CUSTOM FPGA CRYPTOGRAPHIC BLOCKS FOR RECONFIGURABLE EMBEDDED NIOS PROCESSOR

Miloš DRUTAROVSKÝ, Martin ŠIMKA
Department of Electronics and Multimedia Communications, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Park Komenského 13, 041 20 Košice, Slovak Republic,
E-mail: {Milos.Drutarovsky, Martin.Simka}@tuke.sk

SUMMARY

This paper introduces two custom blocks for Nios reconfigurable embedded processor implemented on Altera Field Programmable Gate Arrays (FPGAs). When operations like modular multiplication and modular exponentiation of long integers or other complex algebraic functions are performed on a general-purpose processor they usually consume a lot of processor resources and execution times are not satisfactory. A solution of this problem lies in development of custom coprocessors. The algebraic coprocessor for Montgomery Multiplication (MM) makes possible a fast execution of modular multiplication with large numbers that can be used in several public key cryptographic algorithms. A True Random Number Generator (TRNG) enhances an application of the Nios processor in cryptographic protocols. Until now only few implementations of TRNG on FPGA have been presented in literature. We describe a custom TRNG implementation based on a recently proposed method that reliably extracts intrinsic randomness from low-jitter clock signals synthesized by on-chip FPGA analog PLLs. Both peripheral blocks are connected to the Nios processor through Altera Avalon bus that directly supports scalable connection and different sources of the clock signal. In this way we can optimally use the resources of Altera FPGAs and implement designs customisable with regard to available area resources and desired level of security or timing constraints. Proposed solutions significantly improve security and computational power of System on a Chip (SoC) embedded cryptographic applications based on the Nios processor.

Keywords: *cryptographic algorithm, SoC, embedded coprocessor, TRNG, IP blocks, Montgomery multiplication*

1. INDRODUCTION

The recent rapid increase of the devices complexity and decreasing time-to-market are forcing new ways of system design. Capable solution of this problem lies in application of System on a Chip (SoC) based on pre-designed Intellectual Property (IP) cores and building blocks. IP blocks optimised for target devices, and with standardised interfaces offer fast solution without need for long-term development of often-used parts of designs.

A cryptographic IP block either executes whole cryptographic algorithms, or, as it is in our case, offers significant improvement of computational power or security level of the system. The coprocessor for Montgomery Multiplication (MM) enables a faster execution of large numbers of multiplication needed for several cryptographic public key algorithms. The True Random Number Generator (TRNG) enhances an application of the Nios processor [1] in cryptographic protocols.

Almost all cryptographic protocols require generation and use of secret values that must be unknown to attackers [2]. For example, TRNGs are required to generate public/private key pairs for asymmetric (public key) algorithms including RSA, DSA, and Diffie-Hellman [2]. Unfortunately, standard processors (including synthesizable Nios processor from Altera) are not able to generate true random numbers, as they are deterministic systems. The only way how to get true random numbers, hence true security for cryptographic systems, is to build a generator based on a random physical phenomenon.

Current modern Field Programmable Gate Arrays (FPGAs) provide an alternative hardware platform even for system-level integration of embedded symmetric and asymmetric cryptographic algorithms, but not for high quality TRNGs. Most hardware TRNGs follow unpredictable natural processes, such as thermal (resistance or shoot) noise or nuclear decay. Since these principles of random numbers generation require additional external devices, they cannot be embedded completely inside the FPGAs. Such TRNGs are not compatible with modern FPGAs and do not provide a SoC solution.

This paper describes a custom TRNG IP block based on a recently proposed method [3] that uses on-chip analog PLLs included in Altera APEX FPGAs [4]. The proposed method reliably extracts intrinsic randomness from low-jitter clock signals synthesized by on-chip APEX analog PLLs. The TRNG is developed as a custom IP building block optimised for Nios processor and provides significantly higher system level security for complete embedded cryptographic SoC designs.

Public key algorithms, such as RSA, Diffie-Hellman key exchange algorithm or Elliptic curve cryptography use modular multiplication and modular exponentiation of long integers (up to 2048 bits for RSA). When these operations are performed on a general-purpose processor they consume a lot of time and processor resources. A solution of this problem lies in development of a custom coprocessor for modular long integer multiplication. Chosen MM algorithm provides certain advantages in the implementation of modular multiplication.

The coprocessor has a scalable structure and uses effectively the resources of FPGA. Thanks to the implementation on FPGA we can obtain a very flexible and secure solution.

The paper is organised as follows: in Section 2 we present briefly the Nios processor and its features important to interface of the coprocessor and TRNG. In Section 3 we give a brief description of the TRNG principle and in Section 4 the MM algorithm is presented. In the Section 5 we deal with a way of an implementation of the blocks and their interface to the Nios processor, improvements of the MM coprocessor's structure, and speed and area results of implementations in Altera FPGA devices. In the Section 6 we present our conclusions and propose possible ways of future development.

2. AN OVERVIEW OF EMBEDDED NIOS PROCESSOR

The Nios CPU [1] is a pipelined general-purpose RISC processor that is generated by proprietary Altera VHDL generator (SOPC Builder) and can be synthesised and embedded in all recent Altera FPGAs. The Nios supports both 32-bit and 16-bit architectural variants. Both variants use 16-bit instructions. The principal features of the Nios instruction set architecture are:

Large, windowed register file – Nios implementations can include up to 512 internal general-purpose registers. The compiler uses the internal registers to accelerate subroutine calls and local variable access.

Simple, complete instruction set – both 32-bit and 16-bit Nios variants use 16-bit wide instructions. 16-bit instructions reduce code size and instruction-memory bandwidth.

Powerful addressing modes – the Nios instruction set includes Load and Store instructions that the compiler uses to accelerate structure access and local-variable (stack) access.

Extensibility – users can incorporate custom logic directly into the Nios arithmetic logic unit. The automatically-generated software development kit includes macros for accessing custom instruction hardware for C and assembly-language programs.

Existing Nios peripherals (e.g. UART, Timer...) as well as new custom peripherals can be connected through an Avalon bus [5]. Avalon is a simple bus architecture designed for connecting on-chip processor(s) and peripheral together into a SoC. The principal features of the Avalon bus are:

Simplicity – provides an easy to understand protocol with a short learning curve.

Optimized resource utilization – conserves Logic Elements (LEs) inside the FPGAs.

Synchronous operation – integrates well with other user logic that coexists on the same FPGA, while avoiding complex timing analysis issues.

An example of a SoC structure with user-defined custom peripherals is shown in Fig. 1. Custom peripherals can be included in a system block generated by SOPC Builder what brings optimised implementation of the whole system, or they can be fed to Avalon bus as pre-synthesised blocks. The Fig. 1 depicts a simplified clock distribution, where F_1 is a primary clock signal used for clocking of the Nios and all of peripherals, and F_2 denotes additional clock signals whose function is explained later.

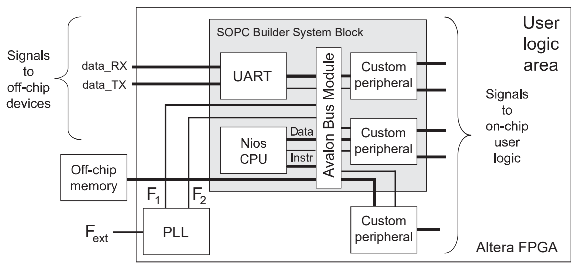

Fig. 1 Example of a system module integrated with custom peripherals into an Altera FPGA

3. BASIC PRINCIPLE OF TRNG IP BLOCK

Several well-known TRNGs require additional off-chip component which decreases a security level of the system. In addition, some practical implementations based on free running oscillators are not secure enough for cryptographic applications as the entropy increase is not sufficiently high, so that the output of the generator can be predicted [6]. In next we shortly describe an implementation completely embedded in Altera FPGAs [3].

Modern Altera FPGAs use reconfigurable analog on-chip PLLs to increase performance and to provide on-chip clock-frequency synthesis. The basic principle behind our method is an extraction of a randomness from the jitter of the clock signal synthesized in the embedded analog PLL [3] (illustrated in Fig. 2).

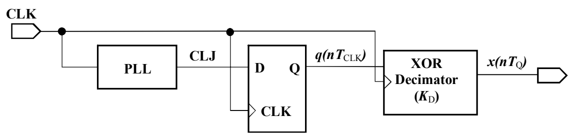

Fig. 2 Basic principle of randomness extraction from the low-jitter clock signal

The jitter is detected by the sampling of a reference clock signal *CLK* (F_1 in Fig. 1) with frequency F_{CLK} using rationally related clock signal *CLJ* (F_2 in Fig. 1) synthesized in the PLL with the frequency:

$$F_{CLJ} = F_{CLK}\frac{K_M}{K_D} = F_{CLK}\frac{m}{n \times k}, \tag{1}$$

where m and k can range from 2 to 160, and n ranges from 1 to 16 [4]. Let values of multiplication factor K_M and division factor K_D be relative primes, so

$$\mathrm{GCD}(K_M, K_D) = 1, \tag{2}$$

where GCD is an abbreviation of Greatest Common Divisor. Equation (2) ensures that the maximum guaranteed distance between the closest edges of *CLK* and *CLJ* (denoted as $\mathrm{MAX}(\Delta T_{\min})$) over the period:

$$T_Q = K_D T_{CLK} = K_M T_{CLJ} \tag{3}$$

is equal to [3]:

$$\mathrm{MAX}(\Delta T_{\min}) = T_{CLK} \frac{\mathrm{GCD}(2K_M, K_D)}{4K_M} = \\ = T_{CLJ} \frac{\mathrm{GCD}(2K_M, K_D)}{4K_D}. \tag{4}$$

By proper choosing of K_M, K_D and T_{CLK} it is possible to guarantee that $\mathrm{MAX}(\Delta T_{\min}) < \sigma_{jit}$, where σ_{jit} is the RMS (root mean square) value of PLL intrinsic jitter. According to [7] and [8], an intrinsic jitter can be approximated by Gaussian distribution and $\sigma_{jit} \geq 15\mathrm{ps}$. Moreover, the proposed method is insensitive to an overall jitter characteristic as far as an intrinsic PLL jitter is included.

In general, the obtained bits are statistically biased random bits. Acceptable TRNG should produce output bits with equal probability. A common way how to reduce the statistical bias is to use a XOR corrector. Non-overlapped pairs of bits are XORed together. In this way, a uniform distribution of generated true random bits with period T_Q (3) is guaranteed [3].

4. SCALABLE WORD-BASED MONTGOMERY MULTIPLICATION ALGORITHM

Basic mathematical operation used by RSA is modular exponentiation [2]:

$$Z = X^E \bmod M, \tag{5}$$

that a binary or general m-ary method can break into a series of modular multiplications. All of these computations have to be performed with large k-bit integers (typically $k \in \{1024, 2048\ldots\}$).

The well-known MM algorithm [9] speeds-up modular multiplication and squaring required for exponentiation (5). It computes the MM product for k-bit integers X, Y as

$$MM(X, Y) = XYR^{-1} \bmod M, \tag{6}$$

where $R = 2^k$, and M is an integer in the range $2^{k-1} < M < 2^k$ such that $\mathrm{GCD}(R, M) = 1$.

Basic MM (6) can be used for efficient computation of (5) by the standard Montgomery exponentiation algorithm [2]. The starting point of this algorithm is the MM. The faster MM is performed, the faster exponentiation process will be accomplished.

For implementation of MM on FPGAs we use a modified version of MM – Multiple Word Radix-2 Montgomery Multiplication (MWR2MM) algorithm [10]. MWR2MM with word width w performs bit-level computations and produces word-level outputs. The scalability and word-orientation of the algorithm makes possible to implement the scalable MM coprocessor. For operands with a k-bit precision, $e = \lceil k/w \rceil$ words are required. MWR2MM algorithm scans word-wise operand Y (multiplicand) and M, and bit-wise operand X (multiplier). Depending on timing requirements, area limitations and connected version of Nios processor we can choose the word width $w = 8, 16, 32, 64\ldots$

For description of the MWR2MM algorithm we use the following denominations:

$$M = \left(M^{(e-1)}, \ldots, M^{(1)}, M^{(0)}\right), \\ Y = \left(Y^{(e-1)}, \ldots, Y^{(1)}, Y^{(0)}\right), \\ X = \left(x_{k-1}, \ldots, x_1, x_0\right). \tag{7}$$

The concatenation of vectors A and B is represented as (A, B). A particular range of bits in a vector A from position i to position j is represented as $A_{j..i}$. The bit position of the k^{th} word of A is represented as $A_i^{(k)}$. Then a fragment of the MWR2MM algorithm can be described as:

```
for $i = 0$ to $k-1$ do
  $\left(C, S^{(0)}\right) = x_i Y^{(0)} + S^{(0)}$
  if $S_0^{(0)} = 1$ then
    $\left(C, S^{(0)}\right) = C + S^{(0)} + M^{(0)}$
    for $j = 1$ to $e-1$ do
      $\left(C, S^{(j)}\right) = C + x_i Y^{(j)} + M^{(j)} + S^{(j)}$
      $S^{(j-1)} = \left(S_0^{(j)}, S_{w-1..1}^{(j-1)}\right)$
    $S^{(e-1)} = \left(C, S_{w-1..1}^{(e-1)}\right)$
```

The computations are performed with precision w bits, what means that they are independent on the operands precision k. What varies is the number of loop iterations needed to compute the result of modular multiplication.

Due to the mutual dependency of the operations inside the j loop the parallel execution is not possible. This is not the case of the instructions in different i loops which can be executed in parallel. This fact makes possible to implement even more flexible and faster scalable pipelined architecture.

5. HARDWARE MAPPING TO THE ALTERA FPGA

The Altera Nios Embedded Processor Development Board with APEX EP20K200 [11] has been used for the implementation of the proposed IP blocks. The board and the provided software form powerful development tools for implementation and testing of the designs. Thanks to the fact that the custom blocks are described by VHDL code we obtained parameterised designs that can be modified according to the desired time and area limits. All presented results have been obtained by using the Altera Quartus II ver.2.2, Altera Nios 2.2, and FPGA Advantage 5.3 tools.

5.1 TRNG

The same Nios development board was also used in [8] for reference PLL measurements so we can expect that jitter characteristics presented in [8] can be directly applied to our design. The configuration of the TRNG block is depicted in Fig. 3. The board features a PLL-capable APEX EP20K200-2X with four on-chip analog PLLs. We used the same PLL organisation as described in the previous design of TRNG [3], the values of parameters K_M and K_D were chosen according to the connection of the TRNG to the Nios processor. The two on-chip PLLs shown in Fig. 3 have been used for generating CLJ and CLK signals.

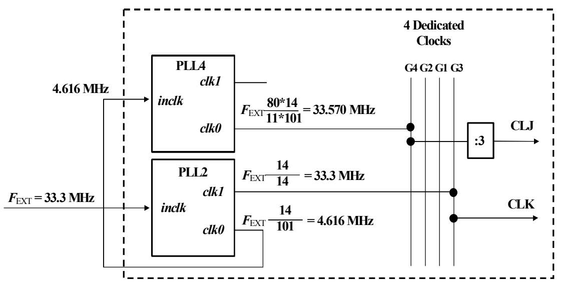
4 Dedicated Clocks
G4 G2 G1 G3
PLL4
clk1
inclk
clk0
4.616 MHz
$F_{EXT}\frac{80*14}{11*101} = 33.570$ MHz
:3
CLJ
PLL2
clk1
inclk
clk0
$F_{EXT} = 33.3$ MHz
$F_{EXT}\frac{14}{14} = 33.3$ MHz
$F_{EXT}\frac{14}{101} = 4.616$ MHz
CLK

Fig. 3 Actual PLL configuration used in the proposed TRNG IP block

The external clock source was $F_{EXT} = 33.3\,\text{MHz}$, on-chip synthesized clocks were $F_{CLK} = 33.3\,\text{MHz}$ and $F_{CLJ} = F_{CLK}(1120/3333) \approx$ 11.9 MHz (can be modified according to other requirements). According to (4) these parameters ensure that $\text{MAX}(\Delta T_{\min}) \approx 6.7\text{ ps} < \sigma_{\text{jit}}$. The output bit-rate of the TRNG is $1/T_Q \approx 10000$ bits/s.

Example of resource requirements of 16-bit and 32-bit Nios with corresponding 16-bit and 32-bit TRNG implementations are shown in Tab.1 (LE — Logic Element is the basic building block of Altera FPGAs).

Name of the block	# LEs	% of total capacity
Nios-16	1140	14
Nios-32	1480	18
UART	170	2
TRNG-16	150	2
TRNG-32	200	2.5
ALL	1669	20
ALL + interface logic	2143	26

Tab. 1 Results of TRNG mapping to Altera APEX EP20K200-2X

5.1.1 TRNG interface

The TRNG peripheral is connected to Avalon bus as a standard memory mapped peripheral. TRNG can be accessed from the Nios processor through three custom registers. Data (read only) and Control/Status registers (write/read access) are mapped into two memory locations shown in Fig. 4.

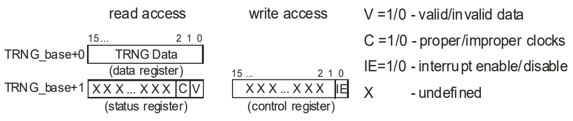

Fig. 4 Data and Control/Status registers of 16-bit TRNG

TRNG can be accessed by the standard memory pooling as well as by using an interrupt service request that can be enabled by an application program. The exact position of a TRNG_base address and TRNG_IRQ can be configured by the SOPC Builder.

5.1.2 TRNG statistical testing

Statistical testing is very important for random number generators dedicated to usage in cryptographic algorithms. More detailed description of applying NIST statistical test suite [12] on presented TRNG can be found in [3].

While in [3] the generator is the only one block implemented on the device, in this paper we present configuration with TRNG connected to the Nios processor. More complex design and logic clocked by different clock sources can affect the jitter distribution. The aim of tests was to detect possible deviations of statistical characteristics. After testing we have obtained similar results of statistical tests as in [3]. This confirms the fact that the reliability of TRNG and the randomness of its output are independent on the design complexity, FPGA configuration, and internal circuit activity.

5.2 Scalable MM coprocessor

The implemented radix-2 MM coprocessor is a unit realizing the MWR2MM algorithm. Input values (X, Y, and M) as well as the result S obtained by the coprocessor are stored in coprocessor's memory. Since input and output values can have high precision (1024 bits or more), we have decided to store inputs and intermediate results in Embedded Memory Blocks (EMBs). EMBs can implement various types of memory with one block size 2048 bits [4]. Therefore, the coprocessor has been split into two blocks: a Radix-2 Montgomery multiplier and a dual port data memory.

In order to reduce storage and arithmetic hardware complexity, data path of the MM coprocessor uses X, Y, and M in a standard non-redundant form. The internal sum S is received and generated in the redundant Carry-Save form [13]. Therefore, the bit resolution of the sum S is effectively doubled and we use denomination ${}_1S$ and ${}_2S$ for it. Then ${}_1S_2^{(j)}$ represents the second bit of the j^{th} word of the first part of S (see Fig. 5). The advantage of Carry-Save form is faster addition process without using the carry logic especially for $w > 16$.

The data path design is based on the structure presented in [10]. The MM unit consists of two layers of carry-save adders as it is shown in Fig. 5 for $w = 3$. Input c represents latched value t that is the least significant bit of the value $S^{(0)} + x_i Y^{(0)}$ and is used to control the addition of M.

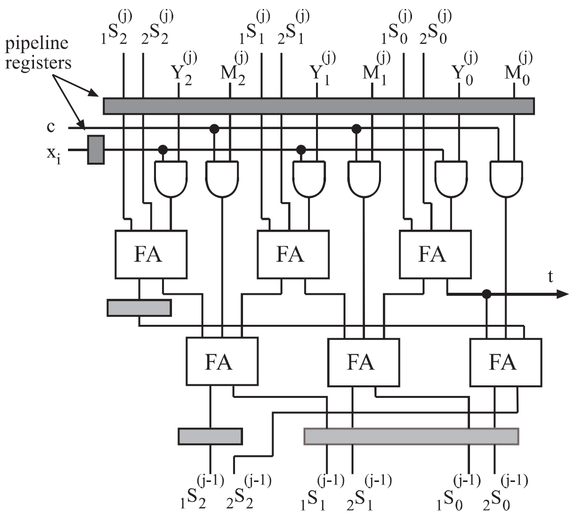
pipeline registers
${}_1S_2^{(j)}$ ${}_2S_2^{(j)}$ ${}_1S_1^{(j)}$ ${}_2S_1^{(j)}$ ${}_1S_0^{(j)}$ ${}_2S_0^{(j)}$
$Y_2^{(j)}$ $M_2^{(j)}$ $Y_1^{(j)}$ $M_1^{(j)}$ $Y_0^{(j)}$ $M_0^{(j)}$
c
x_i
FA
t
${}_1S_2^{(j-1)}$ ${}_2S_2^{(j-1)}$ ${}_1S_1^{(j-1)}$ ${}_2S_1^{(j-1)}$ ${}_1S_0^{(j-1)}$ ${}_2S_0^{(j-1)}$

Fig. 5 Structure of the MM unit for $w = 3$ (FA – Full Adder)

The most important parameter influencing the overall multiplier speed is the memory access time. Since during one cycle previous results ${}_1S$ and ${}_2S$ have to be read and current results from the last stage have to be written to the same memory, we have chosen the EMB configuration as a dual port RAM using the Altera-specific function *lpm_ram_dp* from the Library of Parameterized Modules (LPM). The memory with registered input/output data has been found as the faster implementation.

The pipelined version of the coprocessor has been implemented using changes published in [14]. The data path is organized as a pipelined structure of several MM units shown in Fig. 6. A distribution of signal X to the units has been found as a critical path in the time analysis. Therefore this part has been redesigned by adding an 1-bit register for x_i into the MM unit (see Fig. 6). Thanks to this modification there are no limitations for a number of units as in [15].

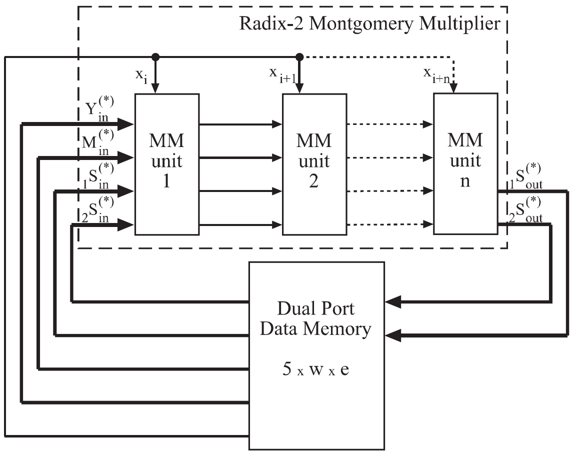

Fig. 6 Block diagram of the MM coprocessor data path

The maximum degree of parallelism that can be obtained with this architecture is found as:

$$n_{\max} = \left\lceil \frac{e}{2} \right\rceil. \tag{8}$$

The number 2 in denominator expresses the number of clock cycles after which the output of the MM unit is valid. When less than $n_{\max}$ MM units are available, the total execution time will increase, but on the other hand the area occupation of the coprocessor can be changed according to the area constraints of a target device. The computation time of the single MM operation when $n \le n_{\max}$ MM units are used is:

$$T_{MM} = \frac{1}{f_{clk\,\mathrm{MM}}} \left(\frac{k^2}{wn} + 2n \right). \tag{9}$$

The MM coprocessor has 3 main parameters (w, e, and n) that can be changed according to the required area of the implemented coprocessor and the required timings for MM computations (n, w)

or the security level (e). This approach gives high flexibility to the processor and coprocessor design.

Area occupation expressed in LEs and maximal possible FPGA clock frequency for Altera APEX 20K200E are given in Table 2 for precisions $k = 1024$, and 2048 bits. The MM coprocessor has the maximal clock frequency significantly higher than the Nios processor ($f_{clk_{\mathrm{Nios}}} = 33.33$ MHz). To achieve higher performance of the system, we use one clock signal ($f_{clk_{\mathrm{Nios}}}$) for the Nios processor, and another one ($f_{clk_{\mathrm{MM}}}$), with higher frequency, for the MM coprocessor.

The computational time (in µs) of the single MM operation (according to the equation 9) is presented in Table 3 for frequency value of the MM coprocessor clock signal $f_{clk_{\mathrm{MM}}} = 99.99$ MHz.

5.2.1 MM coprocessor interface

The way in which the MM coprocessor is connected to the embedded processor is important for the computing process control (the process starting and the processor status checking), and for an exchange of processed data.

In the previous solution [15] the status register has been used for checking the actual status of the coprocessor and the computational process. The current version uses the communication over an interrupt. This solution is more suitable for software control of coprocessors and for a configuration with several MM coprocessors that can be used for parallel computation of MM. After the computation of MM the interrupt signal of the processor is asserted. This state persists until the results are read within the interrupt routine by the processor.

	$k = 1024$		$k = 2048$	
	LEs	$f_{clk_{MM}}$	LEs	$f_{clk_{MM}}$
$n = 1$	542	107.22	551	105.83
$n = 2$	1100	110.43	1136	106.96
$n = 3$	1621	108.34	1644	104.39
$n = 4$	1943	106.67	1980	103.85

Tab. 2 Area occupation (LEs)/max $f_{clk_{MM}}$ (MHz) of the MM coprocessor ($w = 32$, $n = 1..4$)

	$k = 1024$	$k = 2048$
$n = 1$	327.7	1310.72
$n = 2$	163.88	655.38
$n = 3$	109.26	436.94
$n = 4$	82.00	327.76

Tab. 3 Computational time T_{MM} (µs) for $w = 32$, $f_{clk_{MM}} = 99.99$ MHz, $n = 1..4$

Thereafter new operands can be loaded into the memory and the whole process starts again.

The second area where the most suitable solution needs to be found is the interface between the coprocessor's memory block and the bus of the embedded processor. Since there is a need for faster clocking of the coprocessor, we have assumed the case with two clock signals (see Fig. 7).

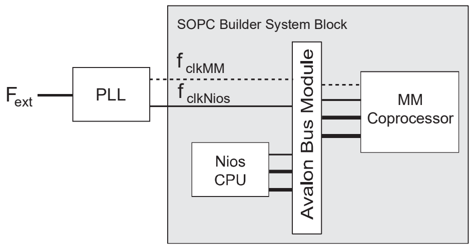

Fig. 7 Block diagram of the interface between the MM coprocessor and the Nios processor

The slower one is connected to the processor, and through the bus and the interface to the coprocessor. All the processes concerning the operations between the processor's bus and the coprocessor's memory block are clocked by this signal ($f_{clk_{\mathrm{Nios}}}$). The operations inside the MM coprocessor are clocked by the external faster clock signal ($f_{clk_{\mathrm{MM}}}$). Thanks to this clock signals organisation almost three times higher performance has been obtained compared to the previous implementation [15].

6. CONCLUSIONS AND FUTURE DEVELOPMENT

The paper has demonstrated the possibility of custom IP blocks development for cryptographic applications in FPGAs. Advancements in FPGAs provide new options for design engineers. FPGAs maintain the advantages of custom functionality, like an ASIC, but avoid high development costs and the inability to make design modifications after productions.

Implemented IP blocks enhance the applicability of the Nios processor in a cryptographic SoC. Both blocks have scalable structure and hence create together with the parameterisable Nios processor very flexible solution. A support for faster modular multiplication and a need of true random numbers are functions required for several currently used cryptographic protocols and algorithms

Thanks to the changes in implemented MM coprocessor we obtained faster solution with optimised features of the interface. The fact that TRNG can be implemented inside the FPGAs represents significant security and system advantage in embedded cryptographic applications.

The future development should include the implementations of other cryptographic algorithms and the improvements of the presented blocks.

REFERENCES

[1] NIOS 2.2 CPU. *Data Sheet*, September 2002, ver. 1.3, pp. 1-14, www.altera.com/nios.

[2] J.A. Menezes, P.C. Oorschot, and S.A. Vanstone: *Handbook of Applied Cryptography*, CRC Press, New York, 1997.

[3] V. Fischer and M. Drutarovský: True Random Number Generator Embedded in Reconfigurable Hardware, *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems – CHES'2002*, Redwood Shores, California, USA, August 2002, pp. 415-430.

[4] APEX 20K Programmable Logic Device Family. *Data Sheet*, February 2002, ver. 4.3, pp. 1-116, www.altera.com.

[5] Avalon Bus Specification, *Altera Reference Manual*, January 2003, ver. 2.1, pp. 1-108, www.altera.com/nios.

[6] M. Dichtl: How to Predict the Output of a Hardware Random Number Generator, In C.D. Walter, C. K. Koc, Ch. Paar (Eds.): *Cryptographic Hardware and Embedded Systems 2003*, LNCS 2779, Springer, Berlin, 2003, pp. 181-188.

[7] Jitter comparison analysis: APEX 20KE PLL vs. Virtex-E DLL, *Technical Brief 70*, January 2001, ver.1.1, pp.1-7, www.altera.com.

[8] Superior Jitter management with DLLs. *Virtex Tech Topic VTT013(v1.2)*, January 21, 2002, pp. 1-6, www.xilinx.com.

[9] C.K. Koc, T. Acar: Analyzing and Comparing Montgomery Multiplication Algorithms, *IEEE Micro*, (16) 3, pp. 26-33, June 1996.

[10] A.F. Tenca, C.K. Koc: A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm, *IEEE Transactions on Computers*, 52(9), pp. 1215-1221, September 2003.

[11] Nios Embedded Processor Development Board, *Altera Data Sheet*, v.2.1, April 2002, pp.1-22, www.altera.com/nios.

[12] A. Rukhin, et al.: A statistical test suite for random and pseudorandom number generators for cryptographic applications, *NIST Special Publication 800-22*, May 15, 2001, pp. 1-153, cs-www.ncsl.nist.gov/rng.

[13] C.K. Koc: RSA Hardware Implementation, www.rsa.com, pp.1-28, August 1995.

[14] M. Drutarovský, V. Fischer: Implementation of Scalable Montgomery Multiplication Co-processor in Altera Reconfigurable Hardware, *Proceedings of 5th International Scientific Conference – DSP MCOM 2001*, Košice, Slovakia, pp. 132-135, November 2001.

[15] M. Šimka, V. Fischer: Montgomery Multiplication Coprocessor for Altera Nios Embedded Processor, *Proceedings of the 5th International Scientific Conference on Electronic Computers and Informatics 2002*, Košice, Slovakia, pp. 206-211, October 2002.

BIOGRAPHY

Miloš Drutarovský was born in 1965 in Prešov, Slovak Republic. He received the MSc degree and PhD degree in radioelectronics from the Faculty of Electrical Engineering and Informatics, Technical University of Košice, in 1988 and 1995, respectively. He defended his habilitation work - Digital Signal Processors in Digital Signal Processing in 2000. He is currently working as an associated professor at the Department of Electronics and Multimedia Communications, Technical University of Košice. Main areas of his research are applied cryptography, digital signal processing, and algorithms for embedded cryptographic architectures.

Martin Šimka was born in 1979 in Košice. He received his MSc. degree in Electronics and Telecommunications in 2002 after defending his Master's Thesis - Conception of connection of embedded processor to arithmetic coprocessor in SOPC Altera. Currently he is a PhD student at the Department of Electronics and Multimedia Communications, Technical University of Košice and his main research area is an implementation of cryptographic blocks on FPGAs.