Cryptographic Applications using FPGA Technology

Alexandru Coman, Radu Frățilă

Military Technical Academy
George Coşbuc Blvd, no 81-83, Sector 5,
Bucharest, zip code 050141
ROMANIA
coman_alex_ionut@yahoo.com, rs2123@gmail.com

Abstract: Cryptographic systems have become a part of our daily life through the need of security of many common activities such as communication, payments, data transfers etc. The best support in design and implementation of cryptographic applications is offered by embedded systems such as ASICs and FPGAs. In the past few years, the increase in performance of FPGAs has made them key components in implementing cryptographic systems. One of the most important parts of the cryptographic systems is the random number generator used. Combinations of PRNG and TRNG are commonly used. A good and efficient TRNG implementation is very important and can be achieved through FPGA technology.

Key-Words: cryptographic algorithm, FPGA, reconfigurable hardware, random number generator, TRNG, oscillator phase noise.

1. Introduction

With the continual and rapid expansion of internet and wireless-based communications across open networks it is becoming increasingly necessary to protect transmitted data. While Cryptography provides the mathematical support to design the necessary data protection services, embedded systems provide the high-speed performance to make these services efficient.
Traditionally, in the design of embedded systems, ASICs (Application-Speciffic Integrated Circuits) have been common components by providing the high performance and/or low power budget that many systems require at the expense of long and difficult design cycles. In the 1980s the use of reprogrammable components, in particular FPGAs (Field Programmable Gate Arrays), was introduced and they allowed for faster design cycles because they enabled early functionality testing.

This is a post conference paper. Parts of this paper have been published in the Proceedings of the 3rd International Conference on Security for Information Technology and Communications, SECITC 2010 Conference (printed version).

Nonetheless, the performance and size of FPGAs did not permit them to substitute ASICs in most applications and thus, they were mainly used to prototype embedded chips small enough to fit in the FPGA. In recent years, however, FPGA manufacturers have come closer to filling the performance gap between FPGAs and ASICs, enabling them, not only to serve as fast prototyping tools but, also to become active players as components in embedded systems [1].
In this context FPGAs have become more and more suitable for cryptographic implementations ranging from high performance encryption algorithms to random number generators.
The reconfigurability of FPGAs offers major advantages when using them for cryptographic applications. In this paper, we will focus on these advantages through examples from various implementations.
We will also make an analysis on TRNGs (True Random Number Generators), their importance in cryptographic applications and some implementation ideas using the FPGA technology.

ISSN 2067 – 4074

2. Advantages of reconfigurable hardware for cryptographic applications

In this section we will present the main advantages of choosing the FPGA technology for implementing cryptographic algorithms. We try to illustrate each main advantage through a small example whenever possible.

2.1 Algorithm flexibility

By algorithm flexibility we are refering to the switching of cryptographic algorithms during operation of the targeted application.The majority of modern security protocols, such as SSL or IPsec, are algorithm independent and allow for multiple encryption algorithms. These encryption algorithms are negotiated
on a per-session basis and a wide variety may be required. For example, IPsec allows among others DES, 3DES, CAST, IDEA, RC4, RC6 and Blowfish, as algorithms, and
future extensions are also possible. some of the advantages of protocols that do not depend on the algorithm are:
the ability to delete broken algorithms
the choice of algorithms according to certain preferences
the possibility of adding new algorithms.
While this kind of flexibility is costly with traditional hardware, FPGAs can be reprogrammed on-the-fly.
A good example of taking advantage of this property is [2] where an Adaptive Cryptographic Engine is proposed. This ACE has the FPGA technology at its core. Besides the FPGA device the system also has a configuration controller and a cryptographic library. The FPGA is configured on the fly by the configuration controller. Subsequently, adaptation to the input key occurs and the data encryption/decryption commences. The configuration controller chooses the proper configuration to be based on the requested security association.

2.2 Algorithm Distribution

It is perceivable that fielded devices are upgraded with a new encryption algorithm at some point. A reason for this could be that the product has to be compatible to new applications. From a cryptographical point of view, the distribution of a new algorithm can be necessary because a current algorithm was broken, the list of ciphers in an algorithm independent protocol (like IPSec) was expanded, or a standard expired (Data Encryption Standard) and a new one was created (Advanced Encryption Standard). Assuming a connection to a network such as the Internet exists, encryption devices equipped with FPGA can upload the new configuration code. It is important to stress that the upgrade of ASIC-implemented algorithms is practically infeasible if many devices are affected or if the systems are not easily accessible, for instance in satellites, while FPGA-implemented ones are easily upgradable.

2.3 Algorithm Modification

Even though most cryptographic algorithms are standardized there are some applications that require some modifications when implementing them. For example, we can introduce permutations at different points or specific substitution boxes if we think of adapting the well known symmetrical algorithms that use this component. These modifications can be easily achieved with reconfigurable hardware.
The UNIX password encryption [3] constitutes an example where a standardized algorithm was slightly changed. In this case DES is used 25 times in a row and a 12-bit salt modifies the expansion mapping (the Unix password encryption has been standardized [4]).
It is also attractive to customize block cipher algorithms such as DES or AES with proprietary S-boxes for certain applications. Furthermore, there are occasions when cryptographic primitives or their operation modes have to be changed according to the application.

2.4 Resource Efficiency

Most security protocols used today are hybrid protocols, like IPSec [5], SSL [6], TLS [7]. This means, that a public-key algorithm is used to transmit the session key. After the key was established a private-key algorithm is used for data encryption. This is done in order to take advantage of both the secure channel provided by the public-key algorithm for symmetrical key exchange, and the faster encryption rates provided by the private-key algorithm.
Since the algorithms are not used simultaneously, the same FPGA device can be utilized for both through run-time reconfiguration. This makes better use of the resources available which is an important factor in many implementations.

2.5 Architecture Efficiency

In some cases, an architecture can be much more efficient if it is designed for a specific set of parameters. The parameters for a cryptographic algorithm can be, for example, the key, the underlying finite field, the coefficients used (the specific curve of an ECC system), and others. In general, the more specific an algorithm is implemented the more efficient it can become.
As an efficient parameter-specific implementation example, we chose the symmetric cipher IDEA [8]. This implementation was presented in [9] and it states that with fixed keys, the main operation in the IDEA becomes a constant multiplication which is far more efficient than a general modular multiplication. In the case of IDEA a general modular shift-and-add multiplication requires 16 partial multiplications while only eight are needed for a fixed key.
Another good example is taken from asymmetric cryptography. We are talking about the arithmetic architectures for Galois
Fields which tend to be way more efficient if the irreducible polynomial and the field order are fixed. Squaring in $GF(2^m)$ takes m=2 cycles with a general architecture, but only one cycle if the architecture is compiled for a fixed field [10]. We should point out that squaring in $GF(2^m)$ is one of the most common operations when implementing elliptic curve cryptosystems defined over fields of characteristic two.
FPGA devices allow this type of design and optimization with a specific parameter set. Due to the way FPGAs work, the application can be changed totally or partially very fast and effective.

2.6 Cost Efficiency

There are two cost things, that have to be considered when analyzing the cost efficiency of FPGAs: development cost and the prices of units. The costs for developing an FPGA implementation of an algorithm
are much lower than for an ASIC implementation, because we are actually able to use the given structure of the FPGA (for example, the look-up table) and to test the reconfigured chip for as many times as we wish without any further costs. This results in a shorter time-to-market period, which is nowadays an important cost factor. The unit prices are not so significant as a cost efficiency factor when comparing them to the development costs. In fact, for high-volume applications, ASIC implementations usually become the more cost-efficient choice.

2.7 Data transfer efficiency

When judging data transfer efficiency we can safely state that general-purpose CPUs are not optimized for fast execution especially in the case of public-key algorithms. That happens mainly because they lack the instructions for modular arithmetic operations on long operands. These operations include, for example, multiplication, squaring, inversion, and addition for elliptic curve cryptosystems (ECC) [11] and exponentiation for RSA [12]. Although they are typically slower than ASIC implementations, the FPGA implementations have the potential of running substantially faster than software implementations. To make an idea of what data rates we are talking

ISSN 2067 – 4074

about here are some examples of functional implementations and their throughput values. The block cipher AES reaches a data rate of 112,3 Mbit/s and 718,4 Mbit/s on a DSP TI TMS320C6201 [13] and Pentium III [14], respectively. In comparison, the FPGA implementation of the same algorithm on a Virtex XCV-1000BG560-6 achieved 12 GBit/s using 12,600 slices and 80 RAMs [15]. On the other hand, an ASIC encrypts at about double the speed of the FPGA, for example, the Amphion CS5240TK which can reach 25.6 Gbit/s at 200MHz.

3 True Random Number Generators

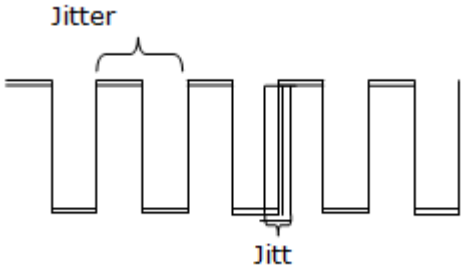
The requirement for random values in cryptography is growing. They are used as initial values, challenges, block paddings, nonces and, of course, cryptographic keys thus creating a need for random number generators. Pure software implementations for Random Number Generators (RNGs) result in Pseudo Random Number Generator (PRNG), because they follow a certain algorithm which. This feature makes the results predictable, so it creates flaws in cryptographic systems which can be exploited by a cryptanalyst. In order to overcome this problem True Random Number Generators (TRNGs) have been created. TRNGs use hardware to generate unpredictable values as input for different algorithms. Because TRNGs work slower than PRNGs they are usually used in pairs: the TRNG generates the seed which will be used by the PRNG to generate other values.
In order to achieve grater speeds for cryptographic systems, and additional security, dedicated devices areas used to encrypt data. Because the operations usually used in cryptographic algorithms are bitwise operations ASICs and FPGAs are used in such devices. TRNGs implemented in FPGA chips use random values from physical phenomena or signal sources such as diode noise, resistive noise, jitter or phase noise, digital circuit artifacts, disk drive turbulence, and noise from A/D converters and radio receivers[16]. The FPGA implementations are gaining popularity because of it's advantages in design time, performance, power consumption, chip area or cost over DSP, VLSI or microcontrollers.

3.1 True Random Number Generators Implementation

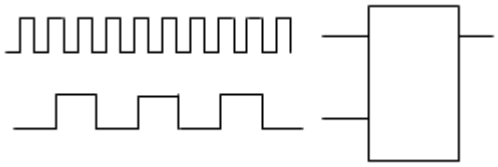
Although Random Number Generators are very important there are few hardware TRNG implementations reported. There are several techniques described in the literature. Such techniques are: direct amplification – which digitizes a source of noise using an amplifier and a comparator; discrete time chaos – which uses chaotic systems to produce random numbers. A commonly described and used technique is the oscillator sampling (oscillator phase noise) which uses two oscillators of different frequency and the jitter noise.

3.2 TRNGs based on oscillator phase noise

This approach uses a slow clock source to sample a faster clock source. A source of randomness in this approach is the jitter noise of the clock sources. Jitter is the variations in the significant instants of a clock or data signal such as the rising edge at the begging of a period or the falling edge. Jitter in digital circuits has different sources such as semiconductor noise, power supply variations, cross talk and electro-magnetic fields in the operating environment. An example would be the period jitter which is the time deviation of clock's period from it's average period, exemplified in Figure 1[17].
Most common implementations for this type of TRNGs use the system clock as the high frequency clock – F_h and an external oscillator as the low frequency clock - F_l. Digital Clock Manager chips, embedded in certain FPGA development boards, are used in order to maximize the jitter from F_l.



In order to sample the high frequency clock an edge triggered D-type Flip-flop is used with F_l as the clock source and F_h as the data input as shown in Figure 2. The random bit succession (R) is the output of the Flip-flop.



In these implementations because the duty cycle of F_h is not 50% the resulted stream of random bits will not have equal probability of being zero or one. Another important aspect is that if F_l's jitter is not large enough (compared to F_h) the two clock sources will be correlated which will give the resulting bit stream a certain periodicity, making it predictable.
To remove these aspects from the resulting stream of bits different de-sckewing techniques can be used, such as: parity filter, Von Neumann de-skew filter or strong mixing functions [18]. If the ratio of ones to zeros in the resulted bit stream is

$$0.5 + e : 0.5 - e, \quad (1)$$

after passing it through a N-tap parity filter the ratio becomes relation 2:

$$1 / 2 * [1 + (2e)^N] : 1 / 2 * [1 - (2e)^N]. \quad (2)$$

If a Von Neumann filter is used it will perform badly if the resulted bits are correlated. Strong mixing functions like hashing and cryptographic functions such as DES, SHA, MD2, MD4 and MD5 output a smaller number of bits than the input with and increased randomness[16].
Other implementations use ring oscillators as the clock sources. Ring oscillators are formed using a buffer and two transparent latches, that add propagation delay, and an inverter configured serially to feed back on itself. The propagation delay is configured in order to obtain two clock sources with different frequencys. This implementation uses a sampler block to create the random stream of bits, with the help of a control block, which also notifies the user that a new bit was obtained from the random bit stream. The control block also helps to eliminate the correlated bits resulted in the first periods of F_h. These bits would apear because F_h also has jitter, having it's source built exactly the same way as the source for F_l.[17]

3.3 Proposed implementation

We propose an approach where the stabilizing time of the composing circuits is exploited in such a way that the actual value contained inside the circuits will not always be stable and so the output will be random as shown in Figure 3. In order to implement this type of TRNG we will use some of the slowest circuits available to propagate a certain bit stream received from an input. The frequency of the clock signal will be above the one specified in the chip's documentation. Another time domain will be necessary in order to be able to send the resulted bit stream to another device as the resulted random bit stream. As the safety borders of the circuits are being exceeded it is expected that the results may vary with environmental factors as temperature and power fluctuations and jitter from adjacent circuits.

ISSN 2067 – 4074

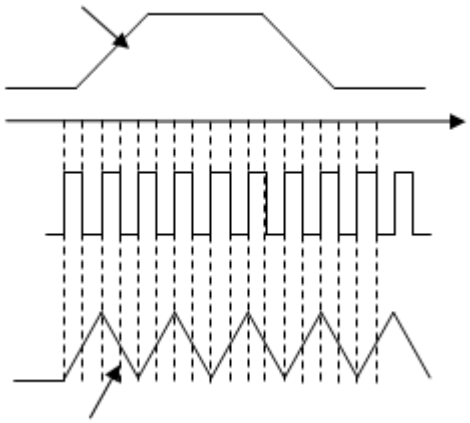


Figure 3. Behavioral time-line diagram of proposed implementation

4 Conclusion

In this paper we tried to make an overview of the features that make reconfigurable hardware a viable tool in the design and implementation of cryptographic applications. We presented general advantages such as algorithm flexibility, modification and distribution combined with the efficiency of architecture, resources, data transfer and non the least, costs. All these advantages were backed up with references and examples of real life implementations.
There was also a chapter about random number generators with some ideas of implementation which we will further develop into a functional system. This stands as an example of how the FPGA architecture can be used in developing cryptographic applications.

References

[1] Wong, S., Vassiliadis, S., and Cotofana, S. 2002. Future Directions of (Programmable and Reconfigurable) Embedded Processors. In Embedded Processor Design Challenges, Workshop on Systems, Architectures, Modeling, and Simulation SAMOS 2002.

[2] Viktor K. Prasanna and Andreas Dandalis, FPGA-based Cryptography for

Internet Security, Department of EE-Systems University of Southern California

[3] Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. Handbook of Applied Cryptography. CRC Press, Boca Raton, Florida, USA. 1997

[4] ANSI. 1981. American National Standards Data Encryption Algorithm X3.92-1981. American National Standards Association.

[5] Kent, S. and Atkinson, R. RFC 2401: Security Architecture for the Internet Protocol. Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group, Reston, Virginia, USA. 1998.

[6] Freier, A. O., Karlton, P., and Kocher, P. C. The SSL Protocol Version 3.0. Transport Layer Security Working Group INTERNET-DRAFT. 1996.

[7] Dierks, T. and Allen, C. RFC 2246: The TLS Protocol Version 1.0. Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group, Reston,Virginia, USA. 1999.

[8] Lai, X. and Massey, J. A proposal for a new block encryption standard. In Advances in Cryptology EUROCRYPT '90, I. B. Damgaard, Ed. Vol. LNCS 473. Springer-Verlag, Berlin,Germany, 389-404. 1990.

[9] Taylor, R. and Goldstein, S. A high-performance flexible architecture for cryptography. In Workshop on Cryptographic Hardware and Embedded Systems | CHES '99,Springer-Verlag, Worcester, Massachusetts, USA, 231-245. 1999.

[10] Wu, H. Low complexity bit-parallel finite field arithmetic using polynomial basis. In Workshop on Cryptographic Hardware and Embedded Systems | CHES 1999, Vol. LNCS 1717. Springer-Verlag, 280 - 291. 1999.

[11] Koblitz, N. Elliptic curve cryptosystems. Mathematics of Computation 48, 203-209. 1987.

[12] Rivest, R. L., Shamir, A., and Adleman, L. 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21, 120-126.

[13] Wollinger, T.,Wang, M., Guajardo, J., and Paar, C. How well are high-end DSPs suited for the AES algorithms? In The Third Advanced Encryption Standard Candidate Conference. National Institute of Standards and Technology, New York, New York, USA, 94-105. 2000.

[14] Lipmaa, H. 2002. Fast Software Implementations of AES. http://www.tcs.hut.fi/~helger/aes/rijndael.html.

[15] Gaj, K. and Chodowiec, P. Fast implementation and fair comparison of the final candidates for Advanced Encryption Standard using Field Programmable Gate Arrays. In Topics in Cryptology - CT-RSA 2001, D. Naccache, Ed. Vol. LNCS 2020. Springer-Verlag, Berlin, 84 - 99. 2001.

[16] Kwok, S.H.M. and Lam, E.Y., "FPGA-based High-speed True Random Number Generator for Cryptographic Applications", TENCON, IEEE Region 10 Conference, 2006;

[17] Paul Kohlbrenner and Kris Gaj, "An embedded true random number generator for FPGAs", International Symposium on Field Programmable Gate Arrays, 2004;

[18] K.H. Tsoi, K.H. Leung and P.H.W. Leong, "Compact FPGA-based True and Pseudo Random Number Generators", 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003.

[19] Renaud Santoro, Olivier Sentieys, Arnaud Tisserand, Philippe Quémerais, Arnaud Carer, Thomas Anger, INRIA - Team Activity Report - Team Cairn, 2009.