Fast Architectures For FPGA-Based Implementation of RSA Encryption Algorithm

Omar Nibouche[1], Mokhtar Nibouche[2], Ahmed Bouridane[3], and Ammar Belatreche[1]

[1] Faculty of Engineering, University of Ulster at Magee, Northland Rd, Derry, BT48 7JL, UK
o.nibouche@ulster.ac.uk, a.belatreche@ulster.ac.uk

[2] Faculty of Computing, Engineering and Mathematical Sciences, University of the West of England, Bristol BS16 1QY, UK
mokhtar.nibouche@uwe.ac.uk

[3] School of computer Science, Queen's University of Belfast, 18 Malone Rd, Belfast BT7 1NN, UK
a.bouridane@qub.ac.uk

Abstract

In this paper, new structures that implement RSA cryptographic algorithm are presented. These structures are built upon a modified Montgomery modular multiplier, where the operations of multiplication and modular reductions are carried out in parallel rather than interleaved as in the traditional Montgomery multiplier. The global broadcast of data lines is avoided by interleaving two or more encryption/decryption operations onto the same structure, thus making the implementation systolic and scalable. The digit approach has been adopted in this paper. This methodology is based on varying the digit size and the level of pipelining of the structures. This parameterised approach presents the designer with an efficient way of choosing the architecture that suits better his/her requirements in terms of speed and area usage, an issue of critical importance to the resources-limited FPGA chips. The results of implementation using FPGA have shown that the proposed RSA structures outperformed those structures built around the traditional Montgomery multiplier in terms of speed, thanks to avoiding global lines broadcast.

I. Introduction:

In the recent past years, the use of software tools and hardware devices for security functions has increased dramatically [8, 10-12, 16, 18-22]. Security issues play a crucial role in wide spreading the use of many computer and communication systems, such as the Internet, which more and more people are using to transmit sensitive information such as credit card numbers. A central tool for achieving system security is the cryptographic algorithms. Privacy and fraud concerns can be addressed through the use of various security primitives such as data encryption, which can be used with the appropriate protocols to construct secure and trusted networks [10-12, 16-22]. Cryptographic algorithms require immense processing power that may cause a bottleneck in high-speed networks. In the past, system developers have had to use software-based techniques in order to achieve the agility necessary to maintain compatibility in the presence of different algorithms and protocols [8, 16, 18-21]. However, software solutions are particularly inefficient for the computationally demanding asymmetric algorithms (also called public key cryptographic algorithms). To achieve hardware agility the use of re-configurable logic is an appropriate solution especially that the recent development of FPGA technology means that these devices are now large enough so that it is possible to implement a reasonable application in a single FPGA device [11, 16, 18-22]. The unlimited re-configurability of an FPGA permits a continuous sequence of custom circuits to be employed, each optimized for the current task. This has led to a relatively new concept for computing where such a device can be used as a co-processor coupled with a host machine in order to speed up the computation of a specific task. Furthermore, in terms of security, another benefit of using dedicated hardware is that security attacks become more difficult, as the secrets can be contained within the coprocessor using nonvolatile memory, which is externally inaccessible [16, 17].

In 1976, Diffie and Hellman introduced the idea of public key cryptography [6], which is now widely used to provide confidentiality, authentication, data integrity and non-repudiation. Since then, numerous public-key cryptosystems have been proposed. All these systems base their security on some mathematical one-way functions. RSA [7] is the most widely used public-key cryptosystem. An RSA operation is a modular exponentiation operation, which requires repeated modular multiplications. For security reasons RSA operand sizes need to be 1024-bits or greater [8, 10-12, 16, 18-22]. In 1985 Montgomery introduced a new method for modular multiplication [9]. The Montgomery multiplication algorithm is the most efficient modular multiplication algorithm available. This method has proved to be very efficient and is the basis of many implementations of modular multiplication, both in software and hardware [9]. It replaces trial division by the modulus with a series of additions and divisions by a power of 2, which is very easy to implement. For this reason, this algorithm has formed the basis of many of the RSA hardware architectures reported to date [3-5, 8, 10-12, 14, 16, 18-22].

This paper proposes new and efficient FPGA based hardware implementations of RSA algorithm based on a modified Montgomery's modular multiplication presented in [3]. A systolic approach for the implementation strategy has been adopted in this paper in order to achieve a high clock frequency. Many systolic architectures of RSA algorithm have been proposed in the literature [18-22]. In this paper, the focus is on avoiding global lines broadcast and using only nearest neighbor communication wherever it is possible. For FPGA implementations, serial and digit architectures that interleave more than one encryption operation and balance area-usage and speed requirements are presented. The paper is organized as follows: section II reviews Montgomery' modular multiplication algorithm and the author's modified version of the algorithm. This modified algorithms uses two conventional multipliers (with one multiplier being slightly different) to carry out the operation of modular multiplication. These two multipliers operate in parallel. A brief discussion about the bound of the result of the

algorithm is presented. As a matter of fact [19-22], the bound of the result of Montgomery algorithm is changed in such a way that, if multiple modular multiplication operations are to be carried out iteratively, with the result of one iteration used in the next one, no subtraction operation would be required. Section III reviews Montgomery architectures and discusses the adopted implementation approach. To design fully systolic architectures, more than one instance is interleaved onto the same structure. Furthermore, the digit approach presents the designer with a trade off to find the best structure that matches the design needs in terms of area-usage, speed requirements, and the number of encryption operations interleaved onto the same structure [1, 3-5]. The implementation results of the proposed architectures using FPGA are analyzed in Section VI, from which the conclusions are derived in section V.

II. Montgomery modular multiplication

RSA algorithm consists of a modular exponentiation operation, which is, on its turn, broken into two modular multiplication operations [7-8, 10-12, 18-22]. There are two classes of modular multiplication algorithms. A clear distinction between these two classes can be based upon the data format. The Most Significant Bit First (MSBF) class of modular algorithms are either comparison/subtraction based algorithms or Look Up Table (LUT) based algorithms [2, 13, 15]. On the other hand, Montgomery's algorithm [9] makes the Least Significant Bit First (LSBF) approach useful when performing numerous successive modular multiplication operations, such as modular exponentiation. This algorithm avoids long chains of carry propagation, and therefore speeds up both the modular multiplication and squaring operations required during the exponentiation process [9]. Rather than using subtraction/division operations, this MSBF method computes the modular product of two numbers multiplied by a scaling factor, which is relatively prime to the modulus. This allows the algorithm to perform divisions by a power of two, which is a shift operation, thus making the design of modulo multipliers easier [9].

Large size operands are used in security applications [3-5, 8, 10-12, 16, 18-22]. Therefore, when implementing Montgomery's algorithm, avoiding the global broadcast of data is very important. Such a global broadcast can lower the clock frequency [3-5]. As shown in [3-5], a way to avoid such global lines is to use a digit implementation and to interleave multiple instances onto the same structure. Furthermore, the digit approach can be the appropriate solution to avoid the large hardware usage of the parallel based implementations and the slow speed that characterises the serial architectures [1, 3-5].

Let the modulus M be an integer within the range $[2^{n-1}, 2^n[$ and let R be 2^n. Montgomery's multiplication algorithm requires R and M to be relatively prime, i.e., $gcd(R,M) = gcd(2^n,M) = 1$, which is satisfied if M is odd as it is required by the algorithm. By exploiting this property, Montgomery algorithm introduces an efficient multiplication scheme, which computes the modular product, P, of two given integers, A and B, as follows [9]:

$$P = \left\langle A\,B\,R^{-1} \right\rangle_M \qquad (1)$$

where R^{-1} is the inverse of R modulo M. In order to describe his algorithm, Montgomery introduced the quantity, M', which is the inverse of $-M$ modulo R, i.e.:

$$M' = \left\langle -M^{-1} \right\rangle_R \qquad (2)$$

The computation of the Montgomery multiplication is carried out using *Algorithm 1*s follows:

Algorithm 1

{

T = AB

$P = (T + \langle TM' \rangle_R M)/R$

If $P \geq M$ *then* $P = P-M$

}

The algorithm uses the multiplication modulo R and the division by R, which are faster and simpler than the computation of AB *mod* M which involves division by M. The algorithm is only efficient when multiple operations are carried out, such as in the modular exponentiation operation when it is broken into modular multiplication operations [12].

For a hardware implementation, a systolic array can be derived from the bit-wise version of Montgomery multiplication as shown in *Algorithm 2* [9]:

Algorithm 2

$$0 \leq A = \sum_{i=0}^{n} a_i 2^i < M, \quad 0<B<R, P_{-1} = 0$$

{

for i= 0 to n-1

$q_i = \langle P_{i-1} + a_i B \rangle_2$

$P_i = P_{i-1} + a_i B + q_i M / 2$ *}*

If $P_{n-1} \geq M$ *then* $P_{n-1} = P_{n-1} - M$

Algorithm 2 interleaves the multiplication steps with the reduction steps. One bit of the partial result is used to select the reduction value. As shown in the algorithm, the LSB of the partial result of the previous iteration, P_{i-1}, together with the bit-product $a_i b_0$, directly selects the modular reduction value, which is either *0* or M. At each iteration, the LSB of P_i equals 0. P_i is then shifted one position to the right. After n iterations of the algorithm, the scaling factor is equal to 2^{-n}. Therefore, the final result is $\left\langle AB2^{-n} \right\rangle_M$ as shown in equation (1). The partial results fall in the range $[0,2M[$ and as such an operation of comparison/subtraction is necessary at the end of the algorithm [2]. The critical delay of Algorithm 2 occurs during the calculation of the P values given by the three input addition, $P_i = P_{i-1} + a_i B + q_i M / 2$. The main contributing factor to this delay is the carry propagation [10-11] or in the case of serial multipliers, the global broadcast line resulting from using very large operands [18-22].

As shown in *Algorithm 2*, the Montgomery's modular multiplication algorithm is equivalent to two interleaved conventional multiplication operations, where the second one is a reduction operation which is required to ensure the partial results within the range $[0, 2M[$.

The modified algorithm shown in [3] uses two conventional multipliers to build a Montgomery structure. In this way, one multiplier carries out the operation of multiplication and the other multiplier carries out the modular reduction operation. These two operations are carried out in parallel. Let T be the product of $A \times B$, i.e.:

$$T = AB = T_0 + 2T_1 R \qquad (3)$$

Algorithm 3:

$0 \le A = \sum_{i=0}^{n+1} a_i 2^i < 2M$, $0 \le B = \sum_{i=0}^{n+1} b_i 2^i < 2M$,

$a_{n+1} = b_{n+1} = 0$, $P_{-1}=0$

$P'_{-1} = 0, c_0 = 1, P''_i = 0$

For i= 0 to n+1

{Step1: $P'_i = P'_{i-1} + 2a_iB$

Step 2: $P'_{i,i} + c_i = \overline{T}_{0,i} + 2c_{i+1}$

Step 3: $q_i = P''_{i,i} \oplus \overline{T}_{0,i}$

Step 4: $P''_i = P''_{i-1} + q_iM$

Step 5: $P_i = (P'_i + P''_i)/2^{i+2}$ }

As shown in *Algorithm 3*, the modular multiplication is broken into two concurrent multiplication operations and computes Montgomery's multiplication by using the two's complement of T_0 to select the reduction value. The results from the two multipliers are then added and a division by $4R$ (or 2^{n+2}) follows. The algorithm start by computing the product AB as in (4). It then uses $\overline{T}_0$, the two's complement of T_0 to compute the reduction value.

However, in the original algorithm (see *Algorithm 2*) after each multiplication a reduction operation is required (the last step in *Algorithm 2*). The input has the restriction A, B < M and the output P is bounded by $P < 2M$. As a consequence, *if* $P > M$, M must be subtracted so that the output can be used as input for the next multiplication. To avoid this subtraction [9, 19-22], a new bound for the inputs is A, $B < 2M$, thus, the output is also bounded by $P < 2M$. In this way, the result P is equal to:

$$P = (T + \langle \mathrm{TM}' \rangle_{4R} M)/4R = A \times B \times 2^{-n-2} < 2M \quad (5)$$

At the $i+1^{th}$ iteration of *Algorithm 3*, the sum $P'_i + P''_i/2^{i+1}$ is equal to:

$$P'_i + P''_i = 3M(1 - 1/2^{i+1}) \quad (6)$$

Therefore, after the $(n+1)^{th}$ iteration of the algorithm:

$$P'_n + P''_n/2^{n+1} = 3M(1 - 1/2^{n+1}) \quad (7)$$

At the $(n+2)^{th}$ (i.e., the last iteration), and since $a_{n+1} = b_{n+1} = 0$, *the result* $P = (P'_{n-1} + P''_{n-1})/2^{n+2}$ is equal to:

$$P = 2M - 3M/2^{n+2} < 2M \quad (8)$$

Thus, the subtraction operation is avoided. In practice, the result of each modular multiplication operation can immediately be used as an input for the next operation, as required by RSA algorithm.

III. Implementation Approach:

The serial approach processes the data serially where at every clock cycle a single data bit is fed to the RSA processor to be processed. In contrast, the parallel approach processes the data bits in a parallel fashion in just one clock cycle. Many RSA implementations are bit-serial based structures [10-11, 16, 18-22]. This is essentially due to their design simplicity and low hardware area-usage requirements. However, when high sample rates are required, the family of bit-serial architectures leads to a slow system speed [1]. To avoid this problem, and thus, reach higher system speed, it is clear that a move towards higher bit-width operations is necessary. The bit-parallel approach presents a solution that overcomes the speed problems of many systems [1]. Unfortunately, this solution comes with its own drawback, as it requires an important area-usage, and pin-out, which may not be satisfied in the resource-limited FPGA chips [1]. For applications for which the bit-serial approach is slow and the bit-parallel one is faster than required and occupies a large area, a trade off must be found [3-5]. The concept of digit-serial arithmetic has been adopted in this paper as a compromise between the bit serial and the bit parallel arithmetic. The systems based on this arithmetic process more than one bit in one clock cycle. The number of bits processed in one clock cycle, which varies from a single bit to the word-length, is referred to as the digit size. Since the digit size is variable, the digit approach provides the designer with a flexible and efficient area-time method to find the speed and the area that match the design needs through an appropriate choice of the digit size. Pipelining the digit structures is of capital importance [1, 3-5]. Traditionally, the digit arithmetic structures, such as Montgomery multipliers, cannot be pipelined beyond the digit level due to the presence of feedback loops. However, based on the use of the feedback loop pipelining technique [3-5], the aim of designing digit structures that are pipelined at the bit level with the minimum number of operations interleaved onto the same structure can be achieved [3-5]. This technique basically pipelines the feedback loops by adding Flip Flops (FFs)/latches to feedback loops.

Algorithm 3 is implemented using two bit serial multipliers, a serial adder for use to perform the addition of step 5, and a serial two's complement circuit that is used to two's complement the product AB, as shown in Figure 1. The LSB cell of the second serial multiplier is different from the remaining cells, which are gated Full Adders [3]. During the n first cycles, the LSB cell generates the bit q_i to select the reduction value. The clock path of this bit serial modular multiplier is equivalent to that of a gated FA (which a FA to which an AND gate is connected) and a latch. However, the data lines, such as the multiplier input a_i, are broadcast to all the cells. Such a global distribution lowers the clock frequency, especially for large operands, which are usually used in cryptography.

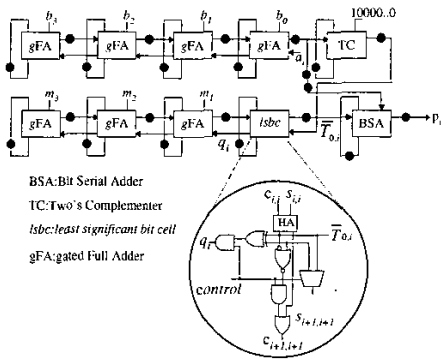


Figure 1. A serial Montgomery Multiplier derived from Algorithm 3 [3]

Another bit-serial structure that implements *Algorithm 3* is shown in Figure 2. The same basic cells used in the multiplier of Figure 1 are used to build this systolic multiplier structure, except that the overall number of latches has increased. It is worth noting that the multiplier uses nearest neighbour communications only and that there are two latches in the feedback loops. Thus this structure is fully systolic and the global distribution lines are avoided.

The design of this architecture has been carried out by interleaving two modular multiplication operations onto the same structure and

by pipelining the feedback loops, a technique that has been used in many works [3-5]. In general, the number of operations interleaved onto the same structure depends on the number of latches added to the feedback loops. The two latches in the loops of the serial multiplier and adder of Figure cope with feeding two different set of operands to the multiplier.
The suggestion made by [22] is that the two interleaved modular multiplication operations are the two operations involved in the modular exponentiation. In this way, the serial multiplier can be used as a bit serial-parallel modular exponentiation structure.

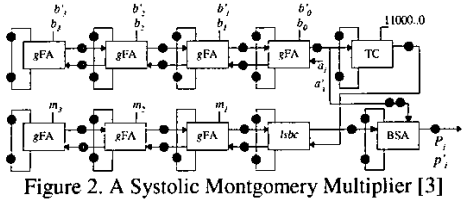


Figure 2. A Systolic Montgomery Multiplier [3]

The number of operations interleaved onto the same structure depends on the number of latches added to the feedback loops. As shown in [3], this structure was derived from Algorithm 3 by using, in the general case, the scheduling vector *[J, 2J]* on the multiplication Dependency Graph (DG) [3]. Therefore, *2J* latches are required in the carry feedback loop, *2J* operations have to be interleaved. In Figure 2 the scheduling vector *[1, 2]* was used. This structure was later unfolded to get digit modular multiplier structures pipelined at the bit level. The Montgomery's digit modular multiplier, as presented in [3], is shown in Figure 3. It uses two digit multipliers, a digit adder, and a digit complement circuit.

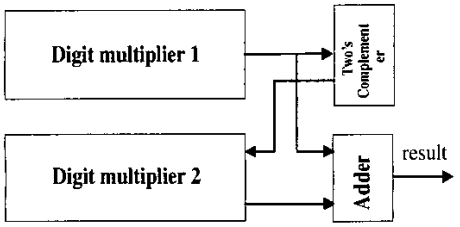


Figure 3. The Digit Montgomery Multiplier described in [3]

As described in [3], the starting point of the design is based on the 4-bit digit multiplier of Figure 4 (or an unfolding factor *J* of *4)*. The second step is to add *2J-1* latches (or 7 latches) to the loops of the multiplier. In the next step, the retiming of the structure is carried out using horizontal and then vertical cutest lines, which leads to the digit multiplier of Figure 5 [3]. The notation in Figure 5: x_j^i is the j^{th} bit of the i^{th} set of data x. However, the level of pipelining achieved is such that there are latches at every output of the cells, the connections from the last row of the digit multiplier of Figure 5 to the top row are not local connections. To circumvent this problem, latches were added to the feedback loops. For an unfolding factor of *J*, *J-1* latches (3 latches in Figure 6) are added to each loop so that the upward lines are no longer broadcast to the top cells and as such, the resulting structure has its connections fully localised. The behaviour of the digit structure changes from a step to another. In Figure 4, the digit multiplier carries out a single operation. In Figure 5, the multiplier interleaves *2J* (*8* in Figure 5) multiplication operations while in Figure 6, *3J-1* (*11* in Figure 6) operations are time multiplexed onto the digit structure [3].

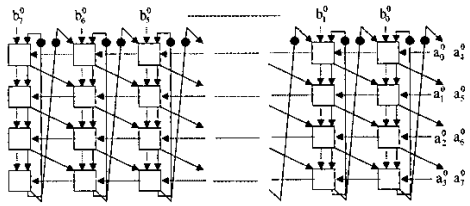


Figure 4. A 4-bit Digit Multiplier

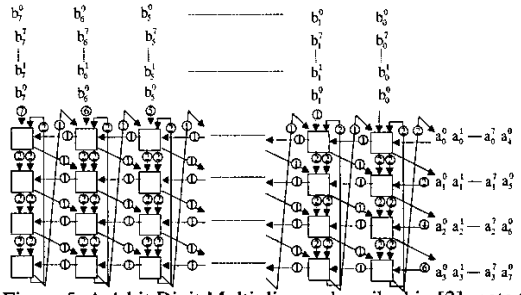


Figure 5. A 4-bit Digit Multiplier as described in [3]: note the global line broadcast though the critical logic path is one FA

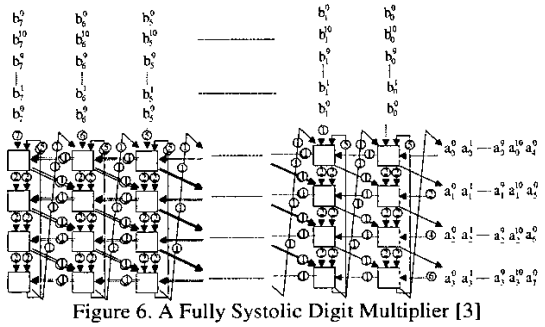


Figure 6. A Fully Systolic Digit Multiplier [3]

IV . Modular Exponentiation and RSA Cryptography

In 1977, Rivest, Shamir and Adleman [7] introduced the cryptographic algorithm that has shown to be the most attractive for implementation in hardware or software [8, 11, 16, 18-22]. It is based on the computation of modular exponentiation using prime numbers. As a matter of fact, the security of the algorithm is based on our inability to efficiently factorise large numbers [7]. For the purpose of keeping the data as secured and secret as possible, current implementations of the RSA are suggested with keys of 1 Kb and up to 2 Kb word-length [8, 18].

As it was suggested by [7], the modulus M of the RSA algorithm is the product of two suitably generated secret prime numbers P and Q: $M = P \times Q$. The public exponent (also known as the encryption key) E is randomly chosen such that it is prime to $(P-1)\times(Q-1)$. The secret exponent (also known as the decryption key) D is computed using the extended Euclidean algorithm such that:

$$\langle E \times D \rangle_{(p-1)(Q-1)} = \langle 1 \rangle_{(p-1)(Q-1)} \qquad (9)$$

P and Q must never be revealed. Once used, the two prime numbers P and Q are no longer required and must be discarded. A message is divided into words of the same word-length as the

product M. If M is encoded on k bits, an $n = km$ bits message is divided into m blocks of k-bit integers each. A k-bit word A is encrypted into a word $Enc(A)$ defined by $Enc(A) = \langle A^E \rangle_M$. An encrypted word B is decrypted into a word $Dec(B)$ defined by $Dec(B) = \langle B^D \rangle_M$.

These two operations, the RSA encryption and decryption, are mutually inverse: $Enc(Dec(A)) = Dec(Enc(A)) = A$ with $0 \le A < M$. This indicates that the original data can be recovered through the RSA encrypt/decrypt process.

The operation of modular exponentiations is carried out iteratively by repeating a modular multiplication and modular squaring operation, as described in algorithm 4 for the case of a LSBF method. The algorithm computes B, which is the remainder of the division of A^E by the modulus M. During every iteration, two modular multiplications are carried out, which can be done in parallel or pipelined using the same multiplier structure [3, 8, 22].

Algorithm 4

$B = \langle A^E \rangle_M$

$E = \sum_{i=0}^{n-1} e_i 2^i$, $P_0 = \langle 2^{n+2} \rangle_M$, $B_0 = \langle 2^{n+2} A \rangle_M$, $B = P_n$

for $i = 0$ *to* $n-1$

$\{ B_{i+1} = \langle B_i^2 \rangle_M$

if $e_i = 1$ *then* $P_{i+1} = \langle P_i B_i \rangle_M$

else $P_{i+1} = P_i$ $\}$

The modular multiplication and squaring operation are based on Montgomery algorithm; therefore, the scaling factor 2^{n+2} is used to neutralize the 2^{-n-2} factor introduced by this algorithm. The range of the result of each iteration is that specified on *Algorithm 3*. Thus no subtraction operation is required at the end of each iteration. At the end of the encryption operation, a multiplication by 1 is required to take away the effect of the 2^{-n-2} factor introduced by *Algorithm 4*. Finally, for the range to fall in the range $[0, M[$, a comparison/ subtraction operation is required.

Two classes of RSA structures that implement *Algorithm 4* are presented. The first class requires two Montgomery multipliers: the first one is used for modular squaring and the second is used for modular multiplication. The general skeleton of this class of structure is depicted in Figure 7. Registers are used to store the result form the modular squarer in a parallel and a serial way. The parallel register associated with squarer is used to feed the parallel operand to the squarer while the serial register is used to feed the serial input to both the squarer and the multiplier. The parallel register associated with multiplier in Figure 7 is used to store the result from the multiplier which is used as the parallel input to this multiplier. On the other hand, the second class of RSA structures uses the general skeleton depicted in Figure 8. This class of structures requires only one Montgomery multiplier. However, multiplexers are required to select the right data to be fed to the multiplier, either for the modular multiplication operation or for the modular squaring operation [22].

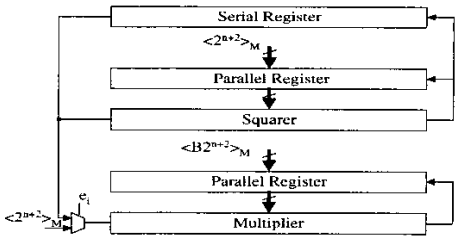


Figure 7. An RSA Structure that uses two Montgomery Multipliers

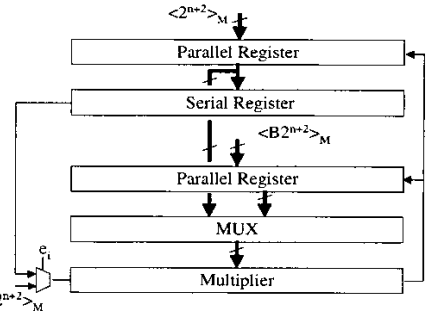


Figure 8. An RSA Structure that uses one Montgomery Multiplier

The Montgomery's multiplier architectures used with these two classes of RSA structures are: a conventional Montgomery multiplier based on *Algorithm 1*, the Montgomery multiplier of Figure 1, a Montgomery multiplier derived from Figure 1 structure by applying the retiming technique as described in Figure 9, a Montgomery multiplier of Figure 2, a 2-bit digit Montgomery multiplier derived from Figure 1 structure via unfolding (see Figure 10), another 2-bit digit Montgomery multiplier derived from Figure 1 structure via unfolding but with its feedback loops pipelined so that 2 operations are interleaved into this structure, a 2-bit digit Montgomery multiplier that interleaves two operations and that has been retimed to avoid global line broadcast (see Figure 11.b), and finally, a 4-bit digit Montgomery multiplier derived form Figure 1 structure via unfolding (with an unfolding factor of 4) and finally another 4-bit digit Montgomery multiplier in which two operations may be interleaved.

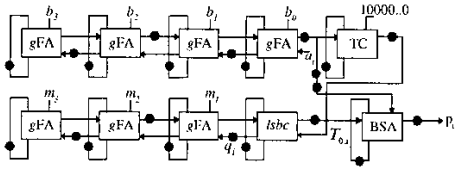


Figure 9. A modified version of the Montgomery Multiplier of Figure 1 after using the retiming technique

The retimed Montgomery multiplier of Figure 1 is depicted in Figure 9. In this structure, global line broadcast are avoided by using $n/2$ latches/FFs on the serial input line for a n-bit multiplier. The retiming cutest lines are vertical to the serial input of Figure 1 and as stated by the retiming algorithm [23], for each FF added to the link of a processor A to a processor B, a FF is removed from the link of the processor B to the processor A. Therefore, the longest path on the multiplier is equivalent to 2 gFAs and a latch/FF. applying the unfolding technique with an unfolding factor of 2 on the structure of Figure 1 leads to the Montgomery multiplier depicted in Figure 10. This multiplier is capable of

processing 2 bits of the serial input per clock cycle. The clock path of this structure is that of two gFAs, a FF.

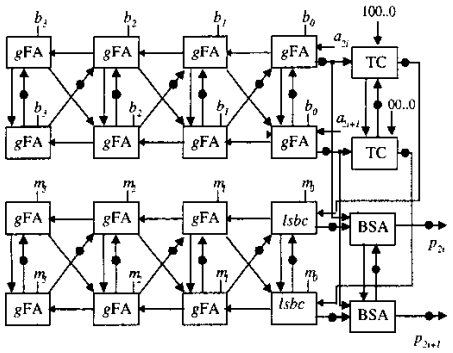
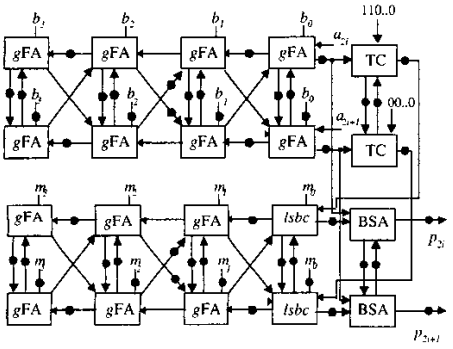
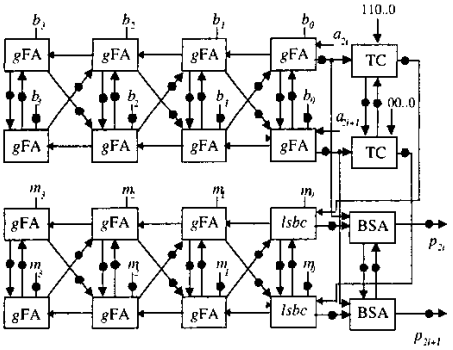


Figure 10. A 2-bit Digit Montgomery Multiplier that Implements Algorithm 3

It is clear from Figure 10 that the global input line will have a major drawback on the clock frequency of this structure. Although the logic path is reduced in the multiplier of Figure 10 to 1 gFA and 1 FF by interleaving two operands into this structure, the drawback of the global lines is not resolved. A solution is shown in Figure 11. By retiming the structure of Figure 11.a, the global lines are once again avoided but at the expense of a longer logic path that consists of 2 gFAs and a latch/FF as shown in Figure 11.b. By avoiding global line broadcast, the placement and routing tools used to implement the RSA algorithm in FPGA one dedicate more efforts into routing the remaining nets of the structures.



(a)



(b)

Figure 11 a. A 2-nit Digit Montgomery Multiplier that Interleaves two operations

b. The same Structure after retiming

Among these Montgomery multipliers, those who interleave two modular multiplication operations are used with the RSA structure of Figure 8. The remaining multipliers, which carry out only one multiplication operation, are used with the RSA structure of Figure 7. As it will be explained further down in this section, all these architecture may be divided into two groups based on avoiding or not avoiding the global line broadcast in the Montgomery multiplier structures.

For the purpose of comparison and analysis, the following proposed structures have been implemented in a Xilinx XC2V8000 device:

Struct 1: RSA architecture of Figure 7 with a conventional Montgomery multiplier as described in *Algorithm* 2 [3].

Struct 2: RSA structure of Figure 7 which uses the Montgomery multiplier depicted in Figure 1.

Struct 3: RSA structure of Figure 8 which uses the Montgomery multiplier of Figure 2.

Struct 4: RSA architecture of Figure 7 with the multiplier depicted in Figure 9.

Struct 5: RSA architecture of Figure 7 which uses the 2-bit digit Montgomery multiplier of Figure 10.

Struct 6: RSA structure of Figure 8 with the multiplier of Figure 11.b that interleaves 2 multiplication operation onto the same multiplier.

Struct 7: the RSA structure of Figure 7 which uses the 4-bit digit Montgomery multiplier obtained by unfolding the multiplier of Figure 1 (with an unfolding factor of 4).

Struct 8: the RSA structure of Figure 8 that uses a 4-bit digit Montgomery multiplier in which two multiplication operations are interleaved.

The implementation results of these 8 structures were obtained using the Xilinx ISE 6.2 package. However, theses architectures were not verified with an actual implementation on a chip. The obtained results are compared against similar work in the literature [11, 20]. In [20], an algorithm combining the Montgomery's technique and the carry save representation of numbers was proposed. A highly modular bit-slice based architecture has been designed for executing the algorithm in FPGA. The serial RSA architecture in [20] used carry save adders to avoid carry propagation during the addition stages of Montgomery algorithm. The architectures in [11] are based on the use carry propagate adders, implemented using fast carry logic resources available in the FPGA chip, with the multiples of A and M' are pre-computed and stored in a RAM. A radix-2 and a radix-16 architecture were proposed in [11].

As mentioned previously, the concept of digit-serial arithmetic was proposed as a compromise between the bit serial and the bit Parallel arithmetic. By an appropriate selection of the digit size, it provides the designer with the best area and speed that match the needs of the system. Therefore, the aim of the analysis presented in this section is to provide the designer with the necessary knowledge and information for finding the best compromise between the cost of the RSA architecture and its performance. The analysis is based on the effect of changing the digit size and the level of pipelining for a full 1024-bit modular exponentiation (i.e. both the key and the public exponent are 1024 bits long). The evaluation parameters are the frequency, the required time to carry out the modular exponentiation operation, and the area usage in terms of the number of FPGA slices.

The proposed structures carry out a modular exponentiation operation in $(2n+4)k/dl$ clock cycles, where n, k, l, and d are the number of bits in the modulus, the number of bits in the

exponent, the level of pipelining, and the digit size, respectively. Therefore, for a full n-bit modular exponentiation operation, $(2n+4)n/dl$ clock cycles are required.

The implementation results of the proposed architectures are shown in Table 1 (the implementation which does not include the pre-mapping and post-mapping operation that convert data to and from the residue system).

A clear distinction can be made between two classes of structures: in one side *Struct 3*, *Struct 4*, and *Struct 6*, and on the other side, the remaining structure. The distinction is made upon the working frequencies of the first class structures, which is much higher than the working frequencies of the architectures of the latter class. This underlines the benefits of using nearest neighbour communications only against the global lines distribution that characterises the architectures of the second class. The effect that pipelining has on the critical path of the different structures can clearly be seen if the frequencies of *Struct 1*, *Struct 5*, and *Struct 7* are compared those of *Struct 2*, *Struct 6*, and *Struct 8*, respectively. The critical logic path of *Struct 1* is 2 FAs and operates at a frequency of 145MHz while the critical logic path of *Struct 2* is one FA, and thus operates at a higher frequency of 151 MHz. The same observation can be made for *struct 5* and *struct 7*.

- *Struct 5* whose critical logic path is 2 FAs and operates at a frequency 112 MHz and *Struct 6* which exhibits a critical path of one FA and has a clock frequency of 255 MHz.
- *Struct 7* that has a critical path of 4 FAs and operates at a frequency of 78 MHz and *Struct 8* whose critical path consists of 2 FAs with an operating frequency of 84 MHz.

In these three previous examples, the improvement in the clock frequency is very clear in the case of *Struct 5* and *Struct 6*. This is due mainly to the fact that *Struct 6* uses nearest neighbour communications only. The small improvement in the remaining two cases can be explained by the effect of inherent routing delay, which is more important than the delay of the logic path.

Another point that is worth to be mentioned is the effect of using the architectures of Figure 7 and Figure 8. For example, *Struct 3* operates at a frequency of 291 while *Struct 4* operates at a frequency of 278 MHz. However, *Struct 4* carries out a 1024-bit modular exponentiation in 7.55 ms, thus faster than *Struct 3*, which requires 14.46 ms to perform the same operation. This is explained by the fact that *Struct 3* interleaves two modular multiplication operations into the same Montgomery multiplier. The same can be said about *Struct 7* and *Struct 8*.

An important decision that can be made from reviewing Table 1 is the choice of the appropriate architecture for the application at hand. For instance, *Struct 4* requires 7.55 ms to carry out the full modular exponentiation operation while *Struct 7* (which uses 4-bit digit Montgomery multipliers) carries out the same operation in 6.78 ms. Nevertheless, the area usage of *Struct 7* is almost twice that of *Struct 4*. By changing the digit size and the level of pipelining, one may have a database of architectures from which he/she can choose the best architecture that matches the design frequency and area usage needs.

	Frequency (MHz)	Period (ns)	Time (ms)	Area (Slices)
Struct 1	145	6.905	14.51	6500
Struct 2	151	6.627	13.92	8500
Struct 3	291	3.440	14.46	12400
Struct 4	278	3.592	7.55	10100
Struct 5	112	8.920	9.37	12200
Struct 6	255	3.928	8.25	14200
Struct 7	78	12.899	6.78	19900
Struct 8	84	11.845	12.44	20300

Table 1. Performances of the proposed architectures

Table 2 shows a comparison between *Struct 4* which is a serial architecture against two serial architectures described in [11] and [20]. The table also compares *Struct 7*, which is a 4-bit digit architecture against a radix-16 architecture proposed in [11]. This comparison is carried out for illustration purposes only as the work in [11] was implemented in an XC40150XV-8 FPGA while the work in [20] was implemented in a Virtex 2000E-8 chip. Probably, using a recent FPGA chip may lead to better performances. Clearly, the architectures presented in this paper are superiors to those in the literature in terms of the processing time. Struct 4 has an 82% and 73% improvement over the work in [11] and [20], respectively. Struct 7 has a 43% improvement over the radix-16 architecture in [11].

	Frequency (MHz)	Time (ms)
Struct 4	278	7.55
Struct 7	78	6.78
Radix-2 in [11]	52	40.05
Radix-4 in [11]	46	11.95
RSA in [20]	78	27.88

Table 2. A comparison with similar work in the literature

V. Conclusion

The modular exponentiation operation is the core of the RSA algorithm that has become the most widely used public key computer security algorithm. New architectures for the implementation of Montgomery modular exponentiation for RSA have been proposed. The new architectures use a modified Montgomery algorithm in which the operations of modular multiplication and modular reduction are carried out separately but in a parallel way. To investigate the best area usage-time trade-off, the digit approach was adopted. The problem of having global lines in the architectures have been circumvented by interleaving more than one instance into the same digit multiplier, which was achieved by pipelining the feedback loops and retiming the whole structures. The implementation results have shown that by pipelining and increasing the digit size of the structures, the global line broadcast is avoided with more bits processed at each clock cycle, which in returns has led to better performances in terms of the processing time. However, this was achieved at the price of extra area usage. This provides designers with the best trade off between speed and area usage and throughput rate.

VI. Reference

[1] O.Nibouche, and M.Nibouche. *"On Designing Digit Multipliers"*, Proceeding of the 9th International IEEE

Conference on Electronics, Circuits, and Systems (ICECS), Dubrovnik 2002.
[2] O. Nibouche. A. Bouridane and M. Nibouche, *"New Iterative Algorithms and Architectures of Modular Multiplication for Cryptography"*, Proceedings of the 8th International IEEE Conference on Electronics, Circuits, and Systems, ICECS Malta 2001.
[3] O. Nibouche, A. Bouridane, and M. Nibouche "Architectures for Montgomery's multiplication", IEE Proceedings, Computers and Digital Techniques, Vol.150, November 2003, Issue 06, p. 361-368.
[4] W. L. Freking and K. K. Parhi. Performance-scalable array architectures for modular multiplication. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pages 149–160. IEEE, 2000.
[5] W. L. Freking and K. K. Parhi, "Ring-Planarized Cylindrical Arrays with Application to Modular Multiplication" IEEE Workshop on Signal Processing Systems Design & Implementation (SIPS 2000), Lafayette, Louisiana, USA, pp 497-506.
[6] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.
[7] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
[8] M. Shand and J. Vuillemin, 'Fast implementation of RSA cryptography', Proceedings of the 11th IEEE Symposium on Computer Arithmetic, 1993.
[9] P. L. Montgomery, "Modular multiplication without trial division," Math. Comp., vol. 44, no. 170, pp. 519-521, 1985.
[10] T. Blum and C. Paar. Montgomery modular exponentiation on reconfigurable hardware. In *Proceedings of 14th IEEE Symposium on Computer Arithmetic*, pages 70–77, Adelaide, Australia, April 14-16 1999.
[11] T. Blum and C. Paar. High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Transactionson Computers*, 50(7):759–764, July 2001.
[12] A. F. Tenca and C¸. K. Koc¸. A scalable architecture for Montgomery multiplication. In C¸. K. Koc¸ and C. Paar, ed-itors, *Proceedings of Cryptographic Hardware and Embed-ded Systems (CHES 1999)*, number 1717 in Lecture Notes in Computer Science, pages 94–108. Springer-Verlag, 1999.
[13] H. Orup. Simplifying quotient determination in high-radix modular multiplication. In *Proceedings of the 12th Sympo-sium on Computer Arithmetic*, pages 193–199. IEEE, 1995.
[14] S. E. Eldridge and C. D. Walter. Hardware implementation of Montgomery's modular multiplication algorithm. *IEEE Transactions on Computers*, 42:693–699, 93.
[15] P. Kornerup. A systolic, linear-array multiplier for a class of right-shift algorithms. *IEEE Transactions on Computers*, 43(8):892–898. August 1994.
[16] J. Goodman, and A. P. Chandrakasan, "an Energy-Efficient Reconfigurable Public Key Cryptography Coprocessor", IEEE journal of solid state circuits, vol. 36, pp. 1808-1320, No. 11, November 2001.
[17] Anderson, R., and Kuhn, M., "Tamper Resisatnce- a Cautionary Note", in The Second USENIX Workshop on Electronic Commerce Proceedings, Oakland, California, November 18-21, 1996, pp 1-11
[18] C. McIvor, M. McLoone, J. McCanny, A. Daly and W. Marnane, "Fast Montgomery Modular Multiplication and RSA Cryptographic Processor Architectures, *"37th Asilomar Conference on Signals, Systems, and Computers,* Nov 2003
[19] A. Mazzeo, L. Romano, G. P. Saggese, and N. Mazzocca, "FPGA Based Implementation of a Serial RSA Processor", proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03), March 03 - 07, 2003 , Munich, Germany pp10582-10589.
[20] A. Cilardo, A. Mazzeo, L. Romano, G. P. Saggese, Carry-Save Montgomery Modular Exponentiation on Reconfigurable Hardware, Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Designers' Forum (DATE'04).
[21] S. B. Ors, L. Batina, B. Preneel, J. Vandewalle, "Hardware Implementation of a Montgomery Modular Multiplier in a Systolic Array", Proceedings of 10th Reconfigurable Architectures Workshop (RAW 2003), Nice, France, April 2003.
[22] J. Sheu, M. Shieh, C. Wu and M. Sheu, "A Pipelined architecture of fast modular multiplication for RSA cryptography", in Proc IEEE ISCAS'98, vol.2 pp.117-180, 1998.
[23] C. E. Leiserson and J. B. Saxe, "Retiming ynchronous circuitry,", *Algorithmica*, vol. 6, pp. 5–35, 1991.