ISSN 2303-4521
Original Research

Periodicals of Engineering and Natural Sciences
Vol. 10, No. 6, December 2022, pp.113-119

Study of effective calculation operation implementation remaining multi-bit numbers division on FPGA

Aqeel Ali. Al-Hilali[1], Laith F. Jumaa[2], Ibrahim A. Amory[2]
[1] Medical Instrumentation techniques, Al-Farahidi University, Iraq
[2] Medical Instrumentation techniques, Al-Esraa University College, Iraq

ABSTRACT

The rapid enhancement in the fields of the computers that leads to rapid breaking for ciphering algorithms and for these reasons most of ciphering algorithm tried to used multidigit for ciphering texts or images. Using multidigit will increase the safety of information and protected it from supercomputer from breaking the ciphering algorithms. The current information systems employ operations on finite fields of various structures (for example, cryptographic systems). In this instance, it's common to have to deal with enormous numbers (128 bits or more). The proposed operation of discovering the remainder of the division of multidigit numbers will considerably improve the speed of such systems if implemented.

Keywords: Programmable logic integrated circuits, distributed arithmetic, fields with a limit, multi-digit numbers.

Corresponding Author:

Aqeel Ali Al-Hilali
Medical Instrumentation techniques
Al-Farahidi University
Address Iraq
E-mail: akeel.alhilali@alfarahidiuc.edu.iq

1. Introduction

The type of sets of remnants from division by pre-chosen bases (coprime common numbers). The assignment of computing the rest of a division turns out to be more muddled when numbers are taken as a positional number huge digit limit, introduced in the alleged "long" math. This type of composing a number it is utilized when the piece lattice of the hardware utilized for calculations is restricted. On the off chance that the digit the matrix is restricted to, say, k pieces, the long number is addressed in the base 2k number framework and is composed successively into memory (for instance, as a cluster). For this situation, the issue of getting the rest of the division becomes nontrivial. Strategies for its answer can be found in [1-3]. In [4], a viable technique is proposed for changing over a double number into a RNS dependent on isolating the first multi-bit parallel number into independent parts, for which a foreordained number of paired digits B are designated. At that point a n-cycle twofold number can be communicated as a mix of weighted (positional) numbers with the measurement B (bits)[5-7]. For this situation, the places of every such part are allocated a specific weight 2j, where j = 0, B, 2B, ..., MB. Direct change of a double number to a secluded number is done utilizing a particular summation modulo residuals pi (I = 1, 2, ... n) - B pieces of n/B parts of a multideity number, considering their loads. In light of the abovementioned, any parallel number can be composed as:

$$X = \sum_{j=0}^{M} \left(\sum_{i=0}^{B-1} x_{jB+i} \, 2^{i} \right) 2^{jB} \tag{1}$$

where B is the number of discharges of one part; M is the number of parts;

x_(jB+i)- - coefficient 0 or 1; j = 0, B, 2B, ..., MB is the position of the part; i is the position of the bit in the part. Using expression (1), we can write the formula for computing a remainder mod p:

$$|X|_p = \left| \sum_{j=0}^{M} \left(\sum_{i=0}^{B-1} x_{jB+i}\, 2^i \right) 2^{jB} \right|_p = \left| \sum_{j=0}^{M} \left(\sum_{i=0}^{B-1} x_{jB+i}\, 2^i \right) 2^{jB} . \left|2^{jB}\right|_p \right|_p \qquad (2)$$

When performing operations, it should be borne in mind that 〖ω_i=|2^jB |〗 _p are pre-considered constants. By freely comparing the remaining pieces, the remaining divisions for each can be found. The next step is to sum modulo p the calculated residuals for each component. This method of calculating residuals allows for the completion of jobs on much smaller piece depths, on the order of the piece width of a single section. Example. Let X = 589249631 and p = 13 be two arbitrary numbers you're working with. To solve, we represent X in binary:

X = 100011 00011111 00111100 01011111. Divide the binary form of X into 4 groups of B = 8 bits, each. Determine the constants ω for use in your computations:

$\omega_0 = |2^0|_{13} = 1;\ \omega_1 = |2^8|_{13} = 9; \omega_2 = |2^{16}|_{13} = 3; \omega_3 = |2^{24}|_{13} = 1$

Further calculations are presented in the following diagram:

$$\begin{tikzcd}
\left\| 01011111 \right|_{1101} \cdot \omega_0 \Big|_{13} \arrow[r] & \left\| 95 \right|_{13} \cdot 1 \Big|_{13} \arrow[r] & 4 \arrow[dr, dash] & & \\
\left\| 00111100 \right|_{1101} \cdot \omega_1 \Big|_{13} \arrow[r] & \left\| 60 \right|_{13} \cdot 9 \Big|_{13} \arrow[r] & 7 \arrow[r, dash] & \bigcirc\!\!\!\!\!+ \arrow[r] & \left|22\right|_{13} = 9 \\
\left\| 00011111 \right|_{1101} \cdot \omega_2 \Big|_{13} \arrow[r] & \left\| 31 \right|_{13} \cdot 3 \Big|_{13} \arrow[r] & 2 \arrow[ur, dash] & & \\
\left\| 00100011 \right|_{1101} \cdot \omega_3 \Big|_{13} \arrow[r] & \left\| 35 \right|_{13} \cdot 1 \Big|_{11} \arrow[r] & 9 \arrow[uur, dash] & &
\end{tikzcd}$$

According to these calculations, |X|_13=9, which is true. We will implement this approach for computing the remainder of the division using the HDL synthesis and analysis software Xilinx ISE (Integrated Synthesis Environment), and then we will verify its efficiency with a precise [8-10]. Hardware and development time for Xilinx Virtex 6 FPGAs using the XC6VLX75T core are estimated at [12].

2. Productive execution of the activity for figuring the rest of a division

Compelling execution of the activity for ascertaining the rest of the division. Advancement of gadgets that carry out present day cryptographic calculations suggests the utilization of superior equipment[13-16]. Programmable Logic Integrated Circuits, specifically FPGAs from Xilinx, are the standard equipment for creating present day elite registering gadgets. FPGA Virtex 6, center Xilinx XC6VLX75T, which has an adequate region, was chosen as a means for the hardware implementation of the algorithm under consideration. To increase the performance, we adapt the considered algorithm to the FPGA architecture[17]. To avoid performing the costly operation of multideity numbers modulo multiplication from the point of view of time and used hardware resources, we store in memory all possible B-bit values previously multiplied by the constants ωi modulo the chosen modulus, and the number of these magnitudes will rely on the choice of parameter B and equal to 2B ... Thus, using this approach allows you to implement modular multiplication tabular. Modulo addition is implemented using a modular adder. Let us apply this method of calculating the modulus for numbers of different widths (32 bits, 64 bits, 128 bits, 256 bits, 512 bits, 1024 bits, 2048 bits) in 32-bit (p = 4294967291), 16-bit (p = 65521) and 8-bit (p = 251) modules, while varying the parameter B. Data on the hardware and time spent on computing the remainder of the division of 32-bit numbers will be written in Table. 1-7 [18-19].

Table 1. Equipment and spent time on executing the X mod p activity for 32-cycle numbers on FPGA Virtex 6, center Xilinx XC6VLX75T

Split Mod		**1 bit**	**2 bit**	**4 bit**	**8 bit**	**IEEE Numeric_std**
32 bit	**Slices**	42	34	30	43	22
	Delay, ns	11,7	10,6	10	11	8,7

Split Mod		**1 bit**	**2 bit**	**4 bit**	**8 bit**	**IEEE Numeric_std**
16 bit	**Slices**	105	71	36	54	265
	Delay, ns	16.3	14,7	12	13	58,6
8 bit	**Slices**	87	46	47	63	287
	Delay, ns	16,8	13,4	14,4	12,2	68.ju4

Table 2. Equipment and spent time on carrying out the X mod p activity for 64-cycle numbers on FPGA Virtex 6, center Xilinx XC6VLX75T

Split Mod		**1 bit**	**2 bit**	**4 bit**	**8 bit**	**IEEE Numeric_std**
32 bit	Slices	42	34	30	43	22
	Delay, ns	11,7	10,6	10	11	8,7
16 bit	Slices	105	71	36	54	265
	Delay, ns	16.3	14,7	12	13	58,6
8 bit	Slices	87	46	47	63	287
	Delay, ns	16,8	13,4	14,4	12,2	68.ю4

While figuring the rest of isolating 32-digit numbers by a 32-bit module, we got the upside of the standard IEEE Numeric_std library calculation both as far as equipment and time costs. When calculating the remainder after dividing 32-bit numbers into 16 and 8-bit modules, the optimal hardware and time costs are to use the parameters B = 4 and 2 bits, respectively. When calculating the remainder of dividing 64-bit numbers into 32, 16 and 8-bit modules, the optimal hardware and time costs are the use of the parameters B = 4, 8 and 8 bits, respectively.

Table 3. Hardware and spent time on X mod p operation for 128-bit numbers on FPGA Virtex 6, core Xilinx XC6VLX75T

Split Mod		**1 bit**	**2 bit**	**4 bit**	**8 bit**	**IEEE Numeric_std**
32 bit	Slices	190	147	112	145	4216
	Delay, ns	24	23,5	20,9	20,2	453,9
16 bit	Slices	380	341	265	186	4172
	Delay, ns	18,6	20,4	17,5	14,7	490,6
8 bit	Slices	222	133	99	124	4399
	Delay, ns	19	15,2	14,8	15, 3	509,6

While ascertaining the rest of isolating 128-bit numbers by a 32-digit module, the ideal equipment and time cost is to utilize the boundary B = 4 pieces, which has a 37-overlay region advantage and a 21-overlap time advantage over the standard calculation. IEEE Numeric_std libraries. While figuring the rest of isolating 128-bit numbers by a 16-digit modulus, the ideal equipment and time cost is to utilize the boundary B = 8 pieces, which has a 22-overlap region advantage and a 32-crease time advantage contrasted with standard calculation of the IEEE Numeric_std library. While computing the rest of partitioning 128-digit numbers by a 8-cycle modulus, the ideal equipment and time cost is to utilize the boundary B = 8 pieces, which has a 35-overlay region advantage and a 34-overlap time advantage over the standard calculation. IEEE Numeric_std libraries .Using the inherent calculation of the IEEE Numeric_std library for ascertaining the rest of the division of 256-bit numbers is incomprehensible because of the restricted assets of the FPGA, while the proposed calculation permits you to figure the rest of the division of numbers whose measurement surpasses 256 bits .As a consequence of looking at the considered strategy for computing the rest of division with the standard calculation of the IEEE Numeric_std library, we can reason that it is prudent to utilize the implicit technique just while figuring the rest of division of a 32-cycle number into a 32-bit module. In any case, it ought to be noticed that when working with numbers with a width of in excess of 128 pieces, the standard elements of the IEEE Numeric_std library are adequately not, while the technique viable permits you to handle quantities of any width, restricted simply by the equipment capacities of the programmable rationale coordinated circuit utilized.

Table 4. Equipment and spent time on carrying out the X mod p activity for 256-cycle numbers on FPGA Virtex 6, center Xilinx XC6VLX75T

Split Mod		**1 bit**	**2 bit**	**4 bit**	**8 bit**	**IEEE Numeric_std**
32 bit	Slices	225	183	152	221	–
	Delay, ns	23,5	26,9	24, 0	23,6	–
16 bit	Slices	493	512	342	244	–
	Delay, ns	22,7	25,0	23,7	21,2	–
8 bit	Slices	288	140	109	133	–
	Delay, ns	21,3	18,0	18,8	16,9	–

While figuring the rest of separating 256-bit numbers by 32, 16, and 8-bit modules, the ideal equipment and time costs are to utilize the boundaries B = 4, 8, and 8 pieces, individually. When calculating the remainder of dividing 512-bit numbers by 32, 16 and 8-bit modules, the optimal hardware and time costs are the use of the parameters B = 8, 8, and 4 bits, respectively. When calculating the remainder after dividing 2048-bit numbers by 32, 16 and 8-bit modules, the optimal hardware and time costs are to use the parameters B = 8, 8, and 4 bits, respectively. Based on the data obtained, we will construct graphs of the dependence of the number of Slices used and the maximum time delays on the partition parameter B when calculating the remainder of dividing multi-bit numbers of various lengths into 32-bit, 16-bit and 8-bit modules.

Table 5. Equipment and spent time on carrying out the X mod p activity for 512-cycle numbers on FPGA Virtex 6, center Xilinx XC6VLX75T

Split Mod		**1 bit**	**2 bit**	**4 bit**	**8 bit**	**IEEE Numeric_std**
32 bit	Slices	225	183	152	221	–
	Delay, ns	23,5	26,9	24, 0	23,6	–
16 bit	Slices	493	512	342	244	–
	Delay, ns	22,7	25,0	23,7	21,2	–
8 bit	Slices	288	140	109	133	–
	Delay, ns	21,3	18,0	18,8	16,9	–

Table 6. Equipment and spent time on carrying out the X mod p activity for 1024-cycle numbers on FPGA Virtex 6, center Xilinx XC6VLX75T

Split Mod		**1 bit**	**2 bit**	**4 bit**	**8 bit**	**IEEE Numeric_std**
32 bit	Slices	1419	796	224	364	–
	Delay, ns	43,0	42,5	29,6	29,7	–
16 bit	Slices	2173	832	535	367	–
	Delay, ns	29,0	31,8	29,8	26,1	–
8 bit	Slices	589	367	198	188	–
	Delay, ns	25,5	23,2	20,5	22,1	–

When calculating the remainder of dividing 1024-bit numbers by 32, 16 and 8-bit modules, the optimal hardware and time costs are to use the parameters B = 4, 8, and 4 bits, respectively.

Table 7. Equipment and spent time on executing the X mod p activity for 2048-piece numbers on FPGA Virtex 6, center Xilinx XC6VLX75T

Split Mod		**1 bit**	**2 bit**	**4 bit**	**8 bit**	**IEEE Numeric_std**
32 bit	Slices	1705	952	623	424	–
	Delay, ns	46,5	46,8	41,8	32,5	–
16 bit	Slices	2621	1469	626	426	–
	Delay, ns	41,4	40,1	31,8	28,3	–
8 bit	Slices	1345	454	259	216	–
	Delay, ns	32,3	25,3	23,5	24,7	–

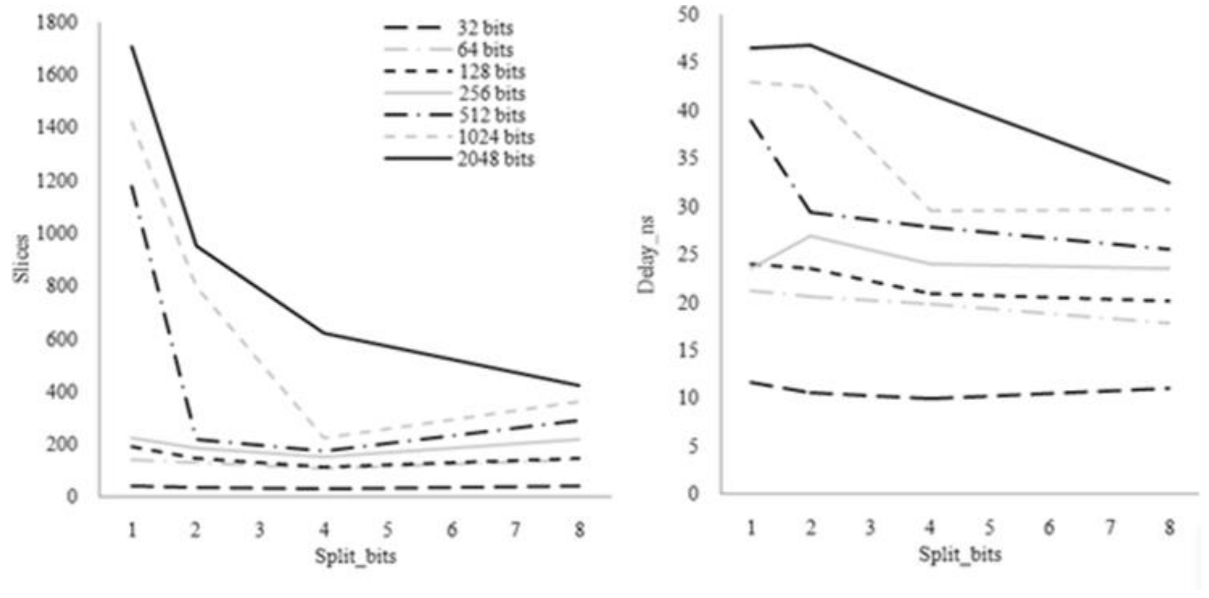

Figure 1. Hardware and time costs for implementing the X mod p operation on a 32-bit FPGA Virtex 6 module, core Xilinx XC6VLX75T a) the number of Slices; b) maximum time delays, ns

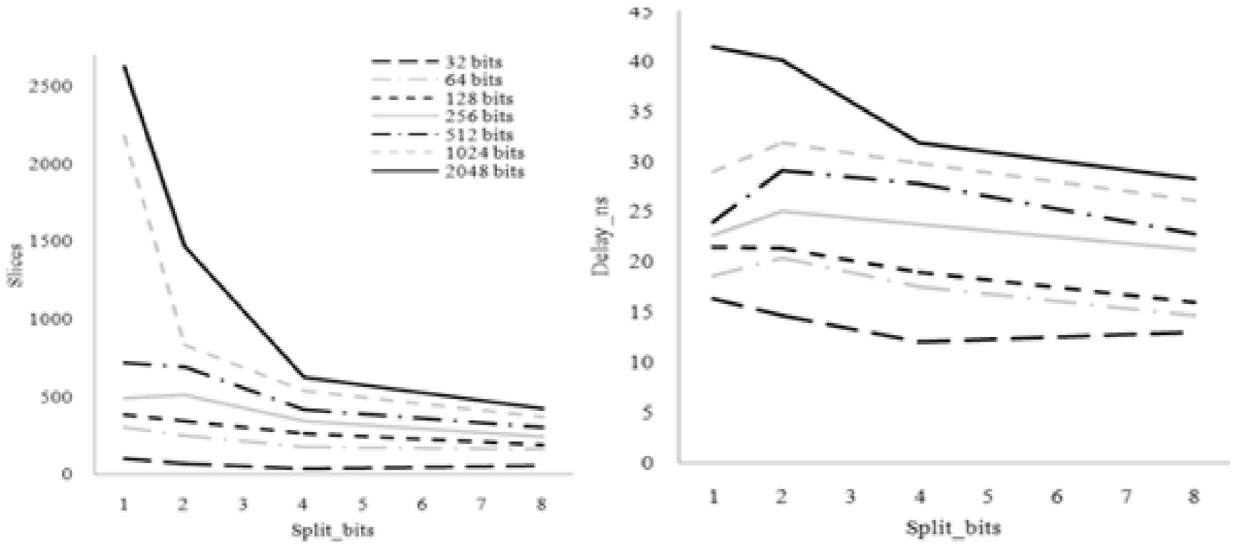

Figure 2. Hardware and time costs for implementing the X mod p operation on a 16-bit FPGA Virtex 6 module, core Xilinx XC6VLX75T a) the number of Slices; b) maximum time delays, ns

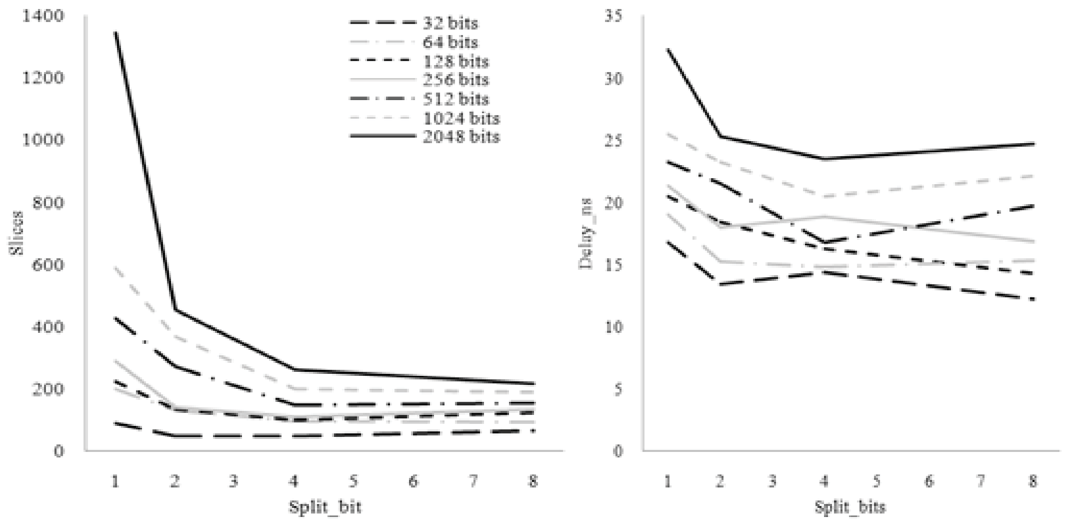

Figure 3. Hardware and time costs for implementing the X mod p operation on an 8-bit FPGA Virtex 6 module, core Xilinx XC6VLX75T a) the number of Slices; b) maximum time delays, ns

From the diagrams acquired (Figures 1-3), we can presume that with an expansion in the digit limit of the prepared numbers, the quantity of Slices and the most extreme time delays in the execution of the activity of ascertaining the rest of the division increment. The quantity of Slices and the most extreme time delays in the execution of the activity for ascertaining the rest of a division arrive at their littlest qualities when the boundary B = 4, 8 pieces, contingent upon the piece width of the number and the modulus.

3. Conclusions

The outcomes got over the span of the examination showed that the utilization of the IEEE Numeric_std library calculation for figuring the rest of division is prudent just on account of computing the rest of isolating a 32-bit number by a 32-bit module, and for preparing numbers with a width of in excess of 128 pieces of standard library capacities IEEE Numeric_std isn't sufficient.

The utilization of conveyed math related to plain particular increase and measured snake permits not exclusively to fundamentally diminish the time spent on the execution of the activity of figuring the rest of division by putting away all conceivable B-cycle esteems to begin with duplicated by the loads for the chose module, yet in addition makes it conceivable to measure multi-digit numbers (in excess of 128 pieces) of any length, restricted simply by the equipment abilities of the programmable rationale incorporated circuit utilized.

Examination of different approaches to carry out the activity of computing the rest of separating multi-digit numbers showed that the ideal according to the perspective of the pre-owned equipment assets and the time spent on the activity of figuring the rest of the division, is to part the enormous piece number into 4-bit or 8-bit parts, contingent upon the piece width numbers and modulus.

Declaration of competing interest

The authors declare that they have no known financial or non-financial competing interests in any material discussed in this paper.

Funding information

No funding was received from any financial organization to conduct this research.

Acknowledgements

The authors have special thanks for Al-Farahidi University and Al-Esraa University College for their support during the period of this research.

References

[1] Xilinx 7 Series DSP48E1 Slice https://www.xilinx.com/support/ documentation/user guides/ug479 7Series DSP48E1.pdf

[2] Xilinx LogiCORE IP v12.0 https://www.xilinx.com/support/ documentation/ip documentation/mult gen/v12 0/pg108-mult-gen.pdf [3] Integer Arithmetic IP Cores User Guide https://www.altera.com/en US/ pdfs/literature/ug/ug lpm alt mfug.pdf

[3] N. Brunie, F. de Dinechin, M. Istoan, G. Sergent, K. Illyes and B. Popa, ”Arithmetic core generation using bit heaps,” 2013 23rd International Conference on Field programmable Logic and Applications, Porto, 2013, pp. 1-8.

[4] J. Beuchat and J. Muller, ”Automatic Generation of Modular Multipliers for FPGA Applications,” in IEEE Transactions on Computers, vol. 57, no. 12, pp. 1600-1613, Dec. 2008.

[5] Ahmet Kakacak, Aydin Emre Guzel, Ozan Cihangir, Sezer Gren, and H. Fatih Ugurdag. 2017. ”Fast Multiplier Generator for FPGAs with LUT based Partial Product Generation and Column/row Compression,”. in Integr. VLSI J. 57, C 2017, 147-157.

[6] M. Kumm, J. Kappauf, M. Istoan and P. Zipf, ”Resource Optimal Design of Large Multipliers for FPGAs,” 2017 IEEE 24th Symposium on Computer Arithmetic (ARITH), London, 2017, pp. 131-138.

[7] E. G. Walters, ”Array Multipliers for High Throughput in Xilinx FPGAs with 6-Input LUTs” in Computers, vol. 5, no. 4, 2016.

[8] M. Kumm, S. Abbas and P. Zipf, ”An Efficient Softcore Multiplier Architecture for Xilinx FPGAs,” 2015 IEEE 22nd Symposium on Computer Arithmetic, Lyon, 2015, pp. 18-25.

[9] H. Parandeh-Afshar and P. Ienne,” Measuring and Reducing the Performance Gap between Embedded and Soft Multipliers on FPGAs,” 2011 21st International Conference on Field Programmable Logic and Applications, Chania, 2011, pp. 225-231.

[10] 7 Series FPGAs Configurable Logic Block https://www.xilinx.com/ support/documentation/user guides/ug474 7Series CLB.pdf

[11] H. Parandeh-Afshar, P. Brisk and P. Ienne, ”Exploiting fast carry-chains of FPGAs for designing compressor trees,” 2009 International Conference on Field Programmable Logic and Applications, Prague, 2009, pp. 242- 249.

[12] Rivest, Ronald L., Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems." Communications of the ACM 21.2 (1978): 120-126.R. L. Rivest, A. Shamir, L. A. Adleman. 1978. pp. 120-126.

[13] Diffie, Whitfield, and Martin Hellman. "New directions in cryptography." IEEE transactions on Information Theory 22.6 (1976): 644-654.Esmaeildoust M. Efficient RNS Implementation of Elliptic Curve Point Multiplication Over GF(p) / M. Esmaeildoust,

[14] Alioto, Massimo. "Editorial on the Opening of the New Editorial Year—The State of the IEEE Transactions on Very Large Scale Integration (VLSI) Systems." IEEE Transactions on Very Large Scale Integration (VLSI) Systems 28.1 (2019): 1-2.

[15] Fournaris, Apostolos P., et al. "An RNS implementation of an Fp elliptic curve point multiplier." IEEE Transactions on Circuits and Systems I: Regular Papers (2009)..

[16] Schinianakis, Dimitrios, and Thanos Stouraitis. "Multifunction residue architectures for cryptography." IEEE Transactions on Circuits and Systems I: Regular Papers 61.4 (2014): 1156-1169.

[17] Chervyakov, Nikolay I., et al. "Comparison of modular numbers based on the chinese remainder theorem with fractional values." Automatic Control and Computer Sciences 49.6 (2015): 354-365. In the meantime, there is no need to worry about it. ”

[18] Wade, Andrew D., et al. "Diagnosis by consensus: a case study in the importance of interdisciplinary interpretation of mummified remains." International journal of paleopathology 24 (2019): 144-153.

[19] M. El Hajjar and L. Hanzo, “A survey of digital television broadcast transmission techniques,” IEEE Commun. Surv. Tut., vol. 15, no. 4, pp. 1924–1949, Fourth Quarter 2013.

[20] O. Bello and S. Zeadally, “Intelligent device-to-device communication in the Internet of Things,” IEEE Syst. J., vol. 10, no. 3, pp. 1172–1182, Sep. 2016.