EFFICIENT HARDWARE ARCHITECTURES FOR MODULAR MULTIPLICATION ON FPGAS

David Narh Amanor, Christof Paar, Jan Pelzl

Horst Görtz Institute for IT Security
Ruhr University Bochum, Germany
email: {amanor,cpaar,pelzl}@crypto.rub.de

Viktor Bunimov, Manfred Schimmler

Technical Computer Science
University of Kiel, Germany
email:{vb,masch}@informatik.uni-kiel.de

ABSTRACT

The computational fundament of most public-key cryptosystems is the modular multiplication. Improving the efficiency of the modular multiplication is directly associated with the efficiency of the whole cryptosystem. This paper presents an implementation and comparison of three recently proposed, highly efficient architectures for modular multiplication on FPGAs: interleaved modular multiplication and two variants of the Montgomery modular multiplication. This (first) hardware implementation of these designs shows their relative performance regarding area and speed.

One of the main findings is that the interleaved multiplication has the least area time product of all investigated architectures.

As a typical cryptographic application, we show that a 1024-bit RSA exponentiation can be performed in less than 6.1ms at a clock rate of 69MHz on a Xilinx Virtex FPGA.

Keywords: modular multiplication, Montgomery multiplication, efficient implementation, hardware, FPGA, cryptography, RSA.

1. INTRODUCTION

With the introduction of public-key primitives for tasks such as digital signatures and key establishment, cryptographic applications have become computational demanding . Nowadays, a single public-key encryption such as, e.g., an RSA or DSA (digital signature) operation can involve thousands of modular multiplications with operands of size of 1024 bits and above. On the algorithmic side, many improvements have been proposed in order to speed up operations such as a typical RSA encryption. Many improvements tackle modular multiplication since it is the basis of most current public-key primitives, such as RSA, DSA, Diffie-Hellman key exchange, and Elliptic Curve Cryptosystems. Making modular multiplication efficient directly relates to an increased overall efficiency of a top-level cryptosystem. Enhancements in algorithms are manifold: some methods provide generic improvements, i.e., regardless of the way of implementation (hard- or software). Examples include the Montgomery modular multiplication [1] and the interleaved modular multiplication [2]. Other methods target special platforms such as, e.g., conventional 32-bit processors, digital signal processors, or custom designed circuits such as ASICs or FPGAs.

In this contribution, we will describe a hardware implementation of two refined algorithmic concepts for the modular multiplication. The Montgomery modular multiplication and the interleaved modular multiplication can be modified such that they yield an area-time efficient design. Since the proposal of such architectures in [3, 4], this is the first contribution providing an actual implementation and, hence, showing a comparison basis of the obtained results. The implementation is done on a typical FPGA. It turns out, that the inherent structure of the (improved) algorithms is very well suited for the implementation on an FPGA. Efficiency gains by table look-ups perfectly meets the availability of look-up tables (LUTs) on a modern commercially available FPGA. Hence, it is possible to efficiently use the LUTs of an FPGA to store pre-computed values with very low area requirements. Furthermore, FPGAs allow for parallel execution of independent instances. We will make use of carry save adders, where several fulladder can work in parallel.

2. MODULAR MULTIPLICATION

Modular Multiplication is the mathematical operation on integers $X \cdot Y \bmod M$ with $X, Y < M$, whereby X and Y are the operands and M is the modulus. In current practical crypto applications X, Y and M are large numbers of 150 bits or more. There are many different algorithms for modular multiplication. In this paper we focus on the two most important ones.

During interleaved modular multiplication the multiplication and the calculation of the remainder of the division are interleaved. The advantage is that the length of the intermediate result is only one or two bits larger than the operands. The disadvantage is the use of subtractions in order to reduce the intermediate results. More detailed information about interleaved modular multiplication is given in Section 2.1.

Montgomery's modular multiplication is the most frequently used algorithm for modular multiplication. The computation is done in the Montgomery domain. The advantage of this calculation is that we do not need subtractions in order to reduce the intermediate results. The disadvantage is the fact that the Montgomery's modular multiplication calculates $X \cdot Y \cdot 2^{-n} \bmod M$ instead of $X \cdot Y \bmod M$ and two Montgomery's modular multiplications are required for one modular multiplication. More detailed information about Montgomery's modular multiplication is given in Section 2.2.

2.1. Interleaved Modular Multiplication

The idea of interleaved modular multiplication is very simple: the first operand is multiplied with the second operand bitwise and added to the intermediate result. The intermediate result is reduced with respect to the modulus. For this purpose two subtractions per iteration are required. Algorithm 1 gives a pseudo code implementation of interleaved modular multiplication. The algorithm specified above has

Algorithm 1 Interleaved Modular Multiplication

INPUT: X, Y, M with $0 \leq X, Y \leq M$
OUTPUT: $P = X \cdot Y \bmod M$
n: number of bits of X
x_i: i^{th} bit of X

1. $P = 0$;
2. for $(i = n - 1; i \geq 0; i = i - 1)\{$
3. $\quad P = 2 \cdot P$;
4. $\quad I = x_i \cdot Y$;
5. $\quad P = P + I$;
6. $\quad$ if $(P \geq M)\ P = P - M$;
7. $\quad$ if $(P \geq M)\ P = P - M;\}$

several drawbacks: The first problem is the comparison of P and M in the steps (6) and (7). In the worst case all bits of M and P must be compared. This problem can be solved by an approximative comparison with 2^n instead of M. The second problem is the number of additions or subtractions in the steps (5), (6), and (7). All these operations can be replaced by one addition only. In order to do so, we pre-estimate the number of Ms to be subtracted, and find out if Y will be added in the next iteration of the loop. There are only few possible values for this estimation. These can be pre-calculated and stored in a look-up table. In each iteration of the loop, the estimation of the previous iteration can be added to the intermediate result. Furthermore, we can replace the slow non-redundant adder by the fast redundant carry save adder. The time complexity of a carry save adder is equal of the time delay of one single full adder. The following data-flow diagram in Figure 1 shows this optimization. The redundant interleaved modular multiplication requires one carry save adder only. Its complete time complexity is equal to n time delays of one full adder. The algorithm for redundant interleaved modular multiplication is described in [4].

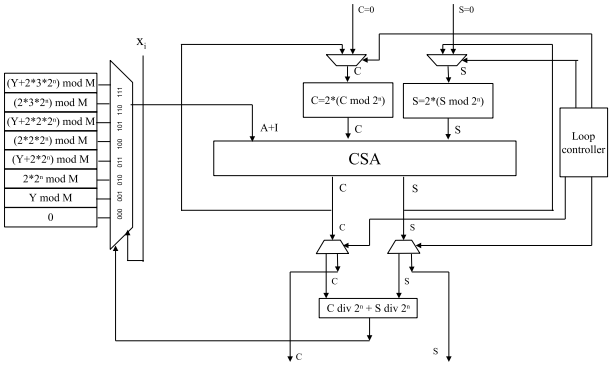

Fig. 1. Redundant interleaved modular multiplication

2.2. Montgomery's Modular Multiplication

Algorithm 2 gives a pseudo code implementation of Montgomery's modular multiplication. A modular multiplication

Algorithm 2 Montgomery Modular Multiplication

INPUT: $X, Y < M < 2^n$, with $2^{n-1} < M < 2^n$ and $M = 2t + 1$, with $t \in \mathbb{N}$.
OUTPUT: $P = X \cdot Y \cdot 2^{-n} \bmod M$.
n: number of bit in X,
x_i: i^{th} bit in X

1. $P = 0$;
2. for $(i = 0; i < n; i++)\{$
3. $\quad P = P + x_i \cdot Y$
4. $\quad$ if $(p_0 = 1)\ P = P + M$;
5. $\quad P = P$ div 2;$\}$
6. if $(P \geq M)\ P = P - M$;

for numbers in standard representation requires two modular Montgomery multiplications. The advantage of Algorithm 2 is that it does not require any subtractions. However, the algorithm requires two additions per iteration of the loop, and these additions are slow, because they are non-redundant. This problem was solved in [5]: The standard additions were replaced by carry save additions. This version requires two carry save additions per iteration of the loop only. The time complexity of each iteration is equal to the time delay of two full adders. Consequently, the time complexity of one Montgomery modular multiplication is equal to $2n$ time delays of one full adder. A further development of Montgomery's modular multiplication was described in [3]. One of four different values ($0, M, Y$, and $Y + M$) can be added during each iteration of the loop. All these values can be stored in

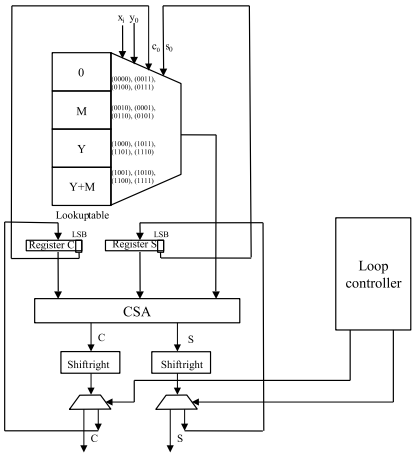

Fig. 2. Montgomery modular multiplication with one carry save adder

a look-up table. The corresponding value can be selected and added to the intermediate result. The advantage of this algorithm is its low time complexity of n time delays of one full adder for the Montgomery modular multiplication.

3. IMPLEMENTATION

The primary objective of this paper is to provide a fair comparison of the efficiency of architectures for the modular multiplication in hardware.

3.1. Methodology

The implementation consists of two essential steps: First, the implementation and simulation of the targeted architectures in a high level programming language (JAVA). Second, the implementation, Simulation, and Synthesis in VHDL. All architectures have been synthetisized for the Xilinx FPGA XVC2000E-6.

The design capture of the modular multiplication architectures using VHDL was done for 32 to 1024 bit. The basic units of the architectures which comprises carry save adders, shift registers and registers were modeled as components which are independently functional. In all the implementations, only standard logic data types are used to provide portability (IEEE 1164). The state machine for the whole architecture consists of 5 states: Idle, Loading, Loading Complete/ Starting, Running, Finished. Precision RTL Synthesis tools were used to map the implemented architectures to Xilinx Virtex 2000E device. The area and timing reports after place and route are to be used to compare the implemented architectures.

3.2. Results

The results of the design as depicted in Tables 1 and 2 were generated after place and route for the Xilinx FPGA for each of the implemented architectures. The tables and figures represent the overall hardware requirements for the complete multipliers including the block of registers for holding the input and output bits. The Montgomery architecture with two carry save adders occupies the largest number of configurable logic block (CLB) slices for all implementations. The CLB requirements for the optimized architectures, namely, Montgomery with one CSA and the redundant interleaved are approximately the same. In Figure 3, the three imple-

Table 1. Percentage of configurable logic block slices (out of 19200) occupied depending on the bitlength (XVC2000E-6)

Precision in bit	Montgomery with two CSAs	Montgomery with one CSA	Redundant Interleaved
32	1.2%	0.7%	0.7%
64	2.4%	1.4%	1.4%
128	4.7%	2.7%	2.7%
256	9.4%	5.4%	5.4%
512	16.1%	8.0%	8.0%
1024	45.0%	21.0%	24.0%

Table 2. Minimum period and maximum frequency depending on the bitlength (XVC2000E-6)

Bit	Montgomery with two CSAs	Montgomery with one CSA	Redundant Interleaved
32	17.7ns (56.5MHz)	16.4ns (61.1MHz)	14.2ns (70.2MHz)
64	20.3ns (49.2MHz)	16.0ns (62.5MHz)	17.3ns (57.7MHz)
128	18.7ns (53.4MHz)	19.5ns (51.3MHz)	20.7ns (48.4MHz)
256	21.3ns (47.0MHz)	21.8ns (46.0MHz)	21.4ns (46.8MHz)
512	24.5ns (40.8MHz)	23.1ns (43.3MHz)	15.4ns (64.8MHz)
1024	23.8ns (42.1MHz)	20.4ns (49.0MHz)	14.4ns (69.4MHz)

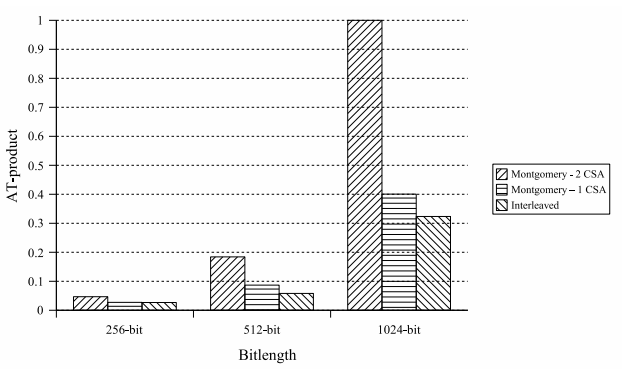

Fig. 3. Comparison of AT-Products (normalized)

mented multipliers are examined in terms of the product of area and time. The percentage of CLBs occupied is used to represent the approximate area. The absolute time for computing a single modular multiplication is obtained by the product of minimum period times number of required clock cycles for completing a modular multiplication.

3.3. A Case Study: RSA

The basic operation in the RSA encryption and signature scheme is inherently based on modular multiplication. I.e., for every public key operation, a modular exponentiation of numbers of size, e.g., of $N = 1024$ bit is computed[1]. For an RSA encryption we need to compute the modular exponentiation $y = x^b \bmod m$, where all operands are approximately of the same size (e.g., $N \approx 1024$ bit). The exponentiation can be done efficiently with the *square and multiply algorithm* using $N \cdot 1.5$ multiplications. For a Montgomery multiplier, we additionally need to convert back to the normal representation, thus, increasing the multiplication count by one. Hence, the time for an N-bit RSA encryption using the three different multiplier architectures can be computed on basis of the time of a single multiplication as $T_{\mathrm{res}} = N \cdot 1.5 \cdot T_{\mathrm{mul},N}$. If we apply the Chinese Remainder Theorem (CRT), the computational complexity of an N-bit exponentiation decreases to two exponentiations of $N/2$-bit. Since the application of the CRT is common in practice, we also provide a time estimate for RSA with CRT. Note that all the running times do not include the preprocessing and a final conversion from the redundant CSA format. However, the required time for the preprocessing and the final conversion (a carry addition) is negligible compared to the overall running time.

Table 3. Running time estimates of a 1024-bit RSA encryption with different multiplier architectures with and without CRT (XVC2000E-6)

Architecture	Frequency	T_{res}	$T_{res,crt}$
Redundant interleaved mult.	69.4MHz	22.7ms	6.1ms
Montgomery mult. with 1 CSA	42.1MHz	37.5ms	9.1ms
Montgomery mult. with 2 CSAs	49.0MHz	32.2ms	9.7ms

4. CONCLUSION

The work at hand presents the first documented implementation of recently proposed hardware architectures for area-time efficient modular multiplication. A thorough investigation of three different multiplier architectures, namely an optimized interleaved multiplier and two variants of a Montgomery multiplier have been made. All architectures have been implemented in VHDL for various different bitlength and were synthetisized for the Xilinx XVC2000E-6 FPGA. The architectures implementing a Montgomery multiplication with one CSA and the interleaved multiplication turn out to be the smallest designs compared to the Montgomery architecture with two CSAs. Regarding the absolute running time of a modular multiplication, the optimized interleaved design is the fastest due to a potentially higher clock frequency. The most efficient multiplier in terms of the product of area and time for all implementations is the optimized interleaved architecture. The implementations represent the proposed asymptotic advantage of the algorithms as described in [4]. A possible application for the presented architectures is, e.g., the acceleration of public-key primitives such as RSA, ElGamal, or elliptic curve operations. With the efficient interleaved multiplier at hand, an RSA public key operation with CRT and with operands of size of 1024 bits takes no longer than 6.1ms on a Xilinx XVC2000E-6 FPGA.

[1] Current protocols such as, e.g., SSL/ TLS predominantly use 1024 bit operands, increasingly 2048 bit operands.

5. REFERENCES

[1] P. Montgomery, "Modular Multiplication without Trial Division," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[2] G. Blakley, "A Computer Algorithm for Calculating the Product $A \cdot B$ modulo M," *IEEE Transactions on Computers*, vol. C-32, no. 5, pp. 497–500, May 1983.

[3] V. Bunimov, M. Schimmler, and B. Tolg, "A Complexity-Effective Version of Montgomery's Algorithm," in *Workshop on Complexity Effective Designs, ISCA'02*, May 2002, http://www.ee.rochester.edu:8080/~albonesi/wced02/.

[4] V. Bunimov and M. Schimmler, "Area and Time Efficient Modular Multiplication of Large Integers," in *IEEE 14th International Conference on Application-specific Systems, Architectures and Processors*, June 2003.

[5] Y. Kim, W. Kang, and J. Choi, "Implementation of 1024-bit modular processor for RSA cryptosystem," School of Electronic and Electrical Engineering, Kyungpook National University, 1370 Sankyok-Dong, Book-Gu, Taegu, Korea, Tech. Rep., 2000, http://www.ap-asic.org/2000/proceedings/10-4.pdf.