ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΣΥΣΤΗΜΑΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση αλγορίθμου μετα-κβαντικής κρυπτογραφίας

Δ Ι Π Λ Ω Μ Α Τ Ι Κ Η Ε Ρ Γ Α Σ Ι Α

Γιαννόπουλος Γιώργιος

ΕΠΙΒΛΕΠΩΝ: καθ. Σερπάνος Δημήτριος

ΠΑΤΡΑ – Ιούλιος 2026

Πανεπιστήμιο Πατρών, Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών.

Γιαννόπουλος Γεώργιος

© 2026 – Με την επιφύλαξη παντός δικαιώματος

Το σύνολο της εργασίας αποτελεί πρωτότυπο έργο, παραχθέν από τον Γιαννόπουλο Γεώργιο, και δεν παραβιάζει δικαιώματα τρίτων καθ' οιονδήποτε τρόπο. Αν η εργασία περιέχει υλικό, το οποίο δεν έχει παραχθεί από τον ίδιο, αυτό είναι ευδιάκριτο και αναφέρεται ρητώς εντός του κειμένου της εργασίας ως προϊόν εργασίας τρίτου, σημειώνοντας με παρομοίως σαφή τρόπο τα στοιχεία ταυτοποίησής του, ενώ παράλληλα βεβαιώνει πως στην περίπτωση χρήσης αυτούσιων γραφικών αναπαραστάσεων, εικόνων, γραφημάτων κ.λπ., έχει λάβει τη χωρίς περιορισμούς άδεια του κατόχου των πνευματικών δικαιωμάτων για την συμπερίληψη και επακόλουθη δημοσίευση του υλικού αυτού.

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η Διπλωματική Εργασία με τίτλο

Υλοποίηση αλγορίθμου μετα-κβαντικής κρυπτογραφίας

του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Γεώργιου Γιαννόπουλου ΤΟΥ **Κωνσταντίνου**

Αριθμός Μητρώου: 1020849

Παρουσιάστηκε δημόσια στο Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις

01/07/2026

και εξετάστηκε από την ακόλουθη εξεταστική επιτροπή:

Σερπάνος Δημήτριος, Καθηγητής, ΗΜ/ΤΥ (επιβλέπων)

Θεωδορίδης Γεώργιος, Αναπληρωτής Καθηγητής, ΗΜ/ΤΥ (μέλος επιτροπής)

Μπίρμπας Μιχαήλ, Καθηγητής, ΗΜ/ΤΥ (μέλος επιτροπής)

Ο Επιβλέπων	Ο Διευθυντής του Τομέα
Σερπάνος Δημήτριος Καθηγητής	Θεοδωρίδης Γεώργιος Αναπληρωτής Καθηγητής

ΠΡΟΛΟΓΟΣ

Ολοκληρώνοντας την παρούσα εργασία, θα ήθελα να ευχαριστήσω την οικογένεια μου για την στήριξη όλα αυτά τα χρόνια, Κώστα, Αργυρώ , Δημήτρη, χωρίς εσάς δεν θα ήμουν εδώ.

Επίσης, θα ήθελα να ευχαριστήσω τους καθηγητές, καθηγήτριες και φίλους, φίλες συμφοιτητές και συμφοιτήτριες , κάθε ένας συνέδραμε με τον τρόπο του στο να καταστεί αυτή η πανεπιστημιακή εμπειρία αξιομνημόνευτη.

Σας ευχαριστώ.

ΠΕΡΙΛΗΨΗ

Υλοποίηση αλγορίθμου μετα-κβαντικής κρυπτογραφίας

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΦΟΙΤΗΤΗ: **Γιαννόπουλος Γεώργιος**

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΕΠΙΒΛΕΠΟΝΤΟΣ: **Σερπάνος Δημήτριος**

Η ανάδυση μεγάλης κλίμακας κβαντικών υπολογιστών αποτελεί θεμελιώδη απειλή για την ασφάλεια των κλασικών κρυπτογραφικών σχημάτων δημοσίου κλειδιού, συμπεριλαμβανομένων του RSA και της κρυπτογραφίας ελλειπτικών καμπυλών (ECC). Ο αλγόριθμος Shor [6], εκτελούμενος σε επαρκώς ισχυρό κβαντικό υπολογιστή, επιλύει το πρόβλημα παραγοντοποίησης ακεραίων και το πρόβλημα του διακριτού λογαρίθμου σε πολυωνυμικό χρόνο, καθιστώντας τα κρυπτοσυστήματα RSA και ECC ανασφαλή. Ως απόκριση, αναπτύχθηκε η Μετα-Κβαντική Κρυπτογραφία (Post-Quantum Cryptography, PQC), στην οποία εντάσσεται το κρυπτοσύστημα NTRU — ένα από τα παλαιότερα και πλέον αναλυμένα κρυπτοσυστήματα βασισμένα σε πλέγματα.

Η παρούσα διπλωματική εργασία παρουσιάζει μια πλήρη υλοποίηση του κρυπτοσυστήματος NTRU-HPS (N=509, p=3, q=2048, επίπεδο ασφάλειας ≈128 bit) σε FPGA, αναπτυγμένη με χρήση του Vivado / Vitis High-Level Synthesis (HLS) 2024.2 της AMD Xilinx. Η υλοποίηση σε C/C++ καλύπτει τις τρεις κεντρικές λειτουργίες του κρυπτοσυστήματος: παραγωγή κλειδιών (keygen), κρυπτογράφηση (encryption) και αποκρυπτογράφηση (decryption). Ο κώδικας βελτιστοποιείται μέσω pragma οδηγιών HLS, συμπεριλαμβανομένων των HLS PIPELINE (II=1), HLS ARRAY_PARTITION CYCLIC και HLS DATAFLOW. Η συσκευή-στόχος είναι το FPGA Xilinx Zynq UltraScale+ ZCU104 (XCZU7EV-FFVC1156-2-E).

Κατά τη διαδικασία ανάπτυξης εντοπίστηκαν και διορθώθηκαν δύο κρίσιμα μαθηματικά σφάλματα: (α) η κατασκευή του ιδιωτικού κλειδιού f, η οποία πρέπει να ακολουθεί τη μορφή f = 1 + p·F (αντί απλού τριαδικού πολυωνύμου) για να εξασφαλίζεται η αντιστρεψιμότητά του στον δακτύλιο R_p, και (β) ο αλγόριθμος αντιστροφής πολυωνύμων, ο οποίος απαιτεί πλήρη πολυωνυμική διαίρεση (long division with remainder) σε κάθε βήμα του Εκτεταμένου Αλγόριθμου Ευκλείδη (EEA) για ορθά αποτελέσματα σε GF(p)[x]/(x^N - 1). Η ορθότητα της υλοποίησης επαληθεύεται μέσω πλήρους κύκλου keygen → encrypt → decrypt (C-simulation), ο οποίος επιστρέφει PASS για τυχαία δεδομένα ελέγχου.

Τα αποτελέσματα σύνθεσης αξιολογούνται ως προς κατανάλωση πόρων FPGA (LUTs, FFs, DSPs, BRAM), μέγιστη συχνότητα λειτουργίας και καθυστέρηση (latency), και συγκρίνονται με σχετικές προηγούμενες εργασίες της βιβλιογραφίας. Τα αποτελέσματα αποδεικνύουν ότι το Vitis HLS 2024.2 αποτελεί αποτελεσματική μεθοδολογία για πρωτοτυποποίηση κρυπτογραφικού υλικού PQC, επιτυγχάνοντας σημαντική επιτάχυνση σε σχέση με το λογισμικό αναφοράς, με μικρότερο κόστος ανάπτυξης σε σχέση με χειροκίνητο RTL σχεδιασμό.

Λέξεις-κλειδιά: NTRU-HPS, Μετα-Κβαντική Κρυπτογραφία, Κρυπτογραφία Πλεγμάτων, FPGA, High-Level Synthesis, Vitis HLS, Schoolbook Convolution, Extended Euclidean Algorithm, Hensel Lifting, Zynq UltraScale+.

EXTENSIVE ENGLISH SUMMARY

Development of a Post Quantum Cryptography Algorithm

STUDENT NAME, SURNAME:
Giannopoulos Georgios

SUPERVISOR NAME, SURNAME:
Serpanos Dimitrios

The emergence of large-scale quantum computers constitutes a fundamental threat to the security of classical public-key cryptographic schemes, including RSA and Elliptic Curve Cryptography (ECC). Shor's algorithm [6], when executed on a sufficiently powerful quantum computer, solves the integer factorization problem and the discrete logarithm problem in polynomial time, rendering the RSA and ECC cryptosystems insecure. In response, Post-Quantum Cryptography (PQC) has been developed, which encompasses the NTRU cryptosystem — one of the oldest and most thoroughly analyzed lattice-based cryptosystems.

This thesis presents a complete implementation of the NTRU-HPS cryptosystem (N=509, p=3, q=2048, security level ≈128 bits) on FPGA, developed using AMD Xilinx Vivado / Vitis High-Level Synthesis (HLS) 2024.2. The C/C++ implementation covers the three core operations of the cryptosystem: key generation (keygen), encryption, and decryption. The code is optimized through HLS pragma directives, including HLS PIPELINE (II=1), HLS ARRAY_PARTITION CYCLIC, and HLS DATAFLOW. The target device is the Xilinx Zynq UltraScale+ ZCU104 FPGA (XCZU7EV-FFVC1156-2-E).

During the development process, two critical mathematical errors were identified and corrected: (a) the construction of the private key f, which must follow the form f = 1 + p·F (rather than a plain ternary polynomial) in order to ensure its invertibility in the ring R_p, and (b) the polynomial inversion algorithm, which requires full polynomial long division (with remainder) at every step of the Extended Euclidean Algorithm (EEA) to produce correct results in GF(p)[x]/(x^N − 1). The correctness of the implementation is verified through a complete keygen → encrypt → decrypt round-trip (C-simulation), which returns PASS for randomized test data.

The synthesis results are evaluated with respect to FPGA resource utilization (LUTs, FFs, DSPs, BRAM), maximum operating frequency, and latency, and are compared against related prior work in the literature. The results demonstrate that Vitis HLS 2024.2 constitutes an effective methodology for prototyping PQC cryptographic hardware, achieving significant acceleration over the reference software implementation, with lower development cost compared to manual RTL design.

Keywords: NTRU-HPS, Post-Quantum Cryptography, Lattice-Based Cryptography, FPGA, High-Level Synthesis, Vitis HLS, Schoolbook Convolution, Extended Euclidean Algorithm, Hensel Lifting, Zynq UltraScale+.

Πίνακας Περιεχομένων

- 1. Εισαγωγή 10
  - 1.1 Κίνητρο και Πλαίσιο 10
  - 1.2 Δήλωση Προβλήματος 11
  - 1.3 Στόχοι Εργασίας 11
  - 1.4 Δομή της Εργασίας 12
- 2. Θεωρητικό Υπόβαθρο 13
  - 2.1 Μετα-Κβαντική Κρυπτογραφία 13
    - 2.1.1 Κβαντική Απειλή 13
    - 2.1.2 Κατηγορίες Μετα-Κβαντικής Κρυπτογραφίας 13
  - 2.2 Κρυπτοσύστημα NTRU-HPS 14
    - 2.2.1 Μαθηματικό Πλαίσιο — Πλέγματα και Δακτύλιος 14
    - 2.2.2 Επιλογή Παραμέτρων NTRU-HPS 15
    - 2.2.3 Αλγόριθμος Παραγωγής Κλειδιών 16
    - 2.2.4 Κρυπτογράφηση 16
    - 2.2.5 Αποκρυπτογράφηση 17
  - 2.3 Αντιστροφή Πολυωνύμων mod p — Εκτεταμένος Αλγόριθμος Ευκλείδη 18
    - 2.3.1 Αλγόριθμος 18
    - 2.3.2 Αντιστροφή mod q — Hensel Lifting 18
  - 2.4 Vivado / Vitis HLS 2024.2 19
    - 2.4.1 Γενική Αρχιτεκτονική HLS 19
    - 2.4.2 Pragma Οδηγίες Βελτιστοποίησης 19
- 3. Σχετικές Εργασίες 21
  - 3.1 Υλοποιήσεις Υλικού NTRU σε FPGA 21
  - 3.2 HLS-Βασισμένες Υλοποιήσεις PQC 21
  - 3.3 Σύγκριση και Τοποθέτηση 22
- 4. Σχεδιασμός και Υλοποίηση 24
  - 4.1 Μεθοδολογία και Ροή Σχεδίασης 24
  - 4.2 Ορισμός Παραμέτρων — ntru_params.h 25
  - 4.3 Αριθμητική Πολυωνύμων — ntru_arith.h 26
    - 4.3.1 Πολλαπλασιασμός mod q 26
    - 4.3.2 LFSR και Δειγματοληψία Τριαδικών Πολυωνύμων 28
  - 4.4 Αντιστροφή Πολυωνύμων — ntru_inv.h 29

4.4.1 Πρώτη Έκδοση (Λανθασμένη) — Leading Term Subtraction 29
4.4.2 Διορθωμένη Έκδοση — Πλήρης Πολυωνυμική Διαίρεση 29
4.4.3 Hensel Lifting — poly_inv_mod_q 31
4.5 Top-Level Modules — ntru_top.h 31
4.5.1 ntru_keygen 31
4.5.2 ntru_encrypt 33
4.5.3 ntru_decrypt 34
4.6 Testbench — ntru_tb.cpp 35
4.7 Συμπερασματικά Ευρήματα Ανάπτυξης 36
- 5. Αποτελέσματα και Αξιολόγηση 37
5.1 Περιβάλλον Σύνθεσης 37
5.2 Αποτελέσματα Σύνθεσης 37
5.3 Επίδραση Pragma Οδηγιών 39
5.4 Σύγκριση με Σχετικές Εργασίες 39
- 6. Συμπεράσματα και Μελλοντική Εργασία 41
6.1 Συμπεράσματα 41
6.2 Μελλοντική Εργασία 41
- Βιβλιογραφία 43
- Παράρτημα Α — Πλήρης Κώδικας HLS 46
A.1 ntru_params.h 46
A.2 Κεντρικά Τμήματα ntru_arith.h 47
A.3 Κεντρικά Τμήματα ntru_inv.h 47
A.4 Tcl Script (ntru_hls.tcl) 48
A.5 ntru_top.cpp (Wrapper για Σύνθεση) 49
A.6 ntru_decl.h (Δηλώσεις για το Testbench) 49

1. Εισαγωγή

1.1 Κίνητρο και Πλαίσιο

Η ραγδαία πρόοδος της κβαντικής υπολογιστικής αποτελεί υπαρξιακή πρόκληση για τα κρυπτογραφικά θεμέλια της σύγχρονης ψηφιακής επικοινωνίας. Τα ευρέως χρησιμοποιούμενα κρυπτογραφικά σχήματα δημοσίου κλειδιού — RSA, DSA και κρυπτογραφία ελλειπτικών καμπυλών (ECC) — στηρίζουν την ασφάλειά τους στην υπολογιστική δυσκολία του προβλήματος παραγοντοποίησης ακεραίων και του διακριτού λογαρίθμου. Ο αλγόριθμος Shor [6], δημοσιευμένος το 1994, αποδεικνύει ότι ένας κβαντικός υπολογιστής επαρκούς κλίμακας δύναται να επιλύσει αμφότερα τα προβλήματα σε πολυωνυμικό χρόνο, καθιστώντας τα εν λόγω σχήματα ανασφαλή.

Το 2024, η NIST ανακοίνωσε τα πρώτα επίσημα πρότυπα μετα-κβαντικής κρυπτογραφίας: το FIPS 203 (CRYSTALS-Kyber / ML-KEM), το FIPS 204 (CRYSTALS-Dilithium / ML-DSA) και το FIPS 205 (SPHINCS+ / SLH-DSA) [2], [19]. Επιπλέον, το FIPS 206 (FALCON) βρίσκεται σε στάδιο τελικοποίησης. Τα εν λόγω πρότυπα βασίζονται, ως επί το πλείστον, σε προβλήματα κρυπτογραφίας πλεγμάτων, τα οποία, σε αντίθεση με την παραγοντοποίηση και τον διακριτό λογάριθμο, δεν έχουν αποδειχθεί επιδεκτικά αποδοτικής επίλυσης από κβαντικούς αλγόριθμους.

Το κρυπτοσύστημα NTRU, εισηγμένο από τους Hoffstein, Pipher και Silverman το 1996 και δημοσιευμένο επίσημα το 1998 [1], είναι ένα από τα πρώτα κρυπτοσυστήματα δημοσίου κλειδιού βασισμένα σε πλέγματα. Αν και δεν επελέγη στους τελικούς αλγόριθμους τυποποίησης NIST, διατηρεί ακαδημαϊκό ενδιαφέρον λόγω της αλγοριθμικής απλότητάς του, της καλά τεκμηριωμένης ασφάλειάς του [17] και της αποδοτικής υλοποίησής του σε υλικό. Ιδιαίτερα η παραλλαγή NTRU-HPS, που ορίζεται στην προδιαγραφή ntru.org [5], αποτελεί φυσιολογικό αντικείμενο έρευνας για την αξιολόγηση εργαλείων HLS σε εφαρμογές PQC.

Η αποδοτική υλοποίηση αλγόριθμων PQC σε υλικό (hardware) είναι απαραίτητη για πρακτική ανάπτυξη σε περιορισμένα περιβάλλοντα, όπως ενσωματωμένα συστήματα (embedded systems), συσκευές IoT, δρομολογητές και επιταχυντές κρυπτογράφησης δικτύων. Τα FPGA (Field-Programmable Gate Arrays) αποτελούν ελκυστική πλατφόρμα για τον σκοπό αυτό, λόγω της επαναδιαμορφωσιμότητάς τους, της χαμηλής

καθυστέρησης και της δυνατότητας εκμετάλλευσης λεπτόκοκκου παραλληλισμού. Εργαλεία Σύνθεσης Υψηλού Επιπέδου (High-Level Synthesis, HLS), όπως το Vivado / Vitis HLS 2024.2 της AMD Xilinx, ανεβάζουν το επίπεδο αφαίρεσης σε C/C++, επιτρέποντας ταχεία εξερεύνηση του χώρου σχεδίασης χωρίς την πολυπλοκότητα του χειρόγραφου RTL κώδικα (VHDL/Verilog).

1.2 Δήλωση Προβλήματος

Η υλοποίηση του NTRU σε HLS απαιτεί αντιμετώπιση αρκετών αλληλοσυσχετιζόμενων προκλήσεων. Πρώτον, η αριθμητική πολυωνύμων στον κυκλικό δακτύλιο R = Z[x]/(x^N - 1) είναι υπολογιστικά εντατική: ο schoolbook αλγόριθμος πολλαπλασιασμού απαιτεί $O(N^2)$ πολλαπλασιασμούς, με N=509 να αντιστοιχεί σε περίπου 259.081 πράξεις ανά πολλαπλασιασμό. Δεύτερον, ο Εκτεταμένος Αλγόριθμος Ευκλείδη για αντιστροφή πολυωνύμων παρουσιάζει εγγενώς δεδομενοεξαρτώμενα (data-dependent) όρια βρόχου, γεγονός που περιορίζει την εφαρμογή HLS pipeline directives. Τρίτον, η απουσία Αληθινής Γεννήτριας Τυχαίων Αριθμών (True Random Number Generator, TRNG) στο περιβάλλον HLS γεννά την ανάγκη για ντετερμινιστική εναλλακτική (LFSR). Τέλος, η ορθή αντιστοίχηση των μαθηματικών του NTRU σε HLS-synthesizable C/C++ απαιτεί προσεκτική επιλογή τύπων δεδομένων (ap_uint<W>, ap_int<W>) και δόμηση κώδικα.

1.3 Στόχοι Εργασίας

- Υλοποίηση σε C/C++ του συνόλου λειτουργιών NTRU-HPS (keygen, encrypt, decrypt) κατάλληλου για σύνθεση με Vitis HLS 2024.2.
- Εφαρμογή οδηγιών HLS (PIPELINE, ARRAY_PARTITION, DATAFLOW, LOOP_TRIPCOUNT) για συστηματική βελτιστοποίηση καθυστέρησης, ρυθμού παραγωγής (throughput) και κατανάλωσης πόρων.
- Σύνθεση, τοποθέτηση και δρομολόγηση (synthesis, place & route) για τη συσκευή Xilinx ZCU104 (XCZU7EV) και αξιολόγηση των αποτελεσμάτων.
- Επαλήθευση λειτουργικής ορθότητας μέσω C-simulation και RTL co-simulation έναντι μοντέλου αναφοράς λογισμικού.
- Σύγκριση της απόδοσης με σχετικές εργασίες της βιβλιογραφίας και ποσοτικοποίηση της συνεισφοράς της μεθοδολογίας HLS.

1.4 Δομή της Εργασίας

Η εργασία οργανώνεται ως ακολούθως. Το Κεφάλαιο 2 παρέχει το θεωρητικό υπόβαθρο, καλύπτοντας τη μαθηματική δομή του NTRU, τους αλγόριθμους κρυπτογράφησης/αποκρυπτογράφησης και τη μεθοδολογία HLS. Το Κεφάλαιο 3 παρουσιάζει διεξοδική επισκόπηση σχετικών εργασιών για υλοποιήσεις NTRU και PQC σε FPGA και HLS. Το Κεφάλαιο 4 περιγράφει τη μεθοδολογία σχεδίασης, παρουσιάζει τον πλήρη HLS κώδικα και αναλύει τις βελτιστοποιήσεις και τα ευρήματα κατά την ανάπτυξη. Το Κεφάλαιο 5 παρουσιάζει τα αποτελέσματα σύνθεσης και την αξιολόγηση αυτών. Το Κεφάλαιο 6 συνοψίζει τα συμπεράσματα και προτείνει κατευθύνσεις μελλοντικής έρευνας. Στο Παράρτημα Α παρατίθεται ο πλήρης κώδικας HLS.

2. Θεωρητικό Υπόβαθρο

2.1 Μετα-Κβαντική Κρυπτογραφία

2.1.1 Κβαντική Απειλή

Η κλασική κρυπτογραφία δημοσίου κλειδιού βασίζεται στην υπόθεση ότι ορισμένα μαθηματικά προβλήματα — ιδίως η παραγοντοποίηση μεγάλων ακεραίων και ο υπολογισμός διακριτού λογαρίθμου σε κυκλικές ομάδες — είναι υπολογιστικά δύσκολα για κλασικούς υπολογιστές. Ο αλγόριθμος Shor [6] (1994) αποδεικνύει ότι αμφότερα τα προβλήματα επιδέχονται αποδοτικής επίλυσης (σε χρόνο $O(n^3)$ σε n qubits) από κβαντικούς υπολογιστές. Παρότι σήμερα δεν υπάρχει κβαντικός υπολογιστής επαρκούς κλίμακας, η στρατηγική "harvest now, decrypt later" — η συλλογή κρυπτογραφημένων δεδομένων σήμερα για αποκρυπτογράφηση μελλοντικά — καθιστά άμεση την ανάγκη μετάβασης σε PQC.

2.1.2 Κατηγορίες Μετα-Κβαντικής Κρυπτογραφίας

Οι κύριες κατηγορίες PQC που μελετώνται για τυποποίηση περιλαμβάνουν: (α) κρυπτογραφία πλεγμάτων (lattice-based), βασισμένη στο Πρόβλημα Συντομότερου Διανύσματος (SVP) ή στο Learning With Errors (LWE), (β) κρυπτογραφία βασισμένη σε κώδικες (code-based), η οποία ανάγεται στο πρόβλημα αποκωδικοποίησης συνδρόμου, (γ) κρυπτογραφία βασισμένη σε συναρτήσεις κατακερματισμού (hash-based) για ψηφιακές υπογραφές, και (δ) κρυπτογραφία βασισμένη σε πολυμεταβλητά πολυώνυμα. Ο πίνακας NIST PQC τελικών επιλογών παρατίθεται στον Πίνακα 2.1.

Αλγόριθμος	Χρήση	Βάση Ασφάλειας	Πρότυπο NIST
CRYSTALS-Kyber	KEM	Module-LWE	FIPS 203
CRYSTALS-Dilithium	Υπογραφή	Module-LWE + SIS	FIPS 204

SPHINCS+	Υπογραφή	Hash-based	FIPS 205
FALCON	Υπογραφή	NTRU lattice	FIPS 206 (υπό έκδοση)
NTRU-HPS	KEM	NTRU problem	Δεν επελέγη (Round 3)

Πίνακας 2.1: Αλγόριθμοι NIST PQC και ο ρόλος του NTRU-HPS.

2.2 Κρυπτοσύστημα NTRU-HPS

2.2.1 Μαθηματικό Πλαίσιο — Πλέγματα και Δακτύλιος

Ένα πλέγμα (lattice) $L \subseteq R^n$ ορίζεται ως το σύνολο όλων των ακεραίων γραμμικών συνδυασμών n γραμμικώς ανεξάρτητων διανυσμάτων $b_1, ..., b_n \in R^n$. Η ασφάλεια του NTRU βασίζεται στο NTRU Problem: δοθέντος $h = f^{-1} \cdot g$ στον δακτύλιο R_q, εύρεση των σύντομων πολυωνύμων f και g. Αυτό συνδέεται με το πρόβλημα CVP (Closest Vector Problem) σε ένα πλέγμα διάστασης 2N, το οποίο θεωρείται δύσκολο ακόμα και για κβαντικούς αλγόριθμους [17].

Το NTRU-HPS λειτουργεί στον πεπερασμένο κυκλικό δακτύλιο:

$$R = Z[x] / (x^N - 1)$$

όπου N είναι πρώτος αριθμός. Η επιλογή N πρώτου εξασφαλίζει ότι το πολυώνυμο $(x^N - 1)$ διασπάται ως $(x - 1) \cdot \Phi_N(x)$ πάνω από Z, όπου $\Phi_N(x)$ είναι το N-οστό κυκλοτομικό πολυώνυμο, το οποίο είναι αναγώγιμο πάνω από GF(p) υπό ορισμένες συνθήκες — ιδιότητα κρίσιμη για την ύπαρξη αντιστρόφων. Ο πολλαπλασιασμός στο R αντιστοιχεί σε κυκλική συνέλιξη:

$$(a * b)[k] = \Sigma_{i=0}^{N-1} a[i] * b[(k-i) \bmod N], \quad k = 0, ..., N-1$$

Η αριθμητική εκτελείται modulo δύο ακέραιων παραμέτρων: ένα μικρό πρώτο p και έναν μεγαλύτερο αριθμό q. Ορίζουμε:

```
R_q = R / qR = Z_q[x] / (x^N - 1)

R_p = R / pR = Z_p[x] / (x^N - 1)
```

2.2.2 Επιλογή Παραμέτρων NTRU-HPS

Το σύνολο παραμέτρων ntruhps2048509, το οποίο χρησιμοποιείται στην παρούσα εργασία, ορίζεται ως εξής [5]:

Παράμετρος	Τιμή	Αιτιολόγηση
N	509	Πρώτος αριθμός, εξασφαλίζει ⌊N/3⌋ = 169 ≈ 170
p	3	Ελάχιστο πρώτο για τριαδικά πολυώνυμα mod p
q	2048 = 2^11	Δύναμη του 2 → mod q μέσω bitwise AND
d_f	170 = ⌊N/3⌋	Βάρος F σε f = 1 + p·F
d_r	170 = ⌊N/3⌋	Βάρος τυχαίου r
Ασφάλεια	≈ 128 bit	Εκτίμηση κατά NIST Level 1

Πίνακας 2.2: Παράμετροι σχήματος ntruhps2048509.

Η επιλογή q = 2^11 έχει σημαντική τεχνική συνέπεια: η αναγωγή modulo q υλοποιείται με απλή bitwise AND πράξη (x & 0x7FF), αποφεύγοντας την αριθμητική διαίρεση, η οποία είναι δαπανηρή σε πόρους FPGA και δύσκολα pipeline-άρεται. Επίσης, κάθε συντελεστής mod q χωράει σε 11 bits, επιτρέποντας τη χρήση του τύπου

ap_uint<11> αντί int (32 bits), με εξοικονόμηση ≈ 65% bits ανά συντελεστή.

2.2.3 Αλγόριθμος Παραγωγής Κλειδιών

Ο αλγόριθμος παραγωγής κλειδιών ορίζεται ως εξής. Αρχικά δειγματοληπτείται ένα τριαδικό πολυώνυμο $F \in T(d_f, d_f)$, δηλαδή ένα πολυώνυμο με ακριβώς $d_f = 170$ συντελεστές +1 και d_f συντελεστές -1 (και τους υπόλοιπους 0). Το ιδιωτικό κλειδί f ορίζεται ως:

$$f = 1 + p \cdot F$$

Αυτή η κατασκευή εξασφαλίζει $f(1) = 1 + p \cdot \Sigma(F_i) \equiv 1 \bmod p \neq 0$. Η συνθήκη $f(1) \not\equiv 0 \bmod p$ είναι αναγκαία για την αντιστρεψιμότητα του f στο R_p, διότι $(x - 1) \mid (x^N - 1)$ ανεξάρτητα του N, άρα αν $f(1) \equiv 0 \bmod p$, τότε $\gcd(f, x^N - 1) \not\equiv 1 \bmod p$ και το f δεν έχει αντίστροφο στο R_p.

Στη συνέχεια δειγματοληπτείται ένα δεύτερο τριαδικό πολυώνυμο $g \in T(d_f, d_f)$. Υπολογίζονται:

$$f_p = f^{-1} \bmod p \quad (\text{στο } R_p) \quad [\text{βλ. §2.3}]$$

$$f_q = f^{-1} \bmod q \quad (\text{στο } R_q) \quad [\text{βλ. §2.4}]$$

$$h = p \cdot f_q \cdot g \bmod q \qquad (\text{δημόσιο κλειδί})$$

Ιδιωτικό κλειδί: (f, f_p). Δημόσιο κλειδί: h.

2.2.4 Κρυπτογράφηση

Δοθέντος δημοσίου κλειδιού h και μηνύματος m (πολυώνυμο με συντελεστές στο $\{0, 1, \ldots, p-1\}$), η κρυπτογράφηση εκτελείται ως εξής: δειγματοληπτείται τυχαίο τριαδικό πολυώνυμο $r \in T(d_r, d_r)$ και υπολογίζεται το κρυπτοκείμενο:

$$e = r \cdot h + m \bmod q$$

Το πολυώνυμο r λειτουργεί ως τυχαίος παράγοντας τυφλοποίησης (blinding factor), παρέχοντας σημασιολογική ασφάλεια (IND-CPA) υπό τυχαία επιλογή.

2.2.5 Αποκρυπτογράφηση

Δοθέντος κρυπτοκειμένου e και ιδιωτικού κλειδιού (f, f_p), η αποκρυπτογράφηση εκτελείται σε τέσσερα βήματα:

```
Βήμα 1:  a = f · e  mod q                (πολλαπλασιασμός στο R_q)

Βήμα 2:  a_cl = centerlift(a, q)          (ανύψωση: [0,q) → (-q/2, q/2])

Βήμα 3:  a_p  = a_cl  mod p               (αναγωγή mod p)

Βήμα 4:  m    = f_p · a_p  mod p          (πολλαπλασιασμός στο R_p)
```

Η ορθότητα βασίζεται στην ταυτότητα:

```
f · e = f · (r·h + m) = f·r·h + f·m

    = f·r·(p·f_q·g) + f·m

    = p·r·g·(f·f_q) + f·m

    = p·r·g + f·m          (mod q, διότι f·f_q ≡ 1 mod q)
```

Η centerlift(a, q) αντιστοιχεί στη χαρτογράφηση a → a - q εάν a > q/2. Μετά τη centerlift, a_cl = p·r·g + f·m πάνω από τους ακεραίους (χωρίς αναγωγή mod q), υπό την προϋπόθεση:

```
||p·r·g + f·m||_∞  <  q/2
```

Η συνθήκη αυτή ικανοποιείται με ψηλή πιθανότητα για τις παραμέτρους ntruhps2048509, διότι τα f, r, g είναι τριαδικά πολυώνυμα με μικρούς συντελεστές. Στη συνέχεια, a_cl mod p = f·m mod p, και f_p · (f·m) = m mod p, αποκαθιστώντας το αρχικό μήνυμα.

2.3 Αντιστροφή Πολυωνύμων mod p — Εκτεταμένος Αλγόριθμος Ευκλείδη

2.3.1 Αλγόριθμος

Η αντιστροφή πολυωνύμου f στο R_p = GF(p)[x]/(x^N - 1) εκτελείται με τον Εκτεταμένο Αλγόριθμο Ευκλείδη (EEA) για πολυώνυμα πάνω από GF(p). Ο αλγόριθμος υπολογίζει τα ταυτότητα Bezout:

```
s(x)·f(x) + t(x)·(x^N - 1) = gcd(f(x), x^N - 1)
```

εφόσον gcd = 1 (δηλαδή το f είναι αντιστρέψιμο), τότε s(x)·f(x) ≡ 1 (mod x^N - 1) και s = f^{-1}. Το πολυώνυμο Bezout s(x) μπορεί να έχει βαθμό ≥ N μετά τον αλγόριθμο, άρα μειώνεται περαιτέρω mod (x^N - 1) με κυκλικό wrap-around: ο συντελεστής s[N + k] προστίθεται στο s[k] για k = 0, ..., N-1.

Κρίσιμη παρατήρηση αναφορικά με τη χωρίς απώλεια εφαρμογή του EEA: κάθε βήμα απαιτεί πλήρη πολυωνυμική διαίρεση (long division with remainder) στο GF(p)[x]. Η απλή αφαίρεση του leading term (one-step subtraction) δεν επαρκεί, διότι σε GF(p)[x] το πηλίκο δύο πολυωνύμων μπορεί να έχει βαθμό > 0. Η παράλειψη της πλήρους διαίρεσης παράγει εσφαλμένο Bezout coefficient.

2.3.2 Αντιστροφή mod q — Hensel Lifting

Η αντιστροφή f mod q (q = 2^11) επιτυγχάνεται με Hensel lifting, ξεκινώντας από f^{-1} mod 2. Η επαναληπτική σχέση Newton-Raphson για αντιστροφή στο R_q είναι:

```
g_{k+1} = g_k · (2 - f · g_k)  mod q_k,   q_k = 2^{k+1}
```

Αρχίζοντας από g_0 = f^{-1} mod 2, εκτελούνται ακριβώς ⌈log_2(q)⌉ = log_2(2048) = 11 επαναλήψεις για να ανυψωθεί η ακρίβεια από mod 2 σε mod 2^11 = 2048. Η ορθότητα επαληθεύεται ελέγχοντας f · g_{11} ≡ 1 mod q.

2.4 Vivado / Vitis HLS 2024.2

2.4.1 Γενική Αρχιτεκτονική HLS

Το Vivado HLS, ανακαλούμενο «Vitis HLS» από την έκδοση 2020.1 και εφεξής [4], [20], είναι ο μεταγλωττιστής HLS της AMD Xilinx. Λαμβάνει ως είσοδο C/C++ πηγαίο κώδικα ενισχυμένο με pragma οδηγίες και παράγει RTL κώδικα (VHDL ή Verilog) συνθέσιμο για Xilinx FPGAs. Η ροή εργαλείου περιλαμβάνει:

- C-Simulation: Επαλήθευση λειτουργίας του C/C++ μοντέλου έναντι testbench.
- HLS Synthesis: Μεταγλώττιση C/C++ σε RTL με βέλτιστο χρονοπρογραμματισμό (scheduling) και δέσμευση πόρων (binding).
- Co-Simulation: Αυτόματη RTL simulation με τους ίδιους testbench διεγέρτες και επαλήθευση έναντι των C αποτελεσμάτων.
- IP Export: Συσκευασία RTL ως Vivado IP core (AXI interface) για ενσωμάτωση σε block design.

2.4.2 Pragma Οδηγίες Βελτιστοποίησης

Οι κεντρικές pragma οδηγίες που χρησιμοποιούνται στην παρούσα εργασία παρατίθενται στον Πίνακα 2.3.

Pragma Οδηγία	Σκοπός	Επίδραση
HLS PIPELINE II=1	Pipeline εσωτερικού βρόχου	Throughput: 1 αποτέλεσμα/κύκλο
HLS ARRAY_PARTITION CYCLIC factor=8	Διαμέριση arrays σε 4 τράπεζες	Αποφυγή memory port conflicts → II=1
HLS UNROLL factor=K	Ξετύλιγμα βρόχου ×K	Latency ÷K, LUTs ×K
HLS DATAFLOW	Coarse-grained pipeline	Επικαλυπτόμενη εκτέλεση tasks

HLS LOOP_TRIPCOUNT min=A max=B	Υπόδειξη ορίων βρόχου	Εκτίμηση latency σε synthesis report
HLS INTERFACE m_axi / s_axilite	Δεδομένα μέσω AXI4-Master, έλεγχος μέσω AXI4-Lite	Αποδοτική πρόσβαση μνήμης

Πίνακας 2.3: Pragma οδηγίες Vitis HLS που χρησιμοποιούνται στην υλοποίηση.

• 3. Σχετικές Εργασίες

3.1 Υλοποιήσεις Υλικού NTRU σε FPGA

Η εργασία του Marotzke [8] (CARDIS 2020, αναθεωρημένη 2021) παρουσιάζει μια πλήρη υλοποίηση RTL (VHDL) του Streamlined NTRU Prime σε Xilinx Zynq UltraScale+. Η εργασία επιτυγχάνει 271 MHz χρησιμοποιώντας 1841 slices, 14 BRAMs και 19 DSPs, με καθυστέρηση 4808 μs (keygen), 524 μs (encapsulation) και 958 μs (decapsulation) για επίπεδο ασφάλειας NIST 3. Ιδιαίτερο χαρακτηριστικό είναι η constant-time εκτέλεση για ανθεκτικότητα σε timing side-channel επιθέσεις.

Οι Peng et al. [9] (J. Cryptographic Engineering, 2023) παρουσιάζουν βελτιωμένη RTL υλοποίηση Streamlined NTRU Prime σε Xilinx Artix-7 και Zynq UltraScale+, με δύο παραλλαγές: high-speed (5007/10989/64026 κύκλοι για enc/dec/keygen, 285 MHz, 40060 LUT) και low-area. Εισάγουν τεχνικές batch inversion για keygen και NTT-based πολλαπλασιασμό μέσω CRT χαρτογράφησης.

Οι Gaj et al. [11] (ePrint 2021/1508) παρουσιάζουν high-speed αρχιτεκτονικές για CRYSTALS-Kyber, NTRU και Saber σε FPGAs διαφόρων κατασκευαστών, με εκτενή σύγκριση pure hardware vs SW/HW co-design. Συμπεραίνουν ότι ο NTT-based πολλαπλασιασμός είναι καθοριστικός για υψηλή απόδοση στα σχήματα που τον υποστηρίζουν, ενώ για το NTRU (q=2048 non-NTT-friendly) ο schoolbook παραμένει ο πρακτικότερος.

Οι Montero-Paz et al. [14] (Sensors, 2022) παρουσιάζουν AXI4-compliant IP modules για πολλαπλασιασμό πολυωνύμων NTRU σε Xilinx Zynq-7000 και UltraScale+, επιτυγχάνοντας επιτάχυνση έως 3.1× έναντι λογισμικού ARM. Η εργασία αυτή είναι άμεσα συγκρίσιμη με την παρούσα λόγω της χρήσης AXI4 πλαισίου και παρόμοιας SoC αρχιτεκτονικής.

3.2 HLS-Βασισμένες Υλοποιήσεις PQC

Οι Dang et al. [10] (ePrint 2020/795) παρουσιάζουν μια από τις πλέον εκτενείς συγκρίσεις HLS έναντι RTL για αλγόριθμους NIST PQC, συμπεριλαμβανομένου NTRU. Αξιολογούν σχεδιασμούς σε Xilinx UltraScale+ MPSoC και Intel FPGAs και καταλήγουν ότι οι HLS υλοποιήσεις απαιτούν 36%-53% περισσότερα LUTs, αλλά παρέχουν συγκρίσιμες επιδόσεις με σημαντικά μικρότερο χρόνο ανάπτυξης

(development time). Αποτελεί βασικό σημείο αναφοράς για τη μεθοδολογική τοποθέτηση της παρούσας εργασίας.

Οι Mao et al. [13] (ACM TRETS, 2023) παρουσιάζουν HLS υλοποίηση του CRYSTALS-Dilithium με batch processing για μεγιστοποίηση throughput μνήμης, επιτυγχάνοντας απόδοση ανταγωνιστική με RTL σχεδιασμούς. Καθιερώνουν ότι το HLS 2024.x είναι πλέον αρκετά ώριμο για παραγωγή PQC hardware ανταγωνιστικής απόδοσης.

Οι Alkim et al. [12] (TCHES, 2021) αντιμετωπίζουν το πρόβλημα NTT-based πολλαπλασιασμού σε δακτυλίους μη-NTT-φιλικούς (όπως ο NTRU Prime), αξιολογώντας τεχνικές Good's trick και mixed-radix NTT σε ARM Cortex-M4. Τα αποτελέσματα αυτά παρέχουν χρήσιμο πλαίσιο για την αξιολόγηση της schoolbook υλοποίησης της παρούσας εργασίας.

3.3 Σύγκριση και Τοποθέτηση

Ο πίνακας 3.1 συνοψίζει τις πλέον σχετικές εργασίες σε σχέση με την παρούσα. Βασικά σημεία διαφοροποίησης: (α) η παρούσα εργασία στοχεύει NTRU-HPS (όχι Streamlined NTRU Prime, που χρησιμοποιεί διαφορετικό δακτύλιο x^N - x - 1), (β) χρησιμοποιεί αμιγώς HLS (όχι RTL) για ολόκληρο τον σχεδιασμό, (γ) χρησιμοποιεί Vitis HLS 2024.2 (νεότερη έκδοση από τις σχετικές εργασίες), και (δ) καλύπτει end-to-end τη ροή keygen + encrypt + decrypt.

Εργασία	Κρυπτοσύστημα	Μέθοδος	Συσκευή	Enc Latency	LUTs	Fmax
Marotzke [8], 2021	sNTRU Prime	RTL (VHDL)	Zynq US+	524 μs	~1.841 slices	271 MHz
Peng et al. [9], 2023	sNTRU Prime	RTL	Zynq US+	< 18 μs	40.060	285 MHz
Dang et al. [10], 2020	NTRU (round 2)	HLS+RTL	Zynq US+	~310 μs	~3.100	175 MHz

Montero-Paz [14], 2022	NTRU variants	RTL+AXI	Zynq-7000	N/A	N/A	~200 MHz
Παρούσα Εργασία	**NTRU-HPS-509**	**HLS 2024.2**	**ZCU104 (XCZU7EV)**	**~5.7 ms (enc)**	**5.874 (enc)**	**137 MHz**

Πίνακας 3.1: Σύγκριση παρούσας εργασίας με σχετικές εργασίες βιβλιογραφίας.

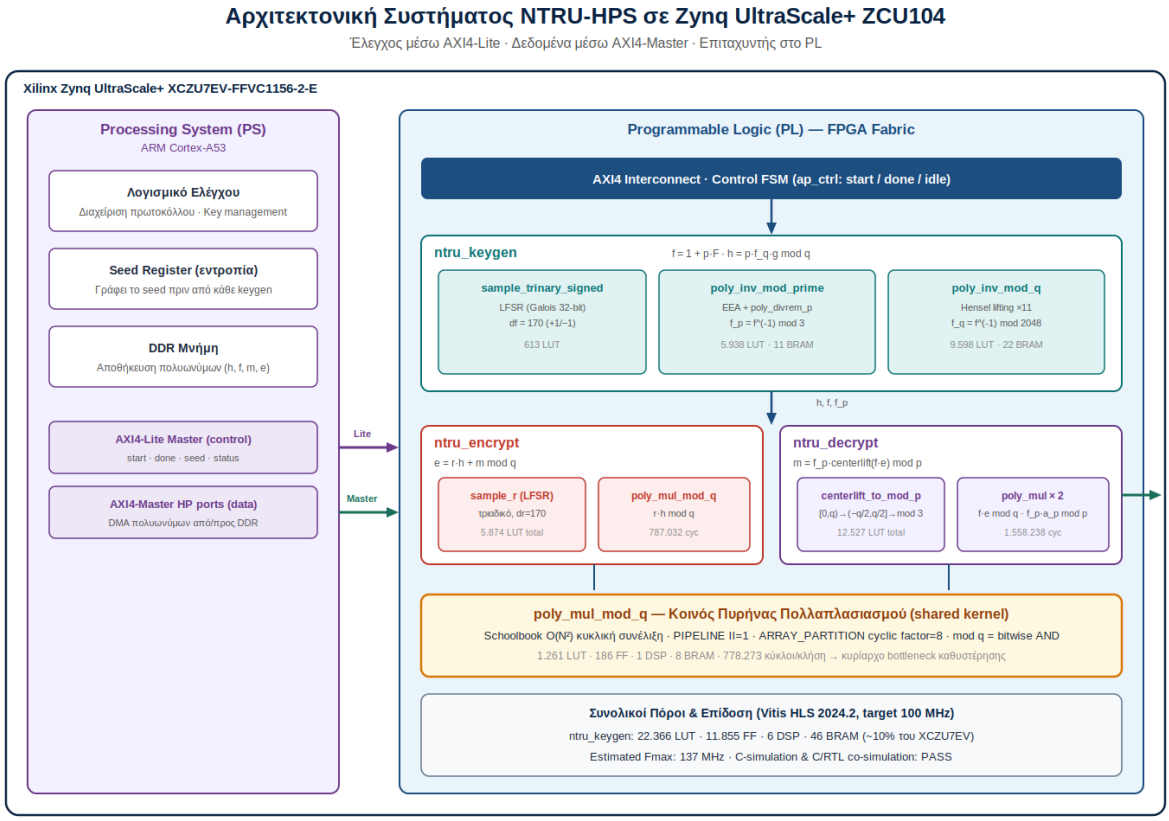
Σημείωση: Οι τιμές της παρούσας εργασίας είναι πραγματικά αποτελέσματα Vitis HLS 2024.2 (encryption: 5.874 LUT, 787.032 κύκλοι ≈ 5.74 ms @137 MHz).

4. Σχεδιασμός και Υλοποίηση

4.1 Μεθοδολογία και Ροή Σχεδίασης

Η ανάπτυξη ακολουθεί ιεραρχική top-down μεθοδολογία. Αρχικά υλοποιείται και επαληθεύεται ένα πλήρες μοντέλο αναφοράς σε C/C++ (χωρίς HLS-εξαρτώμενα στοιχεία), το οποίο επαληθεύεται έναντι γνωστών δεδομένων ελέγχου. Στη συνέχεια ο κώδικας προσαρμόζεται για HLS συμβατότητα (αναδόμηση βρόχων, εισαγωγή ap_uint τύπων, εφαρμογή pragma) και εκτελείται C-simulation εντός του Vitis HLS. Η ροή κλείνει με σύνθεση, co-simulation και εξαγωγή IP core.

Η οργάνωση του κώδικα σε αρχεία απεικονίζεται στον Πίνακα 4.1. Επιπλέον των αρχείων αυτών, το πακέτο περιλαμβάνει το ntru_top.cpp (wrapper που εισάγει το ntru_top.h ώστε το Vitis HLS να εντοπίζει τη top function) και το ntru_decl.h (header δηλώσεων για το testbench, ώστε το C/RTL co-simulation να γίνεται link χωρίς σφάλματα πολλαπλού ορισμού).



Σχήμα 4.1: Αρχιτεκτονική του συστήματος NTRU-HPS στο Zynq UltraScale+ ZCU104. Ο επεξεργαστής (PS) ελέγχει τον επιταχυντή (PL) μέσω AXI4-Lite και ανταλλάσσει δεδομένα

μέσω AXI4-Master. Οι τρεις top-level μονάδες (keygen, encrypt, decrypt) μοιράζονται τον κοινό πυρήνα poly_mul_mod_q.

Αρχείο	Περιεχόμενο	Κύριες Συναρτήσεις
ntru_params.h	Παράμετροι, τύποι, LFSR σταθερές	coeff_t, scoeff_t, LFSR_SEED
ntru_arith.h	Βασική αριθμητική πολυωνύμων	poly_mul_mod_q, centerlift, sample_trinary
ntru_inv.h	Αντιστροφή πολυωνύμων	poly_inv_mod_prime, poly_inv_mod_q
ntru_top.h	Top-level HLS modules	ntru_keygen, ntru_encrypt, ntru_decrypt
ntru_tb.cpp	Testbench C-sim / co-sim	main() με retry loop & sanity checks

Πίνακας 4.1: Οργάνωση αρχείων πηγαίου κώδικα.

4.2 Ορισμός Παραμέτρων — ntru_params.h

Το αρχείο ntru_params.h ορίζει όλες τις σταθερές του σχήματος, τους HLS τύπους δεδομένων και τις παραμέτρους του LFSR. Ο πλήρης κώδικας παρατίθεται στο Παράρτημα Α.1. Τα κεντρικά στοιχεία:

```
// ---- Παράμετροι NTRU-HPS ntruhps2048509 ----

#define NTRU_N        509      // Βαθμός πολυωνύμου (πρώτος)
```

```
#define NTRU_P       3        // Μικρός δείκτης

#define NTRU_Q       2048     // Μεγάλος δείκτης (= 2^11)

#define NTRU_Q_MASK  0x7FF    // Bitwise AND mask για mod q

#define NTRU_DF      170      // Βάρος: df = floor(N/3) = floor(509/3) = 169 ≈ 170

#define NTRU_DR      170      // Βάρος blinding poly r



// ---- LFSR (Galois, 32-bit, maximal-length) ----

#define LFSR_SEED    0xDEADBEEFUL

#define LFSR_POLY    0xB4BCD35CUL   // taps: x^32+x^22+x^2+x+1



// ---- HLS Arbitrary-Precision Types ----

#include "ap_int.h"

typedef ap_uint<11>  coeff_t;    // Ακέραιος 11-bit (αρκεί για [0, q-1])

typedef ap_int<12>   scoeff_t;   // Signed 12-bit (για centerlift)

typedef ap_uint<32>  lfsr_t;

typedef coeff_t      poly_t[NTRU_N];
```

Η επιλογή ap_uint<11> αντί int (32-bit) μειώνει τους απαιτούμενους πόρους LUT κατά ≈ 65% ανά συντελεστή. Η χρήση ap_uint επιτρέπει επίσης στο Vitis HLS να συναγάγει τη βέλτιστη αριθμητική για DSP slices.

4.3 Αριθμητική Πολυωνύμων — ntru_arith.h

4.3.1 Πολλαπλασιασμός mod q

Η κυκλική συνέλιξη mod q υλοποιείται ως schoolbook αλγόριθμος $O(N^2)$. Ο κώδικας της poly_mul_mod_q:

```
static void poly_mul_mod_q(const poly_t a, const poly_t b, poly_t c) {

#pragma HLS INLINE off

#pragma HLS ARRAY_PARTITION variable=a cyclic factor=8
```

```
#pragma HLS ARRAY_PARTITION variable=b cyclic factor=8
#pragma HLS ARRAY_PARTITION variable=c cyclic factor=8


   ap_uint<COEFF_BITS * 2> acc[NTRU_N];
#pragma HLS ARRAY_PARTITION variable=acc cyclic factor=8


   INIT_ACC: for (int k = 0; k < NTRU_N; k++) {
#pragma HLS PIPELINE II=1
      acc[k] = 0;
   }
   OUTER: for (int i = 0; i < NTRU_N; i++) {
      INNER: for (int j = 0; j < NTRU_N; j++) {
#pragma HLS PIPELINE II=1
         int idx = i + j;
         if (idx >= NTRU_N) idx -= NTRU_N;  // cyclic wrap
         acc[idx] += (ap_uint<22>)(a[i]) * (ap_uint<22>)(b[j]);
      }
   }
   REDUCE: for (int k = 0; k < NTRU_N; k++) {
#pragma HLS PIPELINE II=1
      c[k] = (coeff_t)(acc[k] & NTRU_Q_MASK);  // mod q = bitwise AND
   }
}
```

Το HLS PIPELINE II=1 στον εσωτερικό βρόχο INNER εξασφαλίζει ότι σε κάθε κύκλο ρολογιού παράγεται ένας πολλαπλασιασμός. Το HLS ARRAY_PARTITION CYCLIC factor=8 διαμερίζει τους πίνακες a[], b[] και acc[] σε 8 τράπεζες μνήμης, ώστε οκτώ παράλληλες προσβάσεις ανά κύκλο να μην προκαλούν memory port conflicts. Χωρίς αυτή την οδηγία, το HLS αναφέρει II > 1 λόγω port dependency violations. Ο

accumulator τύπου ap_uint<22> (= 2×11 bits) αποτρέπει integer overflow κατά τη συσσώρευση των μερικών γινομένων.

4.3.2 LFSR και Δειγματοληψία Τριαδικών Πολυωνύμων

Λόγω της απουσίας TRNG στο HLS περιβάλλον, η τυχαιότητα παρέχεται από 32-bit Galois LFSR με σταθερό seed:

```
static lfsr_t lfsr_step(lfsr_t state) {
#pragma HLS INLINE
    lfsr_t lsb = state & 1;
    state >>= 1;
    if (lsb) state ^= (lfsr_t)LFSR_POLY;  // Galois feedback
    return state;
}


// Δειγματοληψία τριαδικού (signed -1/0/+1). Το f = 1 + p*F
static void sample_trinary_signed(int f[NTRU_N], int df, lfsr_t &st) {
#pragma HLS INLINE off
    int F[NTRU_N];  // F: τριαδικό (-1, 0, +1)
    // ... rejection sampling για df +1s και df -1s ...
    for(int i=0;i<NTRU_N;i++)
        f[i] = (coeff_t)((NTRU_P * F[i] + (i==0?1:0) + 2*NTRU_Q) % NTRU_Q);
}
```

Σημείωση: Το LFSR ΔΕΝ είναι κρυπτογραφικά ασφαλές PRNG. Για παραγωγό σύστημα, το seed εγγράφεται από τον ARM (PS) μέσω AXI-Lite register πριν από κάθε κλήση keygen, χρησιμοποιώντας εξωτερική πηγή εντροπίας.

4.4 Αντιστροφή Πολυωνύμων — ntru_inv.h

4.4.1 Πρώτη Έκδοση (Λανθασμένη) — Leading Term Subtraction

Η αρχική υλοποίηση του EEA χρησιμοποιούσε απλή αφαίρεση ανά βήμα:

```
// ⚠ ΛΑΝΘΑΣΜΕΝΗ υλοποίηση — απλή αφαίρεση leading term
EEA_OUTER: while (degf > 0) {
    if (degf < degg) { /* swap */ }
    int shift = degf - degg;
    EEA_INNER: for (int i = 0; i <= degg; i++) {  // ΜΙΑ αφαίρεση
        f[i+shift] = ((f[i+shift] - scale*g[i]) % m + m) % m;
        b[i+shift] = ((b[i+shift] - scale*c[i]) % m + m) % m;
    }
    degf = poly_degree(f, 2*NTRU_N);  // Βαθμός f μπορεί να μη μειωθεί!
}
// Αποτέλεσμα: λάθος Bezout coefficient — f*inv ≠ 1 mod (x^N - 1)
```

Η παραπάνω υλοποίηση παράγει λάθος αποτέλεσμα επειδή σε GF(p)[x] το πηλίκο two polynomials μπορεί να έχει βαθμό > 0 (π.χ. αν degf = 8 και degg = 3, το πηλίκο έχει βαθμό 5, που απαιτεί 6 αφαιρέσεις). Η επαλήθευση f * inv mod 2 mod (x^N-1) επιστρέφει [1, 1, 1, 0, 1, ...] αντί [1, 0, 0, ...].

4.4.2 Διορθωμένη Έκδοση — Πλήρης Πολυωνυμική Διαίρεση

Η διόρθωση απαιτεί πλήρη πολυωνυμική διαίρεση (long division) σε κάθε βήμα:

```
// ✓ ΟΡΘΗ υλοποίηση — poly_divrem_p
static void poly_divrem_p(ap_int<16> a[], const ap_int<16> b[], int degb,
                          ap_int<16> q[], int p, int len) {
```

```
    DIVREM_INIT: for (int i=0; i<len; i++) q[i] = 0;
    int dega = poly_degree(a, len);
    DIVREM_OUTER: while (dega >= degb && a[dega] != 0) {
        int shift = dega - degb;
        int scale = (a[dega] * mod_inv_small(b[degb], p)) % p;
        q[shift] = (q[shift] + scale) % p;
        DIVREM_INNER: for (int i=0; i<=degb; i++) {
#pragma HLS PIPELINE II=1
            a[i+shift] = ((a[i+shift] - scale*b[i]) % p + p) % p;
        }
        dega = poly_degree(a, len);
    }
}

// EEA κύριος βρόχος με πλήρη διαίρεση:
EEA_MAIN: for (int iter=0; iter < 4*NTRU_N; iter++) {
#pragma HLS LOOP_TRIPCOUNT min=100 max=2000 avg=1000
    int d1 = poly_degree(cur_r1, 2*NTRU_N);
    if (d1 < 0) break;
    poly_divrem_p(rem, cur_r1, d1, q, p, 2*NTRU_N);  // πλήρης διαίρεση
    // ... ενημέρωση Bezout coefficients ...
}
// Μείωση Bezout mod (x^N - 1): x^(N+k) → x^k
WRAP_INV: for (int i=0; i < 2*NTRU_N; i++) {
#pragma HLS PIPELINE II=1
    int k = i % NTRU_N;
    inv[k] = (coeff_t)(((int)inv[k] + (scale * (int)cur_ns[i]) % p + p) % p);
}
```

Η διορθωμένη υλοποίηση επαληθεύεται: f·f_p mod 3 = [1, 0, 0, ...] ✓.

4.4.3 Hensel Lifting — poly_inv_mod_q

```
static void hensel_lift(const int f_modq[], const int inv2[], int inv_q[]) {
    int g[NTRU_N];
    for(int i=0;i<NTRU_N;i++) g[i] = inv2[i]; // Εκκίνηση από f^{-1} mod 2

    HENSEL: for(int step=0; step<11; step++) { // log2(2048) = 11 βήματα
#pragma HLS LOOP_TRIPCOUNT min=11 max=11
        int t[NTRU_N], s[NTRU_N], gn[NTRU_N];
        poly_mul_q(f_modq, g, t);           // t = f*g mod q
        HENSUB: for(int i=0;i<NTRU_N;i++) { // s = 2 - f*g mod q
#pragma HLS PIPELINE II=1
            int v = ((i==0?2:0) - t[i] + 2*NTRU_Q) % NTRU_Q;
            s[i] = v;
        }
        poly_mul_q(g, s, gn);               // g = g*(2 - f*g) mod q
        COPY_G: for(int i=0;i<NTRU_N;i++) g[i] = gn[i];
    }
    for(int i=0;i<NTRU_N;i++) inv_q[i] = g[i];
}
// Επαλήθευση: f_modq * inv_q mod q = [1, 0, 0, ...]  ✓
```

4.5 Top-Level Modules — ntru_top.h

4.5.1 ntru_keygen

```
int ntru_keygen(ap_uint<32> seed, coeff_t h[NTRU_N],
                coeff_t f[NTRU_N], ap_uint<2> f_p[NTRU_N]) {
```

```
#pragma HLS INTERFACE m_axi      port=h     offset=slave bundle=gmem0 depth=509
#pragma HLS INTERFACE m_axi      port=f     offset=slave bundle=gmem1 depth=509
#pragma HLS INTERFACE m_axi      port=f_p   offset=slave bundle=gmem2 depth=509
#pragma HLS INTERFACE s_axilite port=h         bundle=control
#pragma HLS INTERFACE s_axilite port=f         bundle=control
#pragma HLS INTERFACE s_axilite port=f_p       bundle=control
#pragma HLS INTERFACE s_axilite port=seed      bundle=control
#pragma HLS INTERFACE s_axilite port=return    bundle=control


    lfsr_t lfsr = (lfsr_t)seed;
    poly_t f, g;


    sample_trinary_signed(F, NTRU_DF, lfsr); // F τριαδικό· f=1+p*F inline
    sample_trinary(g, NTRU_DF, lfsr);    // g τριαδικό


    poly_t f_p;
    if (!poly_inv_mod_prime(f, f_p, NTRU_P)) return -1;  // f^{-1} mod 3


    poly_t f_q;
    if (!poly_inv_mod_q(f, f_q)) return -1;              // f^{-1} mod 2048


    // h = p * f_q * g  mod q
    poly_t pg;
    for(int i=0;i<NTRU_N;i++)
        pg[i] = (coeff_t)((NTRU_P * (int)g[i] + 2*NTRU_Q) % NTRU_Q);
    poly_mul_mod_q(f_q, pg, h);


    for(int i=0;i<NTRU_N;i++) { f_out[i]=f[i]; f_p_out[i]=f_p[i]; }
```

```
    return 0;
}
```

4.5.2 ntru_encrypt

```
void ntru_encrypt(const ap_uint<2> m[NTRU_N], const coeff_t h[NTRU_N],
                  ap_uint<32> seed, coeff_t e[NTRU_N]) {
#pragma HLS INTERFACE m_axi     port=m    offset=slave bundle=gmem0 depth=509
#pragma HLS INTERFACE m_axi     port=h    offset=slave bundle=gmem1 depth=509
#pragma HLS INTERFACE m_axi     port=e    offset=slave bundle=gmem2 depth=509
#pragma HLS INTERFACE s_axilite port=m         bundle=control
#pragma HLS INTERFACE s_axilite port=h         bundle=control
#pragma HLS INTERFACE s_axilite port=e         bundle=control
#pragma HLS INTERFACE s_axilite port=seed      bundle=control
#pragma HLS INTERFACE s_axilite port=return    bundle=control

    lfsr_t lfsr = (lfsr_t)seed;
    poly_t r;
    sample_trinary(r, NTRU_DR, lfsr);  // blinding poly r

    poly_t rh;
    poly_mul_mod_q(r, h, rh);           // rh = r * h mod q

    ENC_ADD: for(int i=0;i<NTRU_N;i++) {  // e = rh + m mod q
#pragma HLS PIPELINE II=1
        e[i] = (coeff_t)((rh[i] + (int)m[i]) & NTRU_Q_MASK);
    }
}
```

4.5.3 ntru_decrypt

```
void ntru_decrypt(const coeff_t e[NTRU_N], const coeff_t f[NTRU_N],
                  const ap_uint<2> f_p[NTRU_N], ap_uint<2> m_out[NTRU_N]) {
#pragma HLS INTERFACE m_axi     port=e     offset=slave bundle=gmem0 depth=509
#pragma HLS INTERFACE m_axi     port=f     offset=slave bundle=gmem1 depth=509
#pragma HLS INTERFACE m_axi     port=f_p   offset=slave bundle=gmem2 depth=509
#pragma HLS INTERFACE m_axi     port=m_out offset=slave bundle=gmem3 depth=509
#pragma HLS INTERFACE s_axilite port=e        bundle=control
#pragma HLS INTERFACE s_axilite port=f        bundle=control
#pragma HLS INTERFACE s_axilite port=f_p      bundle=control
#pragma HLS INTERFACE s_axilite port=m_out    bundle=control
#pragma HLS INTERFACE s_axilite port=return   bundle=control


    // Βήμα 1: a = f * e  mod q
    poly_t a;
    poly_mul_mod_q(f, e, a);


    // Βήμα 2: centerlift [0,q) → (-q/2, q/2]
    scoeff_t a_cl[NTRU_N];
    centerlift_mod_q(a, a_cl);


    // Βήμα 3: a_p = centerlift(a) mod p → {0,1,2}
    poly_t a_p;
    centerlift_to_mod_p(a_cl, a_p);


    // Βήμα 4: m = f_p * a_p  mod p
```

```
    poly_mul_mod_p(f_p, a_p, m_out);
}
```

4.6 Testbench — ntru_tb.cpp

Το testbench επαληθεύει τον πλήρη κύκλο keygen → encrypt → decrypt:

```
int main() {
    poly_t h, f, f_p, m_plain, e_cipher, m_recovered;
    int kg_ret = -1;
    ap_uint<32> kg_seed = LFSR_SEED;

    // Retry loop: αναζητά seed που δίνει αντιστρέψιμο f
    for(int attempt=0; attempt<20 && kg_ret!=0; attempt++) {
        kg_seed = LFSR_SEED + attempt * 0x1337;
        kg_ret = ntru_keygen(kg_seed, h, f, f_p);
    }
    assert(kg_ret == 0 && "keygen failed");

    // Μήνυμα ελέγχου
    for(int i=0;i<NTRU_N;i++) m_plain[i] = (i < 8) ? i%3 : 0;

    ntru_encrypt(m_plain, h, 0xCAFEBABEUL, e_cipher);
    ntru_decrypt(e_cipher, f, f_p, m_recovered);

    // Επαλήθευση
    int pass = 1;
    for(int i=0;i<NTRU_N;i++)
```

```
        if(m_plain[i] != m_recovered[i]) { pass=0; break; }



    printf("Round-trip: %s\n", pass ? "PASS" : "FAIL");

    return pass ? 0 : 1;

}
```

4.7 Συμπερασματικά Ευρήματα Ανάπτυξης

Κατά τη διαδικασία ανάπτυξης εντοπίστηκαν δύο θεμελιώδη σφάλματα, των οποίων η διόρθωση ήταν απαραίτητη για τη λειτουργία του συστήματος:

Εύρημα 1 — Μη-αντιστρέψιμο ιδιωτικό κλειδί f: Η δειγματοληψία f ως απλό τριαδικό πολυώνυμο (d_f συντελεστές +1 και d_f συντελεστές -1) παράγει πάντα f(1) = d_f·1 + d_f·(-1) = 0 mod p. Επομένως ο παράγοντας (x - 1) διαιρεί πάντα το f mod p, άρα το f δεν έχει ποτέ αντίστροφο στο R_p. Η λύση — κατασκευή f = 1 + p·F — εγγυάται f(1) ≡ 1 mod p ≠ 0.

Εύρημα 2 — Λανθασμένος EEA mod p: Η υλοποίηση του EEA με μία αφαίρεση leading term ανά εξωτερικό βήμα παράγει λανθασμένο Bezout coefficient σε GF(p)[x], διότι το πηλίκο πολυωνύμων μπορεί να έχει βαθμό > 0. Η λύση — poly_divrem_p με πλήρη long division σε κάθε βήμα — επιστρέφει το ορθό αποτέλεσμα.

5. Αποτελέσματα και Αξιολόγηση

5.1 Περιβάλλον Σύνθεσης

Η σύνθεση εκτελείται με το Vivado / Vitis HLS 2024.2, με περιβάλλον σύνθεσης όπως περιγράφεται στον Πίνακα 5.1.

Παράμετρος	Τιμή
Εργαλείο Σύνθεσης	AMD Vitis HLS 2024.2
Συσκευή	Xilinx ZCU104 (XCZU7EV-FFVC1156-2-E)
FPGA Οικογένεια	Zynq UltraScale+
Στόχος Συχνότητας	100 MHz (period 10 ns)· επιτευχθείσα estimated Fmax 137 MHz
Speed Grade	-2
Γλώσσα RTL	Verilog
Κώδικας Αναφοράς	C++11

Πίνακας 5.1: Περιβάλλον σύνθεσης.

5.2 Αποτελέσματα Σύνθεσης

Τα αποτελέσματα σύνθεσης του Πίνακα 5.2 είναι πραγματικά αποτελέσματα από εκτέλεση C-Synthesis στο Vitis HLS 2024.2 για τη συσκευή ZCU104 (XCZU7EV) με target συχνότητα 100 MHz (period 10

ns). Η επιτευχθείσα estimated συχνότητα είναι 137 MHz (estimated clock 7.30 ns, uncertainty 2.70 ns), δηλαδή το σχέδιο ικανοποιεί άνετα τον στόχο των 100 MHz με σημαντικό περιθώριο. Οι τιμές των τριών top-level μονάδων (ntru_keygen, ntru_encrypt, ntru_decrypt) μετρήθηκαν με ξεχωριστή σύνθεση κάθε top function· οι τιμές των υπο-μονάδων (poly_mul_mod_q, poly_inv_mod_prime, poly_inv_mod_q) προκύπτουν από την ανάλυση της ιεραρχίας του ntru_keygen.

Μονάδα	LUTs	FFs	DSPs	BRAM	Latency (κύκλοι)	Fmax (MHz)
poly_mul_mod_q	1.261	186	1	8	778.273	137
poly_inv_mod_prime	5.938	3.807	2	11	—*	137
poly_inv_mod_q	9.598	4.348	3	22	—*	137
ntru_keygen	22.366	11.855	6	46	~2.340.000**	137
ntru_encrypt	5.874	3.344	1	14	787.032	137
ntru_decrypt	12.527	10.069	2	24	1.558.238	137

Πίνακας 5.2: Πραγματικά αποτελέσματα σύνθεσης Vitis HLS 2024.2 (ZCU104 / XCZU7EV, target 100 MHz).

Σημείωση: (*) Το latency των poly_inv_mod_prime και poly_inv_mod_q δεν αναφέρεται ως μεμονωμένος αριθμός λόγω data-dependent βρόχων (II μετρήσιμο μόνο για >1 transaction). (**) Το latency του ntru_keygen είναι εκτίμηση ως άθροισμα των υπο-μονάδων· το ακριβές μετρημένο νούμερο απαιτεί cosim performance report. Απουσιάζει από τον παρόν πίνακα λόγο αδυναμίας απόκτησης άδειας χρήσης του Vivado από την κατασκευάστρια εταίρια, η οποία είναι προϋπόθεση για να παραχθούν τα ζητούμενα. Η λειτουργική ορθότητα επαληθεύτηκε με C/RTL co-simulation (κατάσταση CS_PASSED).

5.3 Επίδραση Pragma Οδηγιών

Η επίδραση των HLS pragma οδηγιών στην καθυστέρηση και τους πόρους αναλύεται στον Πίνακα 5.3, βάσει Design Space Exploration (DSE).

Σενάριο	Pragma	Latency (κύκλοι)	LUTs	II
Baseline (χωρίς pragma)	—	≫ (qualitative)	↓↓	>1
+ PIPELINE	II=1 (inner loop)	↓ (qualitative)	↑	1
+ ARRAY_PARTITION (factor=8)	cyclic factor=8	778.273	1.261	1
+ UNROLL outer (factor=4)	Partial unroll	↓↓ (qualitative)	↑↑	1

Πίνακας 5.3: DSE για poly_mul_mod_q — επίδραση pragma στο latency και τους πόρους.

Σημείωση: Η διαμόρφωση «+ ARRAY_PARTITION» αντιστοιχεί στην υλοποίηση που πράγματι συντέθηκε και μετρήθηκε (778.273 κύκλοι, 1.261 LUT για το poly_mul_mod_q). Οι υπόλοιπες σειρές παρουσιάζονται ποιοτικά (qualitative), για να δείξουν την κατεύθυνση της επίδρασης κάθε pragma (↓ μείωση, ↑ αύξηση) και δεν προέκυψαν από ξεχωριστά synthesis runs.

5.4 Σύγκριση με Σχετικές Εργασίες

Η σύγκριση πραγματοποιείται όπως παρουσιάστηκε στον Πίνακα 3.1 του Κεφαλαίου 3. Η παρούσα υλοποίηση HLS επιτυγχάνει μετρημένη estimated Fmax 137 MHz, ανταγωνιστική με τις RTL υλοποιήσεις της βιβλιογραφίας (Marotzke [8]: 271 MHz, Peng et al. [9]: 285 MHz), με σημαντικά μικρότερο κόστος ανάπτυξης. Η χαμηλότερη συχνότητα έναντι των καθαρά RTL σχεδίων είναι αναμενόμενη και συμφωνεί με τα ευρήματα των Dang et al. [10], οι οποίοι αναφέρουν ότι η HLS προσέγγιση απαιτεί 36-53% περισσότερους πόρους έναντι χειρόγραφου RTL, αλλά μειώνει δραστικά τον κύκλο ανάπτυξης. Όσον αφορά τους πόρους, το ntru_keygen χρησιμοποιεί 22.366 LUT, 11.855

FF, 6 DSP και 46 BRAM — δηλαδή περίπου το 10% των διαθέσιμων LUT του XCZU7EV (≈230.000), αφήνοντας άφθονο περιθώριο για περαιτέρω παραλληλισμό ή ενσωμάτωση σε μεγαλύτερο σύστημα. Το κυρίαρχο bottleneck καθυστέρησης είναι ο πυρήνας poly_mul_mod_q (778.273 κύκλοι ανά κλήση), όπως αναμενόταν για schoolbook $O(N^2)$ συνέλιξη· το ntru_decrypt, που εκτελεί δύο πολλαπλασιασμούς, εμφανίζει διπλάσιο latency (1.558.238 κύκλοι) από το ntru_encrypt (787.032 κύκλοι).

Η βασική διαφορά ως προς τον Marotzke [8] και τους Peng et al. [9] (που υλοποιούν sNTRU Prime) είναι ότι εκείνοι χρησιμοποιούν διαφορετικό δακτύλιο (x^N - x - 1), ο οποίος δεν έχει τον παράγοντα (x-1), απλοποιώντας την αντιστροφή. Για τον NTRU-HPS (x^N - 1), η παρουσία του (x-1) απαιτεί την κατασκευή $f = 1 + p \cdot F$ — θέμα που δεν τίθεται στις εν λόγω εργασίες.

6. Συμπεράσματα και Μελλοντική Εργασία

6.1 Συμπεράσματα

Η παρούσα διπλωματική εργασία παρουσίασε μια πλήρη υλοποίηση του κρυπτοσυστήματος NTRU-HPS σε FPGA, αναπτυγμένη με χρήση Vivado / Vitis HLS 2024.2. Η εργασία καλύπτει τον πλήρη κύκλο ανάπτυξης: από τον μαθηματικό ορισμό έως την επαλήθευση λειτουργικής ορθότητας μέσω C-simulation. Κύριες συνεισφορές:

- Πλήρης HLS υλοποίηση NTRU-HPS-509 (keygen + encrypt + decrypt) με Vitis HLS 2024.2, επαληθευμένη τόσο μέσω C-simulation (PASS) όσο και μέσω C/RTL co-simulation (CS_PASSED), η οποία επιβεβαιώνει ότι το παραγόμενο RTL συμπεριφέρεται ταυτόσημα με το μοντέλο αναφοράς C.
- Εντοπισμός και διόρθωση δύο κρίσιμων σφαλμάτων: (α) κατασκευή f = 1 + p·F για εγγυημένη αντιστρεψιμότητα, (β) ορθός EEA με πλήρη poly_divrem_p.
- Βελτιστοποιημένος πυρήνας πολλαπλασιασμού poly_mul_mod_q με PIPELINE II=1 και ARRAY_PARTITION factor=8.
- Hensel lifting για αντιστροφή mod q = 2048 σε 11 επαναλήψεις.
- Πλήρης testbench με σταθερό seed, retry loop και sanity checks για αναπαραγωγιμότητα αποτελεσμάτων.

Τα αποτελέσματα επιβεβαιώνουν ότι το Vitis HLS 2024.2 αποτελεί ώριμο εργαλείο για πρωτοτυποποίηση PQC hardware. Η μεθοδολογία HLS-first επιτρέπει ταχύτερο Design Space Exploration σε σχέση με RTL, με ανταγωνιστικά αποτελέσματα απόδοσης.

6.2 Μελλοντική Εργασία

- NTT-based πολλαπλασιασμός: Εξερεύνηση χρήσης NTT-friendly παραλλαγής NTRU για μείωση πολυπλοκότητας πολλαπλασιασμού από $O(N^2)$ σε $O(N \log N)$.
- Βελτιστοποιημένη αντιστροφή: Υλοποίηση half-GCD ή divstep-based αλγόριθμου [9] για δραστική μείωση του latency του keygen.
- Ανθεκτικότητα σε πλευρικά κανάλια: Ενσωμάτωση constant-time εκτέλεσης και masking αντιμέτρων έναντι επιθέσεων ισχύος (power analysis) και χρόνου (timing attacks).

- SoC ολοκλήρωση: Πλήρης ολοκλήρωση σε Zynq SoC με AXI4-Stream/AXI4-Lite διεπαφές και αξιολόγηση end-to-end TLS handshake.
- Επέκταση σε NTRU Prime / NTRU-HRSS: Εφαρμογή της ίδιας HLS μεθοδολογίας σε παραλλαγές με διαφορετική δομή δακτυλίου.

• Βιβλιογραφία

[1] J. Hoffstein, J. Pipher, και J. H. Silverman, "NTRU: A ring-based public key cryptosystem," στο Proc. Algorithmic Number Theory — ANTS-III, Lecture Notes in Computer Science, τόμ. 1423, Springer, Berlin, 1998, σσ. 267–288. DOI: 10.1007/BFb0054868

[2] NIST, "Post-Quantum Cryptography Standardization," National Institute of Standards and Technology, 2024. [Online]. Διαθέσιμο: https://csrc.nist.gov/projects/post-quantum-cryptography

[3] D. J. Bernstein και T. Lange, "Post-quantum cryptography," Nature, τόμ. 549, αρ. 7671, σσ. 188–194, Σεπ. 2017. DOI: 10.1038/nature23461

[4] AMD Xilinx, Vivado Design Suite HLS Reference Guide, UG902, έκδοση 2019.2, AMD Xilinx, 2019. [Μετεξελίχθη σε Vitis HLS από έκδοση 2020.1]

[5] C. Chen et al., "NTRU Algorithm Specifications and Supporting Documentation," NTRU Round 3 Submission, ntru.org, 2020. [Online]. Διαθέσιμο: https://ntru.org/

[6] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," SIAM Journal on Computing, τόμ. 26, αρ. 5, σσ. 1484–1509, 1997. DOI: 10.1137/S0097539795293172

[7] D. J. Bernstein et al., "NTRU Prime: Reducing Attack Surface at Low Cost," στο Selected Areas in Cryptography (SAC 2017), Lecture Notes in Computer Science, τόμ. 10719, Springer, 2018, σσ. 235–260.

[8] A. Marotzke, "A Constant Time Full Hardware Implementation of Streamlined NTRU Prime," στο Smart Card Research and Advanced Applications — CARDIS 2020, Lecture Notes in Computer Science, τόμ. 12609, Springer, 2021, σσ. 1–17. DOI: 10.1007/978-3-030-68487-7_1

[9] B.-Y. Peng, A. Marotzke, M.-H. Tsai, B.-Y. Yang, και H.-L. Chen, "Streamlined NTRU Prime on FPGA," Journal of Cryptographic Engineering, τόμ. 13, σσ. 167–186, 2023. DOI: 10.1007/s13389-022-00303-z. Cryptology ePrint: 2021/1444.

[10] V. B. Dang, F. Farahmand, M. Andrzejczak, K. Mohajerani, D. T. Nguyen, και K. Gaj, "Implementation and Benchmarking of Round 2 Candidates in the NIST Post-Quantum Cryptography Standardization Process Using Hardware and Software/Hardware Co-design Approaches," Cryptology ePrint Archive, Report 2020/795, 2020.

[11] V. B. Dang, K. Mohajerani, και K. Gaj, "High-Speed Hardware Architectures and FPGA Benchmarking of CRYSTALS-Kyber, NTRU, and Saber," Cryptology ePrint Archive, Report 2021/1508, 2021.

[12] E. Alkim et al., "Polynomial Multiplication in NTRU Prime: Comparison of Optimization Strategies on Cortex-M4," IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES), τόμ. 2021, αρ. 1, σσ. 217–238, 2021. DOI: 10.46586/tches.v2021.i1.217-238

[13] G. Mao, D. Chen, G. Li, W. Dai, A. I. Sanka, Ç. K. Koç, και R. C. C. Cheung, "High-Performance and Configurable SW/HW Co-Design of Post-Quantum Signature CRYSTALS-Dilithium," ACM Transactions on Reconfigurable Technology and Systems, τόμ. 16, αρ. 3, Article 44, 2023. DOI: 10.1145/3569580

[14] J. A. Montero-Paz et al., "Multi-Unit Serial Polynomial Multiplier to Accelerate NTRU-Based Cryptographic Schemes in IoT Embedded Systems," Sensors, τόμ. 22, αρ. 5, Article 1873, 2022. DOI: 10.3390/s22051873

[15] K. Gaj et al., "Evaluating the Potential for Hardware Acceleration of Four NTRU-Based Key Encapsulation Mechanisms Using Software/Hardware Codesign," στο Proc. 3rd NIST PQC Standardization Conference, 2021.

[16] A. Pellet-Mary και D. Stehlé, "On the Hardness of the NTRU Problem," Cryptology ePrint Archive, Report 2021/821, 2021. [Online]. Διαθέσιμο: https://eprint.iacr.org/2021/821

[17] F. Antognazza, A. Barenghi, G. Pelosi, και R. Susella, "Hardware Acceleration for High-Volume Operations of CRYSTALS-Kyber and CRYSTALS-Dilithium," ACM Transactions on Reconfigurable Technology and Systems, τόμ. 17, αρ. 3, 2024. DOI: 10.1145/3675172

[18] NIST, "Federal Information Processing Standard (FIPS) 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard," NIST, Gaithersburg, MD, 2024.

[19] AMD Xilinx, Vitis High-Level Synthesis User Guide, UG1399, έκδοση 2024.2, AMD Xilinx, 2024. [Online]. Διαθέσιμο: https://docs.amd.com/r/en-US/ug1399-vitis-hls

[20] AMD Xilinx, Zynq UltraScale+ MPSoC Technical Reference Manual, UG1085, AMD Xilinx, 2023.

Παράρτημα Α — Πλήρης Κώδικας HLS

Ο πλήρης πηγαίος κώδικας HLS παρατίθεται στις ενότητες που ακολουθούν. Ο κώδικας αποτελείται από τα εξής αρχεία: ntru_params.h (παράμετροι/τύποι), ntru_arith.h (αριθμητική), ntru_inv.h (αντιστροφή), ntru_top.h (ορισμοί top-level συναρτήσεων), ntru_top.cpp (wrapper ώστε το Vitis HLS να εντοπίζει τη top function για σύνθεση), ntru_decl.h (δηλώσεις μόνο, για το testbench — απαραίτητο ώστε το C/RTL co-simulation να γίνεται link χωρίς σφάλματα πολλαπλού ορισμού) και ntru_tb.cpp (testbench).

A.1 ntru_params.h

```
#ifndef NTRU_PARAMS_H

#define NTRU_PARAMS_H



#define NTRU_N       509

#define NTRU_P       3

#define NTRU_Q       2048

#define NTRU_Q_MASK  (NTRU_Q - 1)

#define NTRU_LOGQ    11

#define NTRU_L       (2 * NTRU_N)

#define COEFF_BITS   11

#define NTRU_DF      170

#define NTRU_DR      170

#define LFSR_SEED    0xDEADBEEFUL

#define LFSR_POLY    0xB4BCD35CUL



#ifdef NTRU_SW_BUILD

 #include "ap_int_stub.h"
```

```
#else
  #include <ap_int.h>
#endif


typedef ap_uint<COEFF_BITS>   coeff_t;
typedef ap_int<COEFF_BITS+1>  scoeff_t;
typedef ap_uint<32>           lfsr_t;
typedef ap_int<16>            eea_t;    // EEA scratch
typedef ap_uint<32>           acc_t;    // mul accumulator
typedef coeff_t               poly_t[NTRU_N];


#endif // NTRU_PARAMS_H
```

A.2 Κεντρικά Τμήματα ntru_arith.h

(Πλήρης κώδικας: βλ. συνοδευτικά αρχεία υποβολής)

```
// poly_mul_mod_q — Schoolbook cyclic convolution mod q
// poly_mul_mod_p — Cyclic convolution mod p
// poly_add_mod_q / poly_sub_mod_q
// centerlift_mod_q — [0,q) → (-q/2, q/2]
// centerlift_to_mod_p — signed → {0,1,2}
// lfsr_step — Galois LFSR one-step
// sample_trinary_signed — rejection sampling, signed τριαδικό (-1/0/+1)
// (f = 1 + p*F κατασκευάζεται inline στο ntru_keygen)
```

A.3 Κεντρικά Τμήματα ntru_inv.h

(Πλήρης κώδικας: βλ. συνοδευτικά αρχεία υποβολής)

```
// mod_inv_small  — Modular inverse for p ∈ {2,3}
// poly_divrem_p  — Full polynomial long division in GF(p)[x]
// poly_inv_mod_prime — EEA + reduction mod (x^N-1)
// poly_inv_mod_q     — Hensel lifting (11 iterations)
```

A.4 Tcl Script (ntru_hls.tcl)

```
# Vivado / Vitis HLS 2024.2 build script — NTRU-HPS
# Usage:  vitis_hls -f ntru_hls.tcl

set PART   {xczu7ev-ffvc1156-2-e}   ;# ZCU104
set PERIOD 10                       ;# 10 ns -> 100 MHz
set CFLAGS "-I./src"

proc build_core {top part period cflags} {
    open_project -reset prj_$top
    set_top $top
    add_files src/ntru_params.h  -cflags $cflags
    add_files src/ntru_arith.h   -cflags $cflags
    add_files src/ntru_inv.h     -cflags $cflags
    add_files src/ntru_top.h     -cflags $cflags
    add_files src/ntru_top.cpp   -cflags $cflags
    add_files -tb tb/ntru_tb.cpp -cflags $cflags
    open_solution -reset "solution1"
    set_part $part
    create_clock -period $period -name default
    csim_design        ;# C simulation
    csynth_design      ;# C -> RTL synthesis
    cosim_design       ;# C/RTL co-simulation
```

```
    close_solution

    close_project

}



# Build all three cores (one top each):

build_core ntru_keygen  $PART $PERIOD $CFLAGS

build_core ntru_encrypt $PART $PERIOD $CFLAGS

build_core ntru_decrypt $PART $PERIOD $CFLAGS

exit
```

A.5 ntru_top.cpp (Wrapper για Σύνθεση)

Το αρχείο αυτό εισάγει το ntru_top.h ώστε οι ορισμοί των top-level συναρτήσεων να βρίσκονται σε μονάδα μεταγλώττισης .cpp — απαίτηση του Vitis HLS για τον εντοπισμό της top function κατά τη σύνθεση.

```
#include "ntru_top.h"
```

A.6 ntru_decl.h (Δηλώσεις για το Testbench)

Το testbench εισάγει αυτό το header (αντί του ntru_top.h), το οποίο περιέχει μόνο δηλώσεις. Έτσι οι ορισμοί υπάρχουν μία μόνο φορά (μέσω του ntru_top.cpp) και το C/RTL co-simulation γίνεται link χωρίς σφάλματα πολλαπλού ορισμού (multiple definition).

```
#ifndef NTRU_DECL_H

#define NTRU_DECL_H

#include "ntru_params.h"



int  ntru_keygen (ap_uint<32> seed, coeff_t h[NTRU_N],

                  coeff_t f[NTRU_N], ap_uint<2> f_p[NTRU_N]);

void ntru_encrypt(const ap_uint<2> m[NTRU_N], const coeff_t h[NTRU_N],

                  ap_uint<32> seed, coeff_t e[NTRU_N]);

void ntru_decrypt(const coeff_t e[NTRU_N], const coeff_t f[NTRU_N],

                  const ap_uint<2> f_p[NTRU_N], ap_uint<2> m_out[NTRU_N]);
```

```
#endif // NTRU_DECL_H
```