ISSN 2277-3061
Volume 17 Number 1
International journal of Computers and Technology

Frequency Analysis of 32-bit Modular Divider Based on Extended GCD Algorithm for Different FPGA chips

Qasem Abu Al-Haija * [1], Abdullah AlShuaibi [1, 2] and Ahmad Al Badawi [3]

[1] Department of Electrical Engineering, King Faisal University, Al-Ahsa, 31982, Saudi Arabia
[2] Materials Science and Engineering Department, Cornell University, Kimball Hall, ithaca, 14850 NY, United States
[3] Department of Electrical and Computer Engineering, National University of Singapore, 119077 Singapore

Abstract

Modular inversion with large integers and modulus is a fundamental operation in many public-key cryptosystems. Extended Euclidean algorithm (XGCD) is an extension of Euclidean algorithm (GCD) used to compute the modular multiplicative inverse of two coprime numbers. In this paper, we propose a Frequency Analysis study of 32-bit modular divider based on extended-GCD algorithm targeting different chips of field-programmable gate array (FPGA). The experimental results showed that the design recorded the best performance results when implemented using Kintex7 (xc7k70t-2-fbg676) FPGA kit with a minimum delay period of 50.63 ns and maximum operating frequency of 19.5 MHz. Therefore, the proposed work can be embedded with many FPGA based cryptographic applications.

Keywords

Modular Arithmetic; Extended Euclidian's GCD; Field Programmable Gate Array (FPGA), Hardware Synthesize, Coprime numbers, Public Key Cryptography.

1. Introduction

The enormous contemporary technological revolution in electronics, computing and telecommunication facilitated the emergence of communication systems that have the ability to connect people all over the world, and store information about their users. As a result, people started to become more concerned about their privacy and insure guarded access to these communication facilities to prevent unauthorized users to invade their private information.

An effective approach to fulfill the weaknesses to guarantee secured network channel for such purposes is cryptographic techniques, which can be defined as the study of communications over non-secure channels [1]. In other words, the set of means and protocols that can be used to protect users' information transmitted over non-secure communication channels from being accessed by unauthorized parties. In general, cryptography incorporates two main procedures: a) encryption, which is the process of converting ordinary information (called plaintext) into unintelligible text (called Ciphertext), and b) decryption, which is converting the unintelligible text back to plaintext.

Encryption and decryption require the use of a piece of information that is called key. Based on the nature of the key, there are commonly two types of cryptographic systems: a) symmetric-key cryptography where the same key is used in the encryption and the decryption procedures, and b) public-key cryptography where the key is divided into two mathematically related parts: public key and private key; the public-key is distributed by its owner through the network and used by other users to encrypt their messages when trying to communicate with the public-key owner. The public-key owner can easily decrypt the Ciphertext using his private-key. Since public-key and private-key are related by hard mathematical problems, it is safe to say that trying to deduce the private-key from the public-key is computationally infeasible, especially when key size is large enough [3]. Public-key cryptography is known to be more secure and use simple key-management approaches; however, since it is based on hard mathematical problems it requires more computational resources than symmetric-key cryptography.

One well-known public-key cryptographic system is the RSA cryptosystem developed by Rivest, Shamir and Adleman in 1977 [4]. The mathematical relation between public-key and private-key in RSA is based on the factorization problem, which known to be hard if the prime numbers involved are quite large. RSA is one of various cryptosystems that use division operations which require high computational resources and long execution time.

The original Euclidian algorithm (i.e. un-extended) is well-known iterative algorithm used to compute the greatest common divisor (GCD) [5, 14] of two integer numbers. An extended version of this algorithm can be used in the modular arithmetic to compute the modular division (inversion). Our focus in this work is to provide a detailed hardware design of 32-bit modular divider based on the Extended Euclidian's Greatest Common Divider (XGCD) algorithm, which can be integrated smoothly into the FPGA design of RSA cryptosystem as in [6] where XGCD is used to perform the modular inverse operations. The rest of the paper is organized as follows: first, a literature review of modular dividers is presented. Second, we provide an introduction to the substantial background review of modular dividers. Finally, we present our hardware implementation and results of our modular divider followed by the conclusions and future works.

2. Related Work

In the last years, many researches were proposed to address the design and implement of an efficient modular inversion algorithm. One of the well-known solutions was the hardware design using FPGA technology with high level simulations and synthesize. Some of these researches are reviewed here [7-11].

For instance, K. Bigou and A. Tisserand in [7] proposed new RNS modular inversion algorithm based on the extended Euclidean algorithm and the plus-minus trick. In their algorithm, comparisons over large RNS values are replaced by cheap computations modulo 4 and comparisons to an RNS version based on Fermat's little theorem were carried out. They significantly reduced the number of elementary modular operations: a factor of 12 to 26 for multiplications and 6 to 21 for additions. Finally, they implemented their design using Virtex 5 FPGAs and concluded that for a similar area, their plus-minus RNS modular inversion is 6 to 10 times faster.

Also, A. Cilardo in [8] presented a solution for the computation of modular inversion based on speculative addition. Their proposed circuit effectively addressed the problem of long carry chains caused by signed operations and allows fast low-overhead implementations of modular inversion. They concluded that their solution is particularly suitable for devices providing optimized carry-propagation logic such as FPGAs.

Moreover, Md Selim Hossain in [9] implemented high-performance modular inversion for ECC using FPGA technology. A binary inversion algorithm in VHSIC Hardware Description Language (VHDL) has been used for this efficient implementation. The analysis of the design performance and the timing simulation showed that the delay for one modular inversion operation in a modern Xilinx Virtex-7 FPGA is only 2.329 µs at the maximum frequency of 146.389 MHz. We have implemented an area-efficient design which takes a small amount of resources on the FPGA and needs only 1480 slices. Thus, their proposed modular inversion over F_256 provides a better performance than the available hardware implementations in terms of the area and the timing.

Furthermore, Hlaváč J., Lórencz R. in [10] presented hardware architecture of an arithmetic unit intended for computing basic operations over a Galois field GF (p). The arithmetic unit supports addition, subtraction, multiplication, and multiplicative inverse modulo a prime p. To compute the multiplicative inverse, they have used the promising left-shifting algorithm that is based on the extended Euclidean algorithm. Also, they discuss the potential applications of the arithmetic unit, including elliptic curve cryptography. As a result, they concluded that their arithmetic unit can be used whenever there is need to perform modular arithmetic operations efficiently in hardware, such as in elliptic curve cryptography, or for calculations that use a residual number system. Also, they stated that their proposed arithmetic unit outperforms existing designs that do not directly implement the inversion.

Finally, S. Vollala in [11] proposed new algorithm to evaluate modular inverse that can accept any type of integer modulus input without imposing any restriction. The calculated values are stored in a look-up table and hence named LUK-mod-inverse algorithm. Their proposed algorithm uses look-up table structure that has been designed in hardware that explores memorization technique. All the stored modular inverse values can be retrieved in just two clock cycles. They simulated and synthesized their algorithm code using Xilinx-14.6 ISE for usage in FPGA board and the results have shown a positive trend in terms of frequency, clock cycles and throughput. Thus, their proposed hardware design can compute the multiplicative modular inverse within 2 clock cycles only, it can improve the frequency by 32.22% and throughput by 600% for 64-bits integers.

3. Background

To give the reader an adequate insight, we provide here a comprehensive introduction to modular inverse in addition to the Extended Euclidean algorithm. Modular inverse or modular multiplicative inverse [13], of a modulo m is s such that:

$$a.s \equiv 1 \ (mod\ m) \qquad (1)$$

Where a, m and s are integers and GCD (a, m) = 1. It should also be noted that s in equation (1) can be noted a^{-1}.

Extended Euclidean algorithm can be used to compute, in addition to the greatest common devisor of integers a and b, the coefficients s and t of Bezouts identity which stated in equation (2).

$$a.s + b.t = GCD\ (a, b) \qquad (2)$$

When a and b are co-primes, i.e. GCD (a, b) = 1, we say that s is the modular inverse of a modulo b, and t is the modular inverse of b modulo a. The relationship between equations (1) and (2) can be illustrated as follows:

To compute the modular inverse of a modulo m such that the GCD (a, m) = 1.

- Applying equation (2) yields: $a.s + b.t = 1$.
- Reducing this identity *modulo m* gives:

$$a.s + b.t = 1 \ mod\ (mod\ m)$$

This shows that the Extended Euclidean algorithm can be used to compute the modular inverse of integer a modulo m. The modular inversion flowchart as invented in the US patent number 6,609,141 B1 [12] is shown in figure 1.

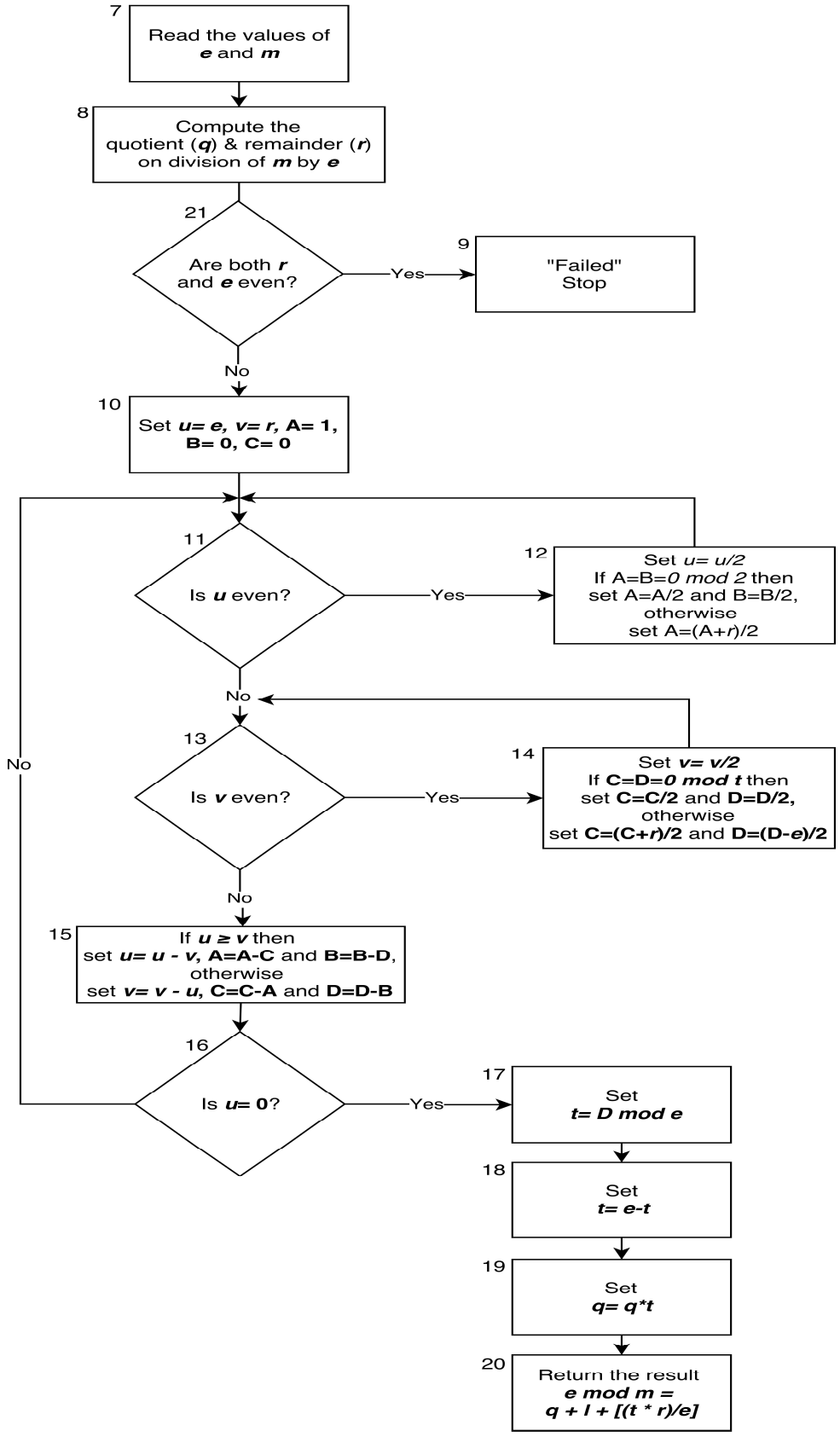

Figure 1. Method of Performing Modular Inversion

The following pseudo code shows the modified Extended Euclidean algorithm used to find the modular inverse. The complete algorithm is given below, followed by numerical example.

Algorithm:	***Mod_Inverse***
Input:	*Integers a and m*
Output:	*Integer s such that*
	a.s ≡ 1 (mod m)
	s = 0; s1 = 1;
	r = m; r1 = a;
	while (r1 ≠ 0)
	quotient = r div r1;
	*(s, s1) = (s1, s - quotient * s1);*
	*(r, r1) = (r1, r - quotient * r1);*
	if r > 1 then return "a is not invertible";
	if s < 0 then s = s + m;
	return s;

To make everything concrete, we provide here a simple numerical example to compute the modular inverse of 18 modulo 65 (18^{-1} mod 65). To solve this example, table 1 shows how Mod Inverse algorithm processed with input a = 18 and m = 65 (Qi is quotient and I is Index). Note that s in the last row is less than 0, so we need to add m to compute the positive modular inverse. The modular inverse of 18 mod 65 is (-18 + 65 = 47).

Table 1. Numerical example of Mod_Inverse trace

I	Qi	s	s1	r	r1
0		0	1	65	18
1	3	1	0 – 3*1= -3	18	65-3*18= 11
2	1	-3	1 - 1*-3= 4	11	18 – 1 * 11= 7
3	1	4	-3 – 1 * 4= -7	7	11 – 1 * 7= 4
4	1	-7	4 – 1 * -7= 11	4	7 – 1 * 4= 3
5	1	11	-7 – 1 * 11= -18	3	4 – 1 * 3= 1
6	3	-18	11 - 3 * -18= 65	1	3 - 3 * 1= 0

4. Implementation Environment and Results

Extended Euclidean algorithm is an extension of the Euclidean algorithm to find the modular multiplicative inverse of two coprime numbers. As mentioned before, the core part of the Extended Euclidean algorithm is taken the succession of remainders as well as the quotients while in the Euclidean algorithm the quotients are ignored, and the remainders kept.

The proposed hardware implementation considers the standard design of 32-bit length as a design precision for input integers and modulus as well as the output result of XGCD based Euclidian's method. Also, the VHDL has been used as hardware description language with target FPGA chip device Altera Cyclone IV (EP4CGX-22CF19C6). The hardware design of XGCD was built using carry save addition technique. For comparisons and benchmarking purposes, in addition to the Altera FPGA device, we have synthesized our VHDL code using five up-to-date Xilinx FPGA devices, namely: Vertix5 (xc5vlx20t-2-ff323), Spartan6 (xc7z010-2-clg400), Artix7 (xc7a100t-2-csg324), Kintex7 (xc7k70t-2-fbg676), Zynq7 (xc7z010-2-clg400).

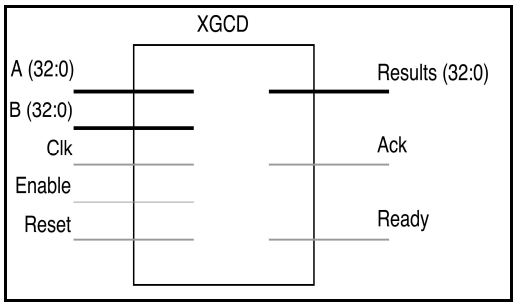

Figure 2. Top View of our 32-bit Modular Divider Chip Processor

The top view of our 32-bit modular divider chip processor is shown in figure 2. XGCD unit includes the following: two 32-bits integers as an input values and three control signals (reset, enable and a synchronized clock) as well as 32-bits integer as an output result reduced by 32-bit modulus along with the acknowledgment and ready signals. Also, state diagram of the XGCD is given in figure 3.

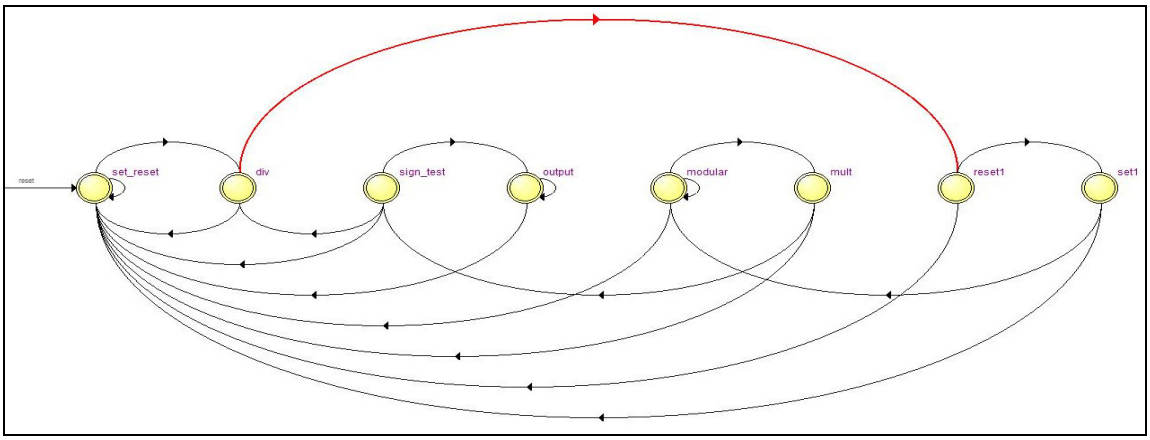

Figure 3. State Diagram of XGCD Algorithm

To generate the experimental results, the proposed design was synthesized using ALTERA Quartus II and Xilinx Synthesizer ISE tools over high-speed CPU architecture (Intel-Core (TM) i7-4770@ 3.40GHz) to improve the simulation and verification time. Table II along with figures 4 and 5 provides the performance results in terms of maximum frequency (FMAX) and the minimum delay period (ns) resulted from synthesizing the proposed VHDL implementation via six different FPGA devices.

Table 2. Maximum Frequency values FMAX (MHz) Vs. Minimum period values TMIN (ns)

FPGA device	FMAX (MHz)	TMIN (ns)
ALTERA CYCLONE IV EP4CGX-22CF19C6	14.85	67.334
Xilinx VERTIX 5 XC5VLX20T-2-FF323	16.40	60.981
Xilinx SPARTAN 6 XC7Z010-2-CLG400	7.323	136.552
Xilinx ARTIX 7 XC7A100T-2-CSG324	12.528	79.824
Xilinx KINTEX 7 XC7K70T-2-FBG676	19.752	50.627
Xilinx ZYNQ 7 XC7Z010-2-CLG400	19.733	50.676

From the obtained results, it is obviously seen that the minimum delay and maximum frequency occurred when the design is implemented using Xilinx Kintex7 (xc7k70t-2-fbg676) device with 50.63 ns and 19.5 MHz respectively. However, Altera Cyclone IV (ep4cgx-22cf19c6) device recorded lower figures (67.334 ns and 14.85 MHz) it still considered as comparable and alternative device due to its simplicity and cost.

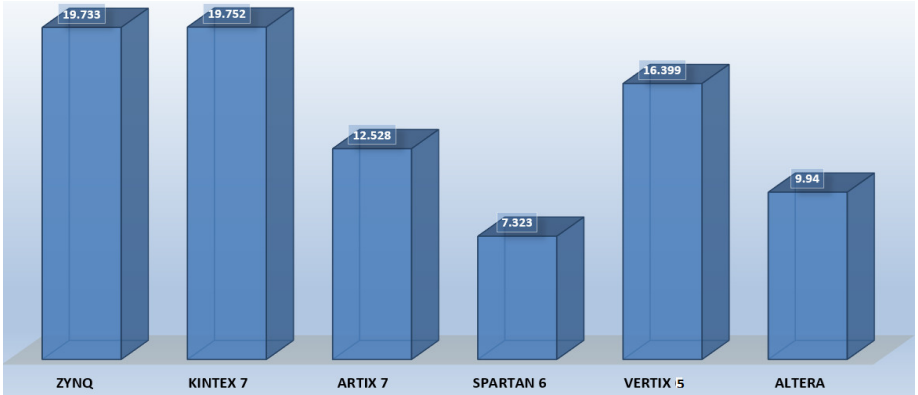

Figure 4. Maximum Frequency values FMAX (MHz)

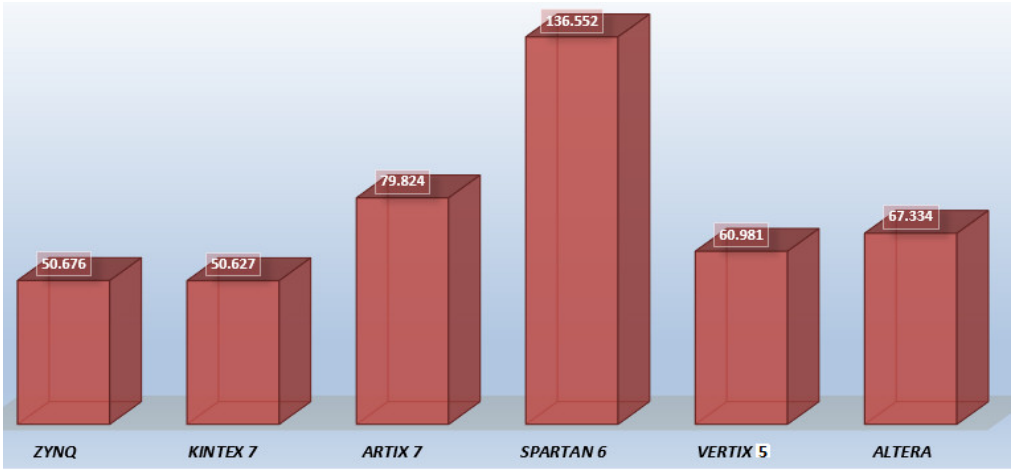

Figure 5. Minimum period values TMIN (ns)

Finally, table III shows the hardware utilization results for the design using Altera's device recoded some attractive results that reflects the scalability of the design as it utilizes small amount of device capabilities. This means that the XGCD precision can be smoothly increased to higher bit lengths.

Table 3. Maximum Frequency values FMAX (MHz) Vs. Minimum period values TMIN (ns)

Hardware factor	Utilization %
Total combinational functions	2714/21280 (13%)
Total number of registers	546/21280 (3%)
Total number of pins	104/167 (62%)
Total LABs: partially or completely	201/ 1330 (15%)
Number of Clock pins	3/6 (50%)

5. Conclusions

In this paper, we propose performance evaluation for the FPGA implementation of XGCD processor based on extended Euclidian's algorithm using different FPGA devices technology. Analysis of two performance factors are majorly considered: the minimum delay (ns) and the maximum frequency (MHz) to compare the VHDL synthesize using various up-to-date FPGA devices. The most compelling evidence, the hardware utilization results are showed for the proposed implementation by using Altera kit devices. The design synthesized for Kintex7 device which beats all other devices in terms of maximum operating frequency.

References

[1] V. S. Miller. *Use of Elliptic Curves in Cryptography*. Exploratory Computer Science. IBM Research, Springer-Verlag, 1998.

[2] D. Kahn. *The Codebreakers*. 1967, ISBN 0684831309.

[3] S. Levy. Crypto: How the Code Rebels Beat the Government Saving Privacy in the Digital Age, 2001, ISBN 0140244328.

[4] R. Rivest, A. Shamir, and L. Adleman. *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*. Communications of the ACM, 1978.

[5] Q. Abu Al-Haija, M. Al-Ja'fari and M. A. Smadi. *A Comparative Study up to 1024 bit Euclid's GCD algorithm FPGA Implementation and Synthesizing*. IEEE 5th International Conference on Electronic Devices, Systems and Applications (ICEDSA), 2016.

[6] Q. Abu Al-Haija, et. al. *Efficient FPGA Implementation of RSA Coprocessor Using Scalable Modules*. Procedia Computer Science, Elsevier, 2014.

[7] Bigou, K., Tisserand, A. *Improving modular inversion in RNS using the plusminus method*. In: Bertoni, G., Coron, J.-S. (eds.) CHES2013. LNCS, vol. 8086, pp. 233–249. Springer, Heidelberg, 2013.

[8] A. Cilardo, *Modular inversion based on digit-level speculative addition*, Electronics Letters (Volume: 49, Issue: 25, December 5 2013), IET.

[9] M.S.Hossain. *High-Performance FPGA Implementation of Modular Inversion over F_256 for Elliptic Curve Cryptography*, IEEE International Conference on Data Science & Data Intensive Systems, ICDSDIS 2015.

[10] J. Hlaváč and R. Lórencz. *Arithmetic Unit for Computations in GF(p) with Left-Shifting Multiplicative Inverse Algorithm*, Architecture of Computing Systems, ARCS 2013. Lecture Notes in Computer Science, vol 7767. Springer, 2013.

[11] S. Vollala, *Hardware design for multiplicative modular inverse based on table look up technique*, International Conference on Computing & Network Communications (CoCoNet), 2015.

[12] P. Montague. *Method of performing modular inversion*. US Patent 6609141, 2003.

[13] T. Koshy. *Elementary number theory with applications*, 2nd edition. ISBN 9780123724878.

[14] Q. Abu Al-Haija, S.M.S Ahmad, I. Alfarran. *FPGA implementation of variable precision Euclid's GCD algorithm*. Journal of Engineering Technology (JoET), American Society for Engineering Education (ASEE): ISSN 0747-9964, Vol.6, Special Issue, 2017.