PARAMETERIZED RSA ARCHITECTURES

OMAR NIBOUCHE
College of Computers and Information Technology
Taif University, AlHawiyah Campus
POB 888 Taif 21974, KSA
o.nibouche@tu.edu.sa

Abstract—**In this paper, new structures that implement the RSA cryptographic algorithm are presented. The core of these architectures is the modular exponential operation based on a modified Montgomery modular multiplier, where the operations of multiplication and modular reduction are carried out in parallel rather than in an interleaved way as in the traditional Montgomery multiplier. The digit approach has been adopted to implement the modified Montgomery multipliers, where by pipelining the feedback loops; one or two modular multiplication operations can be interleaved to use the same multiplier structure. Thus, RSA structures that use a single Montgomery multiplier, termed area-efficient architectures, and architectures that require two Montgomery multipliers, called speed-efficient architectures, are presented. The proposed architectures are scalable and parameterized. Furthermore, by varying the digit size and the level of pipelining, the designer is provided with an efficient way of choosing the architecture that suits better his/her requirements in terms of speed and area usage. The critical propagation path is shortened and data global broadcast can be avoided. The proposed architectures are well suited for description using a VHDL structural style where the user can chose the design parameters such as the word length, modulus, data size, digit size and pipelining level to describe the design. To facilitate the process of generating the proposed architectures for an FPGA-based implementation, an RSA encryption engine that produces a VHDL code has been developed where the design specifications can be entered using the engine's GUI. The results of implementation using FPGA have shown that the proposed structures outperform similar work in the literature. For a 1024-bit exponentiation operation, their speed and area usage performances range from 5.67 ms and 37000 slices, to 16 ms and 8500 slices.**

Keywords-Montgomery multiplication; RSA; FPGA; parameterized architectures.

I. INTRODUCTION

Since the seminal work of Diffie and Hellman was published in 1976, numerous public-key cryptosystems have been devised to provide confidentiality, authentication, data integrity and non-repudiation [5,6,13,14]. All these crypto-systems base their security on some mathematical one-way functions. They are combined with private key encryption primitives and used with the appropriate protocols to construct secure and trusted networks. RSA is the most widely used public-key cryptosystem. An RSA encryption operation is a modular exponentiation operation, which requires repeated modular multiplications. For security reasons, RSA operand sizes need to be 1024-bit or longer [9,10,13-19]. Thus, such an algorithm requires immense processing power that may cause a bottleneck in high-speed networks. A remedy for this drawback can consist of implementing the RSA algorithm in hardware. The developed circuit, either an ASIC or a PLD, can be used as a co-processor coupled with a host machine in order to speed up the computation of the encryption operations. 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 non-volatile memory, which are externally inaccessible [14].

The basis of many RSA hardware architectures reported to date is formed by Montgomery modular multipliers [2-4,7-10,15-19]. A Montgomery multiplier replaces trial division by the modulus with a series of additions and divisions by a power of 2, which is very easy to implement. Many RSA architectures based on Montgomery modular multiplication have been proposed in the literature. The approach adopted in Refs 9-10, is to take advantage of the resources available in FPGA chips such as the fast carry chains to perform the addition operations. The main drawback is that the resulting RSA architecture is specific to the family of FPGA devices they have been devised for. Another class of RSA architectures uses a modified Montgomery multiplier based on a Carry Save Addition (CSA) scheme. By using a carry save representation of numbers, the authors in Ref 18 have been able to shorten the propagation path to a single Full Adder (FA) and a 7-to1 multiplexer with pre-computation of multiples of the operands. Such pre-computation of multiples of the operands is avoided in Ref 16, where recourse to 5-to-2 and 4-to2 CSAs is made. However, the critical path is equal to 3 FAs, 2 XORs and a single AND gate.

A modified version of Montgomery multiplication was proposed in Ref 2. The structure was devised in such a way the modular multiplication operation is divided into two conventional multiplication operations: a multiplication operation and a reduction operation, where the result of the first multiplication operation is used to select a reduction value for the second part. The two's complement of the first multiplier result is added to the result of the second multiplier to produce the result of the Montgomery multiplier. A clear advantage of this architecture is that the propagation path is shortened and the well-known design methodologies applied to conventional multipliers (i.e. unfolding, retiming etc) can be used with the modified structure [1,21,22].

In this paper, new RSA architectures that employ this multiplier are presented. The proposed generic class of RSA architectures that use 2 Montgomery multipliers is termed here Speed-Efficient (SE) architectures. Furthermore, by pipelining the feedback loops of the modified multiplier, two operations can be interleaved to use the same structure. This is exploited in this paper to produce Area-Efficient (AE) RSA architectures with a single Montgomery multiplier. The digit approach has been adopted in the papers, where by increasing the digit size, different performances in terms of speed and area-usage can be obtained; thus providing the designed with a wide range of choices to meet the design requirements. The proposed architectures are scalable and parameterized. Furthermore, by interleaving 2 instances to use the same structure, the proposed architectures have their critical path reduced to up to a single FA and the global lines that broadcast data avoided, thus leading to systolic architectures that use nearest neighbor communications only.

The remainder of the paper is organized as follows: section 2 revisits the modified version of Montgomery modular multiplication algorithm and RSA modular exponentiation. A brief discussion about the bound of the result of the algorithm is presented. As a matter of fact, the bound of the result of Montgomery algorithm is changed in such a way that, if multiple modular multiplication operations were to be carried out iteratively, with the result of one iteration being used in the next one, no subtraction operation would be required. Section 3 presents the RSA architectures and the VHDL engine to describe them. In Section 4, the implementation results of the proposed architectures on FPGA are analyzed and compared to similar work in the literature. Conclusions are drawn in Section 5.

II. Mathematical Background

As it was suggested [6], the modulus M of the RSA algorithm is the product of two suitably generated secret prime numbers P and Q:

$$M = P \times Q \tag{1}$$

The public exponent (also known as the encryption key) E is randomly chosen such that it is prime to $(P\text{-}1)\times(Q\text{-}1)$. The secret exponent (also known as the decryption key) D is computed using the extended Euclidean algorithm such that:

$$\langle E \times D\rangle_{(P-1)(Q-1)} = \langle 1\rangle_{(P-1)(Q-1)} \tag{2}$$

A message is divided into blocks of the same word-length as the product M in equation (1). If M is encoded on k bits, an $n = km$ bits message is divided into m blocks of k-bit each. Thus, a k-bit word A is encrypted into a word $Enc\ (A)$ defined by $Enc(A) = \langle A^E\rangle_M$. A cipher word B is decrypted into a word $Dec\ (B)$ defined by $Dec(B) = \langle B^D\rangle_M$.

The operation of modular exponentiations is carried out iteratively by repeating a modular squaring operation and a modular multiplication, as described in *Algorithm 1*. The algorithm computes B, which is the remainder of the division of A^E by the modulus M.

<u>*Algorithm 1*</u>

```
$B = \langle A^E\rangle_M$
$E = \sum_{i=0}^{k-1} e_i 2^i$, $P_0 = \langle 2^{n+2}\rangle_M$, $B_0 = \langle 2^{n+2}A\rangle_M$
for i = 0 to k-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 $P_{i+1} = P_i$}
```

Algorithm 1 entails a modular squaring operation, $\langle B_i^2\rangle_M$, and a modular multiplication operation, $\langle P_i B_i\rangle_M$. Both modular multiplication and squaring operation can be implemented using the Least Significant Bit First (LSBF) modular multiplication scheme termed Montgomery algorithm. The term $\langle 2^{n+2}\rangle_M$ in P_0 and B_0 is due to the multiplication factor introduced by the Montgomery multiplication scheme. If a most significant bit first multiplication algorithm is used, the factor $\langle 2^{n+2}\rangle_M$ is simply dropped and replaced by *1*. The modulus M in Montgomery algorithm can be an integer within the range $[2^{n-1}, 2^n[$ [8]. The algorithm requires a quantity termed Radix (R), which is equal to 2^n, and M to be relatively prime, which is satisfied as M is odd. It computes the modular product, P, of two given integers, A and B, as follows [8]:

$$P = \langle ABR^{-1}\rangle_M \tag{3}$$

The algorithm uses multiplication modulo R and division by R, which are only shift operations, thus, faster and simpler to implement in software and hardware than the computation of $\langle AB\rangle_M$ which involves division by M. However, the algorithm is only efficient when multiple operations are carried out, such as in the modular exponentiation operation when such an operation is broken into modular multiplication operations [7,11,12,15-20]. For hardware implementation, a systolic array can be derived from the bit-wise version of Montgomery multiplication

<u>*Algorithm 2*</u>

```
$0 \le A = \sum_{i=0}^{n} a_i 2^i < M$, $0 < B < R$, $P_{-1} = 0$
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 $P_{n-1} \ge M$ then $P_{n-1} = P_{n-1} - M$
```

Algorithm 2 interleaves the multiplication steps, $P_{i-1} + a_i B$, with the reduction steps, $P_{i-1} + a_i B + q_i M/2$. One bit of the partial result is used to select the reduction value. As shown in the algorithm, the Least Significant Bit (LSB) of the partial result of the previous iteration, P_{i-1}, together with the bit-product a_ib_0, directly selects the modular reduction value, which is either *0* or *M*. At every iteration, the LSB of P_i equals 0. P_i is then shifted one position to the right. After *n* iterations, the scaling factor is equal to 2^{-n}. Therefore, the final result is $\langle ABR^{-1}\rangle_M$ as shown in equation (3). 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 [8]. The critical propagation delay of *Algorithm 2* occurs during the calculation of the P_i values where the main contributing factor to this delay is the carry propagation [15-19] and the global broadcast line resulting from using very large operands [2-4,15-19].

One can easily argue that *Algorithm 2* consists of two interleaved multiplication operations and an addition operation. This fact has been exploited to build a modified Montgomery structure that uses two conventional multipliers to carry out the multiplications in *Algorithm 2* in a non-interleaved way [2]. Thus, 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 as described in *Algorithm 3* [2].
Let *T* be the product of $A\times B$, i.e.:

$$T = AB = T_0 + 2T_1R \qquad (4)$$

<u>*Algorithm 3:*</u>

```
$0 \le A = \sum_{i=0}^{n+1} a_i 2^i < 2M,\ 0 \le B = \sum_{i=0}^{n+1} b_i 2^i < 2M,\ a_{n+1} = b_{n+1} = 0$
$P'_{-1} = 0,\ c_0 = 1,\ P''_{-1} = 0$
For i= 0 to n+1
{Step1: $P'_i = P'_{i-1} + 2a_iB$
Step 2: $P'_{i,i} + c_i = \bar{T}_{0,i} + 2c_{i+1}$
Step 3: $q_i = P''_{i-1,i} \oplus T_{0,i}$
Step 4: $P''_i = P''_{i-1} + q_iM$
Step 5: $P_i = (P'_i + P''_i)/2$}
```

As shown in *Algorithm 3*, the modular multiplication operation is broken into two concurrent multiplication operations and computes Montgomery product by using the two's complement of T_0 (which is $\bar{T}_0$ in step 2 of *Algorithm 3*) to select the reduction value. The results from the two multipliers are then added and a division by *4R* (or 2^{n+2}) follows. However, in the original algorithm (see *Algorithm 2*), after each multiplication a reduction was needed (the last step in *Algorithm 2*). The inputs have the restriction *A, B < M* and the output *P* is bounded by *P < 2M*. If used to compute repeated multiplication operations as in RSA, *M* must be subtracted from *P* so that the output of a modular multiplication operation can be used as input for the next operation. To avoid this subtraction operation, the 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:

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

Where M' is the multiplicative inverse of *–M* modulo *R*. Therefore, at the *(n+2)th* (i.e., the last iteration), and since $a_{n+1} = b_{n+1} = 0$, the upper bound of the result $P = (P'_{n+1} + P''_{n+1})/2$ is given by:

$$P < 2M - 3M \times 2^{-n-2} < 2M \qquad (6)$$

Thus, it is practicable that the result of each modular multiplication operation can immediately be used as input for the next operation, as required by RSA algorithm. 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 3*. Finally, for the range of the RSA operation result to fall in *[0, M[*, a single comparison/ subtraction operation may be necessary.

III. Implementation Approach

The serial approach processes 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. 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. To avoid this problem, and thus, reach higher system speeds, it is clear that a move towards higher bit-width operations is necessary. Unfortunately, the parallel architectures require a considerable area-usage and pin-out, which may not be satisfied in the resource limited FPGA chips and the size of data bocks used in RSA.

As a trade-off, the concept of digit-serial computation has been adopted in this paper. The proposed system processes more than one bit in one clock cycle. The number of bits processed in one clock cycle 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 [1-4]. However, Montgomery digit multipliers cannot be pipelined beyond the digit level due to the presence of feedback loops. Nevertheless, based on the use of the feedback loop pipelining technique [2-4], one can shorten the propagation path by interleaving two or more data sets into the same structure [2-4]. In the proposed designs, only one or two instances are interleaved into the same architecture.

A. RSA Structures

Two classes of RSA structures that implement *Algorithm 1* are presented. The first generic 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 such a class of structures, termed SE architectures, is depicted in Figure 1. Registers are used to store the result form the modular squarer in both parallel and serial way. The parallel register associated with the 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. Both registers are initialized with $B_0 = \langle 2^{n+2}A\rangle_M$ as in *Algorithm 1* to start the computation of the values B_{i+1} from B_i. The parallel register associated with the multiplier in Figure 1 is initialized with $P_0 = \langle 2^{n+2}\rangle_M$. Using the input from its parallel register and the serial input from the serial register in which the intermediate results B_i are stored, the multiplier computes the multiplication step in *Algorithm 1*, $P_{i+1} = \langle P_i B_i\rangle_M$. The result is fed back to the multiplier's register. The multiplication operation is controlled by the exponent bit e_i. If it is low, rather than freezing the multiplier and its parallel register so that P_i is stored for the next exponentiation step, a multiplication of P_i by *1* takes place. In Montgomery domain, this is equivalent to a multiplication by $\langle 2^{n+2}\rangle_M$ in order to take account of the scaling factor effect introduced by Montgomery multiplier.

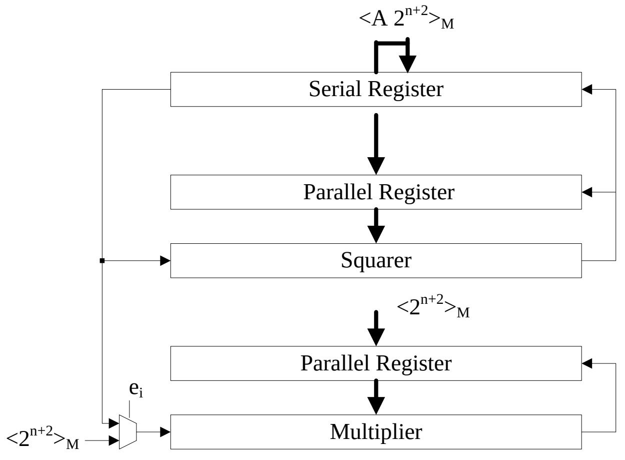

Figure 1. A Speed-Efficient (SE) RSA Structure that uses two multipliers

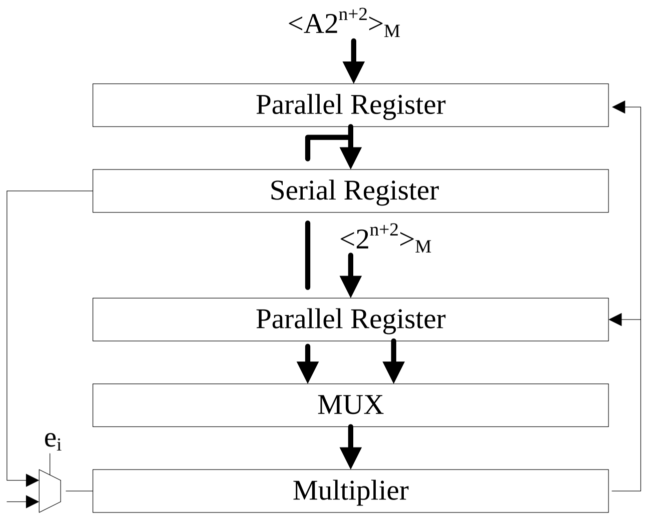

Figure 2. An Area Efficient (AE) RSA Structure that uses one Montgomery Multiplier

The second class of RSA structures, termed AE architectures, employs the general skeleton depicted in Figure 2. This class of structures requires only one Montgomery multiplier which carries out both the multiplication and squaring operations in RSA in an interleaved way. It is worth pointing out that one exponentiation iteration of *Algorithm 1* takes to execute in an AE architecture twice the execution time in its SE counterpart architecture. As a matter of fact, the multiplier in Figure 2 devotes one clock cycle for the modular squaring step in *Algorithm 1* and the next clock cycle for the modular multiplication step every two clock cycles. Since the squaring and multiplication operations are carried out in a serial fashion, the shift process in the serial register in Figure 2 is twice slower than the clock frequency. 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. If a digit RSA is implemented using either of the two general skeletons in Figures 1 and 2, the serial register is replaced by a digit shift register. The parallel registers remain unchanged and the multiplication time is reduced by a factor equal to the digit size *d*.

B. Montgomery Structures

Algorithm 3 can be implemented using two bit serial multipliers, a serial adder 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 3. The First Cell (FC) of the second serial multiplier is different from the remaining Basic Cells (BC), which are gated FAs (a gated FA is a FA to which an AND gate is connected, the AND gate generates the partial products). During the *n* first cycles, the FC generates the bit q_i that selects the reduction value. The clock path of this bit serial modular multiplier is equivalent to that of two FAs and a latch. The issue of data lines broadcast to all the cells is resolved (such as the multiplier input a_i in Figure 3) leading to a fully systolic structure. The insertion of registers in global lines achieves a significant increase in clock frequency, especially for large operands, which are usually used in cryptography. Although the critical propagation path is equal to 2 FAs, such solution is preferred to shortening the critical path to a single FA and removing the registers from the global lines, which can be obtained by retiming the structure of Figure 3 [1-21]. However, a different registers insertion strategy may be used to, in the same time, shorten the critical path to a single FA and circumvent the global broadcast of data. In this approach, the same cells used in the multiplier of Figure 3 are used to build a systolic multiplier structure, except that the overall number of latches increases. The resulting architecture will interleave two modular multiplication operations using the feedback loop pipelining technique [4]. In general, the number of operations interleaved onto the same structure depends on the number of latches added to the feedback loops. As such, an extra register has to be added to the feedback loops and sum- bits of the multiplier in Figure 3 to cope with feeding two different sets of operands to the multiplier. The two interleaved modular multiplication operations can be the two operations involved in the modular exponentiation. In such a way, the resulting serial multiplier can be used as the core bit serial-parallel modular exponentiation element in the architecture of Figure 2. The critical path will be reduced to a single FA and the resulting structure will be fully systolic; requiring nearest neighbor communications only.

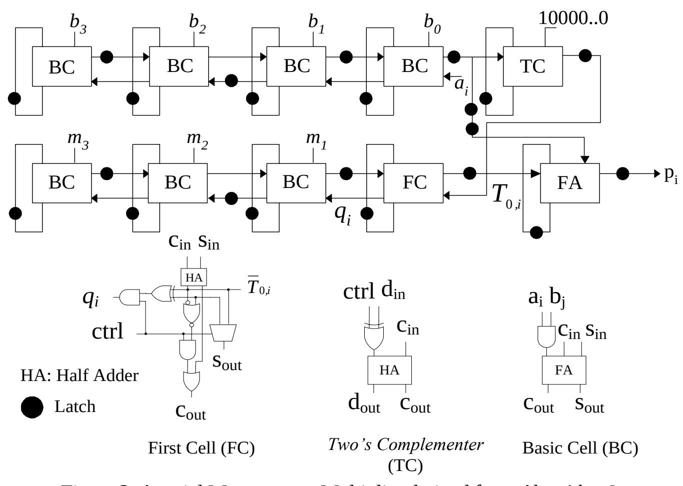

Figure 3. A serial Montgomery Multiplier derived from *Algorithm 3*

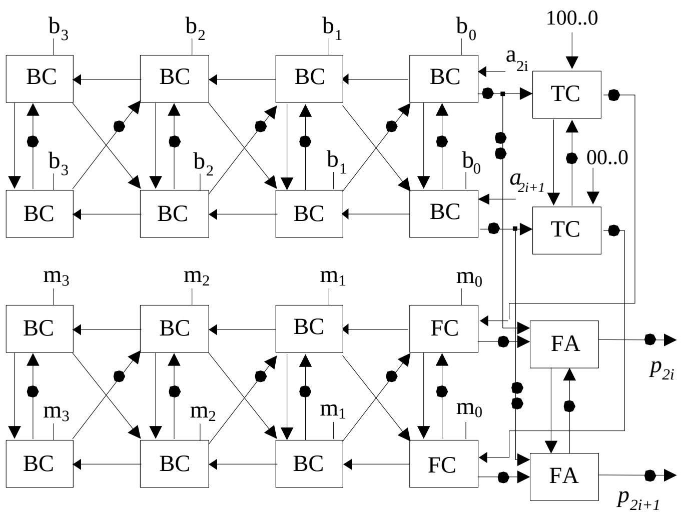

Figure 4. A 2-bit Digit Montgomery Multiplier that Implements *Algorithm 3*

A 2-bit digit Montgomery multiplier structure is depicted in Figure 4. Such architecture processes two bits of the operand per clock cycle, thus requiring half the number of cycles of that of the architecture of Figure 3 to carry out a modular multiplication operation and its critical path is two FAs. It suffers, however, from global data broadcast. Nevertheless, a 2-bit digit multiplier capable of interleaving two data instances can be derived by pipelining the feedback loops in Figure 4 and then retiming the structure. The resulting architecture is fully systolic and exhibits a critical path equal to 2 FAs, thus possessing similar characteristics to that of the architecture of Figure 3. Higher digit sizes multipliers can be obtained by unfolding the structure of Figure 4. To interleave two sets of data into such digit architecture, a FF is added to its feedback loops followed by retiming. An example of 4-bit digit multiplier is depicted in Figures 5 and 6. In the remainder of the paper, the notation *Dd_Ll* is used to indicate a digit size of *d* bits and a level of pipelining *l*.

C. Code Generation Engine

It is clear that the suggested RSA architectures can be described using parameters such as the modulus, which defines the block size, and the public/private key exponent. Such parameters are sufficient to describe the registers, multiplexers and control signals in the developed RSA structures. In addition, Montgomery multipliers are parameterized in terms of the block length, which determines the size of the multiplier, the digit size and level of pipelining. Digit and pipelined multipliers can be derived from the serial structures using the unfolding transformation [1,4,22]. Furthermore, the multipliers in Figures 3 and 4, their counter parts capable of interleaving two operations and the digit multipliers depicted in Figures 5 and 6, are scalable; that is they can be built by replicating a certain number of basic cells. Thus, an automatic code generator can be built to take account of all of the designs details and parameters. The code is generated in a structural VHDL style which is very suitable for a parameterized description of circuits. The user's constraints can be expressed in form of area usage and speed requirements, from which an RSA structure with the adequate digit size and level of pipelining is generated. In addition, the user has the option of picking a predetermined RSA structure which has the design parameters described above.

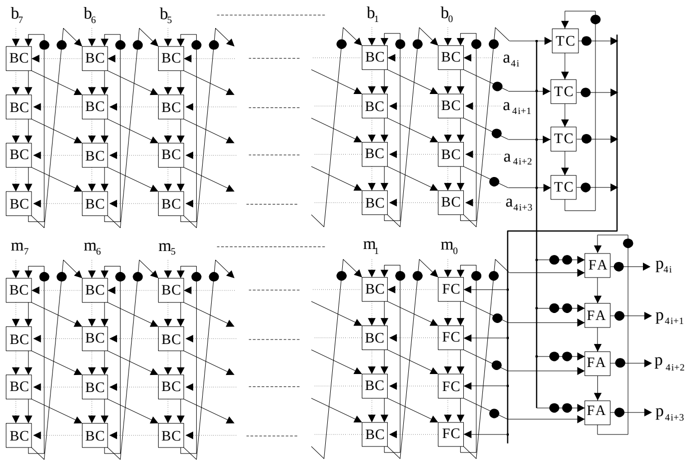

Figure 5. A D4_L1 Digit Multiplier

The proposed system is shown in Figure 7. It consists of a GUI where the user can enter his/her specifications in terms of speed and area usage. The user is also able to pick a predetermined structure which is available in the system library. The library consists of the basic design components such as the registers, multipliers and multiplexers of the two skeletons of RSA architectures. From this basic components and using the design specifications such as the digit size, word length, modulus and level of pipelining, new architectures can be devised. The VHDL code generator can interpret the area usage and processing time specifications to generate a digit architecture that best suits the user by applying transformation techniques such as unfolding, pipelining and retiming on the basic components in the system library. Furthermore, it can also rely on a heavy pipelining of the input/output pins data-lines to further improve the clock frequency. Once the VHDL code has been generated, the Xilinx tools are executed to synthesize, simulate, verify and implement the obtained design in the target FPGA chip.

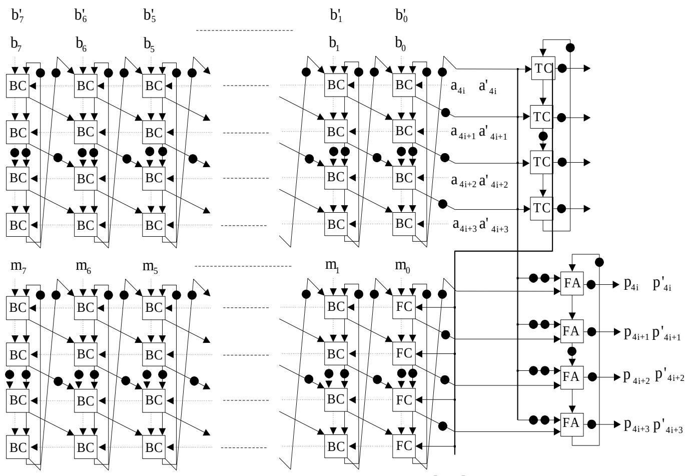

Figure 6. A D4_L2 Digit Multiplier

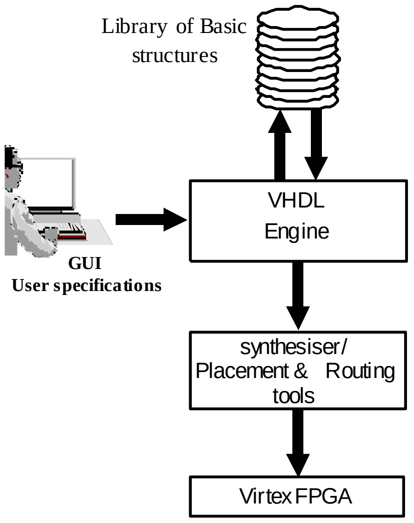
Library of Basic structures
GUI
User specifications
VHDL Engine
synthesiser/ Placement & Routing tools
Virtex FPGA

Figure 7. RSA code generator

IV. IMPLEMENTATION AND RESULTS

The proposed two classes of RSA architectures are compared in terms of their area usage and timing characteristics. To illustrate the fact that the proposed architectures can be parameterized, digit sizes of 1-bit, 2-bit, 4-bit and 8-bit have been selected to generate RSA architectures with either a single Montgomery multiplier or two Montgomery multipliers using the engine of Figure 6 for a modulus of 1024-bit length. The larger digit size of 16-bit does not fit in the target device family. For the purpose of comparison with similar work in the literature, the proposed structures have been implemented in a Xilinx Virtex-4 device.

Table 1 depicts the area usage of the selected 8 structures. As one may have expected, the area usage increases when the digit size increases within the same class of RSA architectures. Thus, D1_L1 and D8_L1 structures use the smallest area and largest area within the Dx_L1 class of architectures, respectively. A similar observation is applicable to the class of Dx_L2 structures. An interesting point which comes out from Table 1 is that up to a digit size of 2 bits, Dx_L1 structures require less area than their Dx_L2 counterparts despite calling for an extra Montgomery multiplier. However, when the Dx_L2 architectures use a single multiplier, they also need extra multiplexers and registers to pipeline the feedback loops. The benefit of AE architectures over the SE architectures in terms of area saving due to using one multiplier becomes clearly apparent for digit sizes 4 bits and 8 bits. Table 1 also shows the critical propagating path within the proposed architectures. The critical path of D1_L1, D2_L1 and D2_L2 is equal to 2 FAs while it is a single FA for the D1_L2 structure. The critical path for the remaining architectures is equal to *d/l* FAs. In addition, Table 1 distinguishes between the proposed architectures depending on whether or not they avoid global line broadcast. Thus, architectures D1_L1, D1_L2 and D2_L2 are fully systolic while the remaining architectures are semi systolic.

Table 2 follows suit of Table 1 in dividing the architectures into semi and fully systolic structures. The fully systolic architectures D1_L1, D1_L2 and D2_L2 exhibit superior clock frequencies with periods ranging from 3,634 ns to 3,936 ns. This is a clear benefit of making the proposed architectures systolic through retiming. The clock periods of the remaining semi systolic architectures are clearly higher; increasing with the critical path from 8.339 ns, in the case of D2_L1 which exhibits a critical path consisting of 2 FAs, to 20.943 ns in the case of D8_L1, which has a critical path of 8 FAs. In the case of a full 1024-bit RSA exponentiation, the processing time of an SE architecture is generally shorter than that of its respective AE counterpart architecture for the same digit size, except for the D2_L2 architecture which requires 8.28 ms to compute a full 1024-bit RSA operation, which is slightly shorter than the 8.79 ms taken by the D2_L1 structure to carry out the computation of the same operation. This means that the gain achieved in terms of clock frequency thanks to shortening the critical propagation path is not proportional to the increase in the number of processing cycles which increases by almost a factor of two when a single Montgomery multiplier is used. Nevertheless, the processing time decreases when the digit size increases making the D8_L1 whose processing time is only 5.58 ms the fastest of the 8 architectures analyzed in Table 2. Another metric used to assess the proposed architectures is their area usage×processing time performances. Such a metric evaluates whether or not the increase in area usage is coupled with a proportional decrease in processing time. From Table 2, Dx_L1 architectures are more area-time efficient than their Dx_L2 counter parts for digit sizes of up to 4 bits. However, for digit size 8-bit, D8_L2 is clearly more area-time efficient than D8_L1 structure. Furthermore, the D1_L1 architecture is the most area-time efficient architecture among the eight structures of Table 2. Thus, the increase in area usage due to using two Montgomery multipliers and increasing the digit size is not proportional to the gain in terms of processing time. The area-time efficiency of the proposed structures worsens when the digit size increases. Taking this into account, an important result can be pointed out from Tables 1 and 2. Among the proposed architectures, see for example D4_L1 and D8_L2, there are those with a shorter processing time and at the same time calls for less area usage. Thus, the digit approach can be seen as more than a compromise between the bit serial and the bit Parallel approaches. Through an appropriate selection of the digit size, such an approach provides the designer with the best area and speed that match the needs of the system.

The obtained results are compared against similar work in the literature [9,10,18]. In Ref 18, an algorithm combining the Montgomery's technique and the carry save representation of numbers was proposed. The serial structure that implements this algorithm computes a full 1024-bit exponentiation in 27.88 ms. The proposed D1_L1 is much faster, carrying out the same operation in just 8.17 ms. The physical device used in Ref 10 to implement a radix-4 RSA processor is not the same as ours. This makes an exact comparison difficult in terms of hardware resource requirements and time performance. The architecture of Ref 10 shows a critical path of a 4-bit adders and its radix-4 multiplication unit processes 4 bits of data per cycle. It is equal to the propagation path of the proposed D4_L1 architecture. However, unlike the architecture of Ref 10, our design does not make use of optimized blocks and has considerable device-independent features.

V. Conclusions

In this paper, architectures that implement the RSA encryption algorithm have been presented. The new architectures use a modified Montgomery algorithm in which the operations of modular multiplication and modular reduction are carried out separately and in a parallel way. To investigate the best area usage-time trade-off, the digit approach has been adopted. The issue of global lines broadcast has been circumvented by interleaving more than one instance onto the same digit multiplier. Such aim has been achieved by pipelining the feedback loops and retiming the whole structures. In addition, this technique has also the merit of shortening the critical propagation path. The resulting RSA architectures use either two Montgomery multipliers and termed Speed-Efficient architectures, or utilize a single multiplier and as such called Area-Efficient architectures. Furthermore, a VHDL cryptographic engine has been proposed so that by varying the digit size and level of pipelining, the designer may select the best architecture that meets the application requirements in terms of speed and area usage.

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 "Architectures for Montgomery's multiplication", IEE Proceedings, Computers and Digital Techniques, Vol.150, November 2003, Issue 06, p. 361-368.

[3]. O. Nibouche, M. Nibouche, A. Bouridane, and A. Belatreche, " Fast architectures for FPGA based implementation of RSA encryption algorithm", Proceedings. IEEE International Conference on Field Programmable Technology, pp: 271- 278, 6-8 Dec. 2004.

[4]. 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.

[5]. W. Diffie and M. E. Hellman, "New directions in cryptography", IEEE Transactions on Information Theory, 22:644–654, 1976.

[6]. 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.

[7]. M. Shnad and J. Vuillemin, "Fast implementation of RSA cryptography", Proceedings of the 11th IEEE Symposium on Computer Arithmetic, 1993.

[8]. P. L. Montgomery, "Modular multiplication without trial division", Math. Comp. vol. 44, no. 170, pp. 519-521, 1985.

[9]. 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.

[10]. T. Blum and C. Paar, "High-radix Montgomery modular exponentiation on reconfigurable hardware", IEEE Transactions on Computers, 50(7):759–764, July 2001.

[11]. H. Orup, "Simplifying quotient determination in high-radix modular multiplication", In Proceedings of the 12th IEEE Symposium on Computer Arithmetic, pages 193–199, 1995.

[12]. S. E. Eldridge and C. D. Walter, "Hardware implementation of Montgomery's modular multiplication algorithm", IEEE Transactions on Computers, 42:693–699, 93.

[13]. 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.

[14]. Anderson, R., and Kuhn, M., "Tamper Resistance- a Cautionary Note", in The Second USENIX Workshop on Electronic Commerce Proceedings, Oakland, California, November 18-21, 1996, pp 1-11.

[15]. 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.

[16]. C. McIvor, M. McLoone, and J.V. McCanny, "Modified Montgomery Modular Multiplication and RSA Exponentiation Techniques", IEE Proceedings – Computers & Digital Techniques, Vol. 151, No. 6, pp 402-408, Nov 2004.

[17]. 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, pp. 10582-10589.

Table 1. Architectures performances

Architecture	D1_L1	D1_L2	D2_L1	D2_L2	D4_L1	D4_L2	D8_L1	D8_L2
Area (slices)	9100	12400	12000	13700	19800	15000	36600	23000
Critical path (FAs)	2	1	2	2	4	2	8	4
Systolicity	Fully	Fully	Semi	Fully	Semi	Semi	Semi	Semi

Table 2. Implementation results

Architecture	D1_L1	D1_L2	D2_L1	D2_L2	D4_L1	D4_L2	D8_L1	D8_L2
Period (ns)	3.883	3.634	8.339	3.963	13.045	10.500	20.943	14.247
Frequency (Mhz)	257	275	114	254	77	95	48	70
Processing Time (ms)	8.17	15.28	8.79	8.28	6.89	11.06	5.58	7.54
Throughput rate (Kb/s)	15.67	8.38	14.56	15.46	18.58	11.57	22.94	16.98
Area-time (ms × slice)	74347	189472	105480	113436	136422	165900	204228	173420

Table 3. A comparison with similar work in the literature

Architecture	D1_L1	D4_L1	RSA [18]	RSA [10]
Frequency (Mhz)	257	77	78	46
Processing Time (ms)	8.17	6.89	27.88	11.95