Fast Montgomery modular multiplication and RSA cryptographic processor architectures

CONFERENCE PAPER · DECEMBER 2003

DOI: 10.1109/ACSSC.2003.1291939 · Source: IEEE Xplore

Maire Mcloone
Queen's University Belfast
90 PUBLICATIONS **977** CITATIONS

J.V. Mccanny
Queen's University Belfast
197 PUBLICATIONS **2,073** CITATIONS

Fast Montgomery Modular Multiplication and RSA Cryptographic Processor Architectures

Ciaran McIvor, Máire McLoone, John V McCanny
The Institute of Electronics, Communications and Information Technology
School of Electrical and Electronic Engineering
The Queen's University of Belfast, Stranmillis Road, Belfast, BT9 5AH
E-mail: c.mcivor@ee.qub.ac.uk, maire.mcloone@ee.qub.ac.uk, j.mccanny@ee.qub.ac.uk

Alan Daly, William Marnane
Department of Electrical and Electronic Engineering
University College Cork
Ireland
E-mail: aland@rennes.ucc.ie, l.marnane@ucc.ie

Abstract - **New, generic silicon architectures for implementing Montgomery's multiplication algorithm are presented. These use Carry Save Adders (CSAs) to perform the large word length additions required by this algorithm when used for RSA encryption and decryption. It is shown that using a four-to-two CSA with two extra registers rather than a five-to-two CSA leads to a useful reduction in the critical path of the multiplier, albeit at the expense of a small increase in circuitry. For operand lengths of 1536-bits and greater, the percentage gain in data throughput rate outweighs the percentage increase in silicon area. Moreover, for a 2048-bit operand length, typical of what is required in many future generation applications, the gain in data throughput is 27.9% compared with a 9.9% increase in area. The practical application of this approach has been demonstrated by applying this to the design of RSA processor architectures with 512-bit and 1024-bit key sizes. The resulting Montgomery multiplier and RSA processor performance results presented are the fastest reported to date in the literature.**

I. Introduction

Public-key cryptosystems are important as they provide *confidentiality*, *authentication*, *data integrity* and *non-repudiation* [1,2]. RSA [3] is the most widely used public-key cryptosystem. An RSA operation is a modular exponentiation, which requires repeated modular multiplications. For security reasons RSA operand sizes need to be 1024-bits or greater in length meaning that high data throughput rates are difficult to achieve.

The Montgomery multiplication algorithm [4,5] is the most efficient modular multiplication algorithm available. It replaces trial division by the modulus with a series of additions and divisions by a power of 2. Thus, it is well suited to hardware implementations. For this reason, this algorithm has formed the basis of many of the RSA hardware architectures reported to date [6,7,8].

To date, several techniques have been proposed in order to avoid carry propagation during the addition stages of the algorithm as this is a key factor in determining performance. One approach proposed by Elbirt and Paar [7] is to break these additions into x-bit stages, where x is an optimal bit length chosen to take advantage of the fast carry chains available on modern FPGAs. However, an inevitable drawback of this approach is that architectures developed can be very heavily technology and implementation dependent. For example, it is unlikely that a design developed for a specific FPGA family will show the same speed advantages if migrated to a modern ASIC technology or indeed an alternative type of FPGA or PLD . An alternative approach presented by Blum and Paar [8] is based on the use of FPGA systolic array multiplier architectures with varying processing element sizes (i.e. 4-bits, 8-bits and 16-bits). However, these systems have again been tailored specifically for the Xilinx XC4000 FPGA series, limiting, we believe, the broader application of the architectural ideas presented.

In this paper we introduce new, generic Montgomery multiplier and associated RSA processor architectures, which can be rapidly reconfigured to accommodate varying operand sizes and are suitable for implementation across a wide range of technology (FPGA, PLD, ASIC) options. These architectures are therefore ideally suited to a multitude of applications, including those that require a very high level of security to those, which only need relatively little protection over a shorter time period. As will be discussed, the designs created are highly competitive and we believe produce the best performance results obtained to date.

As will be discussed, long carry propagation can be avoided by using carry save adders (CSAs) to perform the addition stages of Montgomery's algorithm. The inherently parallel structure of these adders means that a shallow combinational logic depth is guaranteed independent of the target technology onto which the CSAs are implemented. This also makes them ideal for very long integer arithmetic, a key requirement for RSA implementation. As is discussed in Section II, we propose and compare two different multiplier architectures, one using a five-to-two CSA and the other a four-to-two CSA with two additional registers. Each multiplier can perform a Montgomery multiplication in only $k+1$ and $k+2$ clock cycles respectively, where k is the operand bit length. It is shown that the latter architecture outperforms the former in terms of data throughput rate, but vice versa in terms of area, for operand sizes greater than 512-bits. However, the percentage increase in the throughput rate outweighs the percentage increase in area for very large operand lengths in excess of 1536-bits. Thus, the four-to-two CSA architecture will become increasingly attractive in the future when RSA key sizes of 2048-bits and greater in length are needed to protect information.

As will be discussed in Section III, an RSA processor using the five-to-two multiplier can carry out encryption in only $(k+2)(e_k+3)$ clock cycles, where k is the modulus bit length and e_k is the public exponent bit length. Here the Fermat prime, $F_4=2^{16}+1$, is used as the public exponent. RSA decryption can be performed in $(k/2+2)(d_k/2+3)$ clock cycles, where d_k is the private exponent bit length, also using the five-to-two multiplier. The Chinese Remainder Theorem (CRT) technique [9] is used in this case.

The concepts developed have been verified and demonstrated through FPGA implementations, in this case using the Xilinx Virtex2 family of chips. The performance results of these Montgomery multiplier and RSA processor implementations are, to the authors' knowledge, the best reported in the literature to date.

II. Montgomery Multiplier Architectures

A. *Montgomery Multiplication Background*

Given an integer $a < n$, where n is the k-bit modulus, A is said to be its n-residue with respect to r if,

$$A = a*r \ (mod\ n) \tag{1}$$

where $r=2^k$.

Likewise, given an integer $b < n$, B is said to be its n-residue with respect to r if,

$$B = b*r \ (mod\ n) \tag{2}$$

The Montgomery product of A and B can then be defined as,

$$Z = A*B*r^{-1} \ (mod\ n) \tag{3}$$

where r^{-1} is the inverse of r, modulo n.

The radix-2 version of Montgomery's multiplication algorithm [10], which calculates the Montgomery product of A and B, is summarised in the pseudo code below.

Algorithm I: *Montgomery Multiplication (A, B, n)*

```
$S[0] = 0$;
for i in 0 to k-1 loop
    $q_i = (S[i]_0 + A_i * B_0) \bmod 2$;
    $S[i+1] = (S[i] + A_i * B + q_i * n) \text{ div } 2$;
end loop;
return S[k];
```

The critical delay of Algorithm I occurs during the calculation of the S values given by the three input addition,

$$S[i+1] = (S[i] + A_i * B + q_i * n) \tag{4}$$

The main contributing factor to this delay is the *carry propagation* resulting from the very large operand additions. However, this can be avoided by using CSAs to calculate (4). This observation led us to re-write Algorithm I in alternative formats, which we have defined as Algorithm II and Algorithm III in Sections II.B and II.C below. They are explicitly derived so that they can employ different CSA variants when implemented architecturally. The motivation is to produce silicon circuits with a small critical path delay.

B. *Five-to-two CSA Architecture*

The first of these variants is based on a five-to-two CSA, which is used to calculate (4). This type of adder comprises three levels of carry save logic, where each level of logic implements (5) and (6).

$$SUM = X1\ xor\ X2\ xor\ X3 \tag{5}$$

$$CARRY = (X1\ and\ X2)\ or\ (X1\ and\ X3)\ or\ (X2\ and\ X3) \tag{6}$$

The sum of the bit vectors *SUM* and *CARRY* is equal to the sum of the three input bit vectors *X1*, *X2* and *X3*. An outline diagram of the five-to-two CSA operation is shown in Fig. 1. The carry save representation of the five input operands is output from the register after only one clock cycle using this approach.

Algorithm I can be re-written as Algorithm II, given below. Note that the input operands *A* and *B* and the output product *S* are now represented in carry save format as *A1* and *A2*, *B1* and *B2*, and *S1* and *S2* respectively. *CSR* stands for *carry save representation*.

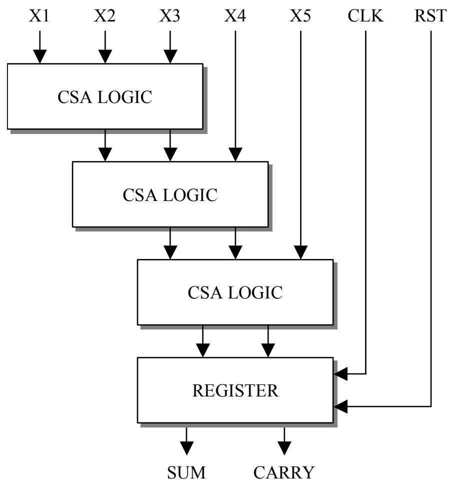

Fig. 1. Block Diagram of Five-to-two CSA

Algorithm II: *Five-to-two CSA Montgomery Multiplication (A1, A2, B1, B2, n)*

```
$S1[0] = 0$;
$S2[0] = 0$;
for i in 0 to k-1 loop
    $q_i = (S1[i]_0 + S2[i]_0) + (A_i * (B1_0 + B2_0)) \bmod 2$;
    $S1[i+1], S2[i+1] =$
    $CSR\ (S1[i] + S2[i] + A_i * (B1 + B2) + q_i * n) \text{ div } 2$;
end loop;
return S1[k], S2[k];
```

The critical delay of Algorithm II occurs during the calculation of the five-to-two carry save addition,

$$S1[i+1], S2[i+1] = CSR\ (S1[i] + S2[i] + A_i * (B1 + B2) + q_i * n) \tag{7}$$

The determination of the correct A_i values means that a full addition of the input operands *A1* and *A2* needs to be carried out. This is calculated "on-the-fly" using a barrel register full adder component (BRFA) as shown in Fig. 2.

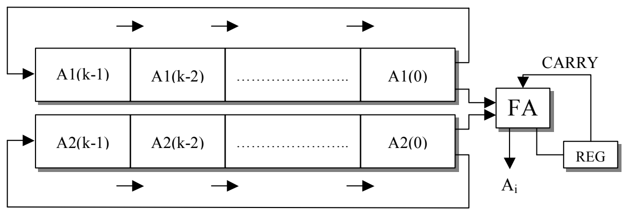

Fig. 2. Barrel Register Full Adder Component

The correct A_i value is ready when needed by adding the least significant bit of the *A1* and *A2* input operands using a single full adder. Once the current A_i value has been used, the *A1* and *A2* operands are barrel shifted one place to the right so that the next A_i value can be determined. In this way no extra clock cycles are required to complete a full addition of the *A1* and *A2* input operands. As a result, (7) can be computed in a single clock cycle. Also, an extra one clock cycle is required to reset the signals *S1[0]* and *S2[0]* to zero at the beginning of each multiplication. Thus, Algorithm II can be executed in only k+1 clock cycles, where k is the bit length of the operands. Also, due to the parallel nature of the carry save adders, a high Montgomery multiplier data throughput rate can be achieved here.

In order to verify this approach, a silicon design based on this was captured generically in VHDL and implemented into the Xilinx Virtex2 series of FPGAs using varying bit lengths. Table I provides performance results for these implementations.

TABLE I
PERFORMANCE RESULTS FOR FIVE-TO-TWO CSA MULTIPLIER

Xilinx Device	Bit Length (k)	Clock Speed (MHz)	Area (Slices)	Throughput Rate (Mb/s)
XC2V1500	512	105.57	4,962	105.36
XC2V3000	1024	71.04	10,332	70.97
XC2V6000	1536	65.57	15,697	65.53
XC2V6000	2048	55.19	20,984	55.16

It should be noted that an increase in the operand bit length by a factor of 4 from 512-bits to 2048-bits results in an increase of only 1.9 in the critical path delay through the multiplier. On the other hand, the area of the multiplier approximately doubles as the bit length doubles. Thus, the architectures are highly scalable and regular, which is a useful characteristic as it ensures that the multiplier design will be of use in the future when operand sizes will need to be greater than 2048-bits in length. It should also be noted that the area figures presented include all the I/O registers required to store the large bit length operands, as shown in Fig. 3.

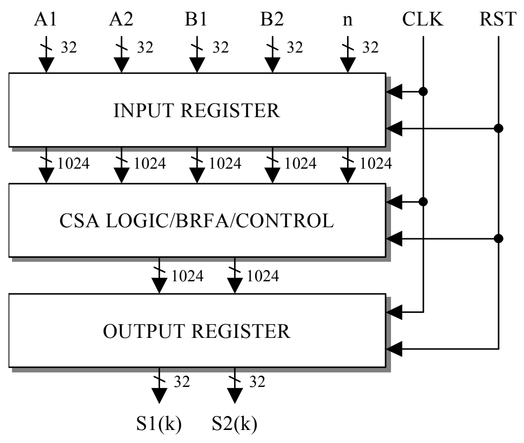

Fig. 3. I/O Interface of Five-to-two CSA Montgomery Multiplier

For a 1024-bit operand length, the inputs are registered into the microchip at a rate of 32-bits per clock cycle. Thus, this process is completed after thirty-two clock cycles. Likewise, the outputs are clocked out of the chip 32-bits per clock cycle over thirty-two cycles. This is necessary due to the limited number of I/O pins available on currently available silicon chips. The same I/O interface technique, as shown in Fig. 3, is used for the four-to-two CSA multiplier architectures and the RSA processor architectures, which are detailed in Sections II.C and III.B respectively.

The approach described can be further extended to improve data throughput rate at the expense of silicon area. This is discussed in the next section.

C. *Four-to-two CSA Architecture*

This architectural approach is based on the use of a four-to-two rather than a five-to-two CSA resulting in a saving of a full level of carry save logic, as shown in Fig. 4.

Again, the carry save representation of the four input operands is output from the register after only one clock cycle. Therefore, by using the four-to-two CSA to calculate (4), Algorithm I can be reformulated as Algorithm III.

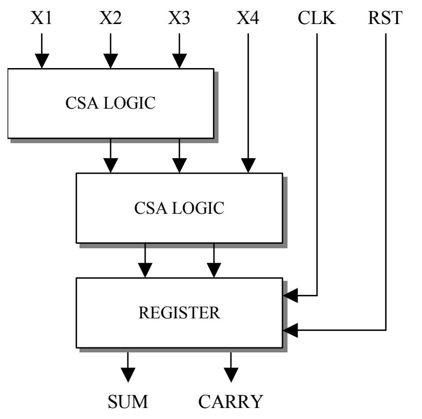

Fig. 4. Block Diagram of Four-to-two CSA

```
Algorithm III: Four-to-two CSA Montgomery Multiplication
                    (A1, A2, B1, B2, n)
D1, D2 = CSR (B1 + B2 + n + 0)
S1[0] = 0;
S2[0] = 0;
for i in 0 to k-1 loop
    $q_i$ = ($S1[i]_0$ + $S2[i]_0$) + ($A_i$ * ($B1_0$ + $B2_0$)) mod 2;
    if $A_i$ = 0 and $q_i$ = 0 then
        S1[i+1], S2[i+1] = CSR (S1[i] + S2[i] + 0 + 0) div 2;
    elsif $A_i$ = 1 and $q_i$ = 0 then
        S1[i+1], S2[i+1] =
        CSR (S1[i] + S2[i] + B1 + B2) div 2;
    elsif $A_i$ = 0 and $q_i$ = 1 then
        S1[i+1], S2[i+1] = CSR (S1[i] + S2[i] + n + 0) div 2;
    else
        S1[i+1], S2[i+1] =
        CSR (S1[i] + S2[i] + D1 + D2) div 2;
    end if;
end loop;
return S1[k], S2[k];
```

The main computation to be performed here is the four-to-two carry save addition,

$$S1[i+1], S2[i+1] = CSR\ (S1[i] + S2[i] + y + z) \qquad (8)$$

The value of y and z in (8) depends on the value of both the A_i and q_i signals from Algorithm III. Indeed, the determination of the state of both these signals simultaneously, represents extra control logic in the form of a more complex multiplexer than is required for the five-to-two CSA case. This needs to be taken into account in determining the critical delay. However, as shown below, this delay is still less than that implied by Algorithm II. This decrease in the critical path delay comes at the expense of two extra registers, *D1* and *D2*. These are used to store the carry save representation of the input operands *B1*, *B2* and *n*, which are added at the beginning of each new multiplication. *D1* and *D2* are possible values of *y* and *z* from (8). The calculation of *D1* and *D2* at the beginning of each multiplication means that an extra clock cycle is required to complete a full Montgomery multiplication. Also, the A_i values required in Algorithm III must again be computed "on-the-fly" using the BRFA component shown in Fig. 2. Thus, Algorithm III can be executed in only k+2 clock cycles.

The four-to-two CSA design was also captured generically in VHDL and implemented into the Xilinx Virtex2 series of FPGAs for varying bit lengths. Table II gives performance results for these implementations. These results can be directly compared with those given in Table I for the five-to-two CSA multiplier. Fig. 5 provides a graphical representation of the comparisons in terms of percentage increase in data throughput rate and area when using the four-to-two CSA architecture rather than the five-to-two design.

For operand lengths greater than 512-bits, the four-to-two architecture has a consistently higher throughput rate than the five-to-two architecture. However, the percentage increase in area is higher than the percentage increase in throughput for operand lengths up to approximately 1536-bits.

TABLE II
PERFORMANCE RESULTS FOR FOUR-TO-TWO CSA MULTIPLIER

Xilinx Device	Bit Length (k)	Clock Speed (MHz)	Area (Slices)	Throughput Rate (Mb/s)
XC2V1500	512	105.56	5,689	105.15
XC2V3000	1024	76.23	11,617	76.08
XC2V6000	1536	70.97	17,300	70.88
XC2V6000	2048	70.64	23,060	70.57

For operand lengths greater than this, however, the throughput benefits begin to outweigh the negative effects of increased area. In fact, for an operand length of 2048-bits, the percentage increase in throughput is 27.9% in comparison to a relatively small increase in area of 9.9%. The overall trend points towards a constant rise in the percentage throughput rate coupled with a decline in the percentage area increase as operand lengths increase. Therefore, it seems that the four-to-two CSA multiplier is of more use for RSA systems with key lengths in excess of 1536-bits and therefore becomes an increasingly attractive option in future RSA systems. For bit lengths less than this then the five-to-two CSA multiplier is the preferable option.

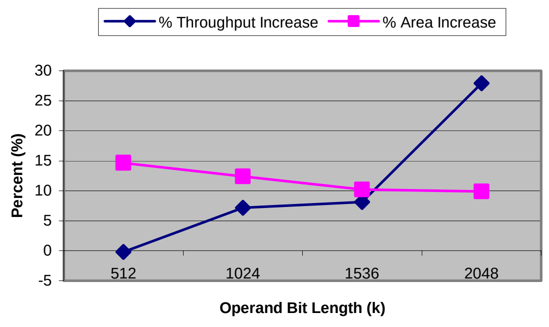

Fig. 5. % Throughput/Area Increase using 4-to-2 CSA instead of 5-to-2 CSA

The Montgomery multiplier performance results reported are, to the authors' knowledge, the fastest reported in the literature to date. Also, the concepts outlined in this section can easily be applied to other Montgomery multiplier designs, which employ different addition techniques to calculate (4).

The next section will demonstrate the practicality of the multiplier architectures through an implementation of the RSA cryptosystem for 512-bit and 1024-bit key sizes. For these key sizes, the five-to-two CSA multiplier is used to calculate the successive modular multiplications.

III. RSA PROCESSOR ARCHITECTURES

A. *RSA and Modular Exponentiation*

The RSA encryption and decryption functions are given by $C=M^e(mod\ n)$ and $M=C^d(mod\ n)$ respectively, where M is a plaintext message block, C is a ciphertext block, n is the k-bit modulus, and e and d are the public and private exponents respectively. The equation $ed=1(mod(p-1)(q-1))$ must also hold, where p and q are two large prime numbers (~ $k/2$-bits in length) and $n=pq$.

Thus, an RSA operation is a modular exponentiation with operands satisfying the conditions stated above. A well known and widely used modular exponentiation algorithm is the square and multiply algorithm given as Algorithm IV below, which computes $M=C^d(mod\ n)$. d_k is the bit length of the private exponent *d*.

```
Algorithm IV: Modular Exponentiation (C, d, n)
K = $2^{2k}$ (mod n);  (computed externally)
P[0] = MontMult (K, C, n);
R[0] = MontMult (K, 1, n);
for i in 0 to $d_k$ loop
    P[i+1] = MontMult (P[i], P[i], n);
    if d[i] = 1 then
        R[i+1] = MontMult (R[i], P[i], n)
    end if;
end loop;
M = MontMult (1, R[k], n);
return M;
```

B. The RSA Cryptoprocessor Architectures

By using the five-to-two CSA multiplier architecture to calculate the MontMult (Montgomery Multiplication) stages of Algorithm IV, we can re-write this algorithm as Algorithm V, given below. As a result, we have been able to develop RSA processor architectures with very impressive data throughput rates.

```
Algorithm V: Five-to-two Multiplier Modular Exponentiation
             (C, d, n)
K = $2^{2k}$ (mod n);  (computed externally)
P1[0], P2[0] = 5to2_MontMult (K, 0, C, 0, n);
R1[0], R2[0] = 5to2_MontMult (K, 0, 1, 0, n);
for i in 0 to $d_k$ loop
    P1[i+1], P2[i+1] =
    5to2_MontMult (P1[i], P2[i], P1[i], P2[i], n);
    if d[i] = 1 then
        R1[i+1], R2[i+1] =
        5to2_MontMult (R1[i], R2[i], P1[i], P2[i], n)
    end if;
end loop;
M1, M2 = 5to2_MontMult (1, 0, R1[k], R2[k], n);
M = M1 + M2;
return M;
```

An analysis of the total number of clock cycles required to compute a full RSA encryption and decryption operation is given in Table III. Again, *k* is the bit length of the modulus and e_k and d_k are the bit lengths of the public and private exponents respectively. An extra clock cycle is now required to complete a full modular multiplication, as shown in Table III. This is to allow the results of the intermediate multiplications and squarings in the *for* loop in Algorithm V to be registered and re-used as inputs to the next iteration of that loop. The pre- and post-processing operations are required to convert the operands in Algorithm V to and from *n*-residue format respectively, as required by the Montgomery multiplication algorithm described in Section II. The final addition operation makes use of the BRFA component, as shown in Fig. 2, to perform a full addition of the carry save result, *M1* and *M2*, of Algorithm V.

TABLE III
CLOCK CYCLES TO COMPUTE ONE RSA EXPONENTIATION

Type of Operation	Corresponding Algorithm 5 Calculations	Number of Clock Cycles for Encryption	Number of Clock Cycles for Decryption
Pre-processing	P1[0],P2[0] R1[0],R2[0]	k+2	k/2+2
For Loop	P1[i+1],P2[i+1] R1[i+1],R2[i+1]	e_k(k+2)	d_k/2(k/2+2)
Post-processing	M1,M2	k+2	k/2+2
Final Addition	M	k+2	k/2+2
Total Number Clock Cycles:		(k+2)(e_k+3)	(k/2+2)(d_k/2+3)

By using the CRT technique [9], two half-size decryption operations can be computed in parallel. Hence, the amount of clock cycles required to compute this operation is decreased by approximately a factor of four, as shown in Table III, thus increasing the potential data throughput rate of the RSA decryption function significantly.

RSA processor architectures based on the five-to-two CSA multiplier described were captured generically in VHDL and implemented using the Xilinx Virtex2 series of FPGAs (using the highest place and route effort level) for 512-bit and 1024-bit key sizes. Tables IV and V provide performance results for these implementations. A public exponent of 2^{16}+1 and a *k*-bit private exponent were used during testing.

TABLE IV
RSA ENCRYPTION RESULTS

Xilinx Device	Bit Length (k)	Clock Speed (MHz)	No. CLB Slices	Encryption Throughput Rate (Mb/s)	One RSA Encryption (ms)
XC2V3000	512	96.27	11,196	4.79	0.11
XC2V6000	1024	93.34	22,075	4.66	0.22

The RSA processor performance results reported here are, to the authors' knowledge, the fastest reported in the literature to date. For instance, our 1024-bit encryption and decryption times are 3.4 and 3.9 times faster respectively than 1024-bit RSA encryption and decryption times recently reported by Blum and Paar [11]. They also reported a 1024-bit *radix-16* decryption time of 3.10 ms, which is 1.2 times slower than our best result. Wu, Hong and Wu [12] also recently reported a VLSI RSA implementation with a 512-bit decryption throughput rate of between 328 kb/s and 578 kb/s. This is based on a 0.6 μm CMOS technology. This result is 2.5 and 1.4 times slower than our 512-bit decryption throughput rate, in the worst and best cases respectively.

TABLE V
RSA DECRYPTION RESULTS

Xilinx Device	Bit Length (k)	Clock Speed (MHz)	No. CLB Slices	Decryption Throughput Rate (kb/s)	One RSA Decryption (ms)
XC2V3000	512	105.98	13,513	812.03	0.63
XC2V6000	1024	100.49	24,767	388.73	2.63

It should also be noted that both of the aforementioned implementations also applied the CRT technique [9] to RSA decryption and that Blum and Paar [11] also used a public exponent of $2^{16}+1$.

C. Further Discussions

As discussed in Section I, our designs are technology independent and thus can be ported to other technologies without difficulty. Therefore, even higher data rates should be achievable when the designs are migrated to modern ASIC technology. Previous experience would indicate [13] that a further doubling in performance is possible if the architectures are implemented in a 0.18μm CMOS. Also, with some further modifications, our Montgomery multiplier designs can be tailored so that they can be implemented as arithmetic co-processors on modern microprocessor based smart cards. Typical RISC processors for such applications require around 80,000 gates with associated co-processors requiring between 80,000 and 190,000 gates depending on the bit length *k*, as shown in Table VI.

TABLE VI
MONTGOMERY MULTIPLIER AREA RESULTS (NUMBER OF GATES)

Multiplier	Bit Length (k)	Area (Gates)
Five-to-two CSA	512	82,929
Five-to-two CSA	1024	165,048
Four-to-two CSA	512	93,743
Four-to-two CSA	1024	187,339

The figures in Table VI are based on the use of a single Montgomery multiplier. As shown, the gate counts required are easily within the realms of what can be integrated on a single chip. Also, by exploiting certain time-area trade-offs such as reducing the amount of carry save logic used to evaluate (7) and (8), then the silicon area of our architectures can be reduced significantly at the expense of some extra latency. For example, the amount of carry save logic used in the five-to-two CSA multiplier architecture can be reduced by two thirds at the expense of two extra clock cycles required to compute (7). Thus, as discussed in Section I, the architectures presented are highly flexible and can be used in a multitude of different industrial applications, including as high-speed network security processors or as low silicon area arithmetic co-processors on embedded smart cards.

IV. CONCLUSION

In this paper we have presented new, generic modular multiplication architectures suitable for implementing Montgomery's multiplication algorithm. CSAs were used to perform the large operand additions required when used in an RSA implementation. It has been shown that by using a four-to-two CSA with two extra registers rather than a five-to-two CSA, then a useful reduction in the critical path delay of the multiplier can be achieved albeit at the expense of some additional silicon. However, for very large operand lengths of 1536-bits and greater, the percentage gain in data throughput rate outweighs the percentage increase in area. For a 2048-bit operand length, the gain in data throughput is 27.9% compared with a 9.9% increase in area. Thus, the four-to-two CSA multiplier is of more use for RSA systems with key lengths in excess of 1536-bits, whereas the five-to-two CSA multiplier is preferable otherwise. In order to demonstrate the practicality of our multipliers, RSA processor architectures were developed for 512-bit and 1024-bit key sizes using the five-to-two CSA multiplier. The resulting FPGA-based Montgomery multiplier and RSA processor performance results presented are, we believe, the fastest reported in the literature to date with future significant performance gains achievable through ASIC implementation.

ACKNOWLEDGMENT

Amphion Semiconductor Ltd. and a Northern Ireland Department of Learning postgraduate studentship in the form of a CAST award have funded this research.

REFERENCES

[1] Stallings, W.: "Network and Internetwork Security Principles and Practice". Prentice Hall International, 1995.
[2] Menezes, A., Oorschot, P., Vanstone, S.: "Handbook of Applied Cryptography". CRC Press, 1997.
[3] Rivest, R.L., Shamir, A., Adleman, L.: "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM, 21(2): 120-126, February 1978.
[4] Montgomery, P.L.: "Modular Multiplication without Trial Division". Math. Computation, Vol. 44, pp. 519-521, 1985.
[5] Koc, C.K., Acar, T., Burton S. Kaliski Jr.: "Analyzing and Comparing Montgomery Multiplication Algorithms". IEEE Micro, 16(3): 26–33, June 1996.
[6] Eldridge, S.E., Walter, C.D.: "Hardware Implementation of Montgomery's Modular Multiplication Algorithm". IEEE Transactions on Computers, Vol. 42, pp. 693–699, July 1993.
[7] Elbirt, A.J., Paar, C.: "Towards an FPGA Architecture Optimized for Public-Key Algorithms". SPIE Symposium on Voice, Video and Communications, Sept 1999.
[8] Blum, T., Paar, C.: "Montgomery Modular Exponentiation on Reconfigurable Hardware". Proceedings 14th Symposium on Computer Arithmetic, pp. 70-77, 1999.
[9] Quisquater, J-J., Couvreur, C.: "Fast Decipherment Algorithm for RSA Public-Key Cryptosystem". Electronics Letters, Vol. 18, pp. 905–907, October 1982.
[10] Walter, C.D.: "Montgomery Exponentiation Needs No Final Subtractions". Electronics Letters, 35(21): 1831-1832, October 1999.
[11] Blum, T., Paar, C.: "High-Radix Montgomery Modular Exponentiation on Reconfigurable Hardware". IEEE Transactions on Computers, Vol. 50, pp. 759-764, July 2001.
[12] Wu, C-H., Hong, J-H., Wu, C-W.: "VLSI Design of RSA Cryptosystem Based on the Chinese Remainder Theorem". Journal of Information Science and Engineering 17, 967-980, 2001.
[13] Amphion Semiconductor Ltd.: "Data Security Data Sheets", http://www.amphion.com