FPGA based architecture evaluation of cryptographic coprocessors for smartcards

Hagen Ploog, Dirk Timmermann, *Member, IEEE*

Abstract— In 1996, about 600 million IC-cards were manufactured worldwide. Due to very small die sizes (max. 25 mm^2) smartcards encounter more severe restrictions than conventional coprocessors. In this paper we study coprocessor architectures for very fast but area efficient modular exponentiation (FME) based on Montgomery multiplication. For assessment purposes we developed an evaluation board containing a 8051-microprocessor, a XILINX FPGA and RAM with variable bus width (8b to 32b). We evaluated these architectures in terms of the main design parameters to ease design decisions for smartcards in arbitrary technologies.

Index terms—modulo multiplication, smartcards

1 INTRODUCTION

Today smartcards offer public and private key cryptoalgorithms using a dedicated coprocessor. The best known public key algorithm is RSA. The message *m* is raised by the power of encryption exponent *e* and then reduced by the system modulus *n*, e.g. $c = m^e \bmod n$. Its security depends on the number of bits representing the modulus, growing from 128 in the early days of RSA until 1024 nowadays.
Several high speed cryptochips have been proposed, but none of them considers the special limitations in smartcards as described in [1]. Therefore, these solutions for high speed ciphering can not be used and new architectures have to be developed. In 1985, Montgomery [2] proposed an algorithm for fast modular multiplication of *ab* mod *n*, which is the fastest known modular multiplication algorithm today [3].
The area constraints when developing small coprocessors suggest the use of multiprecision arithmetic. In multiprecision arithmetic large integers *n* (512, 768, or 1024 bits) are broken down into quantities of length *w*, e.g. 8, 16, or 32 bits. The arithmetic operations are implemented recursively, increasing the index from 0 to (*s*-1), with $s = n/w$. Each intermediate result is stored in external RAM, which requires *Tr* clock cycles to perform a read and *Tw* cycles for writing.

The authors are with the Institute of Applied Microelectronics and Computer Science, University of Rostock, Richard-Wagner-Str. 31, 18119 Rostock, Germany
E-mail: {hp,dtim}@e-technik.uni-rostock.de

2 MONTGOMERY MULTIPLICATION

In [5] Montgomery´s method is analyzed under the aspect of finding an optimal realization for software implementation. We have reanalyzed these algorithms to find out which one is optimal with respect to area and speed for the typical hardware available on smartcards, i.e. multiplier, adder, etc. As a result, we have determined the amount of clock cycles used by these methods as a function of *Tw*, *Tr*, *n*, *w*. As these algorithms are described in depth in [4] and [5], we just give a small outline of two of them.

2.1 Method I

We compute the product *ab* using the algorithm given in Figure 1. For speedup reasons we register *b*[*i*] so that there is no read-cycle for *b*[*i*] in the inner loop. The speedup factor varies with the word length *w* employed.
We then perform the reduction using an idea proposed in [4]. The ADD-function repetitively adds *c* (carry) to the temporary value *t*, starting at *j*+*s*, until no further carry is generated. Thorough simulations demonstrated that the number of these repeated addition varies in average from one to four depending on the word length *w* and *n*. The total number of clocks required is given by (1):

$$\#\text{clock\_cycles} = Tr\,(4s^2+3s+1) + Tw(2s^2+4s+1) + \text{ADD}(s) \qquad (1)$$

```
for i = 0 to s-1    {multiplication}
  c := 0
  for j = 0 to s-1
    (c, s) := t[i+j]+a[j]b[i]+c
    t[i+j] := s
  t[i+s] := c
for i = 0 to s-1    {reduction}
  c := 0
  m := t[i]n* mod w
  for j = 0 to s-1
    (c,s) := t[i+j]+m*n[j]+c
    t[i+j] := s
  ADD(t[j+s], c)
```

Fig. 1 : Main algorithm of method I

2.2 Method II

The second method integrates the multiplication and the reduction step. The comparison in [5] shows that this method is not well suited for software implementation, but it is dedicated for use in DSP-like architectures [4]. The number of clock-cycles needed to perform a modular multiplication using this method can be calculated using (2):

$$\#clock\_cycles = Tr\,(4s^2 - 2s + 3) + Tw\,3s \qquad (2)$$

3 Results

Our study demonstrated that the bottleneck for fast modular exponentiation are the number of read/write-cycles and the number of multiplications. Assuming a read-cycle takes one clock and a write-cycle takes 3 clocks (for fully synchronous designs) and that we can do a *n* by *n* bit multiplication within one clock cycle the total number of clock cycles is shown in Table 1. So for fast modular exponentiation the performance depends on the word length of the coprocessor. In addition, wide RAM-structures instead of standard 8bit RAM are required.
As soon as the number of clock cycles for multiplication exceeds the number of clock cycles for writing the critical parameter will change to the total count of multiplications, since read/writes can be hidden, i.e. pipelined.

TABLE 1
wordsize vs. clock cycles (n=1024)

wordsize w [bit]	Number of clock cycles	
	Method I	Method II
8	165764	66435
16	41924	16835
32	10724	4323
64	2804	1139
128	764	315
256	224	95

4 Implementation issues

For evaluation purposes we developed a VHDL model of our architecture, which can be parameterized by *n*, *w*, *Tr*, and *Tw*. After synthesizing using SYNOPSYS 3.5 we mapped the design to a XILINX XC4020, which is connected to a 8051 microcontroller (for control and external communication). We also spent a 32b-RAM to store intermediate results. As multiplication is the heart of all methods, Table 2 gives the number of CLBs required by a *w***w*-multiplier performing the multiplication in *m*-cycles. As can be seen a 32*32 multiplier using two clock cycles would fit but it would not let enough space for the other components, so we choose the (16,2)-model. As we were not using any XILINX-specific library modules we can map our design directly to any other ASIC library.

Pipelining the multiplier could speed up the design at additional cost of gates (while flip-flops are available for free in XILINX FPGAs as each CLB contains two flip-flops, we do not use them in favor of a more technology invariant result). The use of edge triggered RAM (instead of level sensitive) can reduce *Tw* to two clock cycles and increase the speed of operation.

TABLE 2
CLB-usage

Wordsize w [bit]	Cycles m [#]	CLBs [#]	FPGA utilization [%]
16	2	205	26
	4	123	15
	8	82	10
	16	68	8
32	2	688	87
	4	431	54
	8	235	29
	16	162	20
	32	128	16

5 Conclusions

We studied how modern smartcards can perform high security operations. We emulated the main smartcard algorithms in FPGA hardware and quantified the impact of the main ASIC design parameters on overall speed and area. Using these results, an optimal implementation for a given ASIC technology can be chosen. Since we use FPGA based hardware our implementation is not as fast as commercial available coprocessors in terms of clock frequency. But even if we were dealing with an 16bit model, our implementation becomes comparable in cycle count to current coprocessors [6].

References

[1] Everett, D.B., "Smart Card Technology: Introduction To Smart Cards," www.smartcard.co.uk/tech1.htm

[2] Montgomery, P. L., "Modular multiplication without trial division," Mathematics of Computation, vol. 44, no. 170, pp. 519-521, April 1985

[3] Bosselaers, A., Govaerts, R., and Vandewalle, J., "Comparison of three modular reduction functions," Advances in Cryptology – CRYPTO`93, pp. 166-174, Springer-Verlag, 1993

[4] Kaliski Jr., B. S and Dussé, S. R., "A cryptographic library for the Motorola DSP56000," Advances in Cryptology – EUROCRYPT 90, pp. 230-244, Springer-Verlag, 1990

[5] Koc, C. K, Acar, T., and Kaliski Jr., B. S., "Analyzing and comparing Montgomery Multiplication Algorithms," IEEE Micro, pp. 26-33, June 1996

[6] Dhem, J.F., Veithen D. and Quisquater, J.J., "SCALPS : Smart Card Applied to Limited Payment System," IEEE Micro, pp. 42-51, June 1996