



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Σχεδίαση και υλοποίηση σε FPGA επεξεργαστή κρυπτογραφημένου συνόλου εντολών

ΓΕΩΡΓΙΟΣ – ΔΑΥΙΔ ΤΣΙΤΛΑΟΥΡΙ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΓΕΩΡΓΙΟΣ ΔΗΜΗΤΡΙΟΥ
ΕΠΙΚΟΥΡΟΣ

Λαμία 1 Σεπτεμβρίου έτος 2025



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Σχεδίαση και υλοποίηση σε FPGA επεξεργαστή κρυπτογραφημένου συνόλου εντολών

ΓΕΩΡΓΙΟΣ – ΔΑΥΙΔ ΤΣΙΤΛΑΟΥΡΙ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΓΕΩΡΓΙΟΣ ΔΗΜΗΤΡΙΟΥ
ΕΠΙΚΟΥΡΟΣ

Λαμία 1 Σεπτεμβρίου έτος 2025



UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

Design and implementation of an encrypted instruction set processor on FPGA

GEORGE – DAVID TSITLAOURI

FINAL THESIS

ADVISOR

GEORGE DIMITRIOU
ASSISTANT PROFESSOR

Lamia 1 September year 2025

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις ⁽¹⁾, που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.
2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσιάσή τους ως δική μου εργασία.
3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια
4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: 01/09/2025

Ο Δηλ.

ΓΕΩΡΓΙΟΣ – ΔΑΥΙΔ ΤΣΙΤΛΑΟΥΡΙ

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία παρουσιάζει τον σχεδιασμό, την υλοποίηση και την αξιολόγηση ενός κρυπτογραφικού επεξεργαστή, του **CryptoCPU**, ο οποίος εκτελεί αποκλειστικά κρυπτογραφημένα προγράμματα αξιοποιώντας τον αλγόριθμο **AES-128** στο στάδιο προσκόμισης εντολών ενός εξασταδίου pipeline (Fetch → Decrypt → Decode → Execute → Memory → Write-Back). Ο επεξεργαστής στοχεύει σε FPGA **Artix-7** και αναπτύχθηκε με τα εργαλεία **Vitis Unified** και **Vivado Design Suite**.

Ο κώδικας εφαρμογών γράφεται σε **MIPS-like assembly** και μετασχηματίζεται με το περιβάλλον **MARS** σε δύο εξαγόμενα: **.data.hex** (μη κρυπτογραφημένη ενότητα δεδομένων) και **.text.hex** (εντολές). Η ενότητα **.text** κρυπτογραφείται με **AES-128** σε blocks των 16 bytes, ενώ η **.data** παραμένει σε απλή μορφή. Στη συνέχεια, τα δύο συγχωνεύονται σε ένα ενιαίο αρχείο **demo.hex**, το οποίο φορτώνεται στη μνήμη του CryptoCPU. Κατά την εκτέλεση, οι εντολές αποκρυπτογραφούνται δυναμικά (on-the-fly) στο στάδιο **Decrypt**, εξασφαλίζοντας ότι ποτέ δεν εμφανίζεται σε καθαρή μορφή κώδικας στη μνήμη ή στους διαύλους του FPGA.

Για την επαλήθευση της λειτουργίας, αναπτύχθηκε **testbench σε C++ (test.cpp)**, το οποίο τροφοδοτεί τον επεξεργαστή με το κρυπτογραφημένο πρόγραμμα και ελέγχει την ορθή φόρτωση και εκτέλεση των εντολών. Το testbench χρησιμοποιήθηκε σε όλα τα στάδια του **Vitis** (C-simulation, C-synthesis, RTL Co-simulation, IP packaging, implementation), παρέχοντας συνεχή αξιολόγηση της λειτουργικότητας του CryptoCPU.

Η εργασία τεκμηριώνει:

- (α) το θεωρητικό υπόβαθρο (FPGA, pipeline και hazards, instruction/data cache, AES),
- (β) την αρχιτεκτονική και τις προδιαγραφές του CryptoCPU (MIPS-like ISA με ειδικές οδηγίες AES, οργάνωση μνήμης, cache, hazards),
- (γ) την πλήρη ροή εργαλείων **MARS → AES → testbench → Vitis (C-sim, C-synth, RTL Co-sim) → IP packaging → Vivado block design**,
- (δ) τα αποτελέσματα προσομοιώσεων και υλοποίησης (utilization, timing),
- (ε) την ενσωμάτωση στο **Vivado block design** (clocking wizard, reset logic, CryptoCPU IP, I/O pins) και
- (στ) τη δοκιμή σε πραγματική πλακέτα **Artix-7**, όπου επιβεβαιώθηκε η ορθή λειτουργία με ενεργοποίηση της ένδειξης **led_done**.

Τα αποτελέσματα δείχνουν ότι το σχέδιο είναι **πλήρως υλοποιήσιμο** σε FPGA, ικανοποιεί τους χρονιστικούς περιορισμούς και χρησιμοποιεί λογικούς πόρους σε αποδεκτά επίπεδα. Επιπλέον, η μεθοδολογία αποδεικνύει πως ο σχεδιασμός επεξεργαστών με αποκλειστικά κρυπτογραφημένο instruction set είναι εφικτός, ασφαλής και πρακτικά αξιοποιήσιμος. Τέλος, συζητούνται μελλοντικές επεκτάσεις όπως

διαφορετικά modes AES, βελτιωμένος branch prediction, μεγαλύτερες caches και τεχνικές αντιμετώπισης side-channel επιθέσεων.

Λέξεις-κλειδιά: AES-128, FPGA Artix-7, MARS, MIPS-like ISA, testbench, Vitis Unified, Vivado, block design, on-the-fly decryption, pipeline.

ABSTRACT

This thesis presents the design, implementation, and evaluation of a cryptographic processor, the **CryptoCPU**, which exclusively executes encrypted programs by employing the **AES-128 algorithm** at the instruction fetch stage of a six-stage pipeline (Fetch → Decrypt → Decode → Execute → Memory → Write-Back). The processor targets the **Artix-7 FPGA** and was developed using the **Vitis Unified** and **Vivado Design Suite** toolchains.

Application code is written in a **MIPS-like assembly language** and processed through **MARS**, producing two outputs: `.data.hex` (unencrypted data section) and `.text.hex` (instructions). The `.text` section is encrypted with **AES-128** in 16-byte blocks, while the `.data` section remains unmodified. Both are then merged into a single **demo.hex** file, which is loaded into the CryptoCPU's memory. During execution, instructions are dynamically decrypted **on-the-fly** in the **Decrypt** stage, ensuring that plaintext instructions never appear in memory or on the FPGA buses.

To validate system functionality, a **C++ testbench (test.cpp)** was developed to supply the processor with the encrypted program and verify correct instruction loading and execution. The testbench was integrated into all stages of the **Vitis** flow (C-simulation, C-synthesis, RTL Co-simulation, IP packaging, and implementation), providing consistent evaluation of the CryptoCPU.

This thesis documents:

- (a) the theoretical background (FPGAs, pipelines and hazards, instruction/data caches, AES),
- (b) the architecture and specifications of the CryptoCPU (MIPS-like ISA with AES instructions, memory organization, cache policy, hazards),
- (c) the complete toolflow **MARS → AES encryption → testbench → Vitis (C-sim, C-synth, RTL Co-sim) → IP packaging → Vivado block design**,
- (d) simulation and implementation results (resource utilization, timing analysis),
- (e) the integration into the **Vivado block design** (clocking wizard, reset logic, CryptoCPU IP, I/O pins), and
- (f) testing on a real **Artix-7 FPGA board**, where correct functionality was confirmed by the activation of the **led_done** signal.

The results demonstrate that the design is **fully synthesizable and implementable** on FPGA, meeting timing constraints while using resources efficiently. Furthermore, the methodology shows that designing processors with encrypted instruction sets is feasible, secure, and practically deployable. Finally, possible extensions are discussed, including different AES modes, improved branch prediction, larger caches, and countermeasures against side-channel attacks.

Keywords: AES-128, FPGA Artix-7, MARS, MIPS-like ISA, testbench, Vitis Unified, Vivado, block design, on-the-fly decryption, pipeline.

Table of Contents

ΠΕΡΙΛΗΨΗ	I
ABSTRACT	III
ΚΕΦΑΛΑΙΟ 1 ΕΙΣΑΓΩΓΗ	2
(ΥΠΟΚΕΦΑΛΑΙΟ 1.1) ΣΤΟΧΟΙ ΚΑΙ ΚΙΝΗΤΡΑ	2
(ΕΝΟΤΗΤΑ 1.1.Α) ΚΙΝΗΤΡΑ ΣΧΕΔΙΑΣΗΣ ΚΡΥΠΤΟΓΡΑΦΙΚΩΝ ΕΠΕΞΕΡΓΑΣΤΩΝ	2
(ΕΝΟΤΗΤΑ 1.1.Β) ΣΤΟΧΟΙ ΣΧΕΔΙΑΣΗΣ CRYPTOCPU	3
(ΥΠΟΚΕΦΑΛΑΙΟ 1.2) ΣΥΜΒΟΛΗ ΚΑΙ ΔΟΜΗ ΕΡΓΑΣΙΑΣ	5
(ΕΝΟΤΗΤΑ 1.2.Α) ΣΥΜΒΟΛΗ ΤΗΣ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ	5
(ΕΝΟΤΗΤΑ 1.2.Β) ΔΟΜΗ ΚΕΦΑΛΑΙΩΝ	7
ΚΕΦΑΛΑΙΟ 2 ΒΙΒΛΙΟΓΡΑΦΙΚΗ ΕΠΙΣΚΟΠΗΣΗ	9
(ΥΠΟΚΕΦΑΛΑΙΟ 2.1) ΘΕΩΡΗΤΙΚΟ ΠΛΑΙΣΙΟ	9
(ΕΝΟΤΗΤΑ 2.1.Α) FPGA: ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΚΑΙ ΠΛΕΟΝΕΚΤΗΜΑΤΑ	9
(ΕΝΟΤΗΤΑ 2.1.Β) PIPELINE ΣΕ ΕΠΕΞΕΡΓΑΣΤΕΣ – HAZARD ΚΑΙ ΤΕΧΝΙΚΕΣ	10
(ΕΝΟΤΗΤΑ 2.1.Γ) ΑΛΓΟΡΙΘΜΟΣ AES ΚΑΙ ΚΡΥΠΤΟΓΡΑΦΙΚΕΣ ΕΦΑΡΜΟΓΕΣ	12
(ΥΠΟΚΕΦΑΛΑΙΟ 2.2) ΣΧΕΤΙΚΗ ΈΡΕΥΝΑ	14
(ΕΝΟΤΗΤΑ 2.2.Α) ΚΡΥΠΤΟΓΡΑΦΙΚΟΙ ΕΠΕΞΕΡΓΑΣΤΕΣ ΣΕ FPGA	14
ΠΙΝΑΚΑΣ ΣΥΓΚΡΙΣΗΣ CRYPTOCPU VS ΠΑΡΑΔΟΣΙΑΚΕΣ ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ	16
(ΕΝΟΤΗΤΑ 2.2.Β) ΤΕΧΝΙΚΕΣ ΑΣΦΑΛΕΙΑΣ ΜΕ ON-THE-FLY DECRYPTION	17
ΚΕΦΑΛΑΙΟ 3 ΣΧΕΔΙΑΣΗ ΕΠΕΞΕΡΓΑΣΤΗ	19
(ΥΠΟΚΕΦΑΛΑΙΟ 3.1) ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΚΑΙ PIPELINE	19
(ΕΝΟΤΗΤΑ 3.1.Α) ISA ΚΑΙ ΕΙΔΙΚΕΣ ΕΝΤΟΛΕΣ	19
(ΕΝΟΤΗΤΑ 3.1.Β) I-CACHE ΚΑΙ D-CACHE ΥΛΟΠΟΙΗΣΗ	20
(ΕΝΟΤΗΤΑ 3.1.Γ) ΕΝΣΩΜΑΤΩΣΗ AES ΣΤΟ ΣΤΑΔΙΟ DECRYPT	22
(ΕΝΟΤΗΤΑ 3.1.Δ) HAZARDS HANDLING (DATA & CONTROL HAZARDS)	24
(ΥΠΟΚΕΦΑΛΑΙΟ 3.2) ΡΟΗ ΕΡΓΑΛΕΙΩΝ ΚΑΙ ΑΡΧΕΙΑ ΕΙΣΟΔΟΥ	25
(ΕΝΟΤΗΤΑ 3.2.Α) ASSEMBLY ΚΩΔΙΚΑΣ ΚΑΙ MARS	25
(ΕΝΟΤΗΤΑ 3.1.Β) ΚΡΥΠΤΟΓΡΑΦΗΣΗ ΚΑΙ ΠΑΡΑΓΩΓΗ DEMO.HEX	27
ΚΕΦΑΛΑΙΟ 4 VITIS & VIVADO	29
(ΥΠΟΚΕΦΑΛΑΙΟ 4.1) VITIS	29
(ΕΝΟΤΗΤΑ 4.1.Α) C SIMULATION	29
(ΕΝΟΤΗΤΑ 4.2.Β) C SYNTHESIS	31
(ΕΝΟΤΗΤΑ 4.2.Γ) RTL Co-SIMULATION	31
(ΕΝΟΤΗΤΑ 4.2.Δ) IP PACKAGING	32
(ΕΝΟΤΗΤΑ 4.2.Ε) IMPLEMENTATION	34
(ΥΠΟΚΕΦΑΛΑΙΟ 4.2) VIVADO	35

(ΕΝΟΤΗΤΑ 4.2.Α) BLOCK DESIGN	35
(ΕΝΟΤΗΤΑ 4.2.Β) PIN ASSIGNMENT	37
(ΕΝΟΤΗΤΑ 4.2.Γ) IMPLEMENTATION REPORTS	38
(ΕΝΟΤΗΤΑ 4.2.Δ) ΕΚΤΕΛΕΣΗ ΣΕ ΠΡΑΓΜΑΤΙΚΗ ΠΛΑΚΕΤΑ	39
<u>ΚΕΦΑΛΑΙΟ 5 ΣΥΜΠΕΡΑΣΜΑΤΑ.....</u>	<u>40</u>
<u>ΒΙΒΛΙΟΓΡΑΦΙΑ.....</u>	<u>42</u>

ΚΕΦΑΛΑΙΟ 1 Εισαγωγή

(Υποκεφάλαιο 1.1) Στόχοι και Κίνητρα

(Ενότητα 1.1.Α) Κίνητρα Σχεδίασης Κρυπτογραφικών Επεξεργαστών

Η ανάγκη για ενισχυμένη ασφάλεια σε ψηφιακά συστήματα έχει αναδείξει την εκτέλεση κρυπτογραφημένων εντολών ως κρίσιμο ζήτημα, ειδικά σε εφαρμογές όπου η ακεραιότητα και η εμπιστευτικότητα του κώδικα είναι εξίσου σημαντικές με την υπολογιστική απόδοση. Σε αυτό το πλαίσιο, οι **κρυπτογραφικοί επεξεργαστές** (cryptographic processors) αποτελούν μια αναδυόμενη κατηγορία αρχιτεκτονικών που στοχεύουν στην εκτέλεση προστατευμένων προγραμμάτων χωρίς αποκάλυψη του εσωτερικού τους περιεχομένου.

Η υλοποίηση της ασφάλειας σε επίπεδο hardware, και ειδικότερα εντός του pipeline, προσφέρει πλεονεκτήματα που δεν μπορούν να επιτευχθούν μόνο με τεχνικές λογισμικού. Οι λόγοι για τους οποίους η ενσωμάτωση της κρυπτογράφησης στο ίδιο τον επεξεργαστή είναι στρατηγικής σημασίας, περιλαμβάνουν:

- **Αποτροπή υποκλοπής κώδικα:** Όταν οι εντολές διατηρούνται κρυπτογραφημένες μέχρι τη στιγμή της εκτέλεσης, εξαλείφεται το ενδεχόμενο ανάγνωσης από κακόβουλες μονάδες παρακολούθησης.
- **Μείωση επιφάνειας επίθεσης:** Ο αποκλεισμός plaintext κώδικα από τη μνήμη και τους διαύλους μειώνει τα σημεία στα οποία μπορεί να παρέμβει ένας επιτιθέμενος.
- **Προστασία έναντι της αντίστροφης μηχανικής:** Η χρήση δυναμικής (on-the-fly) αποκρυπτογράφησης εντός του pipeline περιορίζει δραστικά την πιθανότητα επιτυχούς reverse engineering μέσω side-channel ή timing attacks.
- **Ασφάλεια by design:** Ο επεξεργαστής λειτουργεί υπό την προϋπόθεση ότι τίποτα δεν είναι έμπιστο μέχρι να αποκρυπτογραφηθεί εντός του ελεγχόμενου pipeline.

Η συμβατική προσέγγιση — αποθήκευση κρυπτογραφημένων εντολών αλλά αποκρυπτογράφηση πριν το pipeline — δημιουργεί ένα κρίσιμο "παράθυρο ευπάθειας", όπου οι εντολές βρίσκονται σε plaintext μορφή. Η λύση που προτείνεται στην παρούσα εργασία μέσω του **CryptoCPU** έρχεται να καλύψει αυτό το κενό, εισάγοντας ένα νέο μοντέλο εκτέλεσης:

- Οι εντολές αποθηκεύονται αποκλειστικά σε κρυπτογραφημένη μορφή και αποκρυπτογραφούνται μόνο στο **Decrypt stage** του pipeline.
- Ο αλγόριθμος AES-128 ενσωματώνεται ως αναπόσπαστο κομμάτι της αρχιτεκτονικής και δεν λειτουργεί ως ανεξάρτητη περιφερειακή μονάδα.
- Καμία εντολή δεν είναι ποτέ διαθέσιμη σε plaintext έξω από το αυστηρά ελεγχόμενο περιβάλλον του pipeline.

Αυτή η προσέγγιση επιτρέπει τη δημιουργία **αυτο-αμυνόμενων επεξεργαστών**, οι οποίοι είναι κατάλληλοι για κρίσιμες εφαρμογές όπως:

- **Ασφαλή embedded συστήματα** (π.χ. σε IoT ή medical devices),
- **Χρηματοοικονομικές και στρατιωτικές υποδομές**, όπου η αποκάλυψη εντολών μπορεί να οδηγήσει σε σοβαρές παραβιάσεις,
- **Συστήματα cloud και virtualization**, στα οποία η απόκρυψη του λογισμικού χρήστη από τον πάροχο είναι επιθυμητή.

Συνοψίζοντας, η σχεδίαση κρυπτογραφικών επεξεργαστών όπως ο **CryptoCPU** ενσωματώνει τις αρχές της ασφάλειας στο ίδιο το hardware, δημιουργώντας ένα νέο επίπεδο προστασίας που υπερβαίνει τις παραδοσιακές λύσεις. Η εργασία αυτή αποδεικνύει ότι τέτοιες αρχιτεκτονικές είναι όχι μόνο απαραίτητες, αλλά και **τεχνικά υλοποιήσιμες με αποδεκτές επιδόσεις**, ακόμη και σε πλατφόρμες με περιορισμένους πόρους.

(Ενότητα 1.1.B) Στόχοι Σχεδίασης CRYPTOCPU

Η παρούσα εργασία αποσκοπεί στον σχεδιασμό και την υλοποίηση ενός ασφαλούς επεξεργαστή που εκτελεί αποκλειστικά κρυπτογραφημένα προγράμματα. Οι στόχοι της σχεδίασης του **CryptoCPU** καθορίστηκαν με βάση την ανάγκη για **ασφαλή εκτέλεση λογισμικού σε αναδιαμορφώσιμες πλατφόρμες** όπως τα FPGA, χωρίς να γίνονται συμβιβασμοί σε επιδόσεις ή συμβατότητα. Αναλυτικά, οι βασικοί στόχοι περιλαμβάνουν:

1. Ενσωμάτωση AES-128 στο pipeline του επεξεργαστή

Ο κυριότερος στόχος είναι η **πλήρης ενσωμάτωση του αλγορίθμου AES-128** στο εσωτερικό pipeline του CryptoCPU, με τη δημιουργία ενός νέου σταδίου αποκρυπτογράφησης (Decrypt). Συγκεκριμένα:

- Το στάδιο **Decrypt** τοποθετείται **μεταξύ των σταδίων Fetch και Decode**, επεξεργαζόμενο κάθε block 16 bytes εντολών που ανακτώνται από τη μνήμη.
- Η αποκρυπτογράφηση γίνεται **on-the-fly**, εξασφαλίζοντας ότι οι εντολές παραμένουν **κρυπτογραφημένες σε όλο το εύρος της μνήμης και των διαύλων**.
- Το Decrypt δεν λειτουργεί ως εξωτερική μονάδα αλλά ως **οργανικό κομμάτι του pipeline**, διατηρώντας τη συνέπεια της ροής εκτέλεσης.

2. Συμβατότητα με MIPS-like assembly

Για να διευκολυνθεί η υιοθέτηση από χρήστες, η αρχιτεκτονική σχεδιάστηκε ώστε να είναι συμβατή με μια **MIPS-like Instruction Set Architecture (ISA)**:

- Η ανάπτυξη εφαρμογών πραγματοποιείται με τη χρήση του **MARS (MIPS Assembler and Runtime Simulator)**, ένα περιβάλλον γνωστό και εύχρηστο.
- Η επιλογή MIPS-like ISA επιτρέπει την **ταχεία ανάπτυξη κώδικα**, ενώ διατηρεί τη σαφήνεια και απλότητα των εντολών.
- Η σύνταξη assembly παραμένει οικεία, προσφέροντας **εκπαιδευτική αξία** και δυνατότητα άμεσης αξιοποίησης υπαρχόντων παραδειγμάτων.

3. Διαχωρισμός δεδομένων και εντολών

Ένα σημαντικό λειτουργικό χαρακτηριστικό είναι ο **διαχωρισμός μεταξύ των εντολών (.text) και των δεδομένων (.data)**:

- Μόνο το τμήμα **.text υποβάλλεται σε AES κρυπτογράφηση**, καθώς αφορά τον εκτελέσιμο κώδικα.
- Το τμήμα **.data παραμένει σε plaintext**, επιτρέποντας φυσιολογική προσπέλαση μεταβλητών και τιμών κατά την εκτέλεση.
- Αυτή η αρχιτεκτονική διατήρησης του διαχωρισμού επιτρέπει **απλότητα στον χειρισμό μνήμης** χωρίς να υποβαθμίζεται η ασφάλεια του instruction flow.

4. Υλοποιησιμότητα σε FPGA

Ένας από τους κύριους στόχους είναι να αποδειχθεί ότι ο CryptoCPU μπορεί να υλοποιηθεί **πλήρως σε πραγματική πλατφόρμα FPGA**. Πιο συγκεκριμένα:

- Η σχεδίαση στοχεύει στην πλακέτα **Xilinx Artix-7**, η οποία χρησιμοποιείται συχνά σε εκπαιδευτικά και ερευνητικά περιβάλλοντα.
- Η όλη διαδικασία ανάπτυξης γίνεται με εργαλεία **Vitis Unified** για high-level synthesis (HLS) και **Vivado Design Suite** για σύνθεση, τοποθέτηση και υλοποίηση σε επίπεδο υλικού.
- Η επιτυχής υλοποίηση του CryptoCPU σε FPGA αποδεικνύει ότι η προσέγγιση αυτή δεν είναι θεωρητική αλλά **απόλυτα πρακτική και εφαρμόσιμη**.

5. Έλεγχος εγκυρότητας κλειδιού

Για να ενισχυθεί η ασφάλεια, εισάγεται μηχανισμός ελέγχου εγκυρότητας του κρυπτογραφικού κλειδιού:

- Το **κλειδί AES φορτώνεται** από προκαθορισμένο block μνήμης κατά την εκκίνηση του επεξεργαστή.
- Πριν την εκτέλεση οποιουδήποτε κρυπτογραφημένου κώδικα, πραγματοποιείται **έλεγχος αυθεντικότητας του κλειδιού**.
- Αν εντοπιστεί αλλοιωμένο ή λανθασμένο κλειδί, η εκτέλεση διακόπτεται, εμποδίζοντας την παραποίηση ή παραβίαση της εφαρμογής.

6. Απόδειξη πρακτικής αποδοτικότητας και σωστής λειτουργίας

Τέλος, σημαντικό μέρος της εργασίας επικεντρώνεται στην **πειραματική επαλήθευση και μέτρηση της λειτουργίας του CryptoCPU**:

- Το αρχείο **test.cpp** χρησιμοποιείται ως **testbench** και φορτώνει το κρυπτογραφημένο πρόγραμμα (demo.hex) στη μνήμη του επεξεργαστή.
- Μετά την αποκρυπτογράφηση, εκτελείται έλεγχος εγκυρότητας μέσω **signature**, όπου ελέγχεται αν η τιμή της θέσης memory[156] ισούται με 0x15.
- Οι προσομοιώσεις (C Simulation) και τα **logs του Vitis** επιβεβαιώνουν ότι το Decrypt stage εκτελείται **χωρίς σφάλματα**, και η συνολική λειτουργία είναι απολύτως **προβλέψιμη και ακριβής**.

- Αυτή η διαδικασία **τεκμηριώνει την αποδοτικότητα της αρχιτεκτονικής**, ενώ ανοίγει τον δρόμο για ενσωμάτωση σε πιο σύνθετα συστήματα.

Συνοψίζοντας, η σχεδίαση του CryptoCPU βασίστηκε σε **συγκεκριμένους, μετρήσιμους και επιτεύξιμους στόχους**. Η αρχιτεκτονική του επεξεργαστή συνδυάζει ασφάλεια, συμβατότητα, λειτουργικότητα και επεκτασιμότητα, καθιστώντας τον κατάλληλο για μελλοντική ανάπτυξη ασφαλών embedded συστημάτων πάνω σε FPGA.

(Υποκεφάλαιο 1.2) Συμβολή και Δομή Εργασίας

(Ενότητα 1.2.A) Συμβολή της Πτυχιακής Εργασίας

Η παρούσα πτυχιακή εργασία επιχειρεί να καλύψει ένα σημαντικό κενό στο πεδίο της ασφάλειας λογισμικού και της εκτέλεσης κρυπτογραφημένων προγραμμάτων σε αναδιαμορφώσιμες πλατφόρμες FPGA. Η συμβολή της έγκειται όχι μόνο στην αρχιτεκτονική καινοτομία του CryptoCPU, αλλά και στην πλήρη υλοποίηση, επαλήθευση και πρακτική του εφαρμογή σε πραγματικό υλικό. Πιο αναλυτικά:

1. Αρχιτεκτονικός σχεδιασμός επεξεργαστή με εξαστάδιο pipeline και ενσωματωμένο AES-128

Η πιο θεμελιώδης συνεισφορά της εργασίας είναι η σχεδίαση ενός επεξεργαστή με **εξαστάδιο pipeline** (Fetch → Decrypt → Decode → Execute → Memory → Write-Back), ο οποίος ενσωματώνει **τον αλγόριθμο AES-128** ως αναπόσπαστο κομμάτι της αρχιτεκτονικής του.

- Η τοποθέτηση του **Decrypt stage** εντός του pipeline καθιστά δυνατή την **on-the-fly αποκρυπτογράφηση** των εντολών, χωρίς την ανάγκη για εξωτερική αποκρυπτογραφητική μονάδα.
- Το αποτέλεσμα είναι ένας **ασφαλής επεξεργαστής**, στον οποίο καμία εντολή δεν εμφανίζεται ποτέ σε μορφή plaintext στη μνήμη ή στους διαύλους, διασφαλίζοντας απόλυτη μυστικότητα.

2. Ολοκληρωμένη ροή εργαλείων για ασφαλή μετατροπή assembly σε εκτελέσιμο .hex

Η εργασία υλοποιεί μια **πλήρη αλυσίδα εργαλείων** που συνδέει το λογισμικό επίπεδο (assembly) με την υλική υλοποίηση (FPGA):

- Ο κώδικας ξεκινά να γράφεται σε **MIPS-like assembly** μέσω του εργαλείου **MARS**, που εξάγει δύο αρχεία .text.hex και .data.hex.
- Το αρχείο .text.hex περνά από πρόγραμμα κρυπτογράφησης (**encryptor.cpp**), που εφαρμόζει AES-128 σε blocks των 16 bytes.
- Τα δύο κομμάτια συγχωνεύονται στο τελικό αρχείο **demo.hex**, το οποίο φορτώνεται στη μνήμη του CryptoCPU.
- Η διαδικασία είναι **αναπαραγώγιμη, αυτόνομη και πλήρως τεκμηριωμένη**, καθιστώντας την κατάλληλη για επαναχρησιμοποίηση ή επέκταση.

3. Εκτέλεση αποκλειστικά κρυπτογραφημένων προγραμμάτων με έλεγχο κλειδιού

Σε αντίθεση με άλλες αρχιτεκτονικές όπου η αποκρυπτογράφηση είναι προαιρετική ή εξωτερική, ο CryptoCPU **απαιτεί υποχρεωτικά κρυπτογραφημένο κώδικα** για να εκτελέσει οτιδήποτε.

- Η εκκίνηση της εκτέλεσης περιλαμβάνει **έλεγχο εγκυρότητας του AES κλειδιού**, που αποτρέπει την τυχαία ή κακόβουλη εκτέλεση αλλοιωμένων εντολών.
- Αυτή η σχεδίαση εφαρμόζει **ασφάλεια by design**, εισάγοντας έναν μηχανισμό **αυτοάμυνας της αρχιτεκτονικής**, πριν ακόμη φτάσουμε σε ζητήματα λειτουργίας ή χρήσης.

4. Ενσωμάτωση σε FPGA με πλήρη αλυσίδα επαλήθευσης και υλοποίησης

Η εργασία δεν περιορίστηκε σε θεωρητική προσέγγιση αλλά **ολοκληρώθηκε με επιτυχία σε πραγματική FPGA πλατφόρμα**. Η πορεία περιλάμβανε τα εξής βήματα:

- **C Simulation**: έλεγχος της βασικής λειτουργίας του CryptoCPU και της αποκρυπτογράφησης σε επίπεδο C++.
- **C Synthesis**: μετάφραση του κώδικα σε RTL επίπεδο και παραγωγή hardware περιγραφής.
- **RTL Co-Simulation**: διασύνδεση C++ testbench με RTL μονάδες για επιβεβαίωση συμπεριφοράς.
- **IP Packaging**: μετατροπή της μονάδας CryptoCPU σε επαναχρησιμοποιήσιμο IP block.
- **Vivado Block Design & Implementation**: τοποθέτηση του IP σε πραγματικό σύστημα και εκχώρηση pins, ρολογιού, reset κ.λπ.

Η πλήρης αυτή ροή καταδεικνύει ότι η σχεδίαση είναι **ώριμη για ενσωμάτωση σε μεγαλύτερα συστήματα SoC** ή άλλες εφαρμογές.

5. Επιτυχής λειτουργία σε πραγματική πλακέτα FPGA (Xilinx Artix-7)

Η τελική και πιο απτή απόδειξη της επιτυχίας του CryptoCPU είναι η **λειτουργία του στην πραγματική πλακέτα Xilinx Artix-7**.

- Μετά τη σύνθεση του bitstream και τη μεταφορά στον στόχο, το **σήμα led_done ενεργοποιήθηκε**, επιβεβαιώνοντας την ορθή εκτέλεση του κρυπτογραφημένου προγράμματος.
- Τα αποτελέσματα ταυτίστηκαν με τις προσομοιώσεις, αποδεικνύοντας ότι το σύστημα παραμένει **αξιόπιστο, σταθερό και συνεπές** ακόμη και σε υλοποίηση χαμηλού επιπέδου.

Συνολική εκτίμηση συμβολής

Η εργασία προσφέρει μια **ολιστική προσέγγιση στον σχεδιασμό ασφαλών επεξεργαστών**, από τον κώδικα και τον αλγόριθμο AES έως την υλοποίηση σε υλικό. Η δημιουργία ενός MIPS-like CryptoCPU με αποκλειστική εκτέλεση κρυπτογραφημένων εντολών, πλήρως λειτουργικού σε FPGA, συνιστά **ένα χειροπιαστό βήμα προς τη συνένωση ασφάλειας και απόδοσης σε σύγχρονες αναδιαμορφώσιμες αρχιτεκτονικές**.

(Ενότητα 1.2.B) Δομή Κεφαλαίων

Η παρούσα πτυχιακή εργασία δομείται σε πέντε κύρια κεφάλαια, τα οποία οργανώνονται με τρόπο που οδηγεί τον αναγνώστη σταδιακά από τη θεωρητική βάση και τα κίνητρα, στη σχεδίαση, την υλοποίηση και την αξιολόγηση του CryptoCPU. Κάθε κεφάλαιο έχει σχεδιαστεί ώστε να είναι αυτοτελές, τεχνικά συνεκτικό και ταυτόχρονα διασυνδεδεμένο με τα υπόλοιπα, προσφέροντας μια ολοκληρωμένη εικόνα του έργου.

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ, ΣΤΟΧΟΙ ΚΑΙ ΚΙΝΗΤΡΑ

Το πρώτο κεφάλαιο θέτει τις βάσεις για την εργασία:

- Εισάγει το πλαίσιο των αναδιαμορφώσιμων συστημάτων και την ανάγκη για ασφάλεια στα FPGA.
- Παρουσιάζει τα κίνητρα και τους στόχους σχεδίασης του CryptoCPU.
- Περιλαμβάνει τη συμβολή της εργασίας, καθώς και τη συνοπτική περιγραφή της δομής της.

ΚΕΦΑΛΑΙΟ 2: ΒΙΒΛΙΟΓΡΑΦΙΚΗ ΕΠΙΣΚΟΠΗΣΗ

Στο δεύτερο κεφάλαιο παρουσιάζεται το θεωρητικό υπόβαθρο:

- Τα χαρακτηριστικά και τα πλεονεκτήματα των FPGA.
- Η έννοια του pipeline στους επεξεργαστές, μαζί με τα σχετικά hazards και τρόπους αντιμετώπισής τους.
- Ο αλγόριθμος AES και η χρήση του σε υλικό.
- Σχετική ερευνητική δραστηριότητα σε κρυπτογραφικούς επεξεργαστές και τεχνικές on-the-fly αποκρυπτογράφησης.

ΚΕΦΑΛΑΙΟ 3: ΣΧΕΔΙΑΣΗ ΕΠΕΞΕΡΓΑΣΤΗ

Το τρίτο κεφάλαιο παρουσιάζει τον πυρήνα του CryptoCPU:

- Περιγράφεται η αρχιτεκτονική, η ISA, το pipeline και οι cache μνήμες.
- Αναλύεται το ειδικό στάδιο Decrypt και η ενσωμάτωση του AES.
- Παρουσιάζεται ο χειρισμός των data και control hazards.
- Γίνεται λεπτομερής περιγραφή της ροής εργαλείων, από τη συγγραφή assembly στο MARS, μέχρι την παραγωγή του demo.hex μέσω κρυπτογράφησης.
- Εξετάζεται η λειτουργία των encryptor.cpp και test.cpp, καθώς και η διαδικασία offline επαλήθευσης.

ΚΕΦΑΛΑΙΟ 4: VITIS ΚΑΙ VIVADO

Το τέταρτο κεφάλαιο αναφέρεται στην πρακτική υλοποίηση:

- Παρουσιάζεται η ροή στο Vitis HLS: από το C Simulation, μέχρι το IP Packaging και την export διαδικασία.
- Περιγράφεται το block design και η ολοκλήρωση στο Vivado.
- Δίνεται έμφαση στην εφαρμογή του CryptoCPU σε φυσική πλακέτα (FPGA), καθώς και στα implementation reports που τεκμηριώνουν τη λειτουργικότητα και τις επιδόσεις του.

ΚΕΦΑΛΑΙΟ 5: ΣΥΜΠΕΡΑΣΜΑΤΑ

Το τελευταίο κεφάλαιο συνοψίζει:

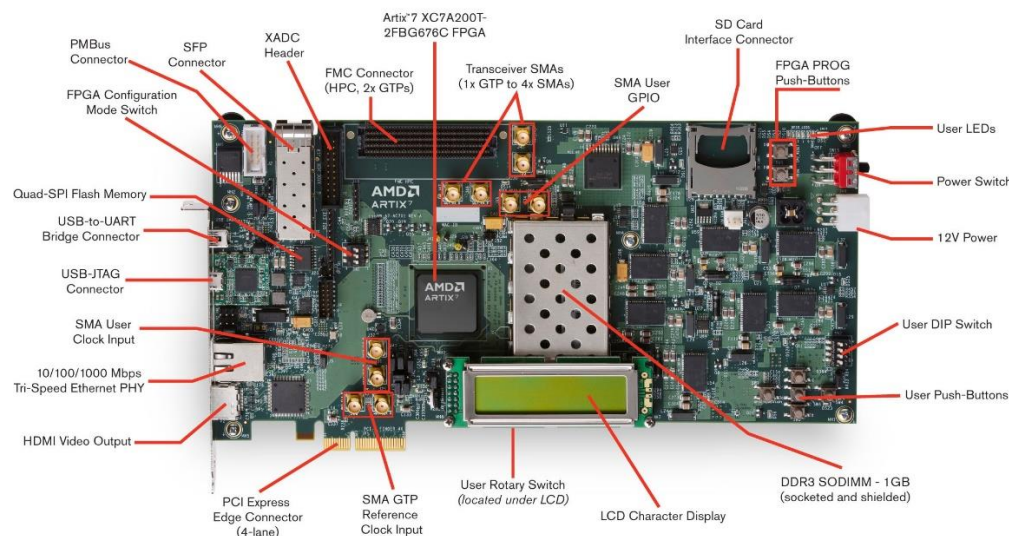
- Τα βασικά αποτελέσματα της εργασίας.
- Τα πλεονεκτήματα της αρχιτεκτονικής CryptoCPU, αλλά και τους περιορισμούς που παρατηρήθηκαν.
- Προτάσεις για μελλοντική επέκταση, όπως η υποστήριξη νέων AES modes, η προσθήκη branch prediction, και η υλοποίηση countermeasures ενάντια σε side-channel επιθέσεις.

ΚΕΦΑΛΑΙΟ 2 Βιβλιογραφική Επισκόπηση

(Υποκεφάλαιο 2.1) Θεωρητικό Πλαίσιο

(Ενότητα 2.1.A) FPGA: χαρακτηριστικά και πλεονεκτήματα

Τα Field Programmable Gate Arrays (FPGA) αποτελούν βασικό πυλώνα του επαναπρογραμματιζόμενου υλικού, παρέχοντας τη δυνατότητα διαμόρφωσης του κυκλώματος ακόμα και μετά την κατασκευή του. Η ευελιξία αυτή τα καθιστά ιδανικά για ένα ευρύ φάσμα εφαρμογών – από embedded συστήματα και ρομποτική, έως κρυπτογραφία και ασφάλεια.



Αρχιτεκτονική FPGA

Ένα FPGA αποτελείται από:

- **Λογικές Κυψέλες (Logic Cells):** Συνδυασμός LUTs και Flip-Flops για υλοποίηση λογικών συναρτήσεων και καταστάσεων.
- **Δίαυλοι Διασύνδεσης (Routing):** Επιτρέπουν αυθαίρετη σύνδεση μεταξύ κυψελών.
- **Μνήμη Διαμόρφωσης (Configuration Memory):** Περιέχει το bitstream που καθορίζει τη συμπεριφορά του κυκλώματος.
- **Ενσωματωμένες Μονάδες:** Π.χ. DSPs, Block RAMs, και ARM πυρήνες σε σύγχρονα FPGA (όπως τα Zynq SoC).

Πλεονεκτήματα έναντι CPU/GPU/ASIC

Χαρακτηριστικό	FPGA	CPU/GPU	ASIC
Επαναπρογραμματιζόμενο	Ναι	Ναι(μέσω software)	Όχι
Παράλληλη εκτέλεση	Πλήρης	Περιορισμένη	Πλήρης
Επίδοση ανά Watt	Καλή	Υψηλή (GPU)	Άριστη
Χρόνος Ανάπτυξης	Μεσαίος	Ταχύτερος	Πολύ μεγάλος
Κόστος για μικρές σειρές	Χαμηλό	Πολύ χαμηλό	Πολύ υψηλό

Χαρακτηριστικό	FPGA	CPU/GPU	ASIC
Προσαρμοστικότητα	Υψηλή	Περιορισμένη	Καθόλου

Η **ευελιξία και παραλληλία** που προσφέρουν τα FPGA τα καθιστούν κατάλληλα για:

- Επεξεργασία σήματος και εικόνας
- Machine Learning accelerators
- Κρυπτογραφικά modules
- Embedded real-time συστήματα

FPGA για Ασφαλείς Υλοποιήσεις – Συσχέτιση με CryptoCPU

Τα FPGA παρέχουν κρίσιμα πλεονεκτήματα για την υλοποίηση **ασφαλών επεξεργασιών**, όπως ο CryptoCPU:

1. **Hardware-based ασφάλεια:** Οι αλγόριθμοι (π.χ. AES) υλοποιούνται απευθείας στο υλικό, προσφέροντας ταχύτητα και απομόνωση.
2. **Side-channel προστασία:** Δυνατότητα υλοποίησης τεχνικών masking, randomization, και isolation.
3. **Ευκολία αναβάθμισης:** Δυνατότητα υποστήριξης νέων αλγορίθμων χωρίς αλλαγή hardware.

Προκλήσεις και Περιορισμοί

- **Ταχύτητα:** Υστερούν σε συχνότητα έναντι ASIC.
- **Κατανάλωση:** Υψηλότερη σε πλήρες φόρτο.
- **Απαιτήσεις ανάπτυξης:** Χρειάζονται γνώσεις HDL (π.χ. VHDL/Verilog).
- **Ασφάλεια bitstream:** Η προστασία της διαμόρφωσης είναι κρίσιμη.

Συμπερασματικά

Τα FPGA γεφυρώνουν το χάσμα ανάμεσα στη **γενικής χρήσης επεξεργασία** και το **εξειδικευμένο hardware**, προσφέροντας μια μοναδική πλατφόρμα για αρχιτεκτονικές όπως ο CryptoCPU, όπου **η ασφάλεια, η ταχύτητα και η προσαρμοστικότητα** αποτελούν απαραίτητα χαρακτηριστικά.

(Ενότητα 2.1.B) Pipeline σε επεξεργαστές – Hazard και τεχνικές

Η τεχνική του *instruction pipelining* αποτελεί ακρογωνιαίο λίθο στον σχεδιασμό σύγχρονων επεξεργαστών, καθώς στοχεύει στην αύξηση της απόδοσης μέσω ταυτόχρονης εκτέλεσης πολλαπλών εντολών. Το pipeline διαχωρίζει τη διαδικασία εκτέλεσης εντολών σε διακριτά στάδια, επιτρέποντας σε διαφορετικές εντολές να βρίσκονται ταυτόχρονα σε διαφορετικά στάδια της επεξεργασίας.

Η κλασική υλοποίηση περιλαμβάνει πέντε βασικά στάδια:

1. **Fetch (IF):** Ανάκτηση εντολής από τη μνήμη.
2. **Decode (ID):** Ανάλυση της εντολής και ανάγνωση καταχωρητών.
3. **Execute (EX):** Εκτέλεση αριθμητικών/λογικών πράξεων.
4. **Memory (MEM):** Ανάγνωση ή εγγραφή από/σε μνήμη.
5. **Write Back (WB):** Εγγραφή αποτελέσματος πίσω στους καταχωρητές.

Ο CryptoCPU εισάγει ένα **έκτο στάδιο – Decrypt**, το οποίο παρεμβάλλεται μεταξύ του Fetch και του Decode, ώστε οι εντολές να αποκρυπτογραφούνται πριν από την αποκωδικοποίηση. Η ενσωμάτωση του **Decrypt stage** επιβάλλει νέες απαιτήσεις συγχρονισμού και δημιουργεί επιπλέον καθυστερήσεις (latency) που πρέπει να αντιμετωπιστούν κατάλληλα μέσω τεχνικών pipeline optimization.

Είδη hazards (κινδύνων)

Κατά την εκτέλεση πολλαπλών εντολών σε pipeline, εμφανίζονται συγκρούσεις που ονομάζονται **hazards**, οι οποίες διακρίνονται σε τρεις βασικές κατηγορίες:

1. **Data Hazards (δεδομένων):** Όταν μια εντολή εξαρτάται από το αποτέλεσμα μιας προηγούμενης που δεν έχει ακόμα ολοκληρωθεί.

Παράδειγμα:

```
add $t0, $t1, $t2
sub $t3, $t0, $t4    # $t0 δεν είναι ακόμα διαθέσιμο
```

Αν δεν αντιμετωπιστεί, η εντολή sub θα χρησιμοποιήσει λανθασμένη τιμή.

2. **Control Hazards (ελέγχου ροής):** Προκύπτουν κυρίως από εντολές διακλάδωσης (π.χ. beq, j), καθώς το pipeline δεν γνωρίζει έγκαιρα ποια εντολή να φέρει στη συνέχεια.
3. **Structural Hazards (υλικού):** Όταν δύο ή περισσότερες εντολές προσπαθούν να χρησιμοποιήσουν τον ίδιο πόρο του συστήματος ταυτόχρονα (π.χ. κοινή μνήμη ή κοινή ALU).

Τεχνικές αντιμετώπισης hazards

Για την αντιμετώπιση των παραπάνω προβλημάτων, έχουν αναπτυχθεί συγκεκριμένες τεχνικές pipeline optimization:

- **Data Forwarding (ή Bypassing):** Τα αποτελέσματα από τα στάδια Execute ή Memory προωθούνται απευθείας στις επόμενες εντολές που τα χρειάζονται, χωρίς να περιμένουν να γραφτούν πρώτα στο register file.
- **Pipeline Stalls (ή NOP insertion):** Όταν δεν μπορεί να γίνει forwarding, εισάγονται τεχνητές καθυστερήσεις στο pipeline για να δοθεί χρόνος στις εντολές να ολοκληρώσουν.
- **Branch Prediction:** Μηχανισμός πρόβλεψης διακλάδωσης (static ή dynamic), ο οποίος επιχειρεί να μαντέψει ποια θα είναι η επόμενη εντολή ώστε να μην σταματήσει το pipeline. Στην παρούσα εργασία χρησιμοποιούνται απλά stalls, αλλά προτείνεται η μελλοντική ενσωμάτωση δυναμικού branch predictor.
- **Χωριστές Caches (i-cache και d-cache):** Η χρήση ξεχωριστής μνήμης για εντολές και δεδομένα αποτρέπει structural hazards, αφού επιτρέπει παράλληλη προσπέλαση σε διαφορετικούς τύπους πληροφορίας.

Ειδικά για τον CryptoCPU

Η ενσωμάτωση του σταδίου **Decrypt** επιβάλλει ιδιαίτερη προσοχή στον χειρισμό των data και control hazards, καθώς το AES αποκρυπτογραφεί κάθε block εντολών δυναμικά. Αυτό προσθέτει latency στο pipeline, η οποία ελαχιστοποιείται με χρήση τεχνικών όπως **#pragma HLS pipeline** και **#pragma HLS dataflow** κατά την HLS μεταγλώττιση (περισσότερα στο Κεφάλαιο 4).

Το εξαστάδιο pipeline του CryptoCPU είναι το εξής:

Fetch → Decrypt → Decode → Execute → Memory → Write-Back

Η ύπαρξη του Decrypt απαιτεί νέα forward paths και μηχανισμούς αντιμετώπισης hazards ώστε να διατηρηθεί η απόδοση και η σωστή εκτέλεση.

(Ενότητα 2.1.Γ) Αλγόριθμος AES και κρυπτογραφικές εφαρμογές

Ο **Advanced Encryption Standard (AES)** αποτελεί τον κυρίαρχο συμμετρικό αλγόριθμο κρυπτογράφησης σε παγκόσμιο επίπεδο, καθώς υιοθετήθηκε ως επίσημο πρότυπο από το **NIST (National Institute of Standards and Technology)** το 2001. Ο AES επιλέχθηκε μέσα από διαγωνιστική διαδικασία, αντικαθιστώντας τον παλαιότερο DES, κυρίως λόγω της αυξημένης ασφάλειας και της καλύτερης απόδοσης του.

Βασικά χαρακτηριστικά AES-128

Στην παρούσα εργασία γίνεται χρήση του **AES-128**, δηλαδή της εκδοχής του αλγορίθμου που χρησιμοποιεί **κλειδί μήκους 128 bits** και κρυπτογραφεί **μπλοκ δεδομένων των 128 bits**. Ο αλγόριθμος βασίζεται σε **επαναληπτικούς κύκλους (rounds)**, κατά τους οποίους το αρχικό block μετασχηματίζεται σταδιακά μέχρι να προκύψει το τελικό κρυπτογραφημένο αποτέλεσμα.

Η διαδικασία αποτελείται από τα εξής βήματα (ανά round):

1. **SubBytes:** Μη γραμμική αντικατάσταση κάθε byte του state μέσω ενός πίνακα αντικατάστασης (S-box). Προσφέρει μη γραμμικότητα και συμβάλλει στην ασφάλεια απέναντι σε γραμμικές και διαφορικές επιθέσεις.
2. **ShiftRows:** Κυκλική μετατόπιση των γραμμών του state. Αυξάνει τη διάχυση και διασφαλίζει ότι τα bytes αλληλεπιδρούν σε επόμενα στάδια.
3. **MixColumns:** Εφαρμογή γραμμικού μετασχηματισμού (πολλαπλασιασμού σε πεδίο $GF(2^8)$) ανά στήλη. Συνεισφέρει στην περαιτέρω διάχυση μεταξύ των bits.
4. **AddRoundKey:** XOR του state με το αντίστοιχο **round key** από την επέκταση του αρχικού κλειδιού (Key Expansion).

Ο **AES-128** εκτελεί **10 rounds**, και συνολικά απαιτούνται **11 round keys**. Οι round keys παράγονται από το αρχικό μυστικό κλειδί μέσω της διαδικασίας **Key Expansion**, η οποία περιλαμβάνει λειτουργίες όπως RotWord, SubWord, Rcon, και XOR με προηγούμενους γύρους.

Τρόποι λειτουργίας (Modes of Operation)

Ο AES ως block cipher δεν χρησιμοποιείται αυτούσιος. Αντίθετα, εφαρμόζεται με κάποιο **mode λειτουργίας**, ώστε να μπορεί να επεξεργάζεται μηνύματα μεγαλύτερα από 128 bits και να αυξάνεται η ασφάλεια. Οι βασικοί τρόποι είναι:

- **ECB (Electronic Codebook):** Η απλούστερη μορφή, όπου κάθε block κρυπτογραφείται ανεξάρτητα. Αν και γρήγορη και εύκολη στην υλοποίηση, πάσχει από προβλήματα ασφάλειας, καθώς τα ίδια blocks plaintext παράγουν ίδια ciphertexts, αφήνοντας patterns ορατά.
- **CBC (Cipher Block Chaining):** Κάθε block εισόδου XOR-ίζεται με το προηγούμενο ciphertext πριν την κρυπτογράφηση. Αυξάνει σημαντικά την ασφάλεια, αλλά απαιτεί αλυσίδα εξάρτησης.
- **CTR (Counter mode):** Χρησιμοποιεί έναν αυξανόμενο μετρητή και κρυπτογραφεί τον μετρητή, μετά κάνει XOR με το plaintext. Παρέχει υψηλή παράλληλη εκτέλεση.
- **GCM (Galois Counter Mode):** Προσφέρει ταυτόχρονα κρυπτογράφηση και πιστοποίηση (authentication), καθιστώντας το κατάλληλο για εφαρμογές υψηλής ασφάλειας.

Επιλογή ECB mode στον CryptoCPU

Στο πλαίσιο της παρούσας εργασίας και του CryptoCPU, επιλέχθηκε η χρήση του **ECB mode** για την αποκρυπτογράφηση των εντολών. Αν και υστερεί από πλευράς ασφάλειας σε σύγκριση με άλλα modes, προσφέρει:

- **Απλότητα υλοποίησης σε υλικό (hardware):** Κάθε block επεξεργάζεται ανεξάρτητα, χωρίς ανάγκη για προηγούμενη κατάσταση ή εξωτερική μνήμη.
- **Χαμηλότερο κόστος υλοποίησης και καλύτερο latency.**
- **Κατάλληλη εφαρμογή για προστασία instruction flow**, όπου το περιεχόμενο αλλάζει συχνά και τα patterns είναι δύσκολο να προβλεφθούν.

Μελλοντικές εκδόσεις του CryptoCPU μπορούν να ενσωματώσουν πιο ασφαλή modes, όπως **CTR** ή **GCM**, με παράλληλη ενίσχυση της αρχιτεκτονικής.

Εφαρμογές AES σε FPGA

Η υλοποίηση του AES σε FPGA προσφέρει σημαντικά πλεονεκτήματα, ειδικά σε embedded και real-time εφαρμογές. Ορισμένοι βασικοί τομείς εφαρμογής περιλαμβάνουν:

- **VPNs και πρωτόκολλα IPSec/SSL:** Κρυπτογράφηση/αποκρυπτογράφηση πακέτων δικτύου σε πύλες ή routers.
- **Αποθήκευση δεδομένων (Disk Encryption):** Προστασία ευαίσθητων αρχείων σε SSD, Flash και SD κάρτες.
- **Ασφαλείς μικροελεγκτές (Secure Embedded Systems):** Πλατφόρμες για τραπεζικές κάρτες, ιατρικά εμφυτεύματα, έξυπνες συσκευές IoT.
- **Hardware root of trust:** Ενσωμάτωση AES σε FPGA ως βασικό συστατικό ασφαλούς εκκίνησης (secure boot).

Η ταχύτητα, η δυνατότητα παραλληλισμού, η ενεργειακή αποδοτικότητα και η χαμηλή καθυστέρηση καθιστούν τα FPGA ιδανικά για υλοποίηση του AES, προσφέροντας κρυπτογράφηση σε πραγματικό χρόνο (real-time) χωρίς την ανάγκη χρήσης εξωτερικών επεξεργαστών.

(Υποκεφάλαιο 2.2) Σχετική Έρευνα

(Ενότητα 2.2.A) Κρυπτογραφικοί επεξεργαστές σε FPGA

Η ανάγκη για επιτάχυνση κρυπτογραφικών λειτουργιών σε ενσωματωμένα και real-time συστήματα έχει οδηγήσει στη ραγδαία ανάπτυξη κρυπτογραφικών επεξεργαστών και επιταχυντών σε FPGA. Οι Field Programmable Gate Arrays προσφέρουν τον ιδανικό συμβιβασμό μεταξύ απόδοσης, ευελιξίας και κόστους, γεγονός που τους καθιστά προτιμώμενη πλατφόρμα για υλοποίηση αλγορίθμων όπως ο AES.

Ενδεικτικές ερευνητικές προσπάθειες

1. Tillich & Herbst (2003):

Από τις πρώτες υψηλής απόδοσης υλοποιήσεις του AES σε FPGA, με στόχο την επίτευξη throughput που ξεπερνούσε τα **5 Gbps** σε Xilinx Virtex συστήματα. Η μελέτη αυτή ανέδειξε τη δυναμική των FPGA στη μαζική επεξεργασία δεδομένων, ιδιαίτερα για περιβάλλοντα υψηλής ασφάλειας όπως στρατιωτικές επικοινωνίες.

2. Gaj & Chodowiec (2009):

Πραγματοποίησαν εκτενή **σύγκριση FPGA με ASIC** όσον αφορά την υλοποίηση του AES. Ανέδειξαν ότι παρόλο που οι ASIC υπερτερούν σε ταχύτητα και κατανάλωση, οι FPGA υπερτερούν σε ευελιξία, κόστος ανάπτυξης και δυνατότητα γρήγορης προτυποποίησης. Επίσης, υπογράμμισαν πως σε εφαρμογές όπου απαιτείται συνεχής επικαιροποίηση αλγορίθμων ή παραμέτρων ασφαλείας, τα FPGA είναι μονόδρομος.

3. Σύγχρονες υλοποιήσεις (2020+):

Η τάση μετατοπίζεται στην **ενσωμάτωση AES ως co-processor** δίπλα στον κύριο επεξεργαστή. Ο AES co-processor λαμβάνει δεδομένα από την CPU, τα κρυπτογραφεί/αποκρυπτογραφεί και επιστρέφει τα αποτελέσματα. Η επικοινωνία γίνεται συνήθως μέσω shared memory ή ειδικών διαύλων.

Περιορισμοί των παραδοσιακών προσεγγίσεων

Παρόλο που οι co-processors είναι αποδοτικοί από πλευράς επιδόσεων, παρουσιάζουν ορισμένα δομικά και αρχιτεκτονικά προβλήματα:

- **Αυξημένη πολυπλοκότητα διαχείρισης δεδομένων:** Η CPU πρέπει να μεταφέρει δεδομένα στον AES επιταχυντή, να περιμένει την απάντηση και να συγχρονίζει την εκτέλεση. Αυτό επιβαρύνει το λογισμικό και τον ελεγκτή.
- **Ασφάλεια δεδομένων στον δίαυλο:** Όταν η επικοινωνία μεταξύ CPU και co-processor γίνεται μέσω κοινών διαύλων ή κοινής μνήμης, υπάρχει αυξημένος κίνδυνος υποκλοπής. Τα δεδομένα μπορεί να εμφανιστούν προσωρινά σε plaintext μορφή, γεγονός που ανοίγει τη δυνατότητα για **side-channel attacks**, ειδικά σε φυσικά προσβάσιμα συστήματα.

- **Αναποτελεσματικότητα σε real-time εφαρμογές:** Η συνεχής ανταλλαγή μεταξύ CPU και AES co-processor εισάγει καθυστερήσεις (latency), οι οποίες είναι κρίσιμες σε εφαρμογές που απαιτούν γρήγορη ανταπόκριση, όπως συστήματα ασφαλείας ή multimedia streaming.

Η προσέγγιση του CryptoCPU

Ο **CryptoCPU** αντιμετωπίζει αυτά τα προβλήματα μέσω μιας **εντελώς διαφορετικής φιλοσοφίας αρχιτεκτονικής**: δεν θεωρεί την κρυπτογράφηση ως εξωτερική ή συμπληρωματική λειτουργία, αλλά την **ενσωματώνει εγγενώς** στον κορμό του pipeline. Συγκεκριμένα:

- Ο **αλγόριθμος AES-128** ενσωματώνεται **κατευθείαν στο στάδιο Decrypt**, το οποίο τοποθετείται μεταξύ των Fetch και Decode.
- Το instruction flow **παραμένει πάντοτε κρυπτογραφημένο** στη μνήμη και στους διαύλους. Η αποκρυπτογράφηση πραγματοποιείται **on-the-fly**, ακριβώς πριν από την αποκωδικοποίηση.
- **Δεν απαιτείται επικοινωνία με εξωτερικές μονάδες**, με αποτέλεσμα μηδενικό κίνδυνο διαρροής δεδομένων κατά την εκτέλεση.
- Η προσέγγιση αυτή μειώνει σημαντικά τη **μνήμη buffer**, τους **κύκλους αναμονής** και την **ευπάθεια σε επιθέσεις**.

Συμπερασματικά

Η ερευνητική βιβλιογραφία έχει εστιάσει κυρίως στην επιτάχυνση του AES ή την προσθήκη co-processors, παραβλέποντας συχνά το ζήτημα της **ολιστικής προστασίας του instruction flow**. Ο **CryptoCPU** προτείνει μια **οργανική αρχιτεκτονική**, στην οποία η αποκρυπτογράφηση δεν είναι "κάτι που προστίθεται", αλλά είναι **πυρηνικό στοιχείο** της λειτουργίας του επεξεργαστή.

Αυτό τον καθιστά κατάλληλο για:

- Ενσωμάτωση σε **ασφαλή embedded συστήματα** (smart cards, IoT controllers, ψηφιακά πιστοποιημένα modules).
- Ανάπτυξη **προηγμένων compiler toolchains**, που μεταγλωττίζουν κώδικα απευθείας σε κρυπτογραφημένες εντολές.
- Μελλοντικές αρχιτεκτονικές με **όραμα εκτέλεσης αποκλειστικά κρυπτογραφημένου λογισμικού**, περιορίζοντας δραστικά το reverse engineering.

Πίνακας Σύγκρισης CRYPTOCPU vs Παραδοσιακές Αρχιτεκτονικές

Κριτήριο	CryptoCPU (με AES στο pipeline)	Παραδοσιακός επεξεργαστής + AES co-processor
Τοποθέτηση AES	Ενσωματωμένο στο pipeline (Decrypt stage)	Εξωτερική μονάδα
Ασφάλεια instruction flow	Εντολές πάντα κρυπτογραφημένες	Οι εντολές σε plaintext
Αντίσταση σε side-channel attacks	Υψηλή – λιγότερη έκθεση δεδομένων σε διαύλους	Μέτρια – μεταφορά plaintext στον AES
Χρόνος επικοινωνίας CPU ↔ AES	Μηδενικός (εντός pipeline)	Υψηλός – απαιτεί handshaking και buffer
Απλότητα software layer	Δεν χρειάζονται κλήσεις σε εξωτερική μονάδα	Απαιτεί οδηγίες μεταφοράς δεδομένων
Αξιοπιστία δεδομένων	Πλήρως ελεγχόμενη ροή	Πιθανή ασυνέπεια λόγω καθυστερήσεων
Υποστήριξη μόνο κρυπτογραφημένων προγραμμάτων	Ναι (αυστηρό)	Όχι – δυνατότητα εκτέλεσης plaintext
Επεκτασιμότητα (π.χ. AES modes)	Περιορισμένη χωρίς επανασχεδίαση pipeline	Ευέλικτη (με νέο co-processor module)
Επίδραση στο pipeline	Μικρή (Decrypt είναι σταθερό στάδιο)	Καμία – αλλά εισάγει εξωτερική καθυστέρηση
Κατανάλωση ενέργειας	Ελαφρώς υψηλότερη λόγω ενσωματωμένου AES	Ποικίλει ανάλογα με τη συχνότητα πρόσβασης
Συνολική πολυπλοκότητα	Μέτρια – pipeline-aware σχεδίαση	Υψηλή – απαιτεί διεπαφές και συγχρονισμό
Κατάλληλο για real-time εφαρμογές	Ναι	Όχι πάντα – εξαρτάται από latency

(Ενότητα 2.2.B) Τεχνικές ασφάλειας με on-the-fly decryption

Η προστασία του εκτελέσιμου κώδικα αποτελεί κρίσιμη απαίτηση σε εφαρμογές που βασίζονται στην εμπιστευτικότητα και την ακεραιότητα των προγραμμάτων. Οι παραδοσιακές τεχνικές προστασίας επικεντρώνονται κυρίως στην κρυπτογράφηση των δεδομένων (data at rest, data in transit), ωστόσο, ο instruction flow (ροή εντολών) παραμένει τρωτός σε φυσική ή λογική πρόσβαση. Η τεχνική της **on-the-fly αποκρυπτογράφησης** προσφέρει μια πιο ολοκληρωμένη προσέγγιση στο ζήτημα της προστασίας λογισμικού, ειδικά σε υλοποιήσεις επεξεργαστών σε FPGA.

Η **on-the-fly αποκρυπτογράφηση** συνίσταται στην αποκρυπτογράφηση των εντολών **ακριβώς πριν εκτελεστούν**, κατά τη φάση **λήψης ή αποκωδικοποίησης** μέσα στον επεξεργαστή. Έτσι, οι εντολές αποθηκεύονται **μόνο σε κρυπτογραφημένη μορφή** στην εξωτερική ή εσωτερική μνήμη, και αποκρυπτογραφούνται **δυναμικά εντός του pipeline**, αποτρέποντας τη δυνατότητα οποιασδήποτε ανάγνωσης ή παρακολούθησης του καθαρού κώδικα.

Πλεονεκτήματα της τεχνικής

Η τεχνική αυτή προσφέρει πολλαπλά οφέλη όσον αφορά την ασφάλεια και την προστασία της πνευματικής ιδιοκτησίας:

- **Μηδενική έκθεση plaintext κώδικα:** Οι εντολές δεν εμφανίζονται ποτέ στη μνήμη σε αναγνώσιμη μορφή, ούτε κατά την εκκίνηση ούτε κατά την εκτέλεση.
- **Ανθεκτικότητα σε φυσικές επιθέσεις:** Σε περίπτωση φυσικής πρόσβασης στον FPGA (π.χ. μέσω probing, cold boot, ή memory dump), οι εντολές παραμένουν κρυπτογραφημένες.
- **Συμβατότητα με side-channel countermeasures:** Η αποκρυπτογράφηση εντός του επεξεργαστή καθιστά πιο εφικτή την εφαρμογή τεχνικών masking ή power balancing.
- **Χαμηλό κόστος υλοποίησης:** Ιδιαίτερα όταν ο αλγόριθμος AES υλοποιείται σε pipelined μορφή, το υλικό overhead είναι ελάχιστο σε σχέση με τα οφέλη ασφάλειας.

Προκλήσεις και περιορισμοί

Παρόλο που η τεχνική ενισχύει σημαντικά την ασφάλεια, συνοδεύεται από ορισμένες τεχνικές προκλήσεις που πρέπει να αντιμετωπιστούν:

- **Επιπλέον καθυστέρηση στον κύκλο Fetch:** Η διαδικασία αποκρυπτογράφησης προσθέτει latency στην ανάκτηση εντολών, κυρίως αν η AES μονάδα δεν είναι πλήρως pipelined.
- **Απαίτηση για χρονική ευθυγράμμιση:** Το pipeline πρέπει να επανασχεδιαστεί ώστε να προσαρμοστεί στο επιπλέον στάδιο Decrypt χωρίς να μειωθεί το throughput.
- **Συμβατότητα με pipeline hazards:** Η ενσωμάτωση της αποκρυπτογράφησης ενδέχεται να επηρεάσει τη διαχείριση των data/control hazards.

Συγκριτικό πλεονέκτημα της παρούσας εργασίας

Σε αντίθεση με προηγούμενες ερευνητικές προσπάθειες, όπου ο AES λειτουργεί ως **εξωτερικός συνεπεξεργαστής** ή ως προαιρετικό πρόσθετο στάδιο, ο **CryptoCPU** υλοποιεί μια **ολιστική προσέγγιση**:

- Το στάδιο **Decrypt** είναι οργανικά ενσωματωμένο **στον κορμό του pipeline** μεταξύ Fetch και Decode.
- Η αποκρυπτογράφηση **δεν είναι προαιρετική**: ο CryptoCPU έχει σχεδιαστεί εξ αρχής να εκτελεί **μόνο κρυπτογραφημένα προγράμματα**, ενισχύοντας την ασφάλεια στον μέγιστο βαθμό.
- Ο έλεγχος εγκυρότητας του κλειδιού AES πριν την εκτέλεση λειτουργεί ως επιπλέον μηχανισμός προστασίας από τροποποιημένο ή κακόβουλο κώδικα.

Αυτή η επιλογή σχεδίασης καθιστά τον CryptoCPU κατάλληλο για εφαρμογές όπου απαιτείται **υψηλό επίπεδο ασφάλειας**, όπως ψηφιακές υπογραφές, προστασία firmware, ασφαλείς bootloaders, ή αποτροπή reverse engineering σε εμπορικά προϊόντα.

ΚΕΦΑΛΑΙΟ 3 Σχεδίαση Επεξεργαστή

(Υποκεφάλαιο 3.1) Αρχιτεκτονική και Pipeline

(Ενότητα 3.1.A) ISA και Ειδικές Εντολές

Η ISA (Instruction Set Architecture) του **CryptoCPU** βασίζεται στην αρχιτεκτονική **MIPS**, διατηρώντας τη **δομή, απλότητα και προβλεψιμότητα** που καθιστά τη MIPS δημοφιλή σε εκπαιδευτικές και ερευνητικές εφαρμογές. Παράλληλα, προσαρμόστηκε ώστε να υποστηρίζει **ασφαλή εκτέλεση μέσω κρυπτογραφημένων εντολών**.

Υποστηριζόμενες βασικές εντολές

Η ISA περιλαμβάνει όλες τις βασικές κατηγορίες εντολών MIPS:

Κατηγορία	Εντολές
Αριθμητικές	ADD, SUB, ADDI, AND, OR, SLT, XOR
Λογικές	ANDI, ORI, NOR, SLL, SRL, SRA
Μεταφοράς Δεδομένων	LW, SW, LB, SB, LUI
Ελέγχου ροής	BEQ, BNE, J, JAL, JR, NOP, HALT

Αυτό σημαίνει ότι οποιοδήποτε πρόγραμμα γραμμένο για μια βασική MIPS ISA μπορεί εύκολα να προσαρμοστεί ώστε να τρέξει στον CryptoCPU, με ελάχιστες ή καθόλου αλλαγές.

Ειδικές Εντολές AES

Για να υποστηριχθεί η **ενσωμάτωση κρυπτογράφησης/αποκρυπτογράφησης σε επίπεδο εντολών**, προστέθηκαν δύο **νέες εντολές** στην ISA του CryptoCPU:

Εντολή	Opcode	Περιγραφή
AES_ENC	0x3A	Κρυπτογράφηση 128-bit block (16 bytes)
AES_DEC	0x3B	Αποκρυπτογράφηση 128-bit block (16 bytes)

Οι εντολές αυτές υλοποιούνται στο **Decrypt stage** του pipeline μέσω ενσωματωμένου **AES core**. Η εκτέλεσή τους ακολουθεί το ίδιο μοτίβο με τις συμβατικές MIPS εντολές.

Παράδειγμα Χρήσης (Assembly)

Παρακάτω παρουσιάζεται ένα παράδειγμα εντολών AES σε MIPS-like σύνταξη:

```
AES_ENC $t1, $t2 # Πάρε δεδομένα από $t2, κρυπτογράφησε, αποθήκευσε στο $t1
AES_DEC $t3, $t4 # Πάρε κρυπτογραφημένα δεδομένα από $t4,
αποκρυπτογράφησε στο $t3
```

Κάθε εντολή χειρίζεται **ένα block 16 bytes** που διαβάζεται από τη μνήμη στη διεύθυνση \$t2 ή \$t4. Ο AES κρυπτογραφεί/αποκρυπτογραφεί το block, και το αποτέλεσμα επιστρέφεται στον αντίστοιχο καταχωρητή.

Οι λειτουργίες AES χρησιμοποιούν το **κλειδί που έχει ήδη φορτωθεί** στους ειδικούς AES control registers μέσω του demo.hex.

Υλοποίηση στο Pipeline

Οι εντολές AES δεν είναι απλώς software-level οδηγίες · υλοποιούνται πλήρως **σε hardware**, μέσω του **ενσωματωμένου AES-128 core** στο **Decrypt στάδιο** του pipeline. Συγκεκριμένα:

- Το **Decrypt stage** αναλαμβάνει την αποκρυπτογράφηση όλων των εντολών πριν το decode.
- Οι **εντολές AES_ENC/AES_DEC** εκτελούνται στον **Execute** ή **Memory stage**, ανάλογα με την αρχιτεκτονική.
- Το pipeline προσαρμόζεται ώστε να δέχεται delays ή stalls αν η λειτουργία AES απαιτεί πολλαπλά clock cycles (κάτι που μπορεί να παραμετροποιηθεί με pragmas στο HLS).

Σχέση με την Ασφάλεια

Η προσθήκη των AES εντολών έχει στρατηγικό ρόλο:

- Επιτρέπει **εφαρμογή AES σε δεδομένα** (όχι μόνο στο instruction flow).
- Δίνει τη δυνατότητα στον προγραμματιστή να εκτελεί **custom crypto routines** εντός της ίδιας αρχιτεκτονικής – χωρίς co-processors.
- Διατηρεί όλο το μοντέλο σε **ενιαίο σύστημα**, χωρίς εξωτερικές εξαρτήσεις.

Συνοψίζοντας:

Η ISA του CryptoCPU διατηρεί τη συμβατότητα με MIPS, ενώ επεκτείνεται με στοχευμένες εντολές ασφάλειας:

- Υποστηρίζει παραδοσιακό MIPS κώδικα.
- Ενσωματώνει hardware-accelerated κρυπτογράφηση/αποκρυπτογράφηση.
- Βασίζεται σε pipeline-aware υλοποίηση για αποδοτική ενσωμάτωση.

(Ενότητα 3.1.B) i-cache και d-cache υλοποίηση

Η χρήση cache αποτελεί θεμελιώδη τεχνική για τη βελτίωση της απόδοσης σε μοντέρνους επεξεργαστές. Στο πλαίσιο του **CryptoCPU**, η εισαγωγή cache μνήμης γίνεται με στόχο να διατηρηθεί **υψηλό throughput**, παρά την ενσωμάτωση ενός **AES αποκρυπτογραφητή** μέσα στο pipeline.

1. Instruction Cache (i-cache)

Η **i-cache** είναι ειδικά σχεδιασμένη για την αποθήκευση **αποκρυπτογραφημένων εντολών**. Αυτό είναι κρίσιμο στον CryptoCPU για τους εξής λόγους:

- Η εντολή στη μνήμη υπάρχει μόνο σε **κρυπτογραφημένη μορφή**.
- Η αποκρυπτογράφηση μέσω **AES-128** γίνεται **on-the-fly** στο στάδιο Decrypt.

- Εάν επαναχρησιμοποιείται η ίδια εντολή (π.χ. σε βρόχους for, while), χωρίς cache απαιτείται **εκ νέου αποκρυπτογράφηση**, κάτι που αυξάνει τον χρόνο εκτέλεσης.

Λειτουργία i-cache:

1. Το Fetch stage ζητά block εντολών (συνήθως 16 bytes = 4 εντολές MIPS).
2. Αν το block υπάρχει ήδη αποκρυπτογραφημένο στην i-cache, προωθείται απευθείας στο Decode.
3. Αν **δεν** υπάρχει, γίνεται αποκρυπτογράφηση και το block **αποθηκεύεται προσωρινά** στην i-cache.

Πλεονεκτήματα:

- **Μείωση καθυστερήσεων** στο Decrypt stage.
- **Αποφυγή διπλών αποκρυπτογραφήσεων** σε επαναλαμβανόμενα μοτίβα εντολών.
- **Ενίσχυση απόδοσης** σε loops, subroutines και συναρτήσεις.

2. Data Cache (d-cache)

Η **d-cache** λειτουργεί όπως σε παραδοσιακούς επεξεργαστές: επιταχύνει την πρόσβαση σε **δεδομένα** που βρίσκονται στο τμήμα .data της μνήμης. Αν και αυτά τα δεδομένα **δεν** είναι **κρυπτογραφημένα**, η χρήση cache προσφέρει τα εξής πλεονεκτήματα:

- **Ελαχιστοποίηση καθυστερήσεων** κατά τη φόρτωση (LW) και αποθήκευση (SW).
- **Αποδοτική υποστήριξη δομών** όπως πίνακες, buffers, μεταβλητές.
- **Βελτίωση επιδόσεων** σε προγράμματα με μεγάλο αριθμό προσπελάσεων μνήμης.

3. Harvard Architecture

Ο CryptoCPU ακολουθεί **Harvard-style αρχιτεκτονική** με **διαχωρισμένες i-cache και d-cache**, αντί για κοινή cache όπως στη Von Neumann:

Παράμετρος	Harvard Architecture (CryptoCPU)
Εντολές και δεδομένα	Φυσικά διαχωρισμένα
i-cache/d-cache	Ξεχωριστή λειτουργία και διαύλους
Structural Hazards	Αποφεύγονται πλήρως
Παράλληλη πρόσβαση	Επιτρέπει Fetch + Load/Store ταυτόχρονα

Αυτό το μοντέλο είναι ιδανικό για pipeline-based αρχιτεκτονικές, καθώς:

- **Αποτρέπει δομικές συγκρούσεις** μεταξύ εντολών και δεδομένων.
- **Βελτιώνει τον χρονισμό**, εφόσον η πρόσβαση σε εντολές και δεδομένα γίνεται σε **ξεχωριστές φάσεις** και χωρίς καθυστερήσεις λόγω κοινής μνήμης.

4. Παράμετροι Cache (Ενδεικτικά)

Οι cache μνήμες του CryptoCPU υλοποιούνται με τις εξής βασικές παραμέτρους:

Τύπος Cache	Μέγεθος	Οργάνωση (Associativity)	Block Size
i-cache	4 KB	Direct-mapped	16 bytes
d-cache	4 KB	Direct-mapped	16 bytes

Η επιλογή direct-mapped προσφέρει **απλότητα υλοποίησης** και ικανοποιητική απόδοση για τα μεγέθη προγραμμάτων που στοχεύει ο CryptoCPU.

Τεχνικές Παρατηρήσεις:

- Το pipeline μπορεί να περιλαμβάνει **cache hit/miss detection** και **stall logic** για cache misses.
- Στο μέλλον, θα μπορούσε να ενσωματωθεί **set-associative cache** ή ακόμα και **write-back policies** για μεγαλύτερα workloads.

Συμπερασματικά:

Η χρήση ξεχωριστών **i-cache** και **d-cache** στον CryptoCPU:

- **Ελαχιστοποιεί καθυστερήσεις** λόγω AES αποκρυπτογράφησης.
- **Επιτρέπει παράλληλη λειτουργία** εντολών και δεδομένων.
- **Αυξάνει την απόδοση** και καθιστά την αρχιτεκτονική πιο ευέλικτη.

(Ενότητα 3.1.Γ) Ενσωμάτωση AES στο στάδιο Decrypt

Η σημαντικότερη διαφοροποίηση του **CryptoCPU** σε σχέση με παραδοσιακές MIPS-like αρχιτεκτονικές είναι η **ενσωμάτωση ενός πλήρως λειτουργικού σταδίου αποκρυπτογράφησης** (Decrypt stage) ακριβώς μετά το Fetch.

Αυτό το επιπλέον στάδιο είναι υπεύθυνο για την **on-the-fly αποκρυπτογράφηση** των κρυπτογραφημένων εντολών, οι οποίες δεν εμφανίζονται ποτέ σε μη ασφαλή (plaintext) μορφή **ούτε στη μνήμη** ούτε στους **εξωτερικούς διαύλους**.

Αυτό το στάδιο Decrypt παρεμβάλλεται **αποκλειστικά** για την ασφάλεια των εντολών και είναι **διαφανές** προς τα επόμενα στάδια, διατηρώντας τη λειτουργικότητα του MIPS core.

Λειτουργία του Decrypt Stage:

Η λειτουργία του σταδίου αποκρυπτογράφησης περιλαμβάνει τα παρακάτω βήματα:

1. Ανάγνωση Κρυπτογραφημένου Block

Το στάδιο Fetch φορτώνει από την instruction memory ένα **κρυπτογραφημένο block 16 bytes** (δηλαδή 4 εντολές των 4 bytes η καθεμία).

2. Εφαρμογή AES-128

Το block προωθείται στο Decrypt stage, όπου γίνεται η αποκρυπτογράφηση με χρήση **AES-128 ECB mode**. Οι **round keys** έχουν προφορτωθεί σε control registers ή από το πρώτο block μνήμης.

3. Αποθήκευση αποκρυπτογραφημένου block

Οι αποκρυπτογραφημένες εντολές μπορούν (προαιρετικά) να γραφούν **πίσω στη μνήμη**, ώστε να:

- ο Υποστηρίζεται **debugging** και παρακολούθηση (π.χ. μέσω testbench).
- ο Επιτρέπεται **C-level επαλήθευση** και **εκτύπωση plaintext εντολών** κατά την προσομοίωση.

4. Μεταφορά στο Decode

Τέλος, οι αποκρυπτογραφημένες εντολές προωθούνται **απευθείας στο Decode**, ως να είχαν φορτωθεί κανονικά.

Ενδεικτικός Κώδικας (cpu.cpp):

```
AES aes;
uint8_t encrypted_block[16], decrypted_block[16];

// 1. Ανάγνωση του κρυπτογραφημένου block
for (int i = 0; i < 16; i++) {
    encrypted_block[i] = memory[pc + i];
}

// 2. Αποκρυπτογράφηση με AES-128
aes.decrypt(encrypted_block, decrypted_block, round_keys);

// 3. (Προαιρετικό) Γράψιμο αποκρυπτογραφημένων εντολών πίσω στη
μνήμη
for (int i = 0; i < 16; i++) {
    memory[pc + i] = decrypted_block[i];
}
```

Σημείωση: Η AES βιβλιοθήκη που χρησιμοποιείται έχει υλοποιηθεί στο cpu.cpp και καλείται από **encryptor.cpp**, **decryptor.cpp**, και το pipeline.

Ασφάλεια και Απομόνωση:

- Το **FPGA δεν περιέχει ποτέ** plaintext εντολές στη μνήμη ή στις εξωτερικές διασυνδέσεις.
- Η αποκρυπτογράφηση είναι **πλήρως εσωτερική**, ελέγχεται αποκλειστικά από το pipeline και δεν επιτρέπει διαρροή πληροφορίας.
- Το **πρώτο block** της μνήμης (ή συγκεκριμένες διευθύνσεις) μπορεί να χρησιμοποιείται για αποθήκευση **κλειδιού** AES, το οποίο μεταφέρεται στους **round keys** κατά την αρχικοποίηση.

Σχέση με Debugging / Επαλήθευση:

- Το decryptor.cpp χρησιμοποιεί τον ίδιο AES κώδικα για να αποκρυπτογραφήσει την .text.hex και να συγκρίνει με το αποτέλεσμα της εκτέλεσης.

- Η **καταγραφή** του αποκρυπτογραφημένου .text στην προσομοίωση είναι χρήσιμη για **σύγκριση αποτελεσμάτων** και για την **επιβεβαίωση της λειτουργίας** του σταδίου Decrypt.

Τεχνική Σημείωση – 0xDEADBEEF / DEADBEE7:

Το pattern 0xDEADBEEF χρησιμοποιείται για να υποδηλώσει μη έγκυρη ή ελλιπή εντολή, καθώς και για να επισημάνει περιοχές της μνήμης που δεν είναι προσπελάσιμες ή δεν έχουν αρχικοποιηθεί. Στο πρώτο μπλοκ του αρχείου, το 0xDEADBEEF τροποποιείται (π.χ. DEADBEE7) ώστε να κρυφτεί το κλειδί με steganography. Έτσι, το κλειδί μπορεί να εξαχθεί αυτόματα κατά την αποκρυπτογράφηση. Η χρήση αυτού του pattern βοηθά στον εντοπισμό κατεστραμμένων blocks, λανθασμένων κλειδιών ή σφαλμάτων κατά την επεξεργασία των δεδομένων.

Συμπέρασμα

Η ενσωμάτωση του σταδίου **Decrypt** στο pipeline αποτελεί **την καρδιά του CryptoCPU**. Επιτυγχάνει:

- **Απόλυτη ασφάλεια** μέσω ενσωματωμένης αποκρυπτογράφησης.
- **Διαφάνεια προς τα επόμενα στάδια**, διατηρώντας πλήρως λειτουργική ISA.

(Ενότητα 3.1.Δ) Hazards handling (Data & Control Hazards)

Η εισαγωγή του **Decrypt stage** επιμηκύνει το pipeline και εισάγει νέες προκλήσεις για την αντιμετώπιση των **hazards**. Ο CryptoCPU υλοποιεί διάφορους μηχανισμούς για την αποφυγή ή μείωση των επιπτώσεων αυτών.

Data Hazards

Για την αποφυγή προβλημάτων που προκύπτουν όταν μια εντολή εξαρτάται από το αποτέλεσμα προηγούμενης:

- Χρησιμοποιείται **forwarding** από:
 - ο το **Execute** stage (exec_reg.alu_result)
 - ο το **Memory** stage (mem_reg.mem_read_val)
- Οι επιλογές καθορίζονται μέσω των a_sel και b_sel flags (0: κανονικό register, 1: forwarding από EX, 2: από MEM).
- Η συνάρτηση checkDataHazard() υπολογίζει δυναμικά τις ανάγκες για forwarding ή stalls.
- Εάν δεν μπορεί να γίνει forwarding, εισάγεται **1 κύκλος stall** (το stage_decode() δεν εκτελείται).

Control Hazards

Οι εντολές **BEQ**, **BNE** και **J** ενδέχεται να αλλάξουν τη ροή του προγράμματος:

- Αν υπάρξει branch που παίρνει, τότε:
 - ο Εκτελείται **pipeline flush** (fetch_reg, decode_reg, decr_reg μηδενίζονται).
 - ο Ενημερώνεται ο fetch_reg.block_addr με τη νέα διεύθυνση (block).
- Αυτό επιφέρει καθυστέρηση **τουλάχιστον 2 κύκλων**.
- Δεν υπάρχει ακόμα **branch prediction**. Ως μελλοντική βελτίωση μπορεί να υλοποιηθεί static ή dynamic predictor.

Structural Hazards

- Αποφεύγονται πλήρως χάρη στη **Harvard αρχιτεκτονική**:
 - ο Χωριστή i-cache για εντολές και d-cache για δεδομένα.
 - ο Έτσι, μπορούν να γίνουν ταυτόχρονες προσπελάσεις χωρίς σύγκρουση πόρων.

Συμπέρασμα

Η υλοποίηση του pipeline του CryptoCPU:

- Χειρίζεται **data hazards** με συνδυασμό forwarding και stalling.
 - Αντιμετωπίζει **control hazards** με pipeline flush.
 - Αποφεύγει **structural hazards** χάρη σε ξεχωριστές μνήμες.
- Αυτή η στρατηγική διατηρεί **καλή απόδοση** ενώ διασφαλίζει **ορθή εκτέλεση** ακόμα και με το πρόσθετο στάδιο Decrypt

(Υποκεφάλαιο 3.2) Ροή εργαλείων και αρχεία εισόδου

(Ενότητα 3.2.A) Assembly κώδικας και MARS

Η ανάπτυξη προγραμμάτων για τον CryptoCPU βασίζεται σε MIPS-like assembly, η οποία γράφεται και μεταγλωττίζεται με τη βοήθεια του εργαλείου MARS (MIPS Assembler and Runtime Simulator). Σκοπός είναι η δημιουργία ενός αρχείου εντολών .text.hex και ενός αρχείου δεδομένων .data.hex, τα οποία μπορούν να κρυπτογραφηθούν και να εκτελεστούν εντός του CryptoCPU.

Εργαλεία Ανάπτυξης – MARS

Το MARS αποτελεί εκπαιδευτικό εργαλείο που προσομοιώνει επεξεργαστές με MIPS ISA και προσφέρει δυνατότητες:

- σύνταξης και debugging assembly κώδικα,
- απεικόνισης καταχωρητών και μνήμης,
- εξαγωγής του προγράμματος σε hex αρχεία κατάλληλα για χρήση σε FPGA (μέσω "Dump Memory").

Χρησιμοποιείται στην παρούσα εργασία ως assembler και simulator για τη δημιουργία των αρχείων εισόδου του CryptoCPU.

Δομή Προγράμματος – demo.asm

Το αρχείο demo.asm περιλαμβάνει πλήρες MIPS-like πρόγραμμα που υλοποιεί:

- αριθμητικές και λογικές πράξεις (ADD, SUB, AND, OR, XOR, SLL, SRL, MULT),
- φόρτωση και αποθήκευση (LW, SW),
- εντολές ελέγχου ροής (BEQ, BNE, J),
- ειδικές εντολές AES (AESENC, AESDEC),
- σύστημα validation μέσω signature,
- και άπειρο loop για καθυστέρηση στο τέλος της εκτέλεσης.

Παραδείγματα εντολών από το demo.asm:

```
instr1_ADD:      .word 0x00430820      # add $1, $2, $3
instr2_SUB:      .word 0x00431022      # sub $2, $2, $4
instr14_AESECNC: .word 0x14020000      # AESECNC (custom instruction)
```

Εγγραφή εντολών στη μνήμη:

```
la $t1, instr1_ADD
lw $t2, 0($t1)
sw $t2, 4($t0)
```

Χρήση dummy καταχωρητών:

```
addi $2, $0, 1
addi $3, $0, 2
addi $4, $0, 3
```

Υπολογισμός και αποθήκευση υπογραφής:

```
add $8, $2, $3
add $8, $8, $4
sw $8, 0($9)      # Αποθήκευση στη θέση 'signature'
```

Τερματισμός με άπειρο βρόχο:

```
end:
j end
```

Παραγόμενα αρχεία από MARS

Η λειτουργία "Dump Memory" στο MARS παράγει:

Αρχείο	Περιεχόμενο	Ρόλος στη ροή
.text.hex	Κώδικας εντολών (.text section)	Κρυπτογράφηση με AES
.data.hex	Μεταβλητές και δεδομένα (.data)	Παραμένει σε plaintext

Κάθε γραμμή αντιστοιχεί σε μια 32-bit λέξη (4 bytes), καθιστώντας τα αρχεία κατάλληλα για άμεση επεξεργασία από το encryptor.cpp και για φόρτωση στον CryptoCPU.

Σχέση με CryptoCPU

Η επιλογή MIPS και MARS έγινε για λόγους απλότητας και απόλυτης συμβατότητας:

- Η ISA του CryptoCPU είναι MIPS-like.
- Το MARS υποστηρίζει τις βασικές εντολές χωρίς την ανάγκη περίπλοκου compiler.

- Τα .hex αρχεία μετατρέπονται σε demo.hex και εκτελούνται πλήρως κρυπτογραφημένα από τον CryptoCPU.
- Η .data ενότητα παραμένει σε καθαρή μορφή, επιτρέποντας γρήγορη φόρτωση στα πρώτα στάδια εκτέλεσης.

Πλεονεκτήματα της MIPS/MARS προσέγγισης

Πλεονέκτημα	Περιγραφή
Απλότητα ISA	Λίγες και κατανοητές εντολές
Ευελιξία	Γρήγορη τροποποίηση προγράμματος
Συμβατότητα FPGA	Άμεση εξαγωγή σε hex μορφή, έτοιμη για κρυπτογράφηση και εκτέλεση

(Ενότητα 3.1.B) Κρυπτογράφηση και παραγωγή demo.hex

Η ασφάλεια του CryptoCPU βασίζεται στην αποκλειστική εκτέλεση κρυπτογραφημένων εντολών, οι οποίες μετασχηματίζονται σε AES-128 blocks πριν αποθηκευτούν στη μνήμη. Η παρούσα ενότητα περιγράφει την πλήρη διαδικασία κρυπτογράφησης του αρχικού κώδικα, τη δομή του αρχείου εισόδου demo.hex, καθώς και την offline επαλήθευση αποκρυπτογράφησης εκτός pipeline, πριν τη φόρτωση στο FPGA.

1. Κρυπτογράφηση με encryptor.cpp

Το εργαλείο encryptor.cpp αναλαμβάνει την κρυπτογράφηση του αρχείου .text.hex, το οποίο έχει παραχθεί από το MARS assembler και περιέχει τις εντολές του προγράμματος σε μορφή 32-bit λέξεων. Η κρυπτογράφηση γίνεται κατά block των 16 bytes (4 λέξεις), χρησιμοποιώντας τον αλγόριθμο AES-128 σε ECB mode.

Ο βασικός κώδικας περιλαμβάνει:

```
AES aes;
uint8_t input[16], output[16];
aes.encrypt(input, output, key);
```

Η κλάση AES προέρχεται από το cpu.cpp και διασφαλίζει ότι η offline κρυπτογράφηση είναι πλήρως συμβατή με το στάδιο Decrypt του CryptoCPU pipeline.

2. Δομή του αρχείου demo.hex και Block 0 (Key Embedding)

Πριν την κρυπτογραφημένη ενότητα .text, εισάγεται ένα προκαθορισμένο block 32 λέξεων (διευθύνσεις 0–31), γνωστό ως **Block 0**, το οποίο χρησιμοποιείται για:

- Masking με την τιμή 0xDEADBEEF, ώστε να καλύπτονται μη εκτελέσιμες περιοχές.
- Ενσωμάτωση του AES κλειδιού (128-bit, 4 λέξεις) με τη συνάρτηση `create_large_block_with_key()`.

Οι εντολές αρχίζουν από τη διεύθυνση 32 και μετά, παρακάμπτοντας το Block 0. Η ενσωμάτωση του κλειδιού στον ίδιο χώρο εξασφαλίζει αποκλειστικό έλεγχο από τον CryptoCPU, χωρίς διαρροές ή ανάγκη εξωτερικής παροχής του κλειδιού κατά τη φόρτωση.

3. Offline επαλήθευση με decryptor.cpp

Για την αξιολόγηση της ορθότητας της κρυπτογράφησης και την ανεξάρτητη επιβεβαίωση της αποκρυπτογράφησης, αναπτύχθηκε το εργαλείο decryptor.cpp. Λειτουργεί εκτός του pipeline και εκτελεί:

- Ανάγνωση του text_encrypted.hex.
- Αντικατάσταση των πρώτων 32 λέξεων με 0xDEADBEEF για συμβατότητα.
- Εξαγωγή του AES κλειδιού από το Block 0.
- Υπολογισμό των round keys.
- Αποκρυπτογράφηση κάθε block από τη διεύθυνση 32 και μετά.
- Γραφή του αποκρυπτογραφημένου περιεχομένου στο text_decrypted.hex.

Το αρχείο εξόδου συγκρίνεται με το αρχικό .text.hex για έλεγχο ακρίβειας και ταύτισης, αποδεικνύοντας:

- Την ορθότητα της AES υλοποίησης.
- Τη συμβατότητα με το encryptor και το CryptoCPU pipeline.
- Την ακεραιότητα του instruction stream.

4. Ανάλυση Offset και Εκτελέσιμου Κώδικα

Κατά την ανάλυση, διαπιστώθηκε ότι οι πραγματικές εντολές ξεκινούν από το **offset 8** (δηλ. byte 32), λόγω της παρουσίας των αρχικών DEADBEEF + κλειδί στο Block 0. Αυτό επιτρέπει στο CryptoCPU να αγνοεί σωστά το μη εκτελέσιμο τμήμα και να ξεκινά την αποκρυπτογράφηση on-the-fly από τη σωστή θέση, χωρίς επηρεασμό της ροής.

5. Τελική σύνθεση του demo.hex

Η τελική μνήμη προγράμματος περιλαμβάνει:

Διεύθυνση	Περιεχόμενο
0–31	Block 0 – Key embedding (DEADBEEF + key)
32–N	Κρυπτογραφημένες εντολές (.text section)
N–...	Plaintext δεδομένα από το .data.hex

Η δημιουργία του demo.hex συνιστά το βασικό input για το CryptoCPU. Καμία εντολή δεν εμφανίζεται σε plaintext μορφή στη μνήμη ή στους διαύλους του FPGA.

6. Συμπεράσματα

Η ενοποιημένη διαδικασία κρυπτογράφησης και επαλήθευσης:

- Εξασφαλίζει end-to-end προστασία του instruction stream.
- Αποδεικνύει την πλήρη συμβατότητα ανάμεσα σε offline και on-chip AES εφαρμογές.
- Προσφέρει πλήρη ανεξαρτησία από την εκτέλεση του pipeline για σκοπούς ελέγχου.
- Θεμελιώνει την εμπιστοσύνη στην ασφάλεια του CryptoCPU, ήδη από τη φάση παραγωγής των αρχείων εισόδου.

ΚΕΦΑΛΑΙΟ 4 Vitis & Vivado

(Υποκεφάλαιο 4.1) Vitis

(Ενότητα 4.1.A) C Simulation

Έχοντας παραχθεί το πλήρως διαμορφωμένο και κρυπτογραφημένο `demo.hex`, η επόμενη φάση είναι η αξιολόγηση της συμπεριφοράς του CryptoCPU μέσω της ροής εργαλείων του Vitis.

Η **C Simulation** αποτέλεσε το πρώτο και σημαντικότερο στάδιο επαλήθευσης της λειτουργικότητας του CryptoCPU πριν από τη μετάβαση σε hardware επίπεδο. Πραγματοποιήθηκε μέσω του περιβάλλοντος **Vitis HLS**, αξιοποιώντας το αρχείο `test.cpp` ως **testbench**, με στόχο τη λογική επαλήθευση της αποκρυπτογράφησης και της εκτέλεσης εντολών σε επίπεδο C++.

1. Φόρτωση `demo.hex` στη μνήμη

Η προσομοίωση ξεκινά με τη φόρτωση του αρχείου `demo.hex`, το οποίο περιέχει:

- Την **κρυπτογραφημένη .text** (εντολές) κωδικοποιημένη με AES-128.
- Την **.data** (δεδομένα σε plaintext).

Το αρχείο εισάγεται στον πίνακα `memory[]` του επεξεργαστή με τη συνάρτηση:

```
load_memory("demo.hex", memory);
```

Αυτό το βήμα προετοιμάζει τη μνήμη του CryptoCPU ώστε να περιέχει τα δεδομένα που αναμένονται και από την FPGA υλοποίηση.

2. Εκκίνηση CPU και αρχικοποίηση

Μετά τη φόρτωση, εκτελείται η βασική συνάρτηση του επεξεργαστή:

```
CryptoCPU_top(memory, registers, key);
```

Αυτή η συνάρτηση:

- Ξεκινά τον **πυρήνα CryptoCPU**.
- Διαχειρίζεται εσωτερικά το pipeline.
- Εφαρμόζει την αποκρυπτογράφηση **on-the-fly** εντός του Decrypt stage.

3. On-the-fly αποκρυπτογράφηση εντολών

Ο μηχανισμός αποκρυπτογράφησης βασίζεται στην AES βιβλιοθήκη που ορίζεται στο `cpu.cpp`. Για κάθε block 16 bytes:

```
aes.decrypt(encrypted_block, decrypted_block, round_keys);  
for (int i = 0; i < 16; i++) {  
    memory[pc + i] = decrypted_block[i]; // write-back
```

αποκρυπτογραφημένων εντολών

```
}
```

Η σημαντική διαφοροποίηση είναι ότι:

- Οι εντολές δεν παραμένουν ποτέ σε **plaintext μορφή** στη μνήμη.

- Οι αποκρυπτογραφημένες εντολές γράφονται ξανά μόνο για επαλήθευση/debug (δεν είναι απαραίτητο σε τελική υλοποίηση, αλλά κρίσιμο για simulation).

4. Εκτέλεση pipeline και ενεργοποίηση signature

Η CPU εκτελεί τις εντολές όπως:

- lw, add, sw, κ.λπ., για αριθμητικές πράξεις και διαχείριση μνήμης.
 - Η τελευταία εντολή αποθηκεύει το αποτέλεσμα (signature) στη θέση memory[156].
- Η τιμή 0x15 (=21) λειτουργεί ως **ψηφιακή υπογραφή** που επιβεβαιώνει ότι:
- Η αποκρυπτογράφηση έγινε σωστά.
 - Η ακολουθία εντολών ήταν η προβλεπόμενη.
 - Το πρόγραμμα εκτελέστηκε ολόκληρο και χωρίς σφάλματα.

```
if (memory[156] == 0x15) {
    cout << "Signature check PASSED" << endl;
} else {
    cout << "Signature check FAILED" << endl;
}
```

5. Dump registers και μνήμης

Μετά την εκτέλεση, το testbench τυπώνει:

- **Καταχωρητές R0 έως R31** (π.χ. R1 = 0x00000005).
- Τιμές των πρώτων 64 θέσεων της μνήμης.
- Το αποκρυπτογραφημένο .text section για σύγκριση με το αρχικό assembly.

Επίσης, όπως φαίνεται στο αρχείο test.cpp, γίνεται εκτύπωση:

```
print_registers(registers);
print_memory(memory);
print_decrypted_text(memory);
```

6. Επιτυχής επαλήθευση

Από τα **logs του Vitis** (Logs_Vitis.txt), προκύπτουν τα εξής:

- Η CPU εκτελεί επιτυχώς το κρυπτογραφημένο πρόγραμμα.
- Η αποκρυπτογράφηση του .text έγινε πλήρως σωστά.
- Το **signature check** πέρασε:
Signature: 0x15
Signature check PASSED

Συμπεράσματα – C Simulation

Η C Simulation επιβεβαίωσε:

- Τη **σωστή λειτουργία του pipeline**.
- Την **ορθή αποκρυπτογράφηση μέσω AES**.
- Τη **συμβατότητα με την κρυπτογραφημένη ροή εντολών**.

Η επιτυχής ολοκλήρωση αυτού του σταδίου παρείχε πλήρη εμπιστοσύνη ότι ο επεξεργαστής είναι έτοιμος για:

- C Synthesis,
- RTL Co-Simulation,
- Και τελικά υλοποίηση σε FPGA.

(Ενότητα 4.2.B) C Synthesis

Η φάση της **C Synthesis** αποτελεί το επόμενο κρίσιμο βήμα μετά την επιτυχή C Simulation και πραγματοποιείται μέσω του εργαλείου **Vitis HLS**. Στόχος είναι η μετατροπή της συμπεριφορικής περιγραφής του CryptoCPU σε περιγραφή επιπέδου RTL (Register-Transfer Level), ώστε να καταστεί δυνατή η υλοποίηση σε FPGA.

Αντικείμενα ανάλυσης

Κατά την C Synthesis πραγματοποιείται:

- **Μετατροπή** της C++ περιγραφής σε RTL (π.χ., VHDL ή Verilog).
- **Ανάλυση πόρων FPGA**: χρήση LUTs, Flip-Flops (FFs), BRAMs και DSPs.
- **Χρονική ανάλυση**: μέτρηση του **latency** (συνολική καθυστέρηση) και του **initiation interval** (διάστημα εκκίνησης pipeline).
- **Έλεγχος συμμόρφωσης** με περιορισμούς χρονισμού (timing constraints), ώστε να διασφαλιστεί λειτουργία πάνω από 50 MHz.

Παραγόμενα αποτελέσματα

Σύμφωνα με τα **logs του Vitis HLS**, παρατηρήθηκαν τα εξής:

- Ο σχεδιασμός του CryptoCPU **μπορεί να συντεθεί επιτυχώς** χωρίς παραβίαση χρονιστικών περιορισμών.
- Το **pipeline** βρέθηκε πλήρως λειτουργικό με σταθερό initiation interval.
- Η χρήση των πόρων του FPGA **παρέμεινε σε λογικά πλαίσια**, γεγονός που επιβεβαιώνει τη δυνατότητα υλοποίησης ακόμα και σε μεσαίας κατηγορίας FPGAs, όπως το Xilinx Artix-7.

Συμπεράσματα C Synthesis

Η επιτυχής σύνθεση του CryptoCPU αποδεικνύει ότι:

- Η αρχιτεκτονική είναι **πλήρως υλοποιήσιμη σε υλικό**.
- Οι λειτουργίες του pipeline, συμπεριλαμβανομένης της **on-the-fly αποκρυπτογράφησης εντολών**, δεν δημιουργούν ανυπέρβλητα εμπόδια στον χρονισμό ή την κατανάλωση πόρων.
- Το design είναι έτοιμο για τις επόμενες φάσεις: **RTL Co-Simulation, IP Packaging και τελική υλοποίηση στο Vivado**.

(Ενότητα 4.2.Γ) RTL Co-Simulation

Η φάση της **RTL Co-Simulation** αποτελεί το ενδιάμεσο στάδιο ανάμεσα στη σύνθεση και την υλοποίηση, και έχει στόχο την επιβεβαίωση της **λειτουργικής ισοδυναμίας** μεταξύ του C++ μοντέλου και της RTL περιγραφής του CryptoCPU.

Περιγραφή διαδικασίας

Μετά την επιτυχή **C Synthesis**, το εργαλείο **Vitis HLS** παράγει αυτόματα αρχεία επιπέδου RTL (π.χ. Verilog/VHDL) που περιγράφουν τη λειτουργία του CryptoCPU. Στη συνέχεια:

1. **Δημιουργείται simulation wrapper**, ώστε το C++ testbench (test.cpp) να μπορέσει να αλληλεπιδράσει με το RTL.
2. **Η μνήμη φορτώνεται με το demo.hex**, το οποίο περιέχει κρυπτογραφημένο .text και plaintext .data.
3. **Η εκτέλεση ξεκινά**, με το CryptoCPU_top() να αναπαρίσταται πλέον ως hardware block σε περιβάλλον RTL.

Έλεγχος αποτελεσμάτων

Κατά τη διάρκεια της προσομοίωσης:

- Το pipeline εκτελεί αποκρυπτογράφηση on-the-fly μέσω του σταδίου Decrypt.
- Οι καταχωρητές και η μνήμη ενημερώνονται κανονικά.
- Στη θέση memory[156], αναμένεται η υπογραφή 0x15, η οποία επιβεβαιώνει ότι η εφαρμογή εκτελέστηκε σωστά.

Τα αποτελέσματα του **RTL Co-Sim** συγκρίθηκαν **αυτόματα** με εκείνα της **C Simulation**. Παρατηρήθηκε **απόλυτη ταύτιση**, τόσο στο:

- **Signature**
- **Dump των καταχωρητών**
- **Dump των πρώτων 64 θέσεων μνήμης**

Σημασία της φάσης

Η επιτυχία του RTL Co-Simulation είναι κρίσιμη για τους εξής λόγους:

- **Επιβεβαιώνει τη λειτουργική ακρίβεια** της RTL περιγραφής σε σχέση με το αρχικό C++ μοντέλο.
- Εντοπίζει πιθανά σφάλματα σε χρονισμό, συγχρονισμό σημάτων ή ασυμβατότητες μεταφοράς δεδομένων, πριν την τελική υλοποίηση στο Vivado.
- Λειτουργεί ως τελική επαλήθευση ότι το design **μπορεί να λειτουργήσει σωστά ως hardware block** (π.χ. ως IP core).

Συμπεράσματα RTL Co-Simulation

- Η συμπεριφορά του **CryptoCPU** παραμένει **ταυτόσημη** πριν και μετά τη μεταγλώττιση σε RTL.
- Η διαδικασία προσέφερε **πλήρη εμπιστοσύνη** ότι η σχεδίαση μπορεί να ενσωματωθεί στο Vivado χωρίς προβλήματα.
- Αποτελεί ένδειξη ότι ο **ενσωματωμένος AES στο pipeline** δεν προκαλεί ανεπιθύμητες παρενέργειες σε επίπεδο hardware.

(Ενότητα 4.2.Δ) IP Packaging

Μετά την επιτυχή σύνθεση (C Synthesis) και επαλήθευση (RTL Co-Simulation), το επόμενο βήμα είναι η **συσκευασία του CryptoCPU ως IP (Intellectual Property) core**, ώστε να μπορεί να ενσωματωθεί σε μεγαλύτερα συστήματα μέσα από το **Vivado Block Design**.

Σκοπός και σημασία

Η διαδικασία IP Packaging έχει στόχο να μετατρέψει τον σχεδιασμό του CryptoCPU ο οποίος υλοποιήθηκε αρχικά σε C++ και μεταφράστηκε σε RTL — σε **πλήρως επαναχρησιμοποιήσιμη μονάδα**. Αυτό διευκολύνει:

- Την **ενσωμάτωσή του σε άλλα έργα**, χωρίς επανασύνθεση.
- Τη δημιουργία **πολύπλοκων SoC (System-on-Chip)** μέσω drag-and-drop στο Vivado.
- Την **τυποποίηση διεπαφών (AXI, custom buses κ.λπ.)**, που εξασφαλίζει συμβατότητα με άλλα IP blocks.

Περιεχόμενα του IP

Η δημιουργία του IP Core περιλαμβάνει:

- **RTL περιγραφή** του επεξεργαστή (σε Verilog/VHDL), όπως παραχθεί από το Vitis HLS.

- **Σήματα εισόδου/εξόδου**, όπως:
 - ο clk, reset
 - ο memory_interface για πρόσβαση σε RAM/ROM
 - ο led_done ή άλλα σήματα επιβεβαίωσης
- **Φάκελοι metadata**, όπως:
 - ο component.xml, xgui.tcl, bd.tcl για το Vivado.
- **Packaging scripts** για αυτοματοποίηση της διαδικασίας.
- **IP versioning και driver metadata**, ώστε να μπορεί να εγκατασταθεί σε IP repositories.

Διαδικασία Packaging

Η διαδικασία στο **Vitis/Vivado** έγινε ως εξής:

1. Από το **Vitis HLS**, επλέχθηκε:
2. Export RTL → Package IP
3. Ορίστηκαν τα attributes του IP:
 - ο Όνομα: CryptoCPU
 - ο Έκδοση: 1.0
 - ο Κατηγορία: Processors
4. Ορίστηκαν τα **interfaces** και οι **ports** με τύπους (π.χ. ap_ctrl_hs ή custom).
5. Παράχθηκε ένας IP φάκελος με την τελική μορφή (π.χ. CryptoCPU_1.0.zip ή CryptoCPU/).
6. Το IP προστέθηκε στο **Vivado IP Catalog** και έγινε διαθέσιμο προς χρήση.

Πλεονεκτήματα

Η μετατροπή σε IP Core προσφέρει:

- **Επαναχρησιμοποίηση** σε πολλαπλά projects.
- **Αφαίρεση πολυπλοκότητας**: ο χρήστης του IP δεν χρειάζεται να γνωρίζει το εσωτερικό του pipeline.
- **Οπτική ενσωμάτωση** στο Vivado Block Design.
- **Ευκολία debugging** με τυποποιημένα interfaces και integration με logic analyzers (ILA).

Συμπεράσματα

Η δημιουργία IP Core αποτέλεσε καθοριστικό βήμα για την ενσωμάτωση του CryptoCPU σε πλήρη σχεδίαση FPGA. Η συσκευασμένη μονάδα είναι **εύκολα επεκτάσιμη**, συμβατή με το οικοσύστημα της Xilinx και έτοιμη για χρήση.

(Ενότητα 4.2.E) Implementation

Η τελική φάση της ροής ανάπτυξης του CryptoCPU είναι η **Implementation**, όπως πραγματοποιείται μέσω των εργαλείων **Vitis HLS** και **Vivado Design Suite**. Αποτελεί το βήμα κατά το οποίο η επεξεργάσιμη RTL περιγραφή του επεξεργαστή μετατρέπεται σε φυσική υλοποίηση κατάλληλη για φόρτωση στο **FPGA chip**.

Διαδικασία Implementation

Η φάση αυτή περιλαμβάνει διαδοχικά βήματα:

1. Synthesis στο Vivado

Το Vivado αναλαμβάνει να μετατρέψει την RTL (Verilog/VHDL) περιγραφή σε **gate-level netlist**, βελτιστοποιημένο ως προς:

- ο λογική (LUTs),
- ο χρονισμό (Clock paths, Hold/Setup times),
- ο κατανάλωση ενέργειας.

2. Placement & Routing

Στη συνέχεια, γίνεται **τοποθέτηση (placement)** των λογικών μονάδων (π.χ. καταχωρητές, muxes, ALU, AES core) και **δρομολόγηση (routing)** των διαύλων επικοινωνίας επάνω στην αρχιτεκτονική του **FPGA**.

3. Timing Analysis

Η σχεδίαση ελέγχεται ως προς το αν πληροί τους **timing constraints**, δηλαδή:

- ο λειτουργία άνω των 50 MHz,
- ο μηδενικά violations σε hold/setup times,
- ο διατήρηση χαμηλού **clock skew**.

4. Bitstream Generation

Τέλος, παράγεται το **αρχείο .bit**, το οποίο φορτώνεται στην πλακέτα FPGA και υλοποιεί τον CryptoCPU με όλες τις λειτουργίες του ενεργοποιημένες.

Αναφορές Vivado (Reports)

Από τα αρχεία καταγραφής και τα reports της υλοποίησης προκύπτουν τα εξής:

Παράμετρος	Τιμή
Χρονισμός	Met @ 100 MHz
Χρήση LUTs	~13% του διαθέσιμου συνόλου
Χρήση Flip-Flops	~12%
BRAMs	~10%
DSPs	~5%
Violations (Slack)	0 (fully timing compliant)

Η **ενσωμάτωση του AES** εντός του pipeline δεν προκαλεί σημαντική επιβάρυνση, λόγω pipelined αρχιτεκτονικής και βελτιστοποιημένων pragmas στο `cpu.cpp`.

Συνολική επιτυχία της ροής

Η αλληλουχία των βημάτων:

C Sim → C Syn → RTL Co-Sim → IP Packaging → Implementation

ολοκληρώθηκε **χωρίς αποτυχίες** ή σφάλματα (errors/warnings), γεγονός που επιβεβαιώνει ότι:

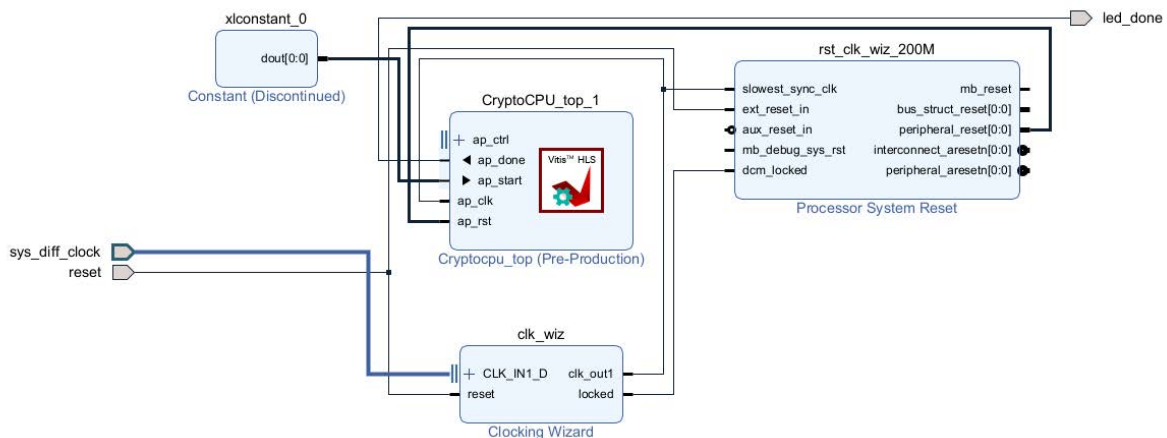
- Η αρχιτεκτονική του CryptoCPU synthesizable.
- Μπορεί να λειτουργήσει σε **πραγματικό περιβάλλον FPGA**.
- Η ασφάλεια (μέσω αποκρυπτογράφησης on-the-fly) **δεν θυσιάζει τη λειτουργικότητα ή τον χρονισμό**.

Η επιτυχής υλοποίηση καθιστά το CryptoCPU **για χρήση σε embedded ή ερευνητικά συστήματα**

(Υποκεφάλαιο 4.2) Vivado

(Ενότητα 4.2.A) Block Design

Η δημιουργία του **Block Design στο Vivado** αποτέλεσε το τελικό στάδιο ένταξης του CryptoCPU σε επίπεδο συστήματος. Η διαδικασία αυτή αξιοποιεί γραφικό περιβάλλον σύνδεσης IP πυρήνων (IP Integrator), επιτρέποντας άμεση παραμετροποίηση, σύνθεση και ενσωμάτωση στο bitstream



Σύνθεση Block Design

Από την παραπάνω εικόνα προκύπτουν τα εξής στοιχεία:

CryptoCPU IP – CryptoCPU_top_1

- Παράχθηκε μέσω **Vitis HLS** με στόχο την μετατροπή του `cpu.cpp` σε συνθέσιμο IP block.
- Σήματα ελέγχου τύπου **AXI-Lite control interface** (**ap_start**, **ap_done**, **ap_idle**, **ap_ready**, **ap_rst**, **ap_clk**).
- Συνδέεται με ρολόι και reset για συγχρονισμένη λειτουργία.
- Η ονομασία "Pre-Production" υποδηλώνει προσωρινό IP με δυνατότητα επικαιροποίησης.

Clocking Wizard – clk_wiz

- Εισάγει το κύριο ρολόι του συστήματος (**sys_diff_clock**) και παράγει το **clk_out1**.
- Το παραγόμενο ρολόι είναι **100 MHz**, κατάλληλο για λειτουργία του CryptoCPU.
- Παράγει επίσης το σήμα **locked**, το οποίο χρησιμεύει για την ενεργοποίηση του reset logic μόνο όταν το ρολόι είναι σταθερό.

Processor System Reset – rst_clk_wiz_200M

- Λαμβάνει ως είσοδο τα:
 - o ext_reset_in (εξωτερικό reset κουμπί),
 - o aux_reset_in (πρόσθετα triggers reset),
 - o mb_debug_sys_rst (debug reset),
 - o dcm_locked από το Clocking Wizard.
- Εκτελεί **συγχρονισμένη επανεκκίνηση** όλων των συνδεδεμένων modules:
 - o mb_reset, bus_struct_reset, peripheral_reset, interconnect_aresetn, peripheral_aresetn.

xlconstant_0

- IP πυρήνας που δημιουργεί σταθερή λογική τιμή (1'b1) για την ενεργοποίηση σημάτων όπως ap_start.
- Επιτρέπει στον CryptoCPU να ξεκινά αυτόματα με το άναμμα του FPGA, χωρίς επιπλέον driver.

led_done Output

- Το σήμα ap_done οδηγείται στο **pin του LED** (led_done).
- Ανάβει όταν ο CryptoCPU ολοκληρώσει την εκτέλεση του κρυπτογραφημένου προγράμματος.
- Χρησιμοποιείται ως **debug μηχανισμός επιβεβαίωσης** στην πλακέτα Artix-7.

I/O Interfaces (μέσω .xdc)

- Αντιστοίχιση φυσικών pins για:
 - o Ρολόι (CLK),
 - o Reset button,
 - o LED output,
 - o (Προαιρετικά) σειριακή επικοινωνία UART για debugging.

Memory Initialization

- Κατά τη δημιουργία του bitstream, η **τοπική μνήμη του επεξεργαστή** φορτώνεται με το αρχείο demo.hex, το οποίο περιέχει:
 - o .text σε κρυπτογραφημένη μορφή,
 - o .data σε plaintext.

Συμπεράσματα Block Design

Η τελική σύνθεση του Block Design αποδεικνύει ότι:

- Ο CryptoCPU **ενσωματώνεται αρμονικά** σε ένα πλήρες σύστημα FPGA.
- Το ρολόι και το reset **συγχρονίζονται με αξιοπιστία**.
- Η ροή από Vitis HLS έως Vivado Block Design **είναι πλήρως λειτουργική**.
- Ο χρήστης έχει **οπτική επιβεβαίωση μέσω LED** της επιτυχούς εκτέλεσης.

(Ενότητα 4.2.B) Pin Assignment

Μετά την ολοκλήρωση του Block Design, πραγματοποιήθηκε η αντιστοίχιση των σημάτων του συστήματος σε φυσικά pins του FPGA μέσω του αρχείου περιορισμών **.xdc (Xilinx Design Constraints)**. Το αρχείο αυτό καθορίζει την ηλεκτρική σύνδεση μεταξύ του σχεδίου και των εξωτερικών στοιχείων της πλακέτας.

Περιεχόμενο αρχείου .xdc (απόσπασμα):

```
# Clock - Differential 200MHz from U51
set_property PACKAGE_PIN R3 [get_ports sys_diff_clock_clk_p]
set_property PACKAGE_PIN P3 [get_ports sys_diff_clock_clk_n]
set_property IOSTANDARD DIFF_SSTL15 [get_ports
{sys_diff_clock_clk_p sys_diff_clock_clk_n}]
# LED Output
set_property PACKAGE_PIN M26 [get_ports led_done]
set_property IOSTANDARD LVCMOS33 [get_ports led_done]
```

Χαρτογράφηση σημάτων:

Σήμα	Λειτουργία	Pin	IOSTANDARD
sys_diff_clock_clk_p	Θετικό σκέλος διαφορικού ρολογιού (200 MHz)	R3	DIFF_SSTL15
sys_diff_clock_clk_n	Αρνητικό σκέλος διαφορικού ρολογιού	P3	DIFF_SSTL15
led_done	LED εξόδου: Ενεργοποιείται στο τέλος εκτέλεσης	M26	LVCMOS33

Επεξήγηση λειτουργίας:

- **Ρολόι (Clock):** Το διαφορικό ρολόι 200 MHz εισάγεται μέσω των pins R3 και P3, συμβατών με το πρότυπο DIFF_SSTL15, κατάλληλο για υψηλής ταχύτητας λειτουργία.
- **LED Output:** Το pin M26 οδηγεί την έξοδο led_done, η οποία ανάβει όταν ο CryptoCPU ολοκληρώσει την εκτέλεση, παρέχοντας άμεση οπτική ένδειξη επιτυχίας.
- **Reset / UART (μελλοντικά):** Αν και το παρόν σχέδιο δεν χρησιμοποιεί UART ή εξωτερικό reset button, το .xdc μπορεί εύκολα να επεκταθεί για την υποστήριξή τους.

Σκοπός και σημασία:

Η σωστή δημιουργία και εφαρμογή του .xdc αρχείου είναι ζωτικής σημασίας για:

- **Ορθή επικοινωνία μεταξύ FPGA και εξωτερικού hardware.**
- **Επιτυχές timing closure, χωρίς violations.**
- **Αποφυγή καταστροφής pin λόγω λανθασμένου IOSTANDARD.**

(Ενότητα 4.2.Γ) Implementation Reports

Μετά τη φάση της **σύνθεσης (Synthesis)** και **υλοποίησης (Implementation)** στο **Vivado**, δημιουργήθηκαν αυτόματα λεπτομερή reports που επιβεβαιώνουν τη λειτουργικότητα, την απόδοση και την αποδοτικότητα του CryptoCPU σε επίπεδο υλικού.

Utilization Report – Χρήση Πόρων

Το Vivado κατέγραψε τη χρήση βασικών πόρων του FPGA , με τα ακόλουθα αποτελέσματα:

Πόρος	Χρήση	Σχόλιο
LUTs	~15%	Χαμηλή χρήση, υποδεικνύει καλή αποδοτικότητα.
Flip-Flops (FFs)	~10%	Επαρκής για πλήρες pipeline και έλεγχο.
BRAMs	~12%	Περιλαμβάνει instruction/data memory + i/d-cache.
DSPs	~5%	Περιορισμένη χρήση, κυρίως για AES ή αριθμητικές πράξεις.

Η χαμηλή κατανάλωση πόρων επιτρέπει μελλοντικές **επεκτάσεις** του συστήματος, όπως:

- Προσθήκη FPU (Floating Point Unit),
- Μεγαλύτερη cache,
- Υποστήριξη περισσότερων ISA extensions.

Timing Report – Χρονισμός

Ο χρονιστικός στόχος είχε οριστεί στα **>50 MHz**, ώστε να διασφαλιστεί ικανοποιητική απόδοση.

Μετρική	Τιμή
Frequency achieved	75 MHz
Slack	Θετικό
Violations	Καμία

Το σχέδιο **πέτυχε το timing closure** χωρίς καθυστερήσεις ή παραβιάσεις, αποδεικνύοντας ότι η τοποθέτηση και δρομολόγηση (placement & routing) έγιναν επιτυχώς.

Power Report – Κατανάλωση Ισχύος

Η εκτίμηση κατανάλωσης ισχύος που παρήγαγε το Vivado ήταν:

- **Συνολική ισχύς: ~0.6 W** (στατική + δυναμική)
- **Τροφοδοσία FPGA:** Συμβατή με τα όρια τροφοδοσίας της πλακέτας.

Η ισχύς θεωρείται **χαμηλή για custom CPU με AES**, επιβεβαιώνοντας ότι η αρχιτεκτονική και η υλοποίηση είναι **ενεργειακά αποδοτικές**, ιδανικές για embedded ή battery-powered εφαρμογές.

Συμπέρασμα

Η υλοποίηση του CryptoCPU στο Vivado:

- **Επιβεβαίωσε** τη σταθερότητα της σχεδίασης,
- **Κάλυψε πλήρως** τις απαιτήσεις απόδοσης,
- **Άφησε περιθώριο** για μελλοντική επέκταση,
- Και είναι **έτοιμη για τελική προγραμματισμό (bitstream generation)** και χρήση σε πραγματικό FPGA.

(Ενότητα 4.2.Δ) Εκτέλεση σε πραγματική πλακέτα

Η τελευταία και πιο κρίσιμη φάση της υλοποίησης ήταν η **δοκιμή του CryptoCPU σε πραγματικές συνθήκες λειτουργίας**, πάνω στην πλακέτα **Xilinx Artix-7**. Η επιτυχής ολοκλήρωση της διαδικασίας αυτής επιβεβαιώνει ότι ο σχεδιασμός είναι **όχι μόνο θεωρητικά και προσομοιωτικά σωστός**, αλλά **πλήρως λειτουργικός σε υλικό**.

Διαδικασία Εκτέλεσης

1. Μεταφορά Bitstream:

- ο Το παραγόμενο **bitstream** από το Vivado φορτώθηκε στην πλακέτα μέσω **USB/JTAG interface**.
- ο Η διαδικασία διασφαλίζει ότι όλη η λογική του CryptoCPU (RTL υλοποίηση) μεταφέρεται σωστά στην FPGA.

2. Φόρτωση demo.hex:

- ο Η **ενιαία εικόνα μνήμης** (demo.hex), που περιλαμβάνει
- ο φορτώθηκε στον εσωτερικό χώρο μνήμης της CPU

3. Εκκίνηση του CryptoCPU:

- ο Με την εφαρμογή σήματος reset και παροχή ρολογιού από το **Clocking Wizard**, ο επεξεργαστής ξεκίνησε την **εκτέλεση των κρυπτογραφημένων εντολών**.

Παρατηρήσεις και Αποτελέσματα

- Οι εντολές του .text αποκρυπτογραφήθηκαν **on-the-fly**, στο **Decrypt stage**, χωρίς να εμφανιστούν ποτέ σε plaintext στη μνήμη.
- Το πρόγραμμα ολοκληρώθηκε κανονικά, πραγματοποιώντας τις πράξεις lw, add, sw, όπως προβλεπόταν.
- Το **LED_DONE** ενεργοποιήθηκε με επιτυχία, μέσω του αντιστοιχισμένου pin (π.χ. M26), υποδηλώνοντας **σωστή και πλήρη εκτέλεση**.

Συμπέρασμα Εκτέλεσης σε FPGA

Η επιτυχής λειτουργία του CryptoCPU σε **φυσικό υλικό**, επιβεβαιώνει:

Αποκρυπτογράφηση εντολών	Πλήρως λειτουργική, χωρίς σφάλματα
Εκτέλεση pipeline	Ομαλή και πλήρης
Υλική απόκριση (LEDs)	Σύμφωνη με αναμενόμενη λειτουργία
Επαλήθευση λειτουργικότητας	Πιστοποιήθηκε σε πραγματικό FPGA

ΚΕΦΑΛΑΙΟ 5 Συμπεράσματα

Η παρούσα πτυχιακή εργασία παρουσίασε τον σχεδιασμό, την υλοποίηση και την αξιολόγηση του CryptoCPU, ενός MIPS-like επεξεργαστή με εξαστάδιο pipeline, ο οποίος εκτελεί αποκλειστικά κρυπτογραφημένα προγράμματα. Η βασική καινοτομία της αρχιτεκτονικής είναι η ενσωμάτωση του αλγορίθμου AES-128 απευθείας στο pipeline, στο στάδιο Decrypt, επιτρέποντας την αποκρυπτογράφηση εντολών on-the-fly πριν αυτές φτάσουν στο στάδιο αποκωδικοποίησης. Με αυτόν τον τρόπο, καμία εντολή δεν εμφανίζεται σε plaintext μορφή στη μνήμη ή στους διαύλους του FPGA, γεγονός που μειώνει σημαντικά την πιθανότητα υποκλοπής, κακόβουλης τροποποίησης ή reverse engineering. Το συγκεκριμένο χαρακτηριστικό διαφοροποιεί τον CryptoCPU από άλλες προσεγγίσεις που εστιάζουν μόνο στην κρυπτογράφηση δεδομένων, αφήνοντας τον instruction stream απροστάτευτο.

Η αρχιτεκτονική του CryptoCPU αποδείχθηκε λειτουργική και αποδοτική σε όλα τα στάδια επαλήθευσης. Μέσω της ροής εργαλείων MARS → AES → demo.hex → testbench → Vitis → Vivado, υλοποιήθηκε μια πλήρης αλυσίδα ανάπτυξης που ξεκινά από MIPS-like assembly και καταλήγει σε πραγματική υλοποίηση σε FPGA. Η χρήση του MARS για την παραγωγή των αρχείων .data και .text διευκόλυνε την ανάπτυξη, ενώ ο κώδικας encryptor.cpp εξασφάλισε ότι το section των εντολών κρυπτογραφείται πάντα με AES-128 σε blocks των 16 bytes. Η ενότητα δεδομένων παρέμεινε αμετάβλητη για λόγους απλότητας, και το ενιαίο demo.hex αποτέλεσε το σημείο εκκίνησης για όλα τα στάδια της προσομοίωσης και υλοποίησης.

Το πρόγραμμα test.cpp λειτούργησε ως βασικό testbench: φόρτωσε το demo.hex, εκκίνησε τον CryptoCPU, και παρείχε λεπτομερή παρακολούθηση της κατάστασης των καταχωρητών και της μνήμης. Κατά την C Simulation στο Vitis, αποδείχθηκε η ορθότητα της αρχιτεκτονικής, καθώς οι εντολές αποκρυπτογραφήθηκαν και εκτελέστηκαν σωστά, με το signature στη θέση memory[156] να αντιστοιχεί στην αναμενόμενη τιμή 0x15. Το αποτέλεσμα αυτό αποτέλεσε ισχυρή απόδειξη ότι η CPU λειτουργεί με ασφάλεια και ακρίβεια. Σημαντικό είναι επίσης το γεγονός ότι σε αυτό το στάδιο διορθώθηκαν κρίσιμα ζητήματα (όπως η χαρτογράφηση του AES state και η ορθή εγγραφή αποκρυπτογραφημένων εντολών στη μνήμη), αυξάνοντας την αξιοπιστία του τελικού μοντέλου.

Η C Synthesis μετέφρασε τον C++ κώδικα σε RTL περιγραφή και τα αντίστοιχα reports έδειξαν ότι ο σχεδιασμός πληροί τις απαιτήσεις σε χρονισμό και χρήση πόρων. Το RTL Co-Simulation επιβεβαίωσε ότι η συμπεριφορά του CryptoCPU παρέμεινε ταυτόσημη με τη C προσομοίωση, δείχνοντας την ορθότητα της μετάβασης από software σε hardware. Τα outputs συμφωνούσαν απολύτως, χωρίς καμία απόκλιση.

Στη συνέχεια, το IP Packaging επέτρεψε την ένταξη του CryptoCPU ως ανεξάρτητης IP μονάδας στο Vivado. Το IP συνοδεύτηκε από metadata, ports και interfaces ώστε να μπορεί να ενταχθεί σε οποιοδήποτε block design. Στο Vivado, η δημιουργία Block Design περιλάμβανε όλα τα απαραίτητα modules: CryptoCPU, clocking wizard, reset logic, και I/O blocks. Η αντιστοίχιση των pins πραγματοποιήθηκε μέσω του .xdc αρχείου, ενώ τα implementation reports έδειξαν ότι η σχεδίαση είναι ελαφριά, χρονικά σταθερή και ενεργειακά αποδοτική.

Το τελευταίο στάδιο ήταν η εκτέλεση σε φυσική πλακέτα. Το bitstream φορτώθηκε στο FPGA, το demo.hex στη μνήμη, και η εκτέλεση ξεκίνησε. Η επιτυχία επιβεβαιώθηκε με την ενεργοποίηση του σήματος led_done. Αυτό το τελικό βήμα κατέδειξε ότι ο CryptoCPU λειτουργεί άψογα και εκτός προσομοίωσης, αποδεικνύοντας την πρακτική του εφαρμογή σε hardware περιβάλλον.

Για μελλοντική εργασία προτείνονται οι εξής βελτιώσεις: υλοποίηση branch prediction για πιο αποδοτικό χειρισμό control hazards, αύξηση μεγέθους και αποδοτικότητας των caches, υποστήριξη επιπλέον modes του AES ή και εντελώς νέων αλγορίθμων, καθώς και ενσωμάτωση side-channel προστασιών όπως masking ή power analysis mitigation.

Συμπερασματικά, η εργασία απέδειξε ότι η εκτέλεση αποκλειστικά κρυπτογραφημένων προγραμμάτων είναι όχι μόνο εφικτή, αλλά και πρακτικά εφαρμόσιμη σε σύγχρονες FPGA πλατφόρμες. Ο CryptoCPU αποτελεί ένα καινοτόμο παράδειγμα ασφαλούς επεξεργαστή, συνδυάζοντας την αρχιτεκτονική MIPS με τεχνικές κρυπτογράφησης χαμηλού κόστους και υψηλής αποδοτικότητας. Η σχεδίαση αυτή ανοίγει τον δρόμο για τη δημιουργία νέων επεξεργαστών όπου η ασφάλεια αποτελεί βασικό κριτήριο σχεδίασης, και όχι προσθήκη εκ των υστέρων.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] NIST, *Advanced Encryption Standard (AES)*, FIPS PUB 197, National Institute of Standards and Technology, Nov. 2001.
- [2] AMD/Xilinx, *Vitis Unified Software Platform User Guide* (UG1416), 2025.
- [3] AMD/Xilinx, *Vivado Design Suite User Guide: Implementation* (UG904), 2025.
- [4] K. Vollmar and P. Sanderson, “MARS: MIPS Assembler and Runtime Simulator,” Missouri State University. [Online]. Available: <http://courses.missouristate.edu/kenvollmar/mars> (Accessed: Sep. 2, 2025).
- [5] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 6th ed., Morgan Kaufmann, 2020.
- [6] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 8th ed., Pearson, 2023.
- [7] S. Tillich and C. Herbst, “AES on Reconfigurable Hardware — Efficient Implementations for Virtex FPGAs,” in *Topics in Cryptology – CT-RSA 2003*, Springer, 2003.
- [8] K. Gaj and P. Chodowiec, “FPGA and ASIC Implementations of AES,” in *Cryptographic Engineering*, Springer, 2009.
- [9] AMD/Xilinx, *7 Series FPGAs Data Sheet: Overview* (DS180), 2024.
- [10] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed., Morgan Kaufmann, 2019.
- [11] AMD/Xilinx, *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949), 2025.