} \in \{1,\ 0\}.$$

Obviously, each digit x_i of X belongs to the set $\{1,\ 0,\ \bar{1}\}$ where the upper bar indicates a negative value. With such a definition, a number may have more than one representation. As an example, the number 7 expressed with four bits has the representations: (0 1 1 1), (1 0 0 $\bar{1}$), (1 0 $\bar{1}$ 1) and (1 $\bar{1}$ 1 1).

Clearly, RSD and carry save representations are in essence quite similar. The main difference is the fact that RSD doesn't need use of two's complement form to handle negative numbers. This representation is thus more "natural" if positive and negative numbers have to be processed. Therefore, the multiplication and division will be easier performed with this representation and the sign test will be directly obtained by testing the first nonzero digit.

There has been little use of RSD arithmetic up to now, perhaps due to the inherent complexity of the basic adder cell [3, 4, 10]. In the next section, we will propose a new addition scheme using RSD representation which can be implemented in the same way carry save adders are. Therefore, it allows to combine the advantages of the two solutions.

3. RSD ADDITION AND MULTIPLICATION

The conventional 1-bit full adder assumes positive weights to all of its 3 binary inputs and 2 outputs. Such adders can be generalized to four types of adding cells by imposing positive and negative weights to the binary input/output terminals [11]. Figure 3 lists the names and logic symbols of the four types of generalized full adders.

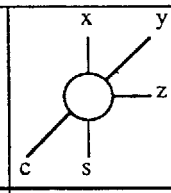
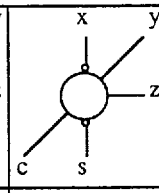
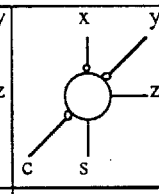
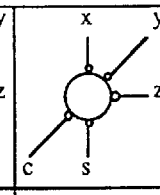
Logic symbol				
Type	0	1	2	3
Function	x+y+z = 2.c+s	-x+y+z = 2.c-s	-x-y+z = -2.c+s	-x-y-z = -2.c-s

Fig. 3. Generalized full adders

Each type of full adder is named by the number of its negatively weighted inputs. The implementation of a generalized full adder (GFA) is quite straightforward: it can be shown that a GFA is simply an usual type 0 full adder with inverters replacing the rings. This implies no extra hardware in CMOS compared to a classical full adder as those inverters are always existent.

The addition of two SD (Signed Digit) numbers Y and Z can be performed by cascading two levels of generalized full adders of types 1 and 2 (Fig. 4).

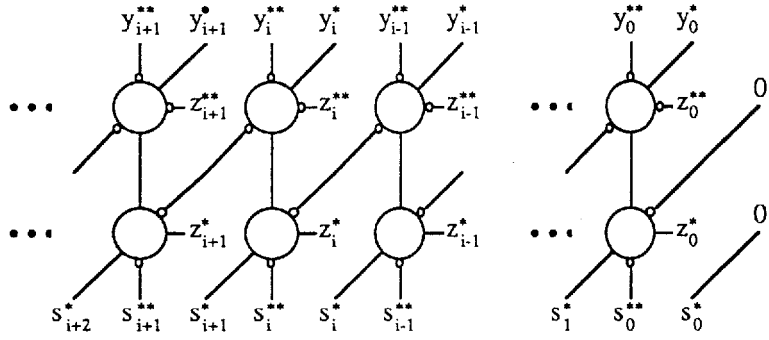


Fig. 4. RSD addition scheme with all data in redundant form

Subtraction may be obtained by permuting the numbers Z* and Z**. The main drawback of this scheme with two RSD numbers is the amount of hardware which is twice than in the carry save case.

Let $Y = Y^* - Y^{**}$ be a number in redundant form, Z a number in two's complement form and $S = S^* - S^{**}$ the result of the operation Y + Z or Y - Z, with $y_i^*, y_i^{**}, z_i, s_i^*, s_i^{**} \in \{0, 1\}$. In the following, we treat the sign of Z by using the well known property of 2's complement numbers [11]:

$$\text{if} \quad Z = -z_n.2^n + \sum_{i=0}^{n-1} z_i.2^i = (z_n \; z_{n-1} \; \ldots \; z_0).$$

$$\text{then} \quad -Z = -(1-z_n).2^n + \sum_{i=0}^{n-1} (1-z_i).2^i + 1 = (1-z_n \; 1-z_{n-1} \; \ldots \; 1-z_0) + 1.$$

So a subtraction reduces to an addition with an inversion of all bits and adding of 1. The details of these two operations are sketched in fig 5 and 6.

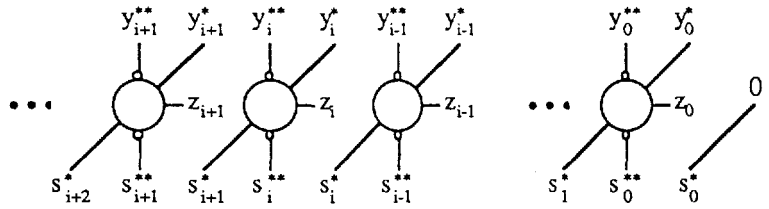


Fig. 5. RSD addition with one operand in 2's complement form

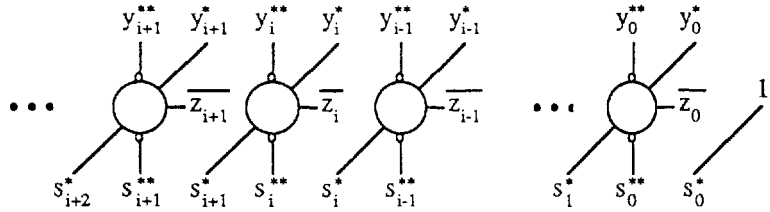


Fig. 6. RSD subtraction with one operand in 2's complement form

In the same way as with the carry save adder, the computation scheme can be implemented with one level of full adders, using in this case generalized full adders of type 1.The addition is fully parallel, thus requiring no carry propagation.

These generalized full adders are used in Pezaris array type multipliers [11] to directly implement the multiplication of two numbers in two's complement form. Pezaris multipliers are the optimal choice for the implementation of classical array multipliers. But as the internal structure is irregular, it is impossible to derive an associated serial-parallel algorithm using one row of adders. This is obviously possible when using rows of RSD adders, as shown in fig. 7.

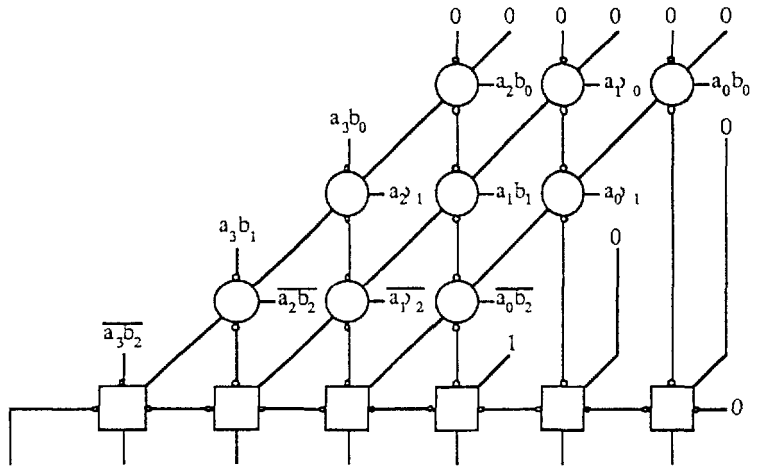


Fig. 7. RSD multiplication scheme

In this example, the multiplicand A = (a_2 a_1 a_0) and the multiplier B = (b_3 b_2 b_1 b_0) in two's complement form are combined to form the product A * B. The array (ring GFA's) contains only GFA of type 1 and the result is in redundant form. If necessary, the back conversion to two's complement form is performed by a carry propagate adder (square GFA's).

4. A NEW CARRY FREE DIVISION ALGORITHM

The classical non restoring division is usually performed using the add/subtract-and-shift-left algorithm [11]. It is governed by the next equations:

$$R^{j+1} = 2 * (R^j - q_j * D)$$

$$-2.D < R^j < 2.D$$

where R^0 is the dividend, R^j is the partial remainder at step j, q_j is the j^{th} quotient bit and D is the divider. At each step, q_j is chosen in order to keep the remainder lower than 2.D in absolute value. Practically, q_j = sign (R^j) hence $q_j \in \{1, \bar{1}\}$. An example using the Robertson diagram [11] is sketched in fig. 8.

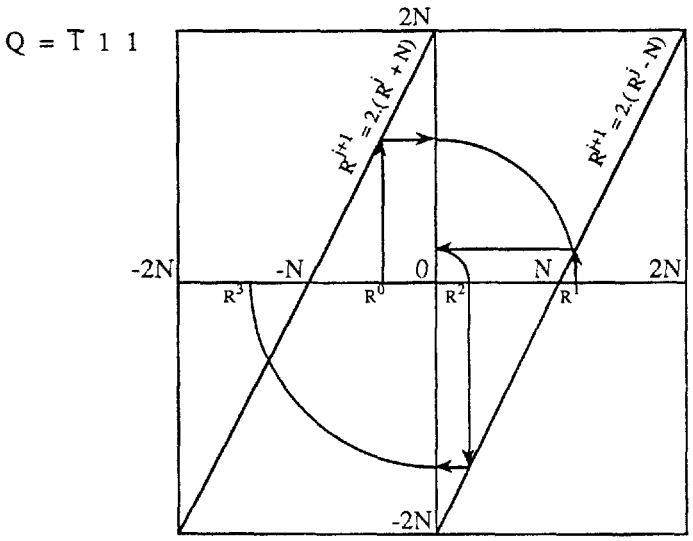


Fig. 8. Robertson diagram of the non restoring division scheme

Starting from R^0, the successive decisions $q_j = \bar{1}, 1, 1$ have to be taken. The problem of this algorithm resides in the testing of the R^j sign because it can only be known through a carry propagation. Indeed, it can be proven that the quotient has a unique representation. Hence, an error in the evaluation of q_j ruins the algorithm. Try for example $q_0 = 1$ in the above example.

If we allow a RSD representation for the quotient and for the partial remainders, the problem can be solved in a quite elegant way. The testing of the three MSD's (Most Significant Digit) of R^j only gives a range which in it resides. For example: R^j is surely negative or R^j is lower than D in absolute value.

Due to the redundant nature of R^j, the two decisions may occur when the ranges overlap. The choice for q_j is given hereunder:

If $R^j < 0$ then $q_j = \bar{1}$.

If $-D < R^j < D$ then $q_j = 0$.

If $R^j > 0$ then $q_j = 1$.

An example using the Robertson diagram is given in fig. 9.

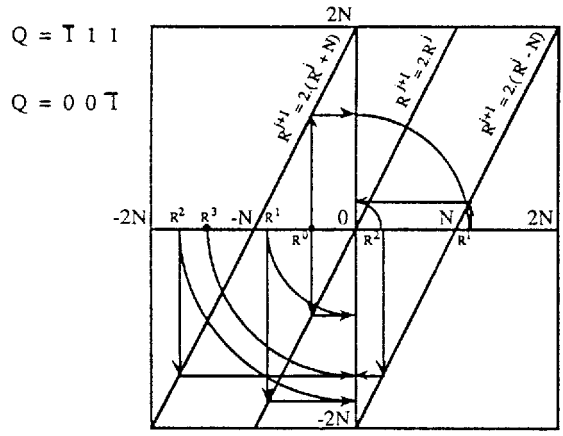


Fig. 9. Robertson diagram of the RSD division scheme

Starting from R^0, we could decide $q_j = 0$ or $\bar{1}$. In fact, this has no effect on the final result: indeed, R^3 is the same because $Q = (\bar{1}\ 1\ 1) = (0\ 0\ \bar{1})$. This technique has been applied with success to a completely different topic: the design of a RSD analog-to-digital converter [12].

More formally, suppose that R^0 is in RSD form and D in two's complement form. Therefore :

$$D = -2^0 .d_0 + \sum_{i=0}^{n} d_i .2^{-i}$$

with $d_i \in \{0, 1\}$ and $d_0 \neq d_1$. Hence: $0.5 \leq |D| \leq 1$.

$$R^0 = \sum_{i=0}^{n+m} r_i^0 .2^{-i} = \sum_{i=0}^{n+m} \left(r^{0\,*}_i - r^{0\,**}_i \right) 2^{-i}$$

with $r_0^0 = 0$, $r_i^0 \in \{1, 0, \bar{1}\}$ and $r^{0\,*}_i$, $r^{0\,**}_i \in \{1, 0\}$. Hence: $-1 < R^0 < 1$. RSD division steps are sketched at fig. 10.

The physical implementation implies that 3 bits are lost after each division step. The classical division scheme with carry propagation is built in order that they can be forgotten. But due to the redundant nature of the intermediate results, this is not guaranteed with the use of RSD even if the arithmetic conditions of the division algorithm are met. This is known as the representation overflow or consistency problem. It implies a further restriction on the choice of the q_j.

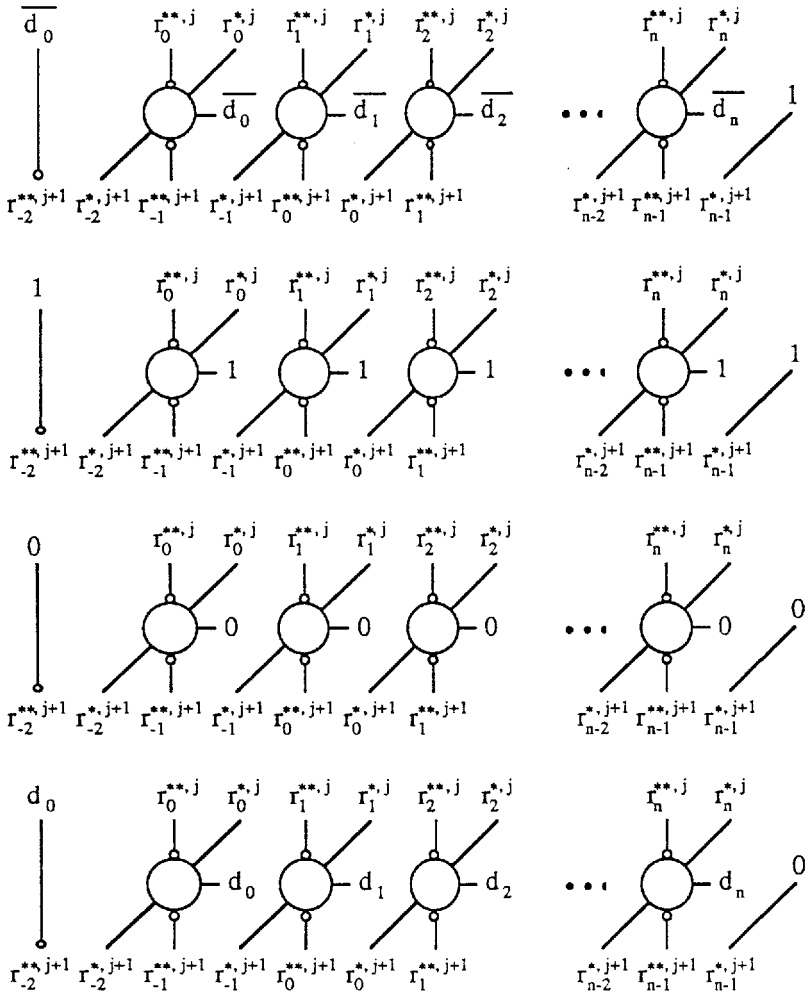


Fig. 10. RSD division steps for $q_j = 1,\ 0,\ \overline{0},\ \overline{1}$

The division algorithm in a redundant number system is governed by two conditions. At each step :

$$-2.|D| < R^{j+1} < 2.|D| . \qquad \text{(Arithmetic condition)}$$

$$R^{j+1} = \sum_{i=0}^{n+m-j} r^{j+1}_i .2^{-i} = \sum_{i=0}^{n+m-j} \left(r^{j+1\,*}_i - r^{j+1\,**}_i \right) 2^{-i} \qquad \text{(Consistency condition)}$$

with $r^{j+1\,*}_i , r^{j+1\,**}_i \in \{1, 0\}$ and $r^{j+1}_i \in \{1, 0, \overline{1}\}$. The consistency condition is met when all r^{j+1}_i are equal to 0 for $i < 0$.

In the following, we suppose that the divisor D is strictly positive. The generalization to negative divisors is immediate. Table I demonstrates the fact that the examination of the three MSD's (Most

Significant Digit) of R^j is sufficient to insure the respect of the two above conditions. The representation overflow is also avoided. From the values of the three most significant bits of R^j, the correct choice of q_j is given as well as the values of the known most significant digits of R^{j+1} in the two distinct cases $d_2 = 0$ or 1. Notice that $R_0^0 = 0$.

r_0	r_1	r_2	q_j	$d_2 = 0$ r_0^{**}	r_0^{*}	r_1^{**}	$d_2 = 1$ r_0^{**}	r_0^{*}	r_1^{**}
1	0	1	1				0	1	1
1	0	0	1				0	0	0
1	0	$\bar{1}$	1				0	0	1
1	$\bar{1}$	1	1				1	1	1
1	$\bar{1}$	0	1				1	0	0
1	$\bar{1}$	$\bar{1}$	0				0	0	0
0	1	1	1	1	1	0	1	1	1
0	1	0	1	1	1	1	1	0	0
0	1	$\bar{1}$	0	0	0	0	0	0	0
0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	1	1	1	1	1
0	0	0	$\bar{0}$	0	0	0	0	0	0
0	0	$\bar{1}$	$\bar{0}$	0	0	1	0	0	1
0	$\bar{1}$	1	$\bar{0}$	1	1	1	1	1	1
0	$\bar{1}$	0	$\bar{1}$	0	0	0	0	1	1
0	$\bar{1}$	$\bar{1}$	$\bar{0}$	0	0	1	0	0	0
$\bar{1}$	1	1	$\bar{0}$				1	1	1
$\bar{1}$	1	0	$\bar{1}$				0	1	1
$\bar{1}$	1	$\bar{1}$	$\bar{1}$				0	0	0
$\bar{1}$	0	1	$\bar{1}$				1	1	0
$\bar{1}$	0	0	$\bar{1}$				1	1	1
$\bar{1}$	0	$\bar{1}$	$\bar{1}$				1	0	0

Table I. Decision table and partial remainders when examining the 3 MSD's

5. RSA ALGORITHM

The RSA cryptosystem is based on the modular exponentiation of large numbers (typically a few hundreds of bits). Let **m** be a message to encipher. The enciphered message **c**
is obtained with the key (e, N) by : $\mathbf{c} = \mathbf{m}^{\mathbf{e}} \pmod{N}$. The deciphering key (**d**, N) allows to recover the original message by $\mathbf{m} = \mathbf{c}^{\mathbf{d}} \pmod{N}$. Both operations are thus identical. Let:

$$\mathbf{e} = \sum_{i=0}^{n-1} e_i.2^i = e_{n-1}.2^{n-1} + ... + e_2.2^2 + e_1.2 + e_0 \text{ with } e_i \in \{0, 1\}.$$

$$\mathbf{m^e} \pmod{N} = (\mathbf{m})^{e_0}.(\mathbf{m^2})^{e_1}.(\mathbf{m^{2^2}})^{e_2} ... (\mathbf{m^{2^{n-1}}})^{e_{n-1}} \pmod{N}$$

The algorithm of the modular exponentiation can be derived from the properties of the modular arithmetic.This leads to the right-to-left binary modular exponentiation and can be expressed as a C program.

```
current = m; result = 1;
for (i = 0; i <= n-1; i++)
{  if (e_i == 1) then result = result * current (mod N);
   current = current * current (mod N);
}
```

The number of modular multiplications steps is n + v(e) where v(e) is the number of 1 in the exponent **e**. For example, with an exponent equal to 65537 = 2^{16} + 1 (the fourth Fermat number), the number of modular multiplications is 19.

Our implementation is based on the RSD algorithms developed in the previous sections. The module operation (mod) is simply a division in which the remainder is kept as result. The basic modular multiplication step is then a multiplication followed by a division (Fig. 11).

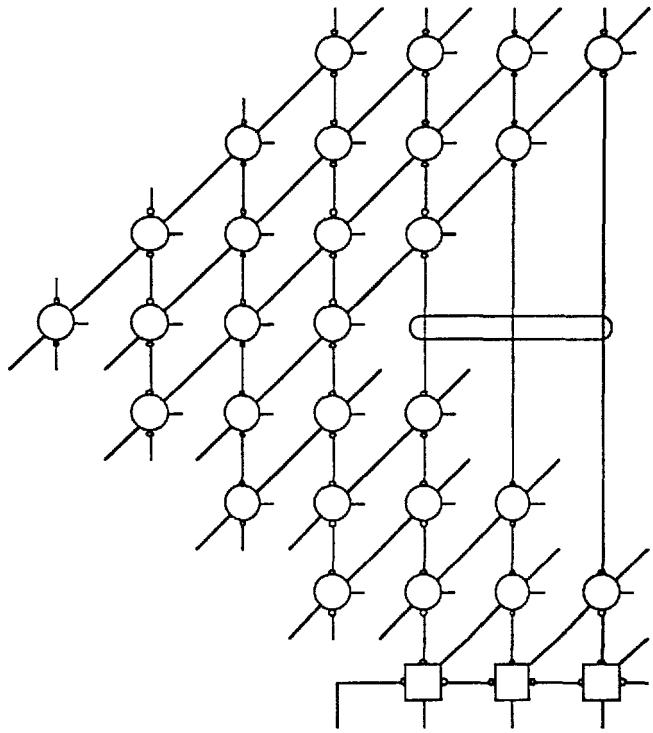


Fig. 11. Modular multiplication step of RSA algorithm

As the size of the operands is large (n = 200 ... 1000 bits), it is impossible to integrate parallel multipliers and dividers on a single chip. Therefore, serial/parallel algorithms are usually used with an adder as basic ALU. Considering a unity add time for the addition (one clock cycle), both multiplication and division are ideally performed in n cycles. In the worst case (all $e_i = 1$), the total number of additions is $4.n^2$ and the associated ideal baudrate is (fclock / 4.n).

The result in redundant form is then converted back to 2's complement form by a carry propagation adder. This conversion is heavily time consuming (≈ 1 µs or 40 cycles with a 25 MHz clock). As this operation seldom occurs, its relative slowness is not a coarse limitation.

6. STATE OF THE ART

The large size of numbers involved in the RSA algorithm leads to large design and processing time. While multiplication algorithms with linear behavior are well known (carry save technique), division usually involves carry propagation because tests are necessary. This considerably degrades the ideal baudrate.

In order to solve this problem, most known realizations use non regular techniques leading to a waste of resources. In two implementations [5, 7], the carry save technique seems to be used for the multiplications while no information is given on the way carry propagation is avoided during division. Anyway, the given throughput does not indicate they use a carry free technique. A recent realization [6] uses a "block" carry save technique while the multiplication and division steps are interleaved. Therefore, the number of divisions steps is higher than N and from the results presented in the paper, the actual baudrate is 2 to 3 times less than the ideal one. On the other hand, the complexity of the basic ALU cell is approximately 1.5 full adder. British Telecom [8, 9] exploits the fact that the maximum carry length is statistically proportional to log(n). By testing each cycle if the carry propagation is completed, the addition time is statistically log(n) cycles. Hence this technique leads to a lower than ideal baudrate (fclock / 4.n.log(n)) while extra circuitry is needed for the test.

7. CHIP ARCHITECTURE

A powerful IC implementation of the RSA algorithm has been carried out, using the method described in the first part of this paper. It is based on a bitslice structure. Each slice contains the minimum hardware to perform such operations i.e. one RSD full-adder, one left/right shift register, 6 registers for keys and partial results storage, one shift register dedicated to I/O and one Manchester carry chain to complete RSD to classical binary form conversion (Fig. 12).

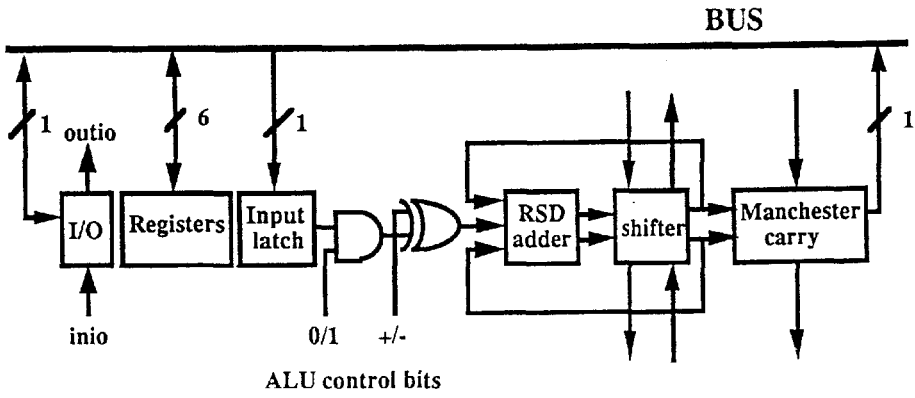


Fig. 12. Bitslice architecture

A combinatorial circuit is dedicated to the control of the ALU (addition or subtraction of 0 or 1) considering the multiplicand bits during the multiplication and the MSB's of the partial remainders during the division. Unlike in other implementations [6, 8], the interleaving between multiplication and division is avoided in order to save extra add cycles. Therefore the LSB's of the partial results must be stored in a RAM (Fig. 13).

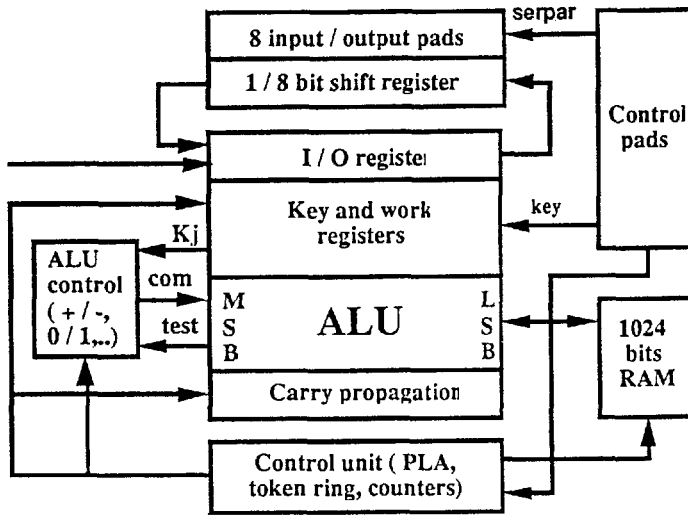


Fig. 13. Chip architecture

The RSD technique leads to a very regular and powerful bitslice realization with a reduced hardware. The final baudrate is near the optimum (fclock / 4.n), except for the few percent lost by the RSD to

classical binary form conversion (carry propagation). To our best knowledge, RSD arithmetic is used for the first time for RSA computation.
The two next sections are devoted to in depth study of specific design problems that have been encountered during the design of the RSA processors.

8. OPERATIVE PART DESIGN

The two main constraints in the design of this RSA chip were a reduced chip area and a low power consumption. These have been achieved by using static memory points and static devices wherever it was possible. As example, two optimizations will be described in some depth: the use of a static I/O shift register and the design of a bitslice structure with a RAM-like access to the registers.
The design of a good interface between a chip and the outside world is a delicate task. The difficulty lies in the fact that the outside world works in an asynchronous way (clocked by ψ_1 and ψ_2) while the chip works strictly synchronously (with ϕ_1 and ϕ_2). The problem has been solved in a quite elegant way by using a static shift register (Fig. 14).

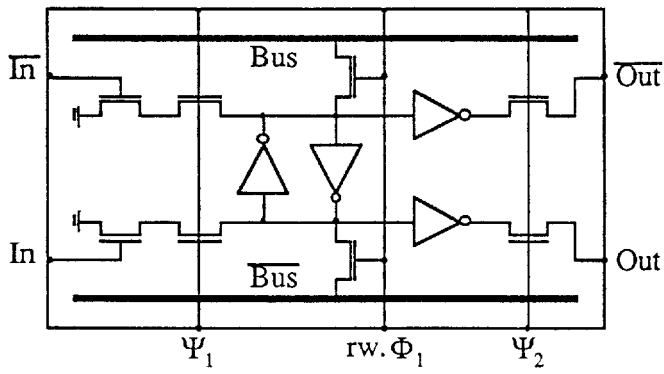


Fig. 14. Static I/O shift register

At the beginning of a RSA computation, a new data is read from the shift register and a result is written to it. During this phase, ψ_1 is low and the accesses are controlled by the command rw.ϕ_1. Then, this command remains low, disconnecting the shift register from the rest of the chip. The cell acts then as a classical shift register controlled by the two non overlapping phases (ψ_1, ψ_2), asynchronous with the chip's own clock. In parallel with a RSA computation, a new data can be shifted in while the stored result is shifted out. This solution is very flexible while quite economical in terms of silicon area.
In the case of multiplication and especially in case of squaring, the same data register must read twice: the first time in parallel (to be written in the input register of the ALU) and serially for the second time, starting with the LSB (to compute the serial/parallel multiplication). A classical way to perform this function is to use a shift register as presented by fig. 14. This solution has two main drawbacks:

- the area of such a register is quite high; notice that, in the RSA algorithm, at least 3 of these registers are necessary;
- in a 1024 bits datapath, the power consumption of these registers becomes critical; indeed with a state change probability of 0.5, 512 cells per register sink current each cycle; this could lead to a dramatical power consumption.

To avoid these drawbacks, we have implemented a RAM-like hardware sketched in fig. 15.

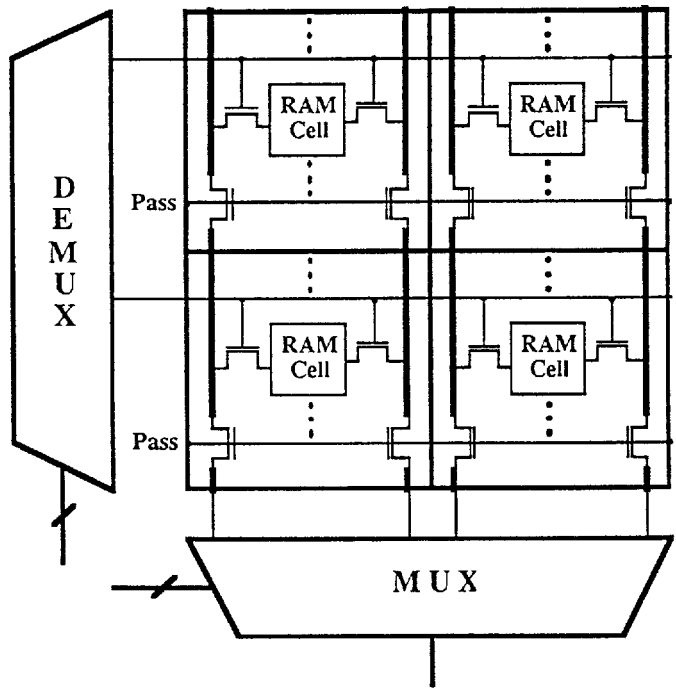


Fig. 15. RAM-like access to the bitslice registers

The registers cell is a 6 T static RAM cell connected to the datapath bus. During the parallel access, the **pass** command is low and each bitslice uses its own bus. Then **pass** is set to high and the entire bitslice acts like a RAM matrix, with wordlines (RW register command) controlled by a demultiplexor and busses as bitlines. The datapath is organized in 8 rows of 128 bits. Each time the serial reading procedure is activated, 128 bits are read from the bitslice and stored at the input of the multiplexor; data are then multiplexed in order to choose the desired bit of the multiplicator. Due to the number of switches on the bitlines, the reading process is rather slow. This is not a real drawback as 128 cycles are available. Compared with shift registers, the power consumption is reduced theoretically by a factor of n (the number of bits); the area inside the datapath is reduced by a factor of 2. At the counterpart, the control of this structure is more complicated than for shift registers but represents only a small silicon area overhead.

9. CONTROL UNIT DESIGN

The RSA program to be implemented had the following features:

- it is rather sequential;
- the number of flags tested at the same step is small (no more than 3);
- loops must be performed.

Aside from the main program, some subprocesses must be executed during multiplication and division loops. As mentioned earlier, reading from bitslice memory is executed 8 times during a 1024 bits multiplication; in the same way, reading/writing to the multiplication tail memory is executed 128 times; these processes are executed in more than one cycle. The control of these processes is devoted to special-purpose units best implemented through "token ring" solution.

The control unit is built around a PLA, a counter datapath and two "token ring" units. The choice of a PLA as heart of the control unit was not so obvious due to the inherent characteristics of the program. A non-classical "token ring" unit seemed to be well-suited. It was in fact rejected for two main reasons:

- it is difficult to adapt the pitch of both command generation plane and "token ring" part;
- this architecture doesn't lead by itself to a regular layout and automatic generation of such units is not easy.

Classical way to design CMOS synchronous PLA's uses multi-phase logic (due to the logical structure of PLA's, it may be shown that at least 3 phases are mandatory) [13]. To fit the PLA in our 2 non-overlapping phases clock scheme, we used the schematic presented at fig. 16; a clocked inverter was inserted between the two planes (AND/OR).

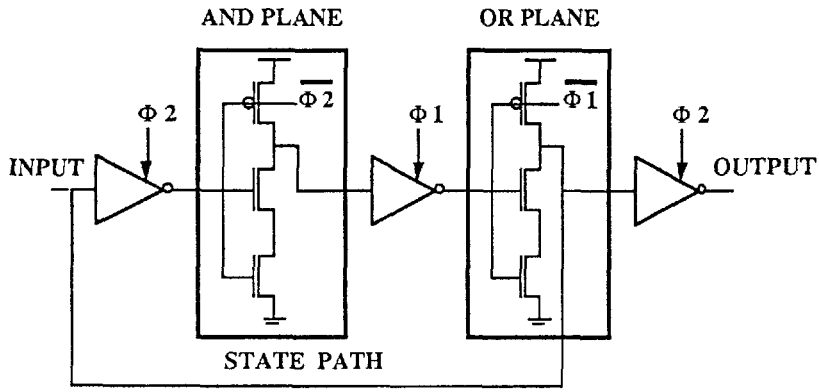


Fig. 16. PLA electrical schematic

These two are both precharged NOR planes which are the fastest logic gates. The area penalty due to this clocked inverter was rather low in our case because of the PLA size (12 flags, 6 state bits, 42 outputs, 82 minterms).

10. MAIN FEATURES

The chip's performances were first evaluated from simulations. All critical cells have been simulated for the worst case at 30 MHz (SPICE) and switch level simulation has proven the correct working of the chip's parts (bitslice, control unit, ...). The chip has been processed and tested (fig. 17-19).

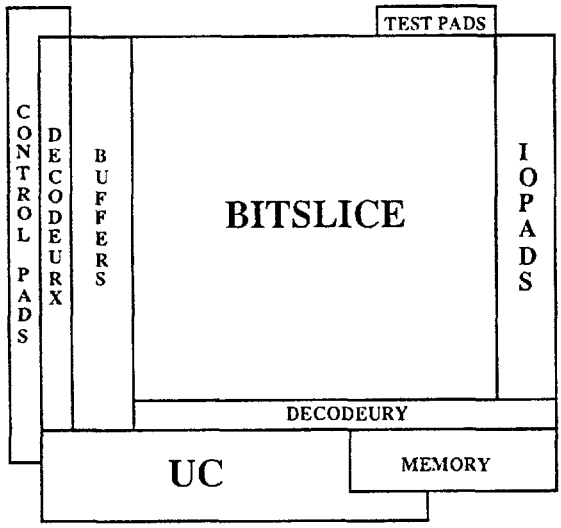


Fig. 17. Chip floorplan

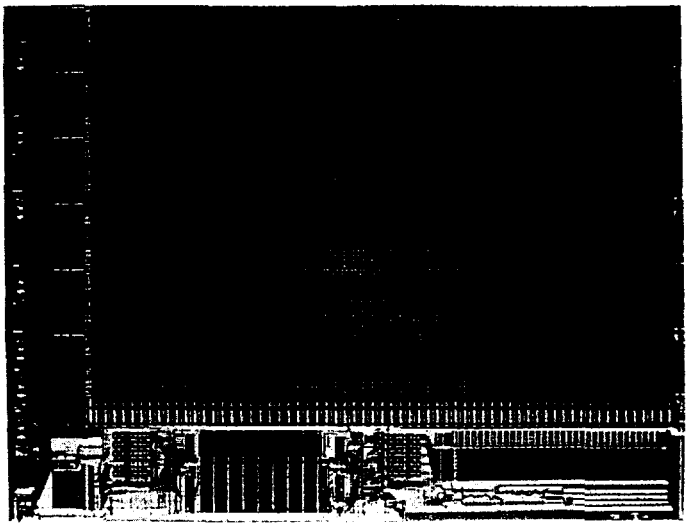


Fig. 18. Chip microphotograph

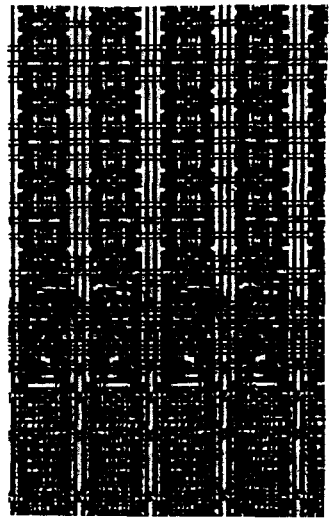


Fig. 19. Detail of the bitslice

The chip is 95 % functional and tests results are in good concordance with the expected baudrates. Table II lists the main technical characteristics of the chip.

Technology:	CMOS 2 µm with 2 metal layers
Transistor count:	180.000.
Chip size:	80 mm^2 (9.3 x 8.7).
Power dissipation at 25 MHz:	500 mW under 5 V.
Baudrate (all 1024 bits words):	8 kbits/s with a 25 MHz clock.
Key storage:	Static, 2 pairs on the chip.
I/O:	Serial or 8 bits parallel, asynchronous.
Control:	On chip.

Table II. Main features of the RSA chip

Coarse comparisons between different realizations is not always obvious. A typical example is the maximum size of numbers that can be processed by one or more IC's. Our implementation and some others [5, 8, 9] can process numbers up to the size of the ALU. In our case, we feel that 1024 bits is sufficient to meet code security requirements, but 256 is surely not! In the case of British Telecom [8, 9], the choice seems to have been dictated by a British law prohibiting the use of RSA with keys greater than 256 bits. On the other hand, the Cylink chip [7] is cascadable in order to process numbers up to 16 kbits. Finally, the CNET chip allows the processing of 1024 bits words with a 256 bits ALU through the use of a multi-precision arithmetic. Therefore, a refined discussion of respective advantages and drawbacks of all realizations will not be presented in this paper.

Hereafter is listed all available information published on RSA single chips (Table III). As cryptographic products appear to be as secret as information they process, full data usually are not available. The number of bits indicates the size of the ALU physically implemented on chip, thus the size of the datapath in bitslice realizations. Baudrates estimations are for 512 bit words, considering they are inversely proportional to the wordlength.

	Technology	#bits	# Trans.	Die size	Clock	Baudrate
MIT (1980)	NMOS 4µm	512	40.000	44 mm^2	4 MHz	1.2 kbits/s
CNET-CNS (1988)	CMOS 2 µm	256	100.000	88 mm^2	10 MHz	1.6 kbits/s
BR. TEL. (1988)	CMOS 2 µm	256				2.5 kbits/s
CYLINK (1987)		1028				5 kbits/s
UCL (1988)	CMOS 2 µm	1024	180.000	80 mm^2	25 MHz	16 kbits/s

Table III. Comparison between single chip RSA processors

11. CONCLUSIONS

A new carry-free division algorithm has been described. Its features like regularity (its hardware structure is identical to parallel multipliers), minimum hardware per bit (one full-adder), small test unit make it suitable for powerful integration in applications like DSP processors, floating point units,...

A 1024 bits RSA chip has also been presented. Thanks to the division algorithm characteristics, high throughput (8 kbits/s) and integration of a 1024 bits datapath in a standard 2 µm technology were achieved on a single chip. This chip includes also security features (no readable keys, no possible attempt to keys integrity,...) needed in cryptographic applications.

Speed improvement by using Booth's algorithm for multiplication and the interleaving between multiplication and division steps is currently studied. This could lead to a four times faster implementation for a 30 % silicon area increase.

REFERENCES

[1] I. Verbauwhede, F. Hoornaert, J. Vandewalle and H. de Man, "Security and performance optimization of a new DES data encryption chip," in *Proc. ESSCIRC '87,* pp 31-34, 1987.

[2] W. Diffie, "The first ten years of public key cryptography", *Proceedings of the IEEE,* Vol. 76, N° 5, pp 560-577, May 1988.

[3] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi and N. Takagi, "Design of high speed MOS multiplier and divider using redundant binary representation," in *Proc. of the 8th Symp. on Computer Arithmetic,* IEEE, pp 80-86, 1987.

[4] H. Edamatsu, T. Taniguchi, T. Nishiyama and S. Kuninobu, "A 33 MFlops floating point processor using redundant binary representation", in *Proc. ISSCC 1988,* pp 152-153, 1988.

[5] R. L. Rivest, "A description of a single chip implementation of the RSA cipher", *Lambda,* Fourth Quarter 1980, pp 14-18.

[6] Ph. Gallay and E. Depret, "A cryptography processor", in *Proc. ISSCC 1988,* pp 148-149, 1988.

[7] D. B. Newman, J. K. Omura and R. L. Pickholtz, "Public key management for network security", *IEEE Network Magazine,* Vol. 1, N° 2, pp 11-16, April 1987.

[8] P. Ivey, A. Cox, J. H. Arbridge and J. Oldfield, "A single chip public key encryption sub-system", in *Proc.ESSCIRC 1988,* pp 241- 242, 1988.

[9] P. Ivey, A. Cox, J. H. Arbridge and J. Oldfield, "A single chip public key encryption sub-system", IEEE J. Solid-State Circuits, Vol. SC-24, N° 4, pp. 1071-1075, June 1988.

[10] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic", *IRE Trans. Electron. Computers,* Vol. EC-10, pp 389-400, Sept. 1961.

[11] K. Hwang, *Computer Arithmetic Principles, Architecture and Design,* John Wiley & Sons, 1979.

[12] B. Ginetti, A. Vandemeulebroecke and P. Jespers, "RSD cyclic analog-to-digital converter," in *Proc. 1988 Symposium on VLSI Circuits,* Tokyo, Japan, 1988, pp 125-126.

[13] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design,* Addison-Wesley, 1985.