Proceedings of the Federated Conference on Computer Science and Information Systems pp. 363–368
DOI: 10.15439/2018F197
ISSN 2300-5963 ACSIS, Vol. 15

An Improved Architecture of a Hardware Accelerator for Factoring Integers with Elliptic Curve Method

Michał Andrzejczak
Wojskowa Akademia Techniczna
ul. Urbanowicza 2, 01-489 Warszawa, Poland
Email: michal.andrzejczak@wat.edu.pl

***Abstract*—Elliptic Curve Method (ECM) is a well-known method for factoring integers, which is usually used in the Number Field Sieve algorithm as a subroutine for factoring smaller integers than the targeted one. ECM is called many times and can be executed in parallel for different inputs. This method mainly consist of simple operations on elliptic curves. Thus, ECM is suitable for hardware implementations that can efficiently reduce computational time. This work describes a new, improved FPGA-based hardware accelerator for ECM, designed for large scale computations. Our accelerator can operate with an on board ARM processor or with an external host computer. This design can factor several numbers at once and can be easily ported to various FPGA boards. Different methods for improving results (e.g. the use of DSP blocks, cache-registers, reorganizing instruction order) are described and their performance is analyzed. As a result, one of the fastest hardware ECM units is achieved.**

I. Introduction

FACTORIZATION is one of the main hard problems used in construction of cryptosystems. One of the most known and the most popular cryptosystem based on factorization problem is RSA. To the present day, several algorithms for factoring integers have been developed and used in various cases. The best algorithm for factoring integers with large factors used in RSA is Generalized Number Field Sieve (GNFS) [2]. This method require to factor a lot of smaller numbers in one of the main steps of the algorithm and the Elliptic Curve Method can be efficiently used for this. ECM performs many operations on a small data, requiring little memory and can be run many times in parallel with the same probability of factoring chosen number. Thus, special purpose hardware can efficiently improve overall factorization time. In this paper an improved architecture of a hardware accelerator for factoring integers with Elliptic Curve Method (ECM) is presented. Analysis of previous architecture is included and detected weaknesses are described with potential improvements. At the end, influences of several changes in initial design are compared, with the best result more than three times better than previously reported in literature.

II. Elliptic Curve Method

The ECM was proposed by H.W. Lenstra [1] (called Phase 1) in late 80's and its principles are based on Pollard (p-1) method. Later, ECM was extended and improved by Brent [3] and Montgomery [4] (called Phase 2).

Let choose a field K with characteristic different from 2 and 3. The elliptic curve $E_{A,B}$ is the set of points $(X, Y) \in K$ such that

$$Y^2 = X^3 + AX + B$$

where $A, B \in K$ and $4A^3 + 27B^2 \neq 0$ with a special point $O_E = (0 : 1 : 0)$ called a "point at infinity".

For more efficient computer implementation, Montgomery's form of elliptic curve is recommended due to lack of number inversion computation. Montgomery's form can be obtained from Weierstrass form presented above by following change of the variables $X \to (3x + a)/(3b)$, $Y \to y/b$, $A \to (3 - a^2)/(3b^2)$, $B \to (2a^3 - 9a)/(27b^3)$. Homogeneous form of this curve is:

$$by^2z = x^3 + ax^2z + xz^2$$

with the triple $(x : y : z)$ represents the point $(x/z : y/z)$ in affine coordinates. Projective coordinates of curve in Montgomery's form allow all intermediate computations to be performed using only x and z coordinate. The y coordinate can be retrived from two others coordinates, but is not necessary in ECM algorithm.

Let q be an unknown factor of N - the number being factorized. The ECM starts with randomly selecting an elliptic curve $E_{a,b}$ and a random point on it. Computations are performed modulo the number N, as if $\mathbb{Z}/n\mathbb{Z}$ was a field. First step of computations can be done just once. In this step, product of all prime numbers and its powers is computed. Most time consuming operation is done in second step, where scalar multiplication of chosen point by computed product is performed. In the last step, greatest common divisor of resulted z coordinate and a factorized N is computed. Pseudocode for ECM is shown in Listing 1.

Algorithm 1 ECM algorithm, phase 1

Require: a composite number N, random point P_0 on random elliptic curve E, integer bound B_1
Ensure: a factor of N or $fail$
$k \leftarrow \prod_{p \leq B_1} p^{\log_p B_1}$
$Q_0 \leftarrow kP_0$;
$q \leftarrow gcd(z_Q, N)$;
if $q > 1$ **then**
  return q
else
  return $fail$
end if

A. Complexity of the ECM

The complexity of ECM is sub-exponential and is described as:

$$\mathcal{O}(n) = e^{\sqrt{\log p \log \log p}\sqrt{2}+o(1)} M(\log n)$$

where $M(\log n)$ is the complexity of multiplication $\mod n$. First part depends only on factors of the chosen integer. The only way to speed up computations is to execute point multiplication as fast as possible, so basically what this paper is about.

III. Initial Design

The initial design was proposed in [8]. The main idea of that hardware accelerator is to spread as many as possible autonomous ECM units in one FPGA chip. The ECM units have Harvard architecture with separable instruction list and data memory. Design can be described in three levels. First of them, the top level, describe FPGA device and interconnections between main modules and external components. Lower level is about design of ECM unit, interconnections between memory, controllers and arithmetic modules. The last level describes architecture and algorithms used in modules building the ECM unit (modular multipliers, adders, controller).

A. Top level

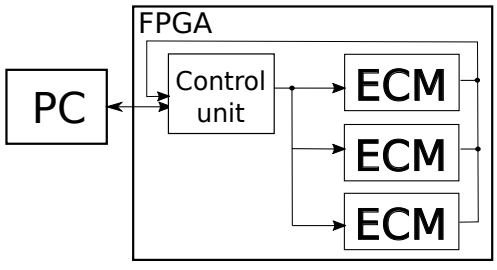

Fig. 1. Top level

In Fig. 1 the top level is shown. FPGA chip is filled by as many ECM units as possible with one global control unit for all of them, responsible for Montgommery Ladder execution, communication with external PC and managing the work units. Connected PC is used for random curves sampling and for last stage of the algorithm, the gcd computation. It is done in this way to maximize logic usage and simplify chip design. Many independent ECM units allow better clock signal distribution. FPGA is responsible only for point multiplication over elliptic curve.

B. ECM level

Every ECM unit is equipped with internal memory, memory controller, microprocessor and 4 arithmetic units (two for modular multiplication, one for addition and subtraction) as shown in 2. During initialization, every ECM unit need random elliptic curve and random point over this curve. Provided curves should be in Montgomery form and every coordinate should be converted to Montgomery domain [5].

The memory controller is responsible for communicating with two way memory bank. Loaded data words are concatenated and put into bus registers. This controller has also internal semaphore table for preventing data override during parallel execution and additional table for storing result address of computed data.

The main controller has ROM memory for instruction and can execute simple commands. Every instruction takes two memory addresses for data input and one address for writing result. There are 5 instructions:

- **ADD** - instruction used for addition
- **SUB** - used for subtraction
- **MULA**- multiplication by first unit
- **MULB**- multiplication by second unit
- **LOADN**- modulus read from memory

These instructions can be used to replace computation path with computations over Edwards curves by simply reprogramming the ROM table.

The ECM unit is equipped with two modular multiplication unit which allow faster point multiplication. This idea was taken from [6].The computation flow for point doubling and addition in Montgomery form ([5]) is shown in Table I) and uses two multipliers in parallel.

C. Module level

Modular multiplication is the most time consuming operation. During every point doubling/addition it is performed 11 times and this number can be reduced to 10. To obtain the lower number of multiplications one coordinate of P_0 must be chosen arbitrarily to simplify computations by selecting $z_{P_0} = z_{P-Q} = 1$. Thus, use of two modular multipliers can increase total throughput. For multiplication, logic based algorithm [7] was used. The aim of that was to have design capable to be deployed on low cost devices without enough DSP modules. This also save logic required for routing to these modules in designs with high percentage of logic usage. Implemented algorithm perform modular multiplication of n - bit numbers in Montgomery form in n clock cycles. Modular reduction in Montgomery's domain is based on efficient hardware bit shift operation and was chosen due to very good performance.

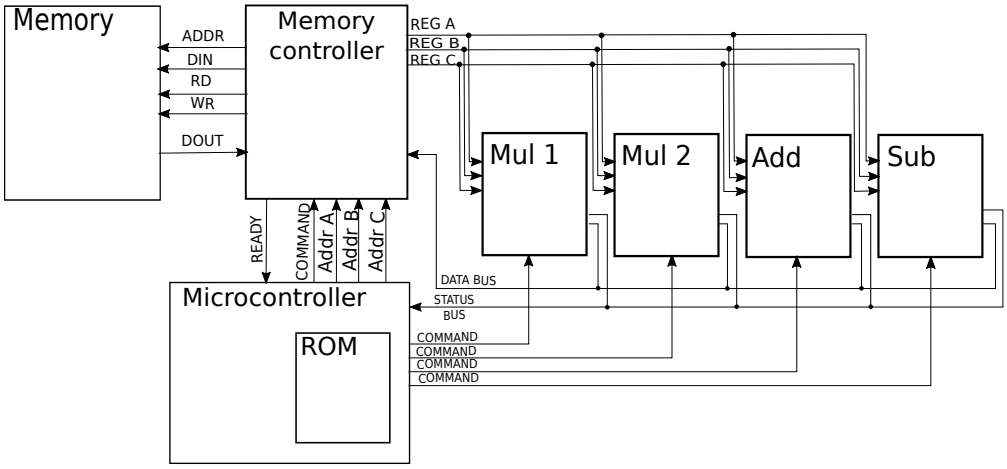

Fig. 2. ECM level

TABLE I
ONE STEP OF SCALAR MULTIPLICATION IN CASE OF $z_{P-Q} = 1$

Adder	Substractor	Multiplier 1	Multiplier 2
$a_1 = x_P + z_P$	$s_1 = x_P - z_P$		
$a_2 = x_Q + z_Q$	$s_2 = x_Q - z_Q$	$m_1 = s_1^2$	$m_2 = a_1^2$
	$s_3 = m_2 - m_1$	$m_3 = s_1 \cdot a_2$	$m_4 = s_2 \cdot a_1$
$a_3 = m_3 + m_4$	$s_4 = m_3 - m_4$	$x_{2P} = m_1 \cdot m_2$	$m_6 = s_3 \cdot a_{24}$
$a_4 = m_1 + m_6$		$x_{P+Q} = a_3^2$	$m_8 = s_4^2$
		$z_{P+Q} = m_8 \cdot x_{P-Q}$	$z_{2P} = s_3 \cdot a_4$

D. Comparision

Basic parameters of this design are shown in Table II in comparison with other results reported earlier in literature. The design was compiled for low cost Altera DE1-SOC board equipped with Cyclone VCSEMA5F31C6 FPGA and for high end Stratix IV targeted for high performance computing.

IV. ANALYSIS AND IMPROVEMENTS

The reported results for initial design are very competitive. However, deep analysis of proposed solution indicates several bottlenecks which may be improved to achieve much better performance.

Fig. 3 shows data dependency graph for point multiplication. Every arrow represents memory read operation and circles are the arithmetic operations with result write to memory. Memory controller is capable only to read from memory to one register at once which result in doubling the same operation. Moreover, data is loaded almost immediately after being write to memory in several cases.

A. Memory improvements

Analysis of the simulation diagrams proved that memory operations give one of the biggest slowdown on design. Every arithmetic operation needs two load operations of operands (which is done by sequential memory loads, concatenated at the end) and one write operation of result, done in similar manner. For 192-bit length numbers and 32-bit size memory, communication overhead takes around 20 clock cycles for one operation.

Simple solution for this problem is to increase memory base size to decrease this overhead. Increasing memory base size from 32-bits to 128-bits can decrease the number of memory calls from 21 to 6 clock cycles. Size of the design increase slightly with this improvements, further called **opt1**.

On the other hand, situation when one variable is loaded twice in a row or written and read in next step is very common. The first issue can be solved by expanding instruction set by load instruction for two operands at once. Several cases when the same data is loaded for two arithmetic units can be improved by splitting arithmetic instructions for more atomic operations. Instead one arithmetic instruction, where attributes are addresses in memory, we use load to register instructions and execute arithmetic operation instruction (without any arguments). .

The second issue needs additional cache registers for temporary results. Adding these registers slightly increases design size, but offered overall performance by FPGA chip is improved. With this change it is possible to replace order list. New orders can be more atomic. With atomic instructions the program size increase, but there is no need in memory controller to be responsible for parallel data access. 4 temporary registers were added to design. With these registers and with direct result to input operation, memory usage is limited only

TABLE II
RESULTS OF THE IMPLEMENTATION COMPARED WITH OTHERS REPORTED IN LITERATURE

Author:	Gaj [6]	Gaj [6]	Zimmermann [11]	Zimmermann [11]	de Meulenaer [10]	Andrzejczak [8]
Device:	S35000	V4LX200	V4SX35	XC4VSX35	XCV4SX25	SGX530
Number length:	198 - bit	198 - bit %	202 - bit	134 - bit	135 - bit	192 - bit
Max. number of modules	13	24	24	24	1	96
Max. clock freq.	80 MHz	104 MHz	200 MHz	200 MHz	220 MHz	150 MHz
Clock cycles in phase 1	1 666 500	1 666 500	1 473 596	797 288	13 750	2 101 400
Time for phase 1	21 ms	16 ms	7.37 ms	3.99 ms	63 s	14 ms
Curves/sec:	624	1 448	3 240	6000	16 000	6822

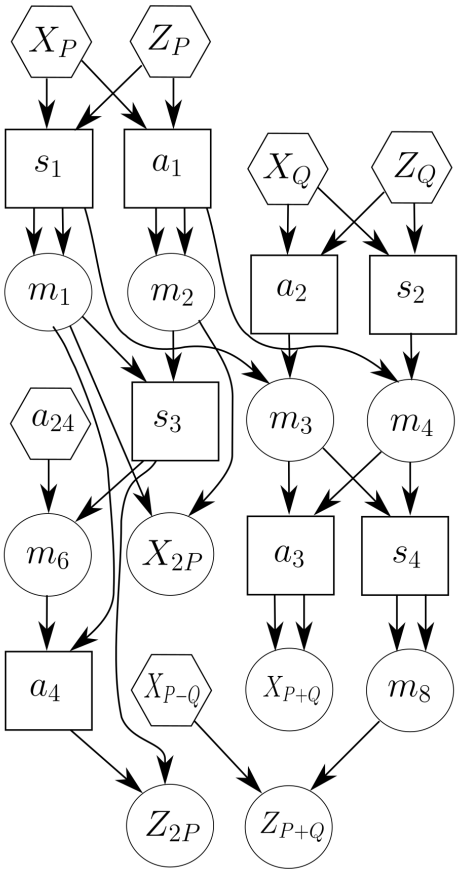

Fig. 3. Data computation graph for point multiplication

to load first coordinates for first bit of ladder and load one coordinate (a_{24}) for every bit. The intermediate results can be stored in these 4 additional temporary registers or can be directly passed to input of another arithmetic unit. The saved memory can be used in Phase 2 to store more pre computation results and improvements from **opt1** are less significant in overall result and are not included in this design, called **opt2**

Instructions have no longer the same format. Arithmetic instructions are not taking any arguments, they operate on data provided to special input registers loaded earlier. Extended instruction set with description is shown in Table III

TABLE III
EXTENDED INSTRUCTION SET WITH FORMAT DESCRIPTION

Name	Description
RLOAD	Load data from one register to another one. Can be used to load data for two registers at once
MLOAD	Memory load to one or two registers
MULA	Start multiplication in unit A
MULB	Start multiplication in unit B
ADD	Start addition
SUB	Start subtraction
WAITFOR	Waits for end of computation in selected module

After memory operation optimization and with the new instruction set, data computation graph is changed. Fig. 4 presents improved data computation graph. All coordinates loaded from memory are marked by gray color and correspond to first improvement. Second improvement is presented with red circles marking data loaded for different modules in one load operation. Double loads for integer squaring or doubling are marked by one pointer. Values stored in temporary registers are marked with dots.

B. Multiplication unit replacement

The other way to increase number of checked elliptic curves is to speed-up multiplication computation. The modular multiplication based on logic gates takes n clock cycles and to decrease this number DSP multiplication algorithms should be used. Algorithm for modular Montgomery multiplication proposed by Itoh [9] was selected. Multiplier is parametrized by radix and multiplication is performed in n^2 steps, where:

$$n = \frac{\text{number length}}{radix}$$

Optimal selection of radix is crucial for overall performance. Bigger radix needs more DSP (Digital Signal Processing) blocks used for integer multiplication. The size of multiplier (in Logic Elements) increase as increase the number of DSP blocks needed, because of longer paths used to route signals to these blocks. The best results have been achieved for radix 32, requiring only 3 DSP blocks per multiplication and executing

TABLE IV
IMPROVEMENTS COMPARISION

Parameter	Initial	opt1	opt2	opt3	Stratix IV opt3
Logic:	30 060	31 394	30 982	25 566	372 951
Logic usage(%):	97 %	98 %	97 %	80 %	92 %
DSP:	-	-	-	66	654
Max. clock freq.	88 MHz	88 MHz	85 MHz	90.1 MHz	151 MHz
Num. of Units	10	10	9	11	109
Clock cycles per bit	1580	1374	1215	481	481
Clock cycles in phase 1:	2 101 400	1 827 420	1 615 950	639 730	639 730
Curves/sec:	418	480	473	1546	25 557
Improvement factor:	**1**	**1.14**	**1.13**	**3.69**	**3.72**

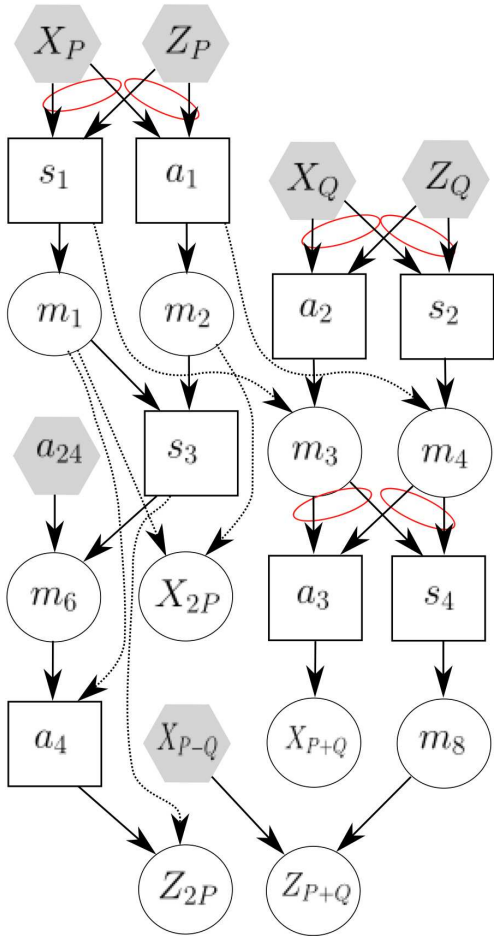

Fig. 4. Data computation graph for improved point multiplication

in 54 clock cycles. Bigger radix significantly decrease the maximum clock frequency, due to longer propagation paths and require more DSP block, which lead to higher logic utilization. Variant with shorter radix executes in more clock cycles and leave many DSP blocks unused in global view.

The replacement of multiplication algorithm saved around 150 clock cycles per each call of this function. Two modules are used at once, so in general around 750 clock cycles are saved compared to previous used logic based algorithm. Moreover, DSP-based module requires less logic element and Quartus Prime compiler was able to fit one more ECM unit in targeted Cyclone V device. For 11 ECM units the compiler needs 25 566 ALM (Adaptive Logic Modules) which is 80% of all available resources. Adding one more ECM unit is not possible. For 12 ECM units compiler can not place all of them close to hardware multipliers, so longer routing path are needed. For targeted Cyclone V, 33 091 ALM is required and it is above 100% of available resources. Multiplier replacement is called **opt3**

Improvements were implemented incrementally. Every next design contains improvement from previous versions. Table IV compares results of constructed modules. The last column contains compilation data for Stratix IV GX530, used to check throughput of initial design. This one is much bigger than the low cost Cyclone V and can be used in practice to factorize numbers in GNFS. Achieved results are 3.72 times better than at the beginning. The total number of sieved curves is the highest one from reported in literature.

V. CONCLUSIONS

An improved hardware architecture for factoring integers has been presented. Careful analysis of algorithm and hardware design lead to architectural changes resulting 3.72 times faster device. The most efficient was the change of multiplication algorithm, which reduced almost half of the total number of computations. Additional temporary registers may be more significant for total computation time if the base of memory was not increased and memory operations will still take almost four times more. With all improvements combined, the fastest architecture is obtained.

Recently Intel introduced new more powerful devices called Stratix 10. Further works will adapt described design for new devices. Preliminary simulations shows around 180 thousands curves per second for one of the biggest devices from the new family.

REFERENCES

[1] H. W. Lenstra, "Factoring Integers with Elliptic Curves" *Annals of Mathematics,* vol. 126, no.2, pp. 694–673, 1985.

[2] A. K. Lenstra and H. W. Lenstra, "The Development of the Number Field Sieve" *Lecture Notes in Math,* Volume 1554, 1993.
[3] R. P. Brent, "Some integer factorization algorithms using elliptic curves," *Australian Computer Science Communications,* vol. 8, pp. 148-163, 1986.
[4] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Mathematics of Computation,* vol. 48, pp. 243-264, 1987.
[5] P. L. Montgomery, "Modular Multiplication Without Trial Division," *Mathematics of Computation,*, vol. 44, pp. 519-519, 1985.
[6] K. Gaj et al., "Area-time efficient implementation of the elliptic curve method of factoring in reconfigurable hardware for application in the number field sieve," *IEEE Transactions on Computers,* vol. 59, pp. 1264-1280, 2010/9
[7] K. Gaj, M. Huang, S. Kwon, T. A. El-Ghazawi, "An Optimized Hardware Architecture for the Montgomery Multiplication Algorithm," *Public Key Cryptography,* , 2008
[8] M. Andrzejczak, "Koprocesor kryptograficzny wspierajacy faktoryzacje liczb metoda krzywych eliptycznych," [Konferencja mlodych naukowcow wiwat 2017, Falenty, Polska, 2017], in press.
[9] K. Itoh, M. Takenaka, N. Torii, S. Temma, Y. Kurihara "Fast Implementation of Public-Key Cryptography on a DSP", [Cryptographic Hardware and Embedded System], 2002.
[10] G. de Meulenaer, F. Gosset, G. de Dormale, J. Quisquater, "Integer Factorization Based on Elliptic Curve Method: Towards Better Exploitation of Reconfigurable Hardware", [15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007)], Napa, CA, 2007, pp. 197-206.
[11] R. Zimmermann, "Optimized Implementation of the Elliptic Curve Factorization Method on a Highly Parallelized Hardware Cluster", Master Thesis, TU Braunschweig, 2009 r.