EXTENDING THE INSTRUCTION SET OF RISC-V PROCESSOR FOR NTRU ALGORITHM

PROJECT REPORT

XILINX OPEN HARDWARE 2020

Team Number:
#128

Participants:

Canberk Topal & Elif Nur İşman

Supervisor:

Sıddıka Berna Örs Yalçın

Table of Contents

1. INTRODUCTION

With the developing technology, interest and investments in developing quantum computers are increasing rapidly[1, 2]. However, this poses a threat to cryptography algorithms used in every system where information security is needed today. The quantum era requires fundamental changes in information security. New cryptography algorithms that can resist post-quantum computers are being developed in order to maintain information security in banking, military and many other areas. In order to be usable and practical in daily life, low area usage and low performance are prioritized in the algorithms created.

N-Truncated Polynomial Ring(NTRU)[3] is one of the most promising post-quantum cryptography algorithms as they are among 28 standardization candidates in the National Institute of Science and Techonology(NIST) competition for public key cryptography[4].

Based on the Reduced Instruction Set Computer (RISC) architecture[5], RISC-V[6] is an open source alternative to a world of proprietary instruction set architectures. Our project aims to increase the performance of a NTRU cryptosystem application on an open source, low-power RISC-V processor. The plan is to increase the performance by extending the instruction set with most commonly used operations in the application.

2. MATHEMATICAL BACKGROUND

The instructions are decided regarding the mathematical base of the project. For which can be mainly divided into post-quantum cryptography and N-Truncated Polynomial Ring (NTRU)[7].

2.1 Post Quantum Cryptography

The most used cryptographic algorithms, such as Rivest-Shamir-Adelman (RSA)[8], Digital Signature Algorithm (DSA) [9] and Elliptic Curve Digital Signa-ture Algorithm (ECDSA) [10] are forming the basis of the today's security sys-tems. However, with the quantum computers being developed, these algorithms will inevitably become vulnerable as proven with Shor's Algorithm [11]. Many cryptosystems with new fundamentals that can withstand said algorithm and many others have been invented due to security concerns raised by quantum computers.

Hash-Based Cryptography: Cryptographic algorithms based on the security of hash functions [12]. A negative side of the hash-based signature schemesis the track of the signed messages must be kept without any leak, since wrongcount will cause insecurity. Another negative side effect is, if signature size is fixed, creation of signatures is limited [13].

Code-Based Cryptography: Security system that algorithmic basis usesan error correcting linear code [14]. An error word or parity check matrix incomputing can be added to it. The classic example is McEliece's hidden-Goppa-code public-key encryption system (1978) [4, 15].

Lattice-Based Cryptography: Similar to other classes, lattice-based cryptography depends on the hypothetic intractability of a mathematical system [16]. Lattice problems like shortest vector problem, closest vector problem and vari-ants of them.

May not be the first example in history but NTRU possibly attracted themost interest [17].

Multivariate-Quadratic Equations Cryptography: Asymmetric cryptographic primitives based on multivariate polynomials over a finite field [4,18].

2.2 N-Truncated Polynomial Ring

NTRU differs from the previously found public key cryptosystems by the foun-dations it is based on which is the shortest vector problem in a lattice [3]. NTRU is shown as an alternative to RSA [8] and Elliptic Curve Cryptography(ECC) [19] by using a lattice-based approach to cryptography. A truncated polynomial ring $R = Z[X]/(X_{N-1})$ that is created based on thedetermined parameters form the backbone of the steps in NTRU cryptosystem. During the process, different polynomials are created by using theRand all ofthem have to have integer coefficients and degree at most N−1.

$$P(x) = a_0 + a_1x + a_2x_2 + \ldots + a_{N-1}x_{N-1} (1)$$

NTRU Keys and Parameters:

N- the polynomials in the ring R have degree N−1. (Non-secret)

q- the large modulus to which each coefficient is reduced. (Non-secret)

p- the small modulus to which each coefficient is reduced. (Non-secret)

f- a polynomial that is the private key.

g- a polynomial that is used to generate the public key h from f (Secret but discarded after initial use)

h- the public key, also a polynomial

r- the random “blinding” polynomial (Secret but discarded after initial use)

d- coefficient

Key Generation:

If two person named Alice and Bob are communicating through a securechannel, sending a secret message from Alice to Bob requires the generation ofa public and a private key. While the public key is known by both sides, privatekey should only be known by the receiver [20].As the first step, two polynomials named f and g in the R

are selected randomly. Chosen polynomials with degree at most N−1 with coefficients [−1,0,1]must be invertible. Then, the inverse offaccording to moduloq(f_q) and modulop(f_p) should be calculated. Operations, especially in the decryption part,will be depend on the f_q and f_p satisfying the equations:

$$f * f_q = 1 \, mod q \quad (2)$$

and

$$f * f_p = 1 \, mod p \quad (3)$$

In the third step, public key h will be calculated with the equation $h = p * (f_q * g) \, mod q$. F and f_p are used to create a longer and protected private key.

Encryption:

As the beginning, message to be transmit will put in the form of polynomialand represented with m, with coefficients [−1,0,1].

$$m = 1 - X_2 + X_5 - X_7 + X_{10} \quad (4)$$

Than, a 'blinding value' is chosen randomly to obscure the message. Blindingvalue is a small polynomial that represented with r.

$$r = 1 + X_1 + X_2 - X_3 - X_9 \quad (5)$$

Last step of the encryption is to calculating the encrypted message by the equa-tion:

$$e = r * h + m \, mod \, q \quad (6)$$

Decryption:

Private key that Bob have is the combination off and f_p, as mentioned before, Private key is the only information Bob has aside from the encrypted message. He can try to solve the message by using f. First, he multiplies theeandf, represent the result with a polynomial a.

$$a = f * e \, mod \, q \quad (7)$$

If equation is rearranged with the equality of e:

$$a = f * (r * h + m) \, mod \, q \quad (8)$$

$$a = f * (r * pf_q * g + m) \, mod \, q \quad (9)$$

$$a = pr * g + f * m \, mod \, q \quad (10)$$

Instead of choosing the coefficients of a between 0 andq–1, they are chosenin the interval $[-_{q2},_{q2}]$. Aim of this is to prevent that the original message maynot be properly recovered since Alice chooses the coordinates of her messagemin the interval $[-_{p2},_{p2}]$. Next step will be calculatingamodulop, result will berepresented with polynomial b:

$$b = a \; mod \; p \quad (11)$$

Since modulo of p r∗g equals to 0,

$$b = f * m \; mod \; p \quad (12)$$

Now, Bob can use the f_p to recapture m, by multiplication of band f_p.

$$c = f_p * b = f_p * f * m \; mod \; p \quad (13)$$

$$c = m \; mod \; p \quad (14)$$

3. INSTRUCTION SET EXTENSION OF RISC-V PROCESSOR

The main goal of the work is to prove a group of vector type of extensions in theISA of the core would increase the performance for the NTRU Cryptosystem byenabling the parallelization of some operations. This would be especially usefulin small embedded applications that uses post-quantum cryptography

3.1 Implementing NTRU Cryptosystem on C Programming Language

C implementation of NTRU Cryptosystem has three main operational functions, which are polymult, polymult2 and polydiv. These three operational functions are used in an ”Extended Euclidean” function. Extended Euclidean algorithm is used for finding the greatest common divisor between two polynomials. All of the mentioned functions involves heavy use of arrays. These arrays are used for implementing the coefficients of the polynomials in such a form that it is easy to do operations between them. Key generation, encryption and decryption main functions have its determined parameters as N= 53, M= 48, p= 3 and q= 101. Predetermined values for secret keys f and g are inside of the key generation function as well. In the main function of the code, first a key is generated than it is used

to encrypt a message and after that encrypted message is decrypted using the generated public key. That means the whole cryptosystem is tested in a single code.

After the implementation of NTRU in C code, the implementation of the core and the research for the extension of instruction set is started.

3.2 Implementing an Open Source RISC-V Processor on FPGA

32-bit Ibex [21] core is chosen as a suitable RV32IMC core because of its hardwaredesign language, SystemVerilog [22], and detailed documentation. Architectureof Ibex can be seen in Fig.1

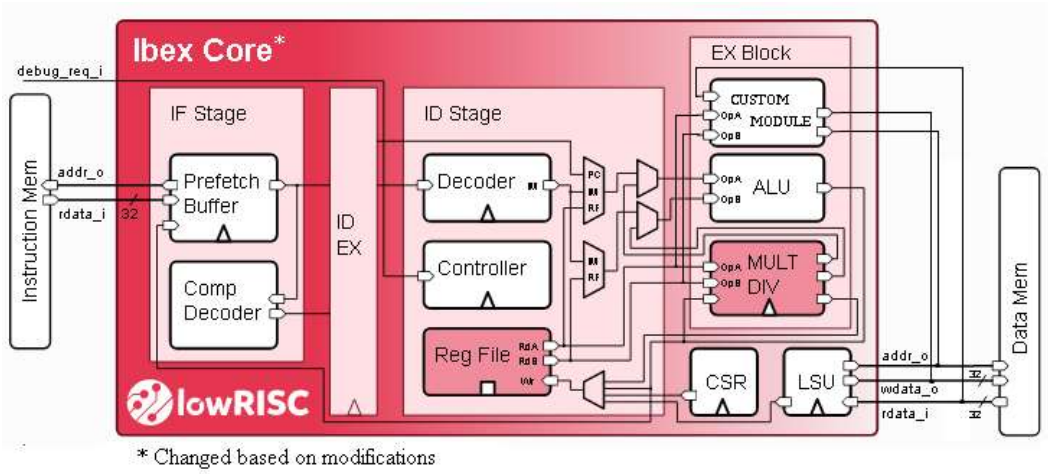

Figure 1: Modified Architecture of IBEX

Xilinx Vivado 2018.1 [23], CMake [24] and other dependencies are installed for the generation of the project files. In the first few months of the project, the project is only used in behavioral simulation for the functionality tests of the core. RV32IMC compiler [25] is built on Ubuntu version 16.04 [26]. This enabled the C code to be compiled on the computer for the core itself. The generated memory file initialized in the memory section of the implementation. Whole project consists of:

–Clock generator for the synchronous processes,

–Core itself,

–Xilinx Internal Logic Analyzer (ILA) [27] for debugging and performance analysis in real-time,

–A memory block as both data and instruction memory

Next step is to write some basic algebraic processes in C and test them in behavioral simulation of the project. Purpose of writing basic codes is to understand the decoding operation of the core and change it so that the new instructions could be fetched and decoded by the core flawlessly. RISC-V ISA makes room for custom instruction and designs them in a way that they can be easily configured[28]. For explaining in detail, one should mention the terms Opcode Space, instruction match and instruction mask terms. Due to not having enough computing power to fully simulate the performance-wise heavy codes, it was needed to program a FPGA to do the operations inreal time. The chosen FPGA was Nexys4-DDR[29] due to its availability in theworking environment. An ILA is also used for the debugging and performance analysis of the project. Detailed explanations for the performance analysis will be mentioned in the later parts.

3.3 Extending Instruction Set of RV32IMC for Multicycle Instructions

Opcode Space is a term for a group of instructions that have the same instruction type and enable the same sort of response in the core to some degree. For example, all 32-bits ALU instructions are in the same opcode space, OP_32[28]. There is also three different opcode spaces for customized instructions. In this project, used the opcode space CUSTOM_0.

3.3.1 Adding Custom Instructions in RISC-V ISA

There are different instruction formats for different needs and for a custom instruction, R-Type instruction format is chosen due to it is common use in ALU related instructions. Modifying the compiler to include the custom function and adding a *.insn* directive to the source code is considered as two different options for changing the machine code of the project. Between these two options, using *.insn* directive is picked for its ease of use. The directive is an assembly directive; in order to combine it with the C code, inline assembly method is used. For 32-bits RISC-V ISA, an instruction is two or four bytes in length and must be aligned on a 2 byte boundary. Compressed(C) extension is designed so that the compressed instructions are two bytes. However, other instructions are four bytes. R-Type instructions has a structure as shown in Fig.2

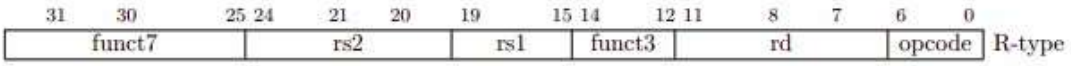

Figure 2: Instruction Format of R-Type Instructions

For CUSTOM_0 opcode space, *funct3* region of the instruction is designed to specify the format. 3'b111 signal shows that the instruction has two source registers and one destination register. Since it is implemented in inline assembly method, registers can be appointed automatically by % symbol. As for the *funct7* region, it is used for specifying the individual instructions in the same opcode space. In the project, 0x03 implies array addition, 0x05 implies array to array equalization and 0x06 implies element-wise modulus operation.

3.3.2 Profiling As A Method For Deciding Candidate Instructions

All three instructions are chosen with regards to their potential improvements on the NTRU implementation. These potential improvements are first measured by theusage frequency of the said operations. These profiling of the code is done bya script that uses a counter to record how many times a particular part of thefunction is called. After the profiling step, candidate instructions are tested ina benchmark C code so that the hardware changes for the instructions can bemeasured efficiently.

Previous versions of the example code parts and updated versions can beseen in Figure 3. This C functions would take array addresses and length of the operation for the instruction to be constructed in a way that complies with the designed hardware. Limitations caused by RISC-V ISA, forced the hardware design to include a constant parameter because there is not any information found about a method that enables the use of multiple operands. This parameter decides how many array elements are processed in a single instruction. For the area performance concerns, this parameter is chosen as three. This choice also affects the calling frequency of the instruction consequently. In order to automatize this, same instruction is being sent size/3 times while each time changing the input addresses accordingly.

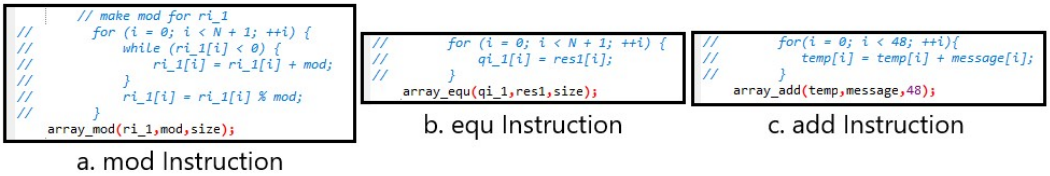

Figure 3: Changes in the C Code for Executing Custom Instructions

3.4 Modification of the Core

The candidate instructions for extension is chosen and according to the choicesmodification in the core are mainly done to the execution stage of the structure. In order to not get the illegal opcode error in the core itself, first thing to dois introducing the custom instructions to the instruction decoding stage of the core. This is done by adding a new state in the decoder hardware of the core forthe specific opcode space CUSTOM_0. Whenever that opcode space is decoded in the decoder, an enable signal and the specific opcode of the instruction is sentto the added module in the execution stage. In the execution stage of the core, there are two modules present in the vanilla version. First one is Arithmetic Logic Unit(ALU) and another module is for the multiplication and division module MUL\DIV. ALU is used for the single cycle operations while MUL\DIV is used for multi cycle operations. In this project, a new custom module is added to the execution stage for enabling the array operations in the core. Main idea of these operations is that any process that reaches the memory structure and pulls multiple data then executes the same operation has to be multi clock cycle. That is because of the nature of the memory structure would allow only one data to be read in one clock cycle. The core already has multi clock cycle instructions from MUL\DIV module. An unfinished instruction should send a signal to the instruction fetch stage of the core, the reason behind it is that if the fetch stage continues to work after one clock cycle, decoder receives another instruction and the core moveson to the next instruction to fetch. That will conclude with erroneous results. The control signals inside of instruction fetch stage are modified to include thecustom module enabling signals and another signal that indicates the validity ofthe custom module outputs. Whenever the custom module is enabled, core would enter a waiting stage just like if it receives a MUL\DIV enabling instruction. Moreover, parallel to MUL\DIV module when the custom valid signal is high the core would start to fetch and decode instruction from where it left off

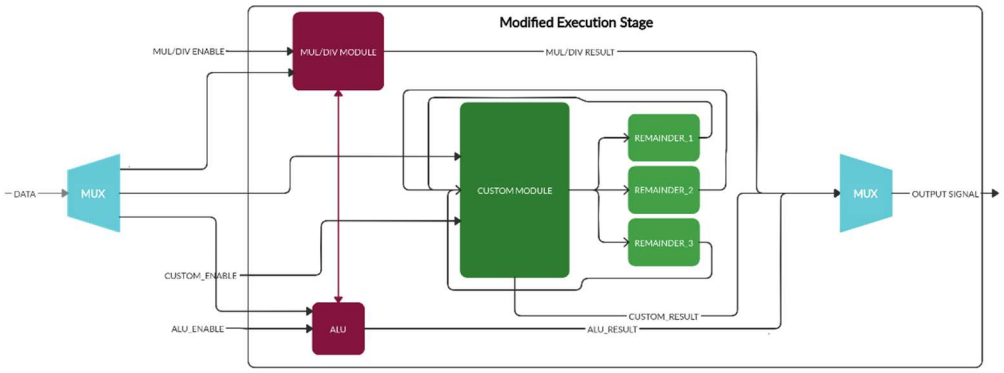

Figure 4: Diagram of Modified Execution Stage

Two new modules are added in the execution stage of the core. Input signals of the execution stage is divided between the original modules and the custom module. Outputs of the execution stage is chosen in respect to the enable signal that comes from the decoder. The diagram that summarizes the connection between the sub-modules of execution stage module is shown in the Figure 4. Remainder module is a module that is used for doing the modulo operation using the non-restoring division algorithm implementation[30]. The reason for not using the inherent REM instruction of RISC-V is that the core itself use both ALU and MUL\DIV modules to execute REM instruction. Thus, parallelization of this structure is costly in terms of area usage. In order to increase the performance by doing the same operation in the same time, three instances of this module is generated in the execution stage.Another module named custom module acts like a driver between the memoryof the system and the remainder module. Also, it does simple algebraic computations like additions and equalization. Main structure of the module consists ofeight states. Which are shown in Figure 5.

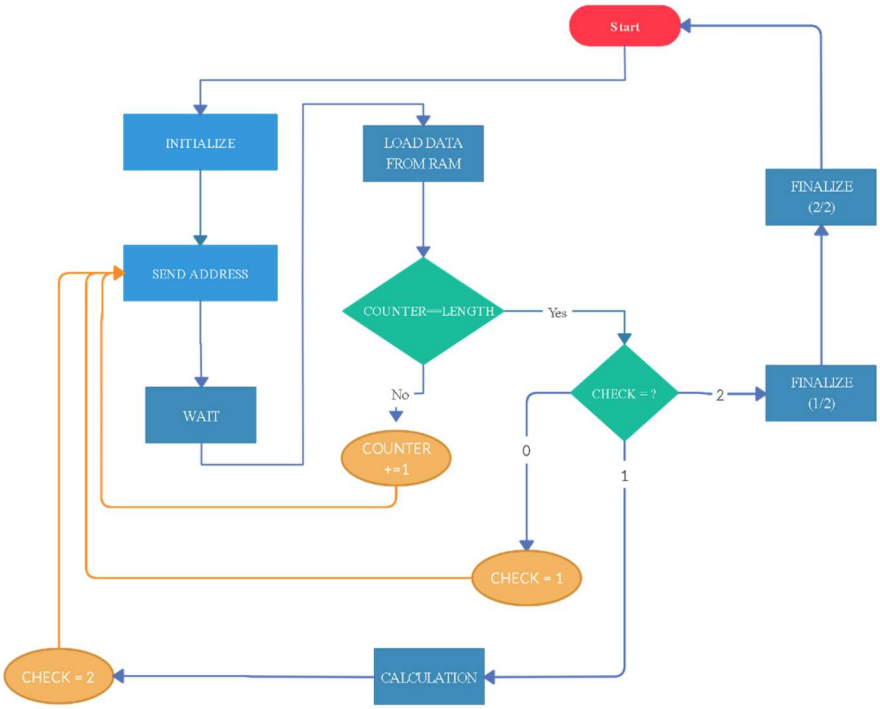

Figure 5: State Flowchart for Custom Module

State zero initializes the counter, temporary registers etc. First state is forsending the address information to the memory and retrieve the corresponding data. In order to read from the memory, one clock cycle has to be passed. So, second state is used for that delay. Third state is for loading the incoming data into local registers datareg1, datareg2. This loading sequence is repeated until the capacity of the local registers are full. In this project, it is decided to use three data at a time because of the trade-off between area and increase in performance. Fourth state is for sending the modulus operands to remainder instances or doing the addition. Action in the fifth state is determined by the unique opcode of the instruction. In the fifth state results are stored in another local register called datareg3. Contents of this register is sent to the memory of the system by changing the control signal checkand return the module to state 1. After valid signal is set to high, the module goes into the end routine, from which it sends the control signals to the core to end the multi cycle waiting process. Also, it gives out the final result to the destination register as intended.

3.5 Performance Anaylsis Method

After the behavioral simulations on the trivial C code examples, NTRU implementation has to be tested in the real-world conditions because heavy computing need of the behavioral simulation would make it practically impossible to simulate the whole operation. In order to solve this problem, a FPGA card is used to implement the whole project and run the C code. The memory structure of the project is quite simple, it has the same memory for both data and instructions. All global arrays that are in the C code is saved in a constant address in the memory. ILA is a module that was used for measuring and reading the values from the project. In order to see the memory input and core output of the project, one probe of the ILA is connected to the wire that connects both of them in the top module.

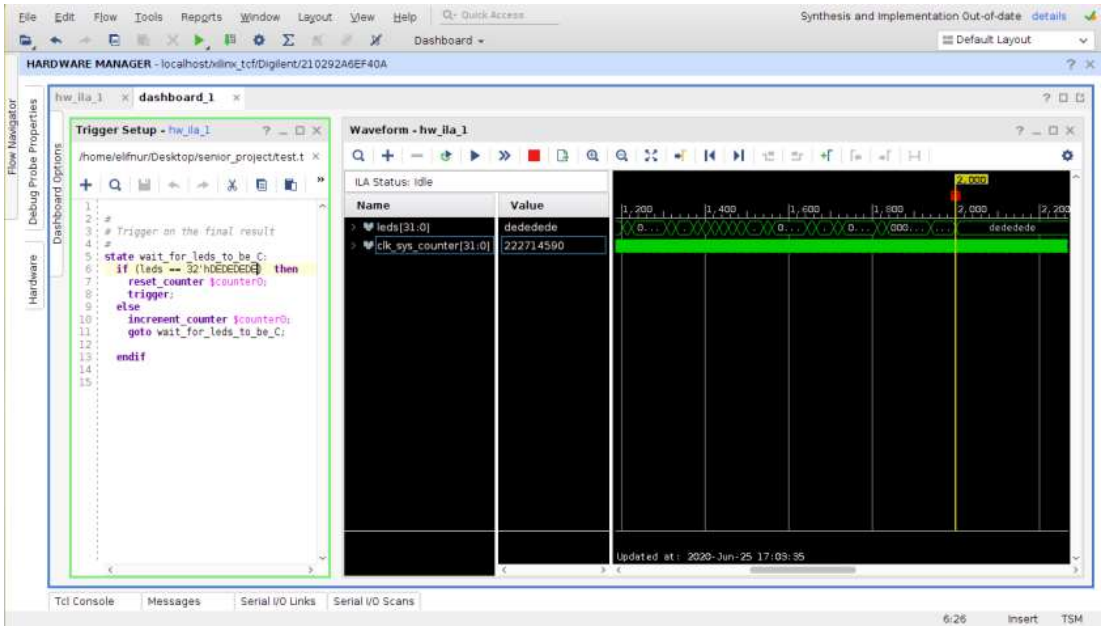

Figure 6: Performance Analysis is done with this interface

As mentioned before, main function of the C code includes both key generation, encryption and decryption algorithms for NTRU. In an effort to measure the performance improvements that are achieved for individual parts of the cryptosystem, a control signal is inserted at the beginning and the end of the function.By simply checking the number of clock cycles or time between the two uniquely designated signals, an accurate measurement is concluded.

4. RESULTS

Performance measurements are made with the previously mentioned method. First measurement was the unmodified NTRU code with the modified core. Array equalization instruction made an unexpected increase in the clock cycles. It is possible that it is because of the disruption of the optimization process of the compiler, caused by the new instruction. However, the overall effect of that instruction in the last configuration shows that it can be a helpful in decreasing the clock cycles in some combinations. It is also worth mentioning that there was no change in the working frequency of the core in the case of modified version.

Added Instruction	Clock Cycles	Improvement
VANILLA	329,281,688	-
MOD	255,087,137	−22.53%
ADD	310,181,046	−5.80%
EQU	345,059,480	+4.79%
MOD + ADD	224,556,529	−31.80%
MOD + EQU	321,277,395	−2.43%
ADD + EQU	312,095,182	–5.22%
MOD + ADD + EQU	222,705,262	-32.37%

Table I: Performance Results

The area measurements are made by implementing unmodified Ibex core and comparing the utilization report of it with the modified core. Increase in the usage of Look-up tables (LUT) and flip-flops (FF) are shown in the Table II.

Added Instruction	LUT	Increase (+%)	FF	Increase (+%)
VANILLA	2991	-	1923	-
MODIFIED	3744	+25.18%	2929	+52.31%

Table II: Area Usage Results

While the number of clock cycles are decreasing the critical path of the whole system is does not change significantly. Results of this project shows that with a directly connected data memory and a custom driver for the data handling in execution stage would improve the overall performance of the core for NTRUcryptosystem implementation.

REFERENCES

1. N. S. Yanofsky and M. A. Mannucci, *Quantum Computing for Computer Scientists*, 1st ed. Cambridge University Press, 11 Aug 2008.
2. M. Mohseni, P. Read, H. Neven, S. Boixo, V. Denchev, R. Babbush, A. Fowler, V. Smelyanskiy, and J. Martinis, "Commercialize quantum technologies in five years," *Nature*, vol. 543, no. 7644, pp. 171–174, 2017.
3. J. Hoffstein, J. Pipher, and J. H. Silverman, "Ntru: A ring-based public key cryptosystem," in *International Algorithmic Number Theory Symposium*. Springer, 1998, pp. 267–288.
4. G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, Y. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, "Status report on the first round of the nist post-quantum cryptography standardization process," National Institute of Standards and Technology, Tech. Rep. 8240, January 2019.
5. S. B. Furber, *VLSI RISC Architecture and Organization*. Routledge, 19 Sep 2017.
6. D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition*, 1st ed. Morgan Kaufmann, 12 May 2017.
7. A. Kamal and A. M. Youssef, "An fpga implementation of the ntru encrypt cryptosystem," in *2009 International Conference on Microelectronics - ICM*, 2009, pp. 209–212.
8. R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signa- tures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
9. D. Johnson and A. Menezes, "The elliptic curve digital signature algorithm (ECDSA)," Department of Combinatorics & Optimization, University of Waterloo, Canada, Tech. Rep. CORR 99-34, February 24 2000, http://www.cacr.math.uwaterloo.ca.
10. A. X9.62, "The Elliptic Curve Digital Signature Algorithm (ECDSA)," http:www.ansi.org.

11. P. W. Shor, “Polynomial-time algorithms for prime factorization and dis- crete logarithms on a quantum computer,” *SIAM Journal on Com- puting*, vol. 26, no. 5, p. 1484–1509, Oct 1997. [Online]. Available: http://dx.doi.org/10.1137/S0097539795293172

12. J. Buchmann, E. Dahmen, and M. Szydlo, *Post-quantum cryptography*. Springer, Berlin, 2009, ch. Hash-based digital signature schemes, pp. 35 – 93.

13. L. Chen, S. Jordan, Y. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith- Tone, “Report on post-quantum cryptography,” National Institute of Standards and Technology, Tech. Rep. 8105, April 2016.

14. R. Overbeck and N. Sendrier, *Code-based cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 95–145. [Online]. Available: https://doi.org/10.1007/978-3-540-88702-7 4

15. R. J. McEliece, “A public-key cryptosystem based on algebraic coding theory,” Jet Propulsion Laboratory, Tech. Rep., 1978.

16. D. Micciancio and O. Regev, *Lattice-based Cryptography*. Berlin, Heidel-berg: Springer Berlin Heidelberg, 2009, pp. 147–191. [Online]. Available: https://doi.org/10.1007/978-3-540-88702-7 5

17. J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko,Raghunathan, and D. Stebila, “Frodo: Take off the ring! practical, quantum- secure key exchange from lwe,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1006–1018. [Online]. Available: https://doi.org/10.1145/2976749.2978425

18. Y.-J. Huang, F.-H. Liu, and B.-Y. Yang, “Public-key cryptography from new multivariate quadratic assumptions,” in *Public Key Cryptography – PKC 2012*, M. Fis-chlin, J. Buchmann, and M. Manulis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 190–205.

19. C. Paar, “Implementation options for finite field arithmetic for elliptic curve cryptosystems,” Presented at The 3rd workshop on Elliptic Curve Cryptography (ECC 1999), November 1-3 1999, http://www.cacr.math.uwaterloo.ca/conferences/1999/ecc99/slides.html.

20. K. Sako, *Public Key Cryptography*. Boston, MA: Springer US, 2011, pp. 996–997.[Online]. Available: https://doi.org/10.1007/978-1-4419-5906-5 22

21. *Ibex Documentation*, lowRISC, April 22 2020, https://ibex- core.readthedocs.io/ /downloads/en/latest/pdf/.

22. SystemVerilog, "Ieee standard for systemverilog–unified hardware design, specification, and verification language - redline," *IEEE Std 1800-2009 (Revision of IEEE Std1800-2005) - Redline*, pp. 1–1346, 2009.

23. *Vivado Design Suite User Guide*, Xilinx, Inc., April 4 2018, https://www.xilinx.com/support/documentation/sw manuals/xilinx2018 1/ug910-vivado-getting-started.pdf.

24. CMake, "Cross platform make," https://cmake.org/cmake/help/v3.3/index.html.

25. RISC-V, "Gnu compiler toolchain," https://github.com/riscv/riscv-gnu-toolchain.

26. Ubuntu, "Ubuntu 16.04 documentation," https://help.ubuntu.com/16.04/ubuntu-help/index.html.

27. *Integrated Logic Analyzer v6.2*, Xilinx, Inc.,October 5 2016, https://www.xilinx.com/support/documentation/ip_documentation/ila/v6_2/pg172-ila.pdf.

28. A. Waterman, Y. Lee, D. Patterson, and K. Asanovic, "The RISC-V instruction set manual," 2016.

29. *Nexys4 DDR FPGA Board Reference Manual*, Digilent, Inc., April 11 2016, https://reference.digilentinc.com/media/reference/programmablelogic/nexys4-ddr/nexys4ddrrm.pdf.

30. Y. Yusmardiah, D. Mohd, A. Karimi, A. Abdul, and A. Kamsani, "Translation of division algorithm into verilog hdl," *ARPN Journal of Engineering and Applied Sciences*, vol. 12, pp. 3214–3217, 05 201