

DESIGN OF A COMPACT MODULAR EXPONENTIATION ACCELERATOR FOR MODERN FPGA DEVICES

Timo Alho, Tampere Univ. of Tech., Finland, timo.a.alho@tut.fi
Panu Hämäläinen, Tampere Univ. of Tech., Finland, panu.hamalainen@tut.fi
Marko Hännikäinen, Tampere Univ. of Tech., Finland, marko.hannikainen@tut.fi
Timo D. Hämäläinen, Tampere Univ. of Tech., Finland, timo.d.hamalainen@tut.fi

ABSTRACT

We present a compact FPGA implementation of a modular exponentiation accelerator suited for cryptographic applications. The implementation efficiently exploits the properties of modern FPGAs. The accelerator consumes 341 logic elements, 1 DSP block, and 13 604 memory bits in Altera Stratix EP1S40. It performs modular exponentiations with up to 2250-bit integers and scales easily to larger exponentiations. Excluding pre and post processing time, 1024-bit and 2048-bit exponentiations are performed in 28.03 ms and 212.09 ms, respectively. Due to its compactness, standard interface, and support for different clock domains, the accelerator can effortlessly be integrated into a larger system in the same FPGA.

KEYWORDS: Modular exponentiation, Montgomery multiplication, cryptography, hardware, FPGA, compact.

1. INTRODUCTION

Modular exponentiation of long integers is required in a number of public-key cryptosystems, e.g. RSA, Digital Signature Algorithm (DSA), Diffie-Hellman key exchange protocol, and Secure Remote Password protocol (SRP). Performing such an operation is computationally very expensive. Efficient exponentiation implementations are especially required in wireless and embedded devices with limited processing capabilities as well as in heavily used network servers and firewalls [1][2]. Compared to software implementations on general-purpose CPUs, the exponentiations can be computed significantly more efficiently with a hardware design tailored for the task [3].

Reprogrammable logic circuits, specifically Field Programmable Gate Arrays (FPGA), offer a cost-effective and high-performance hardware alternative to Application Specific Integrated Circuits (ASIC) in low- and mid-volume products. Furthermore, FPGAs are becoming important building blocks in embedded systems in general [3]. According to the recent trend, they are no longer used as single parts of embedded systems but rather as System-on-Chip (SoC) platforms for implementing complete applications. Modern, commercial FPGA devices contain large functional blocks, such as high-speed multipliers, embedded multiport memories, Phase Locked Loops (PLL), and even programmable CPU cores. For cryptographic applications FPGAs offer high performance, possibility to modify and change algorithms in already fielded devices, and potential to share resources through run-time reconfiguration [3].

By carefully mapping designs to the resources of modern FPGA devices, compact and high-performance implementations can be realized. Specifically, the computationally expensive modular exponentiations, which are widely required in cryptographic applications, can significantly benefit from these resources. In this paper we present the design of a compact exponentiation accelerator for modern FPGA devices. Due to the compact size, standard interface, and support for different clock domains, the accelerator can effortlessly be used as a part of a SoC implemented in the same FPGA device. Furthermore, the design can easily be scaled for larger operand sizes by reserving more embedded memory.

2. MODULAR ARITHMETIC

Modular exponentiations are typically calculated using repeated square-and-multiply algorithms with modular reductions in between. The most basic type of these algorithms is the binary modular exponentiation that has two variations, right-to-left and left-to-right, presented as Algorithm 1 and Algorithm 2. More efficient algorithms reducing the number of multiplications and squarings also exist [4] but they are more complex and require more resources as a hardware implementation. For the compact area and simplicity we chose to use the left-to-right binary exponentiation in this work. Compared to the right-to-left algorithm, only one temporary result (P) needs to be stored instead of two (P and Z). The advantage of the right-to-left algorithm is that it allows calculating multiplications and squarings in parallel. However, this requires two separate arithmetic units, increasing the area of the hardware.

Since the exponentiation algorithm consists of repeated multiplications and squarings, an efficient multiplication algorithm is required. Montgomery Multiplication (MM) [5] is a widely used method for performing combined modular reductions and multiplications [3]. The version of MM that we utilize in this work is based on Algorithm 3 [6]. The algorithm calculates $MM(A, B, M) = ABR^{-1} \bmod M$ which is, with a suitable choice for R , significantly more efficient to compute on a typical processor than $AB \bmod M$.

In order to cope with the extra term R^{-1} , we need to transform the operands to a special form called M -residue respect to R , defined as $A^* = AR \bmod M$ [4]. This transformation can be performed with a single MM operation since $MM(A, R^2, M) = AR^2R^{-1} \bmod M = AR \bmod M = A^*$. The inverse transformation can also be performed with MM since $MM(A^*, 1, M) = ARR^{-1} \bmod M = A$. Despite of the conversions, MM is specifically beneficial in modular exponentiation, in which the MM results are repeatedly multiplied using the same modulus. The conversions are required only for the initial input and the final result.

Algorithm 4 presents the digit-serial modification of the MM algorithm used in this work. A similar variation of the algorithm, which is referred to as Finely Integrated Operand Scanning (FIOS), is presented in [11]. The algorithm uses multi-precision integers with k -bit digits as inputs and the intermediate variable c of size $k+1$ bits. The notation $(c, s) \leftarrow x$ denotes that k least significant bits of x are assigned to s and the most significant $k+1$ bits to c . The division by 2^k in Algorithm 3 is satisfied by delaying the inputs b_i and a_j one cycle. In [6] it is shown that the final result $S(n+3)$ is bounded by $2\tilde{M}$, which guarantees that it can be fed back as an input for the next multiplication. Also, the intermediate result $S(i)$ is bounded by $2^k A + \tilde{M}$ and fits into $n+3$ k -bit digits. Therefore, the last step of Algorithm 4 is valid even though c is one bit wider than s .

3. FPGA APPROACHES FOR MONTGOMERY MULTIPLICATION

A typical hardware implementation approach is to partition the MM algorithm so that the n -bit multiplicand B is decomposed into n/k digits, 2^k being the chosen radix, while the multiplier A and modulus M are used in their full widths. However, when performing the additions of Algorithm 3 with these long integers, the carry propagation delay becomes a problem. In order to

Input : Positive integers
 $x, M, E = (e_{n-1}e_{n-2} \dots e_1e_0)_2$
Output : $P = x^E \bmod M$

$P \leftarrow 1, Z \leftarrow x$
for $i = 0$ **to** $n-1$ **do**
 $Z \leftarrow Z \times Z \pmod{M}$
if $e_i = 1$ **then** $P \leftarrow P \times Z \pmod{M}$
end for

Algorithm 1. Right-to-left binary modular exponentiation.

Input : Positive integers
 $x, M, E = (1e_{n-2} \dots e_1e_0)_2$
Output : $P = x^E \bmod M$

$P \leftarrow x$
for $i = n-2$ **to** 0 **do**
 $P \leftarrow P \times P \pmod{M}$
if $e_i = 1$ **then** $P \leftarrow P \times x \pmod{M}$
end for

Algorithm 2. Left-to-right binary modular exponentiation.

Input : $M > 2$ and M is odd.
 Positive integers k and n such that
 $4\tilde{M} < 2^{k(n+2)}$, where $\tilde{M} = (M' \bmod 2^k)M$
 and $M' = -M^{-1} \bmod 2^{k(n+2)}$.
 A and $B = \sum_{i=0}^{n+2} 2^{ik} b_i$, where
 $b_i \in \{0, 1, \dots, 2^k - 1\}$, $b_{n+2} = 0$, $0 \leq A, B < 2\tilde{M}$.

Output : $S(n+3) \equiv ABR^{-1} \bmod M$,
 where $S(n+3) < 2\tilde{M}$,
 and $2^{k(n+2)} R^{-1} \bmod M = 1$.

```

S(0) ← 0
for  $i = 0$  to  $n + 2$  do
   $q_i \leftarrow S(i) \bmod 2^k$ 
   $S(i+1) \leftarrow (S(i) + q_i \times \tilde{M}) / 2^k + b_i \times A$ 
end for

```

Algorithm 3. Montgomery modular multiplication.

Input : $\tilde{M} = \sum_{i=0}^{n+2} 2^{ik} \tilde{m}_i$, $\tilde{m}_i \in \{0, 1, \dots, 2^k - 1\}$,
 $\tilde{m}_{n+2} = 0$, $\tilde{m}_{n+1} = 0$.
 $A = \sum_{i=0}^{n+1} 2^{ik} a_i$, $a_i \in \{0, 1, \dots, 2^k - 1\}$
 $B = \sum_{i=0}^{n+2} 2^{ik} b_i$, $b_i \in \{0, 1, \dots, 2^k - 1\}$, $b_{n+2} = 0$,
 where $\tilde{M} = MM'$, $M > 2$, M is odd,
 $M' = -M^{-1} \bmod 2^k$, and $0 \leq A, B < 2\tilde{M}$.

Output : $S(n+3) \equiv ABR^{-1} \bmod M$,
 where $S(n+3) < 2\tilde{M}$ and $2^{k(n+2)} R^{-1} \bmod M = 1$.

```

S(0) ← 0
for  $i = 0$  to  $n + 2$  do
   $q_i \leftarrow s(i)_0$ 
   $(c, -) \leftarrow s(i)_0 + q_i \times \tilde{m}_0$ 
  for  $j = 0$  to  $n + 1$  do
     $(c, s(i+1)_j) \leftarrow s(i)_{j+1} + q_i \times \tilde{m}_{j+1} + b_j \times a_j + c$ 
  end for
   $s(i+1)_{n+2} \leftarrow c$ 
end for
where  $S(i) = \sum_{j=0}^{n+2} 2^{jk} s(i)_j$ ,
 $s(i)_j \in \{0, 1, \dots, 2^k - 1\}$  and  $c \in \{0, 1, \dots, 2^{k+1} - 1\}$ 

```

Algorithm 4. Digit-serial Montgomery multiplication.

overcome this, a redundant number system can be used [7][8] or the additions can be performed in a systolic array [9][10]. Typically, these methods result in high performance but also in high consumption of computational resources in FPGAs as each addition is performed fully parallel.

As an alternative partitioning supporting compact designs, all the MM inputs can be decomposed into n/k digits and the computations performed a digit at a time (digit-serial). The method resembles a software approach in which the intuitive choice for radix 2^k is the word size of the processor [11]. With this partitioning the amount of FPGA resources dedicated for computation can be decreased at the expense of performance. Ref. [14] presents an implementation in which the multiplicand is in the radix-2 form, the multiplier is broken up into k -bit digits, and the computations are performed in the serial fashion.

Ref. [12] presents an instruction set extension for modular arithmetic that enables computing the operation $x \times y + z + w$ in one clock cycle and thus speeds up the digit-serial loop of Montgomery's algorithm. The paper reports considerable speedup on a MIPS32-compatible CPU when the instruction set extension is used. In addition, the approach scales well to different key sizes as well as algorithms. The method can also easily be adapted to FPGA implementations based on soft-core CPUs, such as NIOS II of Altera [13]. In this paper we utilize a related approach in our MM data path.

According to our knowledge, the only modular exponentiation implementation efficiently exploiting the embedded multipliers of a modern FPGA device is reported in [7]. The implementation uses a large number of multipliers in parallel on a Xilinx FPGA, resulting in a high-performance design. However, unlike in our implementation, the amount of consumed resources is very high and thus the implementation allows integrating only very little other functionalities into the same FPGA.

Clock cycles for a modular multiplication	$(n+3)(n+4)$
Number of modular multiplications for modular exponentiation	$(l+p)$, where l is the bit length of the exponent and p is the number of ones in its binary representation
Required pre processing	given $x < M$, $M < 2^{kn}$, $R = 2^{k(n+2)}$ calculate: $xR = xR \bmod M$ and $\tilde{M} = MM'$
Required post processing	given $P \equiv x^e R \bmod M$ calculate: $x^e \bmod M = PR^{-1} \bmod M$

Table 1. Summary of accelerator features.

4. ACCELERATOR ARCHITECTURE

This section describes the architecture of our modular exponentiation accelerator. As the accelerator is intended to be used as a part of a larger SoC containing also a general-purpose processor, it only computes the most time consuming parts of the modular exponentiation. That is, the pre and post transformations must be performed outside the accelerator. The features of our accelerator design are summarized in Table 1.

4.1. FPGA Platform

We used Altera Stratix EP1S40 as the target device. It is a modern FPGA containing a large number of programmable Logic Elements (LE) as well as RAM and Digital Signal Processing (DSP) blocks. In our design the RAM blocks are used as Dual-Port RAM (DPRAM) and shift registers, and the DSP blocks are configured to a multiplier-adder mode. Similar building blocks can also be found in the Stratix II and Cyclone II devices of Altera as well as the Virtex II, Virtex 4, and Spartan 3 devices of Xilinx.

4.2. Exponentiation Architecture

Figure 1 depicts the high-level architecture of the accelerator. Initially, the accelerator inputs are stored into the DPRAMs xR -ram, M -ram, and E -ram as k -bit digits. In addition, the number of iterations for a single multiplication (n) and the total number of bits in the exponent are passed to the control logic. During the exponentiation, $s1$ -ram and $s2$ -ram are used as temporary storages, one storing the intermediate result ($S(i)$ of Algorithm 4) during multiplication and the other one holding the result of the previous multiplications (P of Algorithm 2).

The exponentiation control logic, Exp control, manages the four multiplexers. Output of p_1 is always selected to be the result of the previous multiplication (P) and the output of p_2 to be the intermediate result of the active multiplication ($S(i)$). The multiplexers p_a and p_b select which multiplication, $xR \times xR$, $xR \times P$, or $P \times P$, is calculated. The registers before the multiplexers p_a and p_b are used for delaying b_i and a_j by one clock cycle.

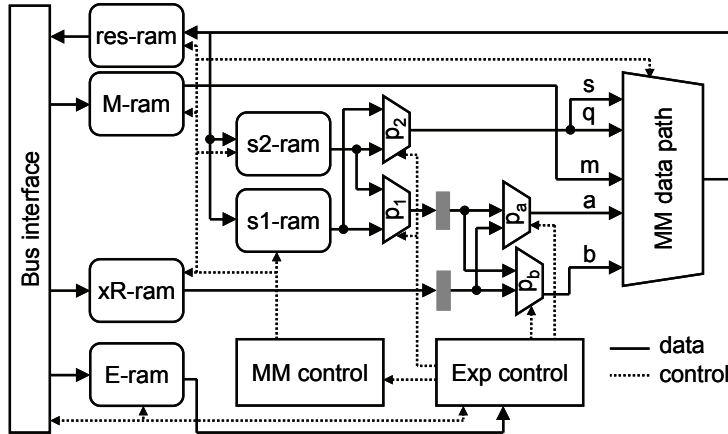


Figure 1. Accelerator architecture.

4.3. Modular Multiplier

For MM the integers A , B , and the pre-calculated \tilde{M} are broken up to operands of length k -bits as required by Algorithm 4. The modular multiplier entity consists of a control state machine, *MM control*, and a data path performing the computations of the body of the inner loop in Algorithm 4. *MM control* counts the values of the loop variables i and j and uses them for addressing the operand memories. In addition, it maintains a delayed version of the counter j for addressing *res-ram*, *s1-ram*, or *s2-ram* to write back the results in parallel with operand reading. At the beginning of the each iteration of i , MM control loads the values of $q_i = s(i)_0$ and b_i into the data path input and clears the intermediate variable c . During the first iteration of i the data path input is cleared. The data path is shown in Figure 2(a). The result of the data path addition always fits into $2k+1$ bits.

4.4. Mapping to FPGA

Figure 2(b) depicts the mapping of the data path to our target FPGA device, data path being 18 bits wide. The intermediate variable c is divided into two registers, r and r' , and always computed during one clock cycle. The data path is pipelined into five stages for minimizing the combinatorial logic delays. However, the pipelining does not cause any stalls in the operation, since the only data dependencies between back-to-back operations are due to the intermediate variable c .

Each DSP block in EP1S40 can be configured to support either eight 9-bit, four 18-bit, or one 36-bit multipliers. In addition, the DSP blocks contain various other elements such as registers and adder/accumulator blocks. We chose to use the data path width of 18 bits, which allows us to perform two multiplications, sum the results of the multiplications, and pipeline the calculations using only half of the resources of one DSP block. The shift register for the input s is mapped into a RAM block and the two additional adders and the rest of the registers into LEs.

The operand, intermediate, and result memories are mapped into RAM blocks configured to DPRAM mode with one read and one write port. This enables simultaneous writing of results and reading of the next operands. The DPRAMs of the accelerator interface (*E-ram*, *xR-ram*, *M-ram*, and *res-ram*) are also configured to support two clock domains, allowing running the accelerator at a higher clock speed than the rest of the system. The support for different clock domains explains also the extra *res-ram*, which could otherwise be replaced by *s1-ram* or *s2-ram*.

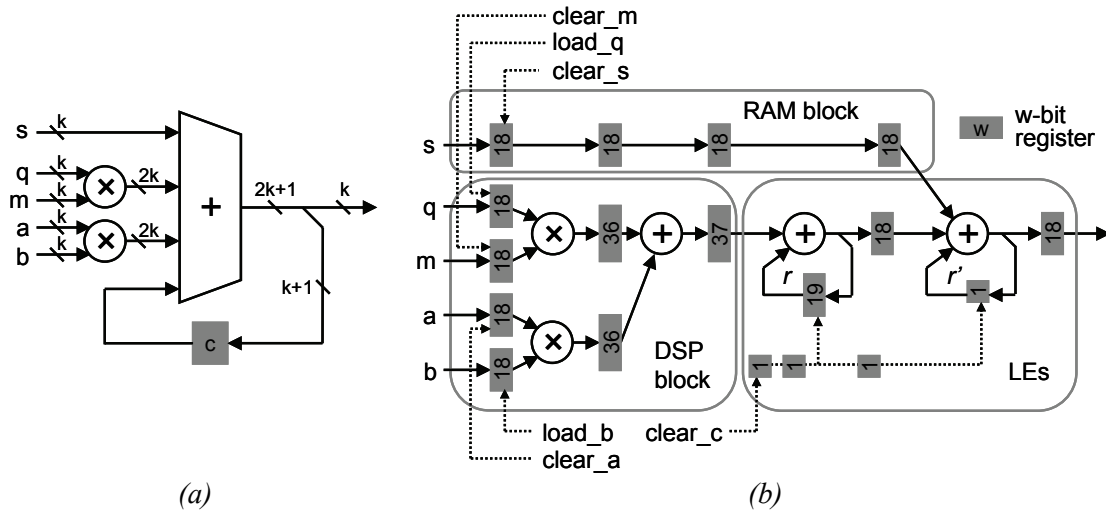


Figure 2. Montgomery multiplication data path: (a) architecture and (b) mapping to EP1S40.

5. RESULTS AND COMPARISON

The synthesis results of our accelerator implementation including a fully functional, standard bus interface to NIOS II are summarized in Table 2. The accelerator was described in VHDL and synthesized using Altera Quartus II 5.0. The memory sizes were chosen to be 128 x 18 bits, which supports computing full exponentiations with up to 2250-bit operands. The design can be scaled for longer exponentiations simply by increasing the amount of memory. As can be seen, our accelerator consumes only a very small amount of resources on the target FPGA.

For the execution time comparison, we measured that a full 1024-bit exponentiation takes over a second in the 32-bit NIOS II at 100 MHz with a C-language OpenSSL implementation. By carefully hand tuning the software for NIOS II, the performance could be improved, but not to the level of our accelerator. Ref. [15] presents a highly optimized software implementation in a DSP processor. They report that a 1024-bit RSA signing can be performed in 11.7 ms at 200 MHz by exploiting the Chinese Remainder Theorem (CRT), which makes direct execution time comparisons impossible. Compared to our data path, the DSP processor data path would also very likely map less cost-efficiently to the FPGA resulting in larger resource consumption.

Table 2 compares our implementation with other MM-based modular exponentiation FPGA implementations relevant to this work. It should be noted that a fair comparison is difficult as different FPGA technologies are used. All the reference implementations support 1024-bit exponentiations. One Xilinx *slice* corresponds roughly to two Altera LEs. Ref. [9] presents a radix-16 systolic array design mapped carefully to slices for high performance. A similar design for radix-2 with tuned control on an Altera device is reported in [10]. Ref [14] is a serial implementation that exploits the characteristics of slices for compact size. The execution times in Table 2 exclude the pre and post processing, which varies among the implementations. We have calculated the full execution time for [14] since it has not explicitly been reported.

The comparison shows that the consumption of the general-purpose resources (LEs/slices) in our implementation is considerably lower than in the reference implementations. Whereas [7] utilizes 62 multipliers, we only need two, residing in a single DSP block. Our execution time is significantly better than that of the most compact reference [14]. Compared to [7], [9], and [10], the execution time of our design for the 1024-bit exponentiation is longer but we also support longer exponentiations with a single implementation.

If the target application is parallelizable, its performance can be further improved by

Design	FPGA device	Logic blocks	Memory bits	DSP elements	Max. freq. [MHz]	Op. length [bits]	Execution time [ms]
Ours	Altera Stratix EP1S40	341 LEs	13 604 ⁽¹⁾	1 DSP ⁽²⁾	198	1024	28.03
		(0.8%)	(0.4%)	(7.1%)		2048	212.09
Ref. [7]	Xilinx Virtex-II XC2V3000	14 334 slices	- ⁽³⁾	62 18-bit multipliers	90	1024	2.33
Ref. [9]	Xilinx XC40250XV	6 633 CLBs ⁽⁴⁾	- ⁽³⁾	-	45	1024	11.95
Ref. [10]	Altera EPXA10	13 960 LEs	4096	-	63	1024	25.06
Ref. [14]	Xilinx Virtex-E 2000-8	1 188 slices	- ⁽³⁾	-	86	1024	208

⁽¹⁾ Memory bits are mapped to 11 M512 (2.9%) and four M4K (2.2%) RAM blocks.

⁽²⁾ Half of the resources of the DSP block are free, including two 18-bit multipliers.

⁽³⁾ A slice can be configured to two 16 x 1 bit memories. Thus, dedicated memory blocks are not necessary.

⁽⁴⁾ One XC40250XV Configurable Logic Block (CLB) equals to one Virtex slice.

Table 2. Results and comparison (percentages of total resources in parenthesis).

utilizing multiple parallel accelerators. This is the case e.g. with RSA, in which the modulus M is a product of two or more primes [4]. When the primes are known, the exponentiation can be divided into smaller, parallel exponentiations using CRT.

6. CONCLUSION

In this paper we presented the design of a compact modular exponentiation accelerator suited for a number of cryptographic applications. By mapping the design to the resources of modern FPGA devices, we achieved high performance with very low resource consumption. In addition, our implementation is effortlessly scalable to exponentiations with larger operands. Due to its compact size, standard bus interface, and support for different clock domains, the accelerator can easily be integrated into a larger SoC implemented in the same FPGA.

7. REFERENCES

- [1] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: design challenges," *ACM Trans. Embedded Comp. Systems*, vol. 3, no. 3, pp. 461-491, Aug. 2004.
- [2] L. Zhao, R. Iyer, S. Makineni, L. Bhuyan, "Anatomy and performance of SSL processing," in *Proc. IEEE Int. Symp. Performance Analysis of Systems and Software*, March 2005, pp. 197-206.
- [3] T. Wollinger, J. Guajardo, and C. Paar, "Security on FPGAs: state-of-the-art implementations and attacks," *ACM Trans. Embedded Comp. Systems*, vol. 3, no. 3, pp. 534-574, Aug. 2004.
- [4] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*, Boca Raton (CA): CRC Press, 1997.
- [5] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519-521, Apr. 1985.
- [6] H. Orup, "Simplifying quotient determination in high-radix modular multiplication," in *Proc. 12th Symp. Computer Arithmetic*, July 1995, pp. 193-199.
- [7] S. Tang, K. Tsui, and P. Leong, "Modular exponentiation using parallel multipliers," in *Proc. IEEE Int. Conf. Field-Programmable Technology*, Dec. 2003, pp. 52-59.
- [8] S. E. Elridge and C. D. Walter, "Hardware implementation of Montgomery's modular multiplication algorithm," *IEEE Trans. Comput.*, vol. 42, no. 6, pp. 693-699, June 1993.
- [9] T. Blum and C. Paar, "High radix Montgomery modular exponentiation on reconfigurable hardware," *IEEE Trans. Comput.*, vol. 50, no. 7, pp. 759-764, July 2001.
- [10] P. Hämäläinen, N. Liu, M. Hännikäinen, and T. D. Hämäläinen, "Acceleration of modular exponentiation on system-on-a-programmable-chip", in *Proc. IEEE Int. Symp. System-on-Chip*, Nov. 2005, pp. 14-17.
- [11] Ç. K. Koç and B. S. Kalinski Jr., "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, pp. 26-33, June 1996.
- [12] J. Großschädl, "Instruction set extension for long integer modulo arithmetic on RISC-based smart cards," in *Proc. 14th Symp. Computer Architecture and High Performance Computing*, Oct. 2002, pp. 13-19.
- [13] Altera Corporation, "Nios II Embedded Processor," [Online document], [cited 2005 Nov 21], Available at HTTP: <http://www.altera.com/products/ip/processors/nios2/ni2-index.html>
- [14] A. Mazzeo, L. Romano, and G. P. Saggese, "FPGA-based implementation of a serial RSA processor," in *Proc. Design, Automation and Test in Europe*, 2003, pp. 582-587.
- [15] K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara, "Fast implementation of public-key cryptography on a DSP TMS320C6201," in *Proc. 1st Int. Workshop Cryptographic Hardware and Embedded Systems (CHES 1999)*, 1999, pp. 61-72.