ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ &
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός και υλοποίηση SoC σε αρχιτεκτονική ARM

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Νίκου Ραφαήλ

Επιβλέπων: Δρ. Δασυγένης Μηνάς

Εργαστήριο Ρομποτικής, Ενσωματωμένων και Ολοκληρωμένων Συστημάτων

ΚΟΖΑΝΗ/ΙΟΥΛΙΟΣ/2024

HELLENIC DEMOCRACY
UNIVERSITY OF WESTERN MACEDONIA

FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL &
COMPUTER ENGINEERING

SoC design and implementation in ARM architecture

DIPLOMA THESIS

Nikou Rafail

SUPERVISOR: Dr. Minas Dasygenis

Assistant Professor

Robotics, Embedded and Integrated Systems Laboratory

KOZANI/JULY/2024

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο “Σχεδιασμός και υλοποίηση SoC σε αρχιτεκτονική ARM” καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Δασυγένη Μηνά, Επίκουρου Καθηγητή αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Υπογραφή Φοιτητή: Νίκου Ραφαήλ

Περίληψη

Η ταχεία εξέλιξη του ηλεκτρονικού εμπορίου και της ψηφιακής τεχνολογίας έχει αυξήσει την ανάγκη για ασφαλή διαχείριση των δεδομένων χρηστών. Οι Συστοιχίες Πυλών Προγραμματιζόμενες στο Πεδίο (Field Programmable Gate Arrays - FPGAs) διαδραματίζουν κρίσιμο ρόλο στην ανάπτυξη τέτοιων συστημάτων, προσφέροντας προσαρμοστικότητα και υψηλές επιδόσεις λόγω της δυνατότητάς τους για προγραμματισμό στο πεδίο λειτουργίας.

Στο πλαίσιο αυτής της διπλωματικής εργασίας, εξετάζεται η σχεδίαση και η υλοποίηση ενός περιφερειακού συστήματος κρυπτογράφησης, εστιάζοντας στον αλγόριθμο SHA-256 της οικογένειας SHA-2, και η διασύνδεσή του με επεξεργαστή αρχιτεκτονικής ARM. Η επιλεγμένη πλατφόρμα για την υλοποίηση είναι η πλακέτα Kria KR260 Robotics, η οποία διαθέτει ισχυρές δυνατότητες προγραμματισμού και ευρεία υποστήριξη. Ο στόχος της εργασίας είναι η δημιουργία ενός συστήματος που θα επιτυγχάνει την κρυπτογράφηση οποιασδήποτε εισόδου δεδομένων με την καλύτερη δυνατή ταχύτητα, συγκρινόμενο με αντίστοιχες υλοποιήσεις σε άλλες πλακέτες. Ο κύριος στόχος της εργασίας είναι η ανάπτυξη ενός αποδοτικού και ταχύτατου συστήματος κρυπτογράφησης SHA-256 σε FPGA. Οι επιμέρους στόχοι περιλαμβάνουν: μελέτη και ανάλυση του αλγορίθμου SHA-256, σχεδίαση του συστήματος σε FPGA, διασύνδεση με επεξεργαστή ARM, και αξιολόγηση των επιδόσεων.

Η εργασία αυτή αποδεικνύει τη δυνατότητα αξιοποίησης των FPGAs για την υλοποίηση αλγορίθμων κρυπτογράφησης υψηλής απόδοσης. Μέσω της διασύνδεσης με επεξεργαστή ARM, επιτυγχάνεται ένα ολοκληρωμένο σύστημα που προσφέρει σημαντικά πλεονεκτήματα στην ασφάλεια και την ταχύτητα επεξεργασίας δεδομένων, συμβάλλοντας στη βελτίωση των εφαρμογών ψηφιακής ασφάλειας και ηλεκτρονικού εμπορίου. Η μελλοντική έρευνα μπορεί να εστιάσει σε επέκταση σε άλλους αλγορίθμους κρυπτογράφησης, βελτιστοποίηση ενεργειακής κατανάλωσης και εφαρμογές σε πραγματικούς κόσμους.

Λέξεις Κλειδιά: FPGA, κρυπτογράφηση, αλγόριθμος, επεξεργαστής, δεδομένα, διεπαφή

Abstract

The rapid evolution of e-commerce and digital technology has increased the need for secure management of user data. Field Programmable Gate Arrays (FPGAs) play a critical role in the development of such systems, offering adaptability and high performance due to their ability to be field programmable.

In this thesis, the design and implementation of a peripheral encryption system, focusing on the SHA-256 algorithm of the SHA-2 family, and its interface with an ARM architecture processor is considered. The chosen platform for the implementation is the Kria KR260 Robotics board, which has powerful programming capabilities and broad support. The goal of the work is to create a system that achieves encryption of any data input at the best possible speed, compared to similar implementations on other boards. The main objective of the work is to develop an efficient and fast SHA-256 encryption system on FPGA. The sub-objectives include study and analysis of the SHA-256 algorithm, design of the system on FPGA, interface with ARM processor, and performance evaluation.

This work demonstrates the potential to exploit FPGAs to implement high-performance encryption algorithms. By interfacing with an ARM processor, an integrated system is achieved that offers significant advantages in security and data processing speed, contributing to the improvement of digital security and e-commerce applications. Future research may focus on extension to other encryption algorithms, energy consumption optimization and real-world applications.

Keywords: FPGA, encryption, algorithm, processor, data, interface

Ευχαριστίες

Η παρούσα διπλωματική εκπονήθηκε στα πλαίσια του προπτυχιακού προγράμματος σπουδών της Πολυτεχνικής σχολής του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, του πανεπιστημίου Δυτικής Μακεδονίας. Στο σημείο αυτό θα ήθελα να εκφράσω τις ευχαριστίες μου προς τα πρόσωπα που διαδραμάτισαν καθοριστικό ρόλο στην εκπόνηση της παρούσας διπλωματικής εργασίας.

Θα ήθελα να ευχαριστήσω όλους τους καθηγητές του τμήματος που με βοήθησαν στο να αποκομίσω τις γνώσεις που θα συνδράμουν στην εξέλιξη μου στον τομέα του Ηλεκτρολόγου Μηχανικού και Μηχανικού Υπολογιστών. Ιδιαίτερα, θα ήθελα να εκφράσω ένα μεγάλο ευχαριστώ στον επιβλέποντα της διπλωματικής μου τον Δρ. Μηνά Δασυγένη για την εμπιστοσύνη που μου έδειξε στην ανάληψη της παρούσας διπλωματικής καθώς και για την ανελλιπή και πολύτιμη βοήθεια του οποιαδήποτε ώρα της ημέρας για την επίτευξη και την καθοδήγηση στο τελικό αποτέλεσμα. Παράλληλα, θα ήθελα να ευχαριστήσω τον Δρ. Ασημόπουλο Νικόλαο για της γνώσεις στα ψηφιακά κυκλώματα που μου μετέφερε από τον πρώτο εξάμηνο και σε όλο το πέρας τον σπουδών μου, τον Δρ. Πλόσκα Νικόλαο για την συνεισφορά των γνώσεων στο πεδίο του προγραμματισμού αλγορίθμων στην γλώσσα C, καθώς και τον Δρ. Λαζαρίδη Βασίλειο για τις γνώσεις στο πεδίο της ασφάλειας.

Επίσης, θα ήθελα να ευχαριστήσω όλο το προσωπικό του εργαστηρίου Ρομποτικής και Ενσωματωμένων Συστημάτων, διδάκτορες και φοιτητές, για οποιαδήποτε βοήθεια μου παρείχανε καθώς και για τις ώρες που μου επέτρεψαν να βρίσκομαι στο χώρο του εργαστηρίου για το πειραματικό μέρος της παρούσας διπλωματικής.

Τέλος, θα ήθελα να πω ένα τεράστιο ευχαριστώ στην οικογένεια μου, καθώς και στους φίλους μου στην σχολή και εκτός για την αμέριστη στήριξή τους και την συμπαράσταση στην εκπόνηση και συγγραφή του παρόντος κειμένου χωρίς την συμβολή τους το παρών έγγραφο δεν θα είχε δρομολογηθεί ποτέ.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Πρόλογος

Στις επόμενες σελίδες, παρουσιάζεται αναλυτικά η έρευνα και η ανάπτυξη ενός συστήματος κρυπτογράφησης βασισμένου στον αλγόριθμο SHA-256 σε συνδυασμό με έναν επεξεργαστή τύπου ARM. Το προγραμματιστικό μέρος της παρούσας διπλωματικής εργασίας υλοποιήθηκε με τη χρήση λογισμικού της εταιρείας Xilinx, το οποίο διατίθεται δωρεάν. Το πειραματικό σκέλος διεξήχθη στο Εργαστήριο Ρομποτικής και Ενσωματωμένων Συστημάτων του Πανεπιστημίου Δυτικής Μακεδονίας, στην πόλη της Κοζάνης. Η διεξαγωγή της έρευνας και των πειραμάτων που περιγράφονται στο παρόν έγγραφο δεν θα ήταν δυνατή χωρίς την επίβλεψη, την καθοδήγηση και την υποστήριξη των μελών του εργαστηρίου του πανεπιστημίου, καθώς και του επιβλέποντα καθηγητή μου.

Κεφάλαιο 1: Εισαγωγή

Για λόγους πληρότητας, το παρόν κεφάλαιο περιλαμβάνει μια συνοπτική περιγραφή των FPGAs και της χρήσης τους. Επιπλέον, παρέχεται μια ιστορική ανασκόπηση της εξέλιξής τους σε σύγκριση με τους μικροεπεξεργαστές, θέμα που θα αναλυθεί εκτενέστερα σε επόμενο κεφάλαιο. Γίνεται επίσης μια σύγκριση με προ υπάρχουσες τεχνικές και μελέτες στο αντίστοιχο πεδίο. Τέλος, παρουσιάζεται ο σκοπός της παρούσας διπλωματικής εργασίας, καθορίζοντας το πλαίσιο για τις επόμενες αναλύσεις και εφαρμογές.

1.1 Αντικείμενο της διπλωματικής

Οι Συστοιχίες Πυλών Προγραμματιζόμενες στο Πεδίο (Field Programmable Gate Arrays - FPGAs) έχουν ενσωματωθεί στον τομέα του σχεδιασμού ψηφιακών συστημάτων ως εναλλακτικά στοιχεία προς τους μικροεπεξεργαστές. Οι μικροεπεξεργαστές προσφέρουν τη δυνατότητα αξιοποίησης σε διάφορα προγραμματιστικά περιβάλλοντα, αλλά δεδομένου ότι βασίζονται κυρίως σε λογισμικό για την υλοποίηση των συναρτήσεών τους, συχνά είναι πιο αργοί και καταναλώνουν περισσότερη ενέργεια σε σύγκριση με τα εξατομικευμένα ολοκληρωμένα κυκλώματα. Παρόλο που τα FPGAs έχουν χαμηλότερη απόδοση και υψηλότερη κατανάλωση ενέργειας σε σχέση με τα πλήρως εξατομικευμένα συστήματα, έχουν αρχίσει να καθίστανται αναγκαία στην ανάπτυξη πολύπλοκων υπολογιστικών συστημάτων λόγω της αυξανόμενης προγραμματιστικής λογικής και πολυπλοκότητάς τους [1]. Αξιοποιούν τόσο προγραμματιζόμενα λογικά στοιχεία όσο και προγραμματιζόμενες διασυνδέσεις, με στόχο τη δημιουργία πολυεπίπεδων λογικών συναρτήσεων.

Η ανακάλυψη των Συστοιχιών Πυλών Προγραμματιζόμενου στο Πεδίο (FPGAs) προσδίδεται στον Ross Freeman [1]. Τα αρχικά FPGAs που ανέπτυξε περιλάμβαναν προγραμματιζόμενα λογικά στοιχεία και ρυθμιζόμενες διασυνδέσεις, με τον προγραμματισμό να επιτελείται μέσω στατικών μνημών SRAM. Η καινοτομία του Freeman προέκυψε από τη δημιουργία ενός συστήματος που επέτρεπε τον προγραμματισμό επί του κυκλώματος, χρησιμοποιώντας μνήμη flash, η οποία τη δεδομένη περίοδο το 1984 δεν είχε ευρεία αποδοχή. Ο Ross Freeman ήταν συνιδρυτής της Xilinx, μιας εταιρείας που αργότερα αγοράσθηκε από την AMD®. Οι δύο κύριες εταιρείες που δραστηριοποιούνται στην κατασκευή, προγραμματισμό και διανομή των FPGAs είναι η Xilinx και η Altera® [2]. Τα FPGAs που παράγουν βασίζονται σε μνήμες SRAM. Μια άλλη εταιρεία, η Actel, πρότεινε μια διαφορετική αρχιτεκτονική βασισμένη σε τεχνολογίες αντιασφαλειών. Σε αντίθεση με τις Xilinx και Altera®, τα ολοκληρωμένα συστήματα της Actel δεν μπορούσαν να επαναπρογραμματιστούν επί του πεδίου, ένα γεγονός που ενδέχεται να θεωρηθεί ως πλεονέκτημα σε περιπτώσεις που δεν απαιτείται προγραμματιστική αλλαγή [3], καθώς το κόστος τους είναι χαμηλότερο. Τα ολοκληρωμένα κυκλώματα της Actel χρησιμοποιούν λογική που βασίζεται σε πολυπλέκτες, τοποθετημένους γύρω από τα κανάλια της καλωδίωσης.

1.2 Παρόμοια έρευνα στο αντικείμενο της διπλωματικής

Η κρυπτογράφηση αποτελεί ένα πεδίο που βιώνει σημαντική εξέλιξη στην εποχή του σύγχρονου Διαδικτύου. Η σημαντικότητά της αναδεικνύεται ιδιαίτερα σε περιβάλλοντα όπου η ανταλλαγή πληροφοριών απαιτεί υψηλά επίπεδα ασφάλειας, πιστοποίησης και ακεραιότητας. Η ευρεία διάδοση της κρυπτογράφησης σε όλους τους τομείς της σύγχρονης κοινωνίας οδηγεί σε συνεχείς εξελίξεις στον τομέα της χρήσης των Συστοιχιών Πυλών Προγραμματιζόμενων στο Πεδίο (FPGAs) σε συνδυασμό με αλγορίθμους κρυπτογράφησης [4]. Η συνεργασία αυτή στοχεύει στην επίτευξη ασφαλών αποτελεσμάτων και στη χρονική βελτιστοποίηση των αλγορίθμων, προσφέροντας ένα ευρύ φάσμα εφαρμογών με υψηλή απόδοση και αξιοπιστία. Για παράδειγμα ο [5] εξετάζει τις δυνατότητες των FPGA και αξιοποιεί την κρυπτογραφική ισχύ και την υπολογιστική αποδοτικότητα του αλγορίθμου Keccak-512, δίνοντας έμφαση στις δυνατότητες των ολοκληρωμένων συστημάτων. Μια άλλη εφαρμογή χρήσης των FPGAs αποτελεί η κατασκευή ενός συστήματος κρυπτογράφησης από τον [6] με προδιαγραφές παρόμοιες με αυτές της παρούσας εργασίας, επικεντρωμένο στην υλοποίηση SW/HW (Software/Hardware) της συνάρτησης κατακερματισμού SHA-256 και την επίτευξη εξοικονόμησης ενέργειας στην πλακέτα Xilinx Zynq 7000 σε σύγκριση με αποκλειστικά λογισμικές υλοποιήσεις. Επιπλέον, η έρευνα του [7] περιγράφει την ανάπτυξη και δοκιμή μιας παρόμοιας αρχιτεκτονικής για τον αλγόριθμο SHA-256, η οποία υλοποιείται σε γλώσσα περιγραφής υλικού (HDL) και δοκιμάζεται στην πλακέτα Spartan 6 SP605 Evaluation Board. Αυτή η προσέγγιση επιτρέπει την εκτεταμένη παραμετροποίηση και τη βελτιστοποίηση της απόδοσης του συστήματος, προσφέροντας μια λύση υψηλής απόδοσης και ασφάλειας.

1.3 Στόχος της διπλωματικής

Η παρούσα διπλωματική εργασία, με τίτλο «Σχεδιασμός και υλοποίηση SoC σε αρχιτεκτονική ARM», αναδεικνύει ως κύριο στόχο τη δημιουργία ενός περιφερειακού συστήματος και τη διασύνδεσή του με έναν επεξεργαστή αρχιτεκτονικής τύπου ARM. Ως περιφερειακό επιλέγεται η υλοποίηση ενός αλγορίθμου κρυπτογράφησης της οικογένειας SHA-2 και συγκεκριμένα τον αλγόριθμο SHA-256, ο οποίος θα αναλυθεί εκτενώς σε επόμενο κεφάλαιο. Ως προγραμματιζόμενη πλακέτα δοκιμής επιλέγεται η πλακέτα Kria KR260 Robotics [8], η οποία διαθέτει τις κατάλληλες δυνατότητες προγραμματισμού για τον επεξεργαστή της και υποστηρίζεται ευρέως τόσο από προγράμματα όσο και από την διαδικτυακή κοινότητα. Το SOM (System-on-Module) που χρησιμοποιείται είναι σχεδιασμένο για απλότητα και συμπαγή δομή, περιλαμβάνοντας όλα τα βασικά στοιχεία όπως επεξεργαστή Zynq® UltraScale+™ MPSoC-based, μνήμη, boot και security module. Ο επεξεργαστής της πλακέτας είναι ένας Quad Arm Cortex-A53 με Dual Arm Cortex-R5F, κατασκευασμένος με τεχνολογία 16nm FinFET με επιπρόσθετη υποστήριξη Programmable Logic [9]. Λεπτομερείς πληροφορίες σχετικά με την προγραμματιζόμενη πλακέτα θα παρουσιαστούν σε επόμενο κεφάλαιο. Η πλακέτα, ως προϊόν της εταιρίας Xilinx, υποστηρίζεται από προγράμματα της εταιρίας, ακόμη και από τις δωρεάν εκδόσεις της. Στο πλαίσιο της παρούσας εργασίας, για τον προγραμματισμό και τις δοκιμές χρησιμοποιήθηκαν πολλά εξειδικευμένα περιβάλλοντα. Το ModelSim αποτελεί ένα από τα κύρια εργαλεία που χρησιμοποιήθηκαν για την προσομοίωση των κυκλωμάτων. Επιπλέον, το Vitis, αν και χρησιμοποιήθηκε η προηγούμενη έκδοση, αναμένεται να αντικατασταθεί από ένα ενοποιημένο IDE που αναπτύσσεται από την AMD®, συμπεριλαμβανομένων και άλλων εφαρμογών. Το Vivado, σε έκδοση 2023.2 Standard ML, αποτελεί ένα ακόμα σημαντικό εργαλείο που χρησιμοποιήθηκε για το σχεδιασμό και την ανάπτυξη του συστήματος. Επιπλέον, χρησιμοποιήθηκε ένα Linux image έκδοσης 20.04, το οποίο παρέχει το απαραίτητο περιβάλλον για τη λειτουργία της πλακέτας. Τέλος, για την εξέλιξη των λειτουργιών

της πλακέτας και τον προγραμματισμό σε γλώσσα Python, χρησιμοποιήθηκε μια ειδική έκδοση της βιβλιοθήκης PYNQ, η οποία προσφέρει σημαντική υποστήριξη στην προγραμματιστική διαδικασία μέσω του Jupyter Notebooks.

1.4 Οργάνωση της διπλωματικής

Πριν εμβαθύνουμε στο θέμα της παρούσας διπλωματικής εργασίας, είναι απαραίτητο να καθορίσουμε και να κατανοήσουμε το δομικό πλαίσιο της μελέτης τόσο σε προγραμματιστικό όσο και σε πειραματικό επίπεδο. Το παρόν έγγραφο αποτελείται από πέντε κεφάλαια. Στο Κεφάλαιο 2, με τίτλο «Θεωρητικό Υπόβαθρο», θα μελετήσουμε ενδελεχώς και θα παρουσιάσουμε τα προγράμματα που χρησιμοποιήθηκαν για την διεκπεραίωση της παρούσας διπλωματικής, αναλύοντας τις δυνατότητες και τους περιορισμούς που μας επέβαλαν και θα κάνουμε μια σύντομη παρουσίαση των FPGA καθώς και μια σύγκριση με ανάλογα στοιχεία της αγοράς. Στη συνέχεια, στο Κεφάλαιο 3 με τίτλο «Ανάλυση του Αλγορίθμου», θα αναλύσουμε και θα περιγράψουμε τον αλγόριθμο κρυπτογράφησης που βασίζεται το περιφερειακό μας σύστημα, εξετάζοντας τη λειτουργία του τόσο σε θεωρητικό όσο και σε προγραμματιστικό επίπεδο. Στόχος είναι η κατανόηση των εννοιών που θα χρησιμοποιηθούν στην υλοποίηση του συστήματος, παρουσιάζοντας το θεωρητικό υπόβαθρο καθώς και την πρακτική εφαρμογή του στο προγραμματιστικό περιβάλλον, και συζητώντας τα διάφορα αντικείμενα και τις δομές που χρησιμοποιήθηκαν. Ακολουθεί το Κεφάλαιο 4, το οποίο στοχεύει να γεφυρώσει το χάσμα μεταξύ του θεωρητικού και του πρακτικού υπόβαθρου. Σε αυτή την ενότητα, παρουσιάζονται λεπτομερώς στοιχεία για τον επεξεργαστή και την πλακέτα, περιγράφοντας την υλοποίηση τόσο στο επίπεδο του υλικού (hardware) όσο και του λογισμικού (software). Ιδιαίτερη έμφαση δίνεται στα πρωτόκολλα επικοινωνίας και στον προγραμματισμό του επεξεργαστή, που συμπληρώνουν τη σύζευξη του συστήματος κρυπτογράφησής μας. Τέλος, το Κεφάλαιο 5, με τίτλο «Συμπεράσματα-Επίλογος», διατυπώνει μια κριτική άποψη για το παρόν σύστημα, τονίζοντας την αποτελεσματικότητά του και προτείνοντας βελτιώσεις για κάθε στάδιο της διαδικασίας. Το παρόν έγγραφο είναι δομημένο με σχολαστικότητα, επιδιώκοντας την ουσιαστική εμβάθυνση και κατανόηση της θεματολογίας. Προσκαλεί τους αναγνώστες να εντρυφήσουν στο σύγγραμμα, προσεγγίζοντας το αντικείμενο με σαφήνεια, ακρίβεια και ενθουσιασμό, συμβάλλοντας στην περαιτέρω εξέλιξη του τομέα των ενσωματωμένων συστημάτων και των προγραμματιζόμενων συσκευών.

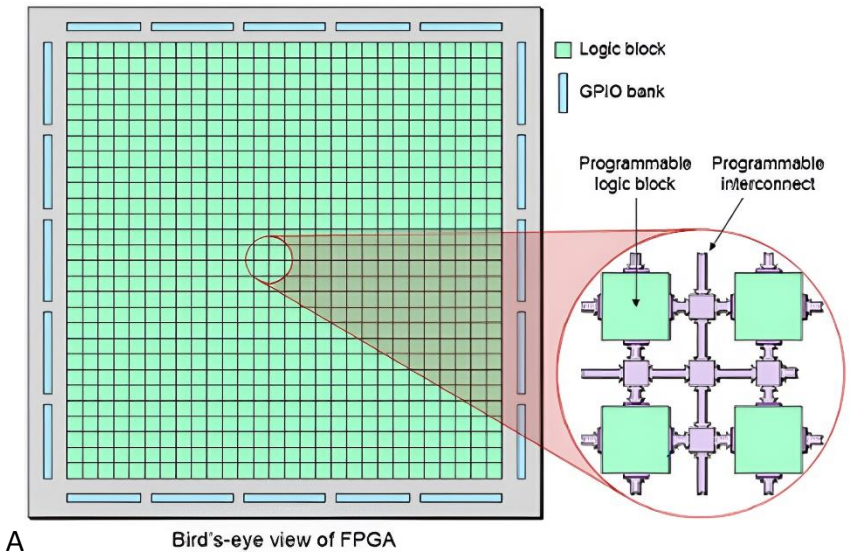
Κεφάλαιο 2: Θεωρητικό Υπόβαθρο

Στο παρόν κεφάλαιο, παρουσιάζονται γενικές πληροφορίες σχετικά με τα FPGA καθώς και τα λογισμικά που χρησιμοποιήθηκαν κατά τη διάρκεια της παρούσας διπλωματικής εργασίας. Επιπλέον, αναλύονται τα βασικά σημεία που συνέβαλαν στην ανάπτυξη του τελικού συστήματος, παρέχοντας μια ολοκληρωμένη εικόνα των εργαλείων και των μεθοδολογιών που εφαρμόστηκαν.

2.1 Εργαλεία ανάπτυξης και προγραμματισμού του συστήματος

2.1.1 FPGA

Η διάταξη προγραμματιζόμενων πυλών στο πεδίο, γνωστή ως FPGA (Field-Programmable Gate Array), αποτελεί έναν τύπο ολοκληρωμένου κυκλώματος (IC) που επιτρέπει την ανάπτυξη προσαρμοζόμενης λογικής για τη δημιουργία πολύ γρήγορων και ταχύτατων πρωτοτύπων, τα οποία μπορούν να οδηγήσουν στον τελικό σχεδιασμό των συστημάτων Εικόνα 1. Σε αντίθεση με άλλα προσαρμοζόμενα ή ημί-προσαρμοζόμενα ολοκληρωμένα συστήματα, τα FPGA διακρίνονται για την ευελιξία τους στον τομέα του προγραμματισμού και του επαναπρογραμματισμού, μέσω της επαναδιαμόρφωσης του λογισμικού τους [10]. Η δυνατότητα αυτή τους επιτρέπει την προσαρμογή στις εξελισσόμενες ανάγκες των εφαρμογών τους, όπως η λειτουργία σε εξωτερικούς χώρους με δυσμενείς συνθήκες, καθώς και στις νέες απαιτήσεις και προκλήσεις που προκύπτουν από την τεχνολογική εξέλιξη.



Εικόνα 1 FPGA με την ματιά του μικροσκοπίου

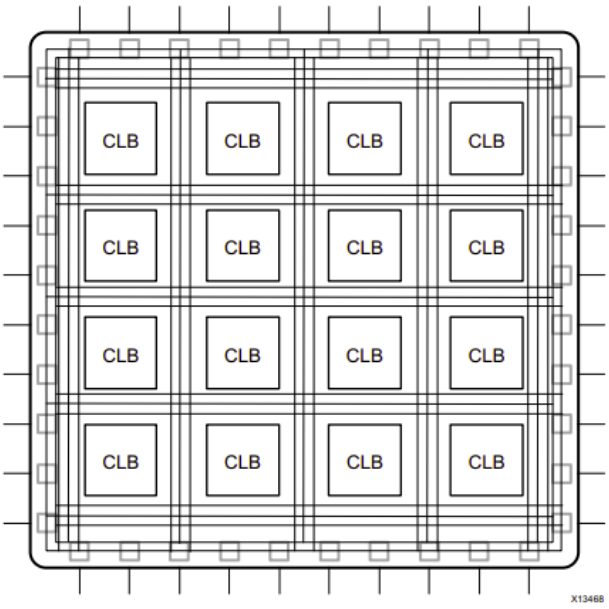
Επίσης, θα πρέπει να τονιστεί ότι τα FPGA διαδραματίζουν πρωταγωνιστικό ρόλο σε ταχέως αναπτυσσόμενες εφαρμογές και τεχνολογίες [10], όπως το edge computing, η τεχνητή νοημοσύνη (AI), η ασφάλεια συστημάτων μέσω κρυπτογράφησης και αποκρυπτογράφησης μεγάλων δεδομένων, το 5G, καθώς και σε βιομηχανικές εφαρμογές αυτοματισμού και σύγχρονης ρομποτικής. Η ικανότητά τους να προσαρμόζονται και να ανταποκρίνονται σε εξελισσόμενες απαιτήσεις, τα καθιστά απαραίτητα σε αυτούς τους τομείς, ενισχύοντας την αποδοτικότητα και την ασφάλεια των συστημάτων.

2.1.2 Αρχιτεκτονική FPGA

Η βασική δομή οποιουδήποτε FPGA δομείται από τα επακόλουθα στοιχεία-components:

- ✓ Look-up Table (LUT): Αυτό το συγκεκριμένο στοιχείο είναι υπεύθυνο για την εκτέλεση λογικών πράξεων.
- ✓ Flip-Flop (FF): Αυτό το στοιχείο καταχωρητή είναι υπεύθυνο για την αποθήκευση των αποτελεσμάτων των LUT.
- ✓ Καλώδια: Τα στοιχεία αυτά πραγματοποιούν τις συνδέσεις αναμεταξύ όλων των στοιχείων.
- ✓ Επιφάνειες εισόδου/εξόδου (I/O): Αυτές οι φυσικά διαθέσιμες θύρες έχουν ως ρόλο την εισαγωγή και την εξαγωγή δεδομένων από το FPGA.

Ο επιτυχής συνδυασμός των παραπάνω στοιχείων που αναφέραμε, έχει ως άμεσο αποτέλεσμα τη βασική αρχιτεκτονική του FPGA που σκιαγραφείται στην Εικόνα 2. Παρόλο που η παρακάτω δομή αποτελεί έναν επαρκή τρόπο υλοποίηση οποιουδήποτε αλγορίθμου όσο περίπλοκος και αν είναι αυτός, δεν μπορεί να θεωρηθεί ως η καλύτερη, μιας και σε θέματα αποδοτικότητας είναι αρκετά περιορισμένη από άποψη υπολογιστικής απόδοσης, απαιτούμενων πόρων και στοχευμένης συχνότητας ρολογιού [11].



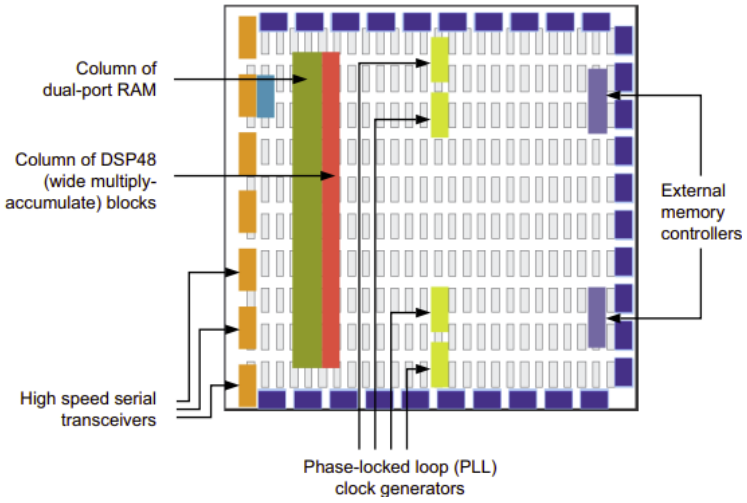
Εικόνα 2 Βασική αρχιτεκτονική FPGA[1]

[1] Η εικόνα προέρχεται από το https://docs.amd.com/v/u/en-US/ug998-vivado-intro-fpga-design-hls .

Στον αντίποδα, οι σύγχρονες και καλύτερες αρχιτεκτονικές FPGA ενσωματώνουν τα βασικά στοιχεία που αναφέραμε παραπάνω σε συνδυασμό με πρόσθετα υπολογιστικά μπλοκ καθώς και μπλοκ αποθήκευσης δεδομένων, που σε επόμενο στάδιο αυξάνουν την υπολογιστική ισχύ, πυκνότητα και αποδοτικότητας της πλακέτας. Τα επιπρόσθετα στοιχεία είναι τα παρακάτω:

- Ενσωματωμένες μνήμες που στοχεύουν στην κατανεμημένη αποθήκευση δεδομένων.
- Phase-locked Loops (PLL) για την οδήγηση του FPGA σε διαφορετικούς ρυθμούς ρολογιού.
- Σειριακοί πομποδέκτες που αγγίζουν υψηλές ταχύτητες.
- Ελεγκτές μνήμης εκτός του ίδιου του τσιπ.
- Μπλοκ πολλαπλασιασμού-συσσώρευσης.

Ο συνδυασμός αυτών των στοιχείων παρέχει στους προγραμματιστές των FPGA την ευελιξία να υλοποιούν οποιοδήποτε αλγόριθμο λογισμικού που μπορεί να εκτελεστεί σε έναν επεξεργαστή, οδηγώντας σε μια σύγχρονη FPGA αρχιτεκτονική [11], όπως φαίνεται στην Εικόνα 3.



Εικόνα 3 Σύγχρονη αρχιτεκτονική FPGA [2]

2.1.3 Διαφορές μεταξύ ASIC και FPGA

Θα πρέπει να υπογραμμίσουμε τις κύριες διαφορές ανάμεσα στα FPGA (Field-Programmable Gate Arrays) και τα ASIC (Application-Specific Integrated Circuits). Είναι σημαντικό να τονιστεί ότι αποτελούν διαφορετικές προτάσεις με μοναδική αξία, απαιτώντας σαφή και εκτενή αξιολόγηση του πεδίου εφαρμογής κάθε ενός. Παραδοσιακά, τα FPGA χρησιμοποιούνταν κυρίως για σχέδια χαμηλότερης ταχύτητας, πολυπλοκότητας και όγκου [12]. Ωστόσο, τα σύγχρονα FPGA, που αναπτύσσονται από κορυφαίες εταιρείες του κλάδου, ξεπερνούν συχνά το φράγμα απόδοσης των 500 MHz [12]. Η τεράστια αύξηση της λογικής πυκνότητας, καθώς και άλλων χαρακτηριστικών όπως οι ενσωματωμένοι επεξεργαστές και τα DSP μπλοκ, σε συνδυασμό με τους βελτιωμένους χρονισμούς και τις σειριακές συνδέσεις υψηλών ταχυτήτων, καθιστούν τα FPGA μια ιδανική επιλογή για την ανάπτυξη έξυπνων και σύγχρονων συστημάτων.

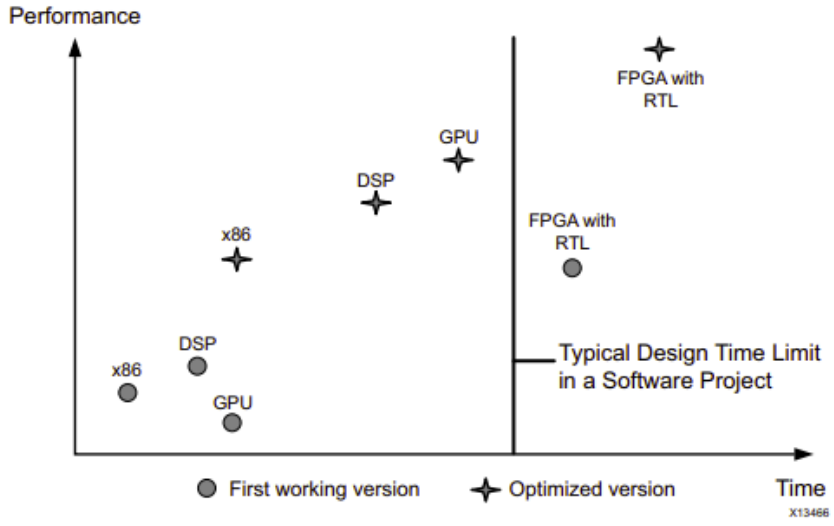
Σε αντίθεση με τα ASIC, τα FPGA προσφέρουν μεγαλύτερη ευελιξία στον προγραμματισμό και επαναπρογραμματισμό, επιτρέποντας προσαρμογές στις εξελισσόμενες απαιτήσεις των

[2] Η εικόνα προέρχεται από το https://docs.amd.com/v/u/en-US/ug998-vivado-intro-fpga-design-hls .

εφαρμογών. Επιπλέον, τα FPGA παρέχουν αυτή την ευελιξία σε χαμηλότερο κόστος, γεγονός που τα καθιστά προτιμητέα επιλογή έναντι των ASIC σε πολλές περιπτώσεις [12]. Συνολικά, η συνδυασμένη αύξηση των δυνατοτήτων και η μείωση του κόστους καθιστούν τα FPGA μονόδρομο στην ανάπτυξη προηγμένων και σύγχρονων συστημάτων σε διάφορους τομείς.

2.1.4 FPGA αντί για άλλους τύπους ολοκληρωμένων κυκλωμάτων

Το κύριο πλεονέκτημα των FPGA είναι η δομή προγραμματισμού τους, η οποία επιτρέπει στον προγραμματιστή-σχεδιαστή να προγραμματίζει και να επαναπρογραμματίζει τη συσκευή με ταχύτητα και ευκολία, προσαρμόζοντάς την σε διάφορες λειτουργίες και ανάγκες. Αυτή η δυνατότητα επαναπρογραμματισμού, που μπορεί να πραγματοποιηθεί μέσω ενημερώσεων λογισμικού ακόμη και μετά την ενσωμάτωση της συσκευής σε μια εφαρμογή, εξηγεί την ονομασία τους ως Field-Programmable Gate Arrays (FPGA). Η ευελιξία που προσφέρουν τα FPGA τα καθιστά εξαιρετικά πολύτιμα στην παγκόσμια αγορά, καθώς επιτρέπουν στους προγραμματιστές να εξοικονομούν χρόνο και να ανταποκρίνονται άμεσα στις εξελισσόμενες ανάγκες της αγοράς και των συνθηκών χρήσης. Η ικανότητα των FPGA να αναπτύσσονται παράλληλα με τις απαιτήσεις της αγοράς και τις ανάγκες εφαρμογών, τα εδραιώνει ως αναντικατάστατα εργαλεία στην ανάπτυξη σύγχρονων και προσαρμοστικών συστημάτων. Ένα επιπλέον πλεονέκτημα των FPGA είναι η δυνατότητα παράλληλης επεξεργασίας, που επιτυγχάνεται μέσω της αρχιτεκτονικής "Sea of Gates" [13]. Αυτή η αρχιτεκτονική επιτρέπει την ταυτόχρονη επεξεργασία δεδομένων, σε αντίθεση με τη διαδοχική εκτέλεση, αυξάνοντας σημαντικά την αποδοτικότητα. Η δυνατότητα αυτή καθιστά τα FPGA ιδιαίτερα κατάλληλα για υπολογιστικές εφαρμογές υψηλών επιδόσεων, όπως η τεχνητή νοημοσύνη (AI) και η μηχανική μάθηση, τομείς που έχουν γνωρίσει ραγδαία ανάπτυξη τα τελευταία χρόνια στον τομέα των προγραμματιζόμενων συσκευών. Η παράλληλη επεξεργασία επιτρέπει την επίτευξη υψηλότερων αποδόσεων με χαμηλότερες ταχύτητες ρολογιού, γεγονός που συνεπάγεται σημαντικά χαμηλότερη κατανάλωση ενέργειας [11]. Αυτό το χαρακτηριστικό καθιστά τα FPGA μια αποδοτική και βιώσιμη επιλογή για σύγχρονες εφαρμογές που απαιτούν υψηλή απόδοση και ενεργειακή αποδοτικότητα Εικόνα 4.



Εικόνα 4 Χρόνος σχεδιασμού vs. Χρόνος εφαρμογής με την είσοδο του RTL[3]

Διάφοροι άλλοι τύποι ολοκληρωμένων συστημάτων, όπως οι μικροελεγκτές (MCU), τα ολοκληρωμένα κυκλώματα ειδικών εφαρμογών (ASIC), οι μικροεπεξεργαστές (MPU) καθώς και τα τυποποιημένα προϊόντα ειδικών εφαρμογών (ASSP) έχουν κάθε φορά μια συγκεκριμένη-σταθερή

[3] Η εικόνα προέρχεται από το https://docs.amd.com/v/u/en-US/ug998-vivado-intro-fpga-design-hls .

εφαρμογή και λειτουργούν διαδοχικά, δηλαδή το ένα μετά το άλλο έχοντας πολλές φορές εξαρτήσεις από άλλα στοιχεία. Μάλιστα, σε αυτά τα συστήματα θα πρέπει να υπογραμμίσουμε ότι η αδυναμία έλλειψης προγραμματισμού κατά την εφαρμογή στο πεδίο χρήσης μπορεί να επιφέρει την μείωση της διάρκειας ζωής του συστήματος μας [14]. Επίσης, πρέπει να τονιστεί ότι η σειριακή επεξεργασία μπορεί να επιφέρει υψηλή κατανάλωση ενέργειας, εφόσον τα ολοκληρωμένα κυκλώματα κάνουν χρήση ρολογιών υψηλότερης ταχύτητας για να συμβαδίσουν με το φόρτο εργασίας που μπορεί να προκαλέσει η επεξεργασία των δεδομένων.

2.1.5 Προγραμματισμός και ρύθμιση ενός FPGA

Τα FPGA Εικόνα 5 απαιτούν κατάλληλη παραμετροποίηση, ώστε τα λογικά κυκλώματα και οι εσωτερικές διασυνδέσεις της συσκευής να λειτουργούν σωστά στην υλοποίηση συγκεκριμένων εφαρμογών. Για το λόγο αυτό, οι προγραμματιστές-σχεδιαστές χρησιμοποιούν εξειδικευμένο λογισμικό, το οποίο είναι προσαρμοσμένο από τον εκάστοτε προμηθευτή του FPGA. Αυτό το λογισμικό διευκολύνει τον σχεδιασμό και την κατασκευή της ψηφιακής λογικής που θα υλοποιηθεί εσωτερικά στο FPGA.

Η διαδικασία υλοποίησης βασίζεται κυρίως σε δύο μεθόδους:

1. **Γραφική σύλληψη του σχεδιασμού:** Αυτή η μέθοδος χρησιμοποιείται κυρίως σε μικρότερες πλακέτες FPGA και περιλαμβάνει τη χρήση γραφικών εργαλείων για την απεικόνιση και τον σχεδιασμό των λογικών κυκλωμάτων [15].

2. **Γλώσσα Περιγραφής Υλικού (Hardware Descriptive Language, HDL):** Αυτή η μέθοδος αξιοποιεί γλώσσες περιγραφής υλικού, όπως η VHDL ή η Verilog [16], για τη λεπτομερή περιγραφή της ψηφιακής λογικής. Η HDL επιτρέπει μεγαλύτερη ευελιξία και ακρίβεια στον σχεδιασμό και χρησιμοποιείται συχνά για πιο πολύπλοκα και μεγάλης κλίμακας έργα.

Περισσότερες λεπτομέρειες για τον τρόπο προγραμματισμού των FPGA θα παρουσιαστούν αναλυτικότερα σε επόμενη ενότητα.

Εικόνα 5 Παράδειγμα πλακέτας FPGA

Η διαδικασία προγραμματισμού των FPGA περιλαμβάνει διάφορα στάδια, τα οποία είναι απαραίτητα για τη σωστή υλοποίηση των σχεδιασμένων λογικών κυκλωμάτων και διασυνδέσεων. Ακολουθεί η περιγραφή των βασικών σταδίων αυτής της διαδικασίας:

Μεταγλώττιση και Σύνθεση

Σε πρώτο στάδιο, το εξειδικευμένο λογισμικό συνθέτει τη σχεδίαση μέσω της διαδικασίας της «μεταγλώττισης». Κατά τη διάρκεια αυτού του σταδίου, το λογισμικό εκτελεί τις εξής λειτουργίες:

1. **Σύνθεση (synthesis):** Μετατρέπει τον περιγραφικό κώδικα (είτε από γραφική σύλληψη, είτε από HDL) σε ένα ενδιάμεσο αναπαραστασιακό επίπεδο λογικών στοιχείων.

2. **Τοποθέτηση και Δρομολόγηση (place and route):** Ενσωματώνει τα λογικά στοιχεία στο συγκεκριμένο FPGA, καθορίζοντας τη φυσική θέση τους και τις διαδρομές των συνδέσεων μεταξύ τους [17].

Παραγωγή Bitstream

Μετά την επιτυχή σύνθεση και δρομολόγηση, το λογισμικό παράγει ένα αρχείο bitstream. Αυτό το αρχείο περιέχει όλες τις απαραίτητες πληροφορίες για τον προγραμματισμό του FPGA και τη διαμόρφωση των λογικών κυκλωμάτων.

Μεταφόρτωση και εκτέλεση

Το επόμενο βήμα περιλαμβάνει τη μεταφόρτωση του αρχείου bitstream στην πλακέτα του FPGA. Όταν το bitstream φορτωθεί επιτυχώς, η συσκευή καθίσταται λειτουργική και έτοιμη να εκτελέσει τις καθορισμένες εργασίες, όπως ορίστηκαν από τον προγραμματιστή κατά το στάδιο του προγραμματισμού.

Εργαλεία Ανάπτυξης

Οι δύο κύριες εταιρείες που δραστηριοποιούνται στον χώρο των FPGA, όπως η Xilinx (πλέον μέρος της AMD®), προσφέρουν ολοκληρωμένες σουίτες εργαλείων για τον σχεδιασμό και τη δημιουργία bitstream αρχείων [18]. Αυτές οι σουίτες περιλαμβάνουν εργαλεία για:

- **Σχεδιασμό κυκλωμάτων:** Παρέχουν γραφικά και κειμενικά εργαλεία για τη δημιουργία και την επαλήθευση λογικών κυκλωμάτων.

- **Σύνθεση και Δρομολόγηση:** Επιτρέπουν τη μετατροπή των σχεδίων σε λειτουργικά bitstream αρχεία.

- **Ανάπτυξη και Δοκιμές:** Περιλαμβάνουν προγράμματα για την ανάπτυξη κώδικα και την εκτέλεση διαφόρων δοκιμών στα σχεδιασμένα κυκλώματα.

Στην παρούσα διπλωματική εργασία, χρησιμοποιήθηκαν κυρίως τα προγράμματα που διανέμονται από την Xilinx-AMD®. Αυτά τα εργαλεία διευκόλυναν τον σχεδιασμό, την υλοποίηση και τον έλεγχο των λογικών κυκλωμάτων που αναπτύχθηκαν.

Στις επόμενες ενότητες, θα παρουσιαστούν αναλυτικά τα προγράμματα που χρησιμοποιήθηκαν και ο τρόπος με τον οποίο συνέβαλαν στην επιτυχία της διπλωματικής εργασίας.

2.1.6 Εφαρμογές FPGA

Η ευελιξία στον προγραμματισμό των FPGA τα καθιστά ιδανικές λύσεις για διάφορες αγορές και εφαρμογές. Η AMD®, ως ένας από τους ηγέτες στον τομέα, προσφέρει ολοκληρωμένες λύσεις FPGA που περιλαμβάνουν προηγμένο και σύγχρονο λογισμικό. Αυτό το λογισμικό είναι εξαιρετικά διαμορφώσιμο και συνοδεύεται από έτοιμους πυρήνες IP (Intellectual Property), οι οποίοι μπορούν να αγοραστούν και να ενσωματωθούν σε διάφορες εφαρμογές [19]:

- Αεροδιαστημική & Άμυνα: FPGAs με αντοχή στις ακτινοβολίες και τις θερμοκρασίες καθώς και σε δυσχερές κλιματικές συνθήκες αξιοποιούνται για την επεξεργασία εικόνας, για την παραγωγή και την μορφοποίηση κυματομορφών καθώς και για μερική αναδιαμόρφωση SDR.

- ASIC Prototyping: Το prototyping με FPGAs δίνει την δυνατότητα στους σχεδιαστές για ταχύτατη και ακριβής μοντελοποίηση συστημάτων SoC και μάλιστα επιτρέπει και την επαλήθευση του ενσωματωμένου λογισμικού

- Αυτοκινητοβιομηχανία: Αξιοποιούνται για συστήματα υποβοήθησης του οδηγού ενός οχήματος, καθώς και για την παροχή άνεσης, ευκολίας και ψυχαγωγίας μέσα στο όχημα.

- Καταναλωτικά ηλεκτρονικά: Εταιρίες όπως η Apple, η Google, η Samsung, η Huawei και πολλές άλλες έχουν ξεκινήσει τα τελευταία χρόνια την παροχή στο ευρύ καταναλωτικό κινητά με εφαρμογές AI, ψηφιακές επίπεδες οθόνες, οικιακή δικτύωση και παράλληλα αποκωδικοποιητές υψηλών ταχυτήτων.

- Κέντρα μεγάλων Δεδομένων: Χρησιμοποιούνται για διακομιστές υψηλού εύρους ζώνης με απώτερο σκοπό την χαμηλή καθυστέρηση δημιουργώντας ένα κλίμα ανάπτυξης για εφαρμογές δικτύωσης και εφαρμογές που αρμόζουν σε ένα σύγχρονο κέντρο δεδομένων.

- Ασφάλεια: Η AMD® και η Altera® προσφέρουν εξειδικευμένες λύσεις ασφαλείας τόσο για το καταναλωτικό κοινό όσο και για εταιρείες που δραστηριοποιούνται στον τομέα της ασφάλειας. Οι εφαρμογές αυτές καλύπτουν ένα ευρύ φάσμα, από συστήματα ελέγχου πρόσβασης μέχρι ολοκληρωμένα συστήματα ασφάλειας και επιτήρησης [19].

- Επεξεργασία βίντεο και εικόνας: Τα FPGA της AMD® σε συνδυασμό με τις στοχευμένες πλατφόρμες σχεδίασης, παρέχουν ένα υψηλό επίπεδο ευελιξίας και ταχύτερο χρόνο διάθεσης στην αγορά για εφαρμογές επεξεργασίας βίντεο και εικόνας [20]. Επιπλέον, συμβάλλουν στη μείωση του συνολικού μη επαναλαμβανόμενου κόστους μηχανικής, καθιστώντας τις ιδανικές για ένα ευρύ φάσμα εφαρμογών.

- Επεξεργασία εικόνας και βίντεο: Καινούργιες FPGA της AMD® και άλλων εταιριών με μια πληθώρα από εφαρμογές ειδικά σχεδιασμένες για τον τομέα αυτό επιτρέπουν υψηλότερο βαθμό ευελιξίας στο πεδίο καθώς και ταχύτερο χρόνο διάθεσης στην παγκόσμια αγορά με χαμηλότερο κόστος. Οι πλακέτες αυτές υποστηρίζουν εφαρμογές επεξεργασίας εικόνας και βίντεο με ειδικά διαμορφωμένα περιβάλλοντα που διανέμονται από την ίδια την AMD®. Η πλακέτα που πραγματοποιήθηκε η παρούσα διπλωματική έχει τέτοιες εφαρμογές και θα επανέλθουμε πάνω στο συγκεκριμένο θέμα σε επόμενο κεφάλαιο.

- Ενσύρματες και ασύρματες επικοινωνίες: Προσφέρονται λύσεις από άκρο σε άκρο για επαναπρογραμματιζόμενες κάρτες γραμμής δικτύωσης και επεξεργασίας πακέτων

(Framer/MAC, σειριακές backplanes) καθώς και λύσεις RF και μεταφοράς δικτύωσης για ασύρματο εξοπλισμό που απευθύνεται σε πρότυπα όπως HSDPA, WiMAX, WCDMA κτλ.

2.2 Εργαλεία ανάπτυξης και προγραμματισμού του συστήματος

2.2.1 Modelsim

Το Modelsim αποτελεί ένα πακέτο λογισμικού που αναπτύχθηκε από την Mentor Graphics με σκοπό την προσομοίωση HDL modules σε εξειδικευμένο περιβάλλον. Το πρόγραμμα αυτό υποστηρίζει προσομοίωση γλωσσών περιγραφής υλικού, όπως VHDL, Verilog και SystemC, προσφέροντας ένα ολοκληρωμένο σύνολο εργαλείων για την ανάπτυξη και την επαλήθευση ψηφιακών συστημάτων [17].

Λειτουργίες και Εργαλεία του Modelsim

1. **Επεξεργαστής Κειμένου:**
    - Ο επεξεργαστής κειμένου του Modelsim είναι ειδικά διαμορφωμένος για τις γλώσσες VHDL, Verilog και SystemC.
    - Παρέχει δυνατότητες όπως σύνταξη χρώματος (syntax highlighting), αυτόματη συμπλήρωση κώδικα και έλεγχο σφαλμάτων, διευκολύνοντας την ανάπτυξη κώδικα HDL.
2. **Πρόγραμμα Προβολής Κυματομορφών (Waveforms):**
    - Το πρόγραμμα προβολής κυματομορφών επιτρέπει την οπτικοποίηση των σημάτων και των κυματομορφών που παράγονται κατά τη διάρκεια της προσομοίωσης.
    - Παρέχει δυνατότητες όπως η μεγέθυνση, η μέτρηση χρονικών διαστημάτων και η παρακολούθηση συγκεκριμένων σημάτων, επιτρέποντας την εις βάθος ανάλυση της συμπεριφοράς του σχεδίου.
3. **Γεννήτρια RTL (Register Transfer Level):**
    - Η γεννήτρια RTL του Modelsim παράγει την ενδιάμεση αναπαράσταση του σχεδίου σε επίπεδο καταχωρητών.
    - Αυτή η αναπαράσταση χρησιμοποιείται για την επαλήθευση και την ανάλυση της λογικής του σχεδίου, διασφαλίζοντας τη σωστή λειτουργία του πριν την υλοποίηση σε υλικό.

Στην παρούσα διπλωματική εργασία χρησιμοποιήθηκε η έκδοση ModelSim - Intel FPGA Starter Edition Model Technology ModelSim - Intel FPGA Edition vsim 2020.1 (Quartus Prime 20.1), με φοιτητικές άδειες που κατέβηκαν και ενσωματώθηκαν από την επίσημη ιστοσελίδα διάθεσης του λογισμικού. Η κύρια χρήση της εφαρμογής ήταν η αποσφαλμάτωση και ο έλεγχος της λειτουργικότητας του κώδικα σε επίπεδο υλικού.

Σκοπός και Χρήση της Εφαρμογής

Η εφαρμογή ModelSim χρησιμοποιήθηκε για την επαλήθευση και την αποσφαλμάτωση του περιφερειακού συστήματος. Η διαδικασία αποσφαλμάτωσης περιλαμβάνει δύο κύρια στάδια:

1. **Αποσφαλμάτωση πριν τη Σύνθεση (Synthesis):**
    - Σε αυτό το στάδιο, γίνεται έλεγχος όλων των μονάδων HDL για την ορθή λειτουργία τους και την παραγωγή των αναμενόμενων αποτελεσμάτων.
    - Παρά το γεγονός ότι οι μονάδες μπορεί να λειτουργούν σωστά σε αυτό το στάδιο, δεν υπάρχει εγγύηση ότι θα λειτουργήσουν το ίδιο καλά μετά την υλοποίηση σε υλικό, λόγω πιθανών άλλων προβλημάτων.
2. **Αποσφαλμάτωση Μετά τη Σύνθεση (Post-Synthesis):**
    - Μετά τη σύνθεση του κώδικα, εμφανίζονται συχνά προβλήματα που απαιτούν περαιτέρω έλεγχο.
    - Η διαδικασία αυτή περιλαμβάνει τη χρήση του προβολέα κυματομορφών, όπου οι διάφορες ενότητες και οντότητες ελέγχονται για την ορθή λειτουργία τους.

Εργαλεία και Δυνατότητες του ModelSim

Το ModelSim περιλαμβάνει διάφορα ενσωματωμένα εργαλεία που διευκολύνουν τη διαδικασία αποσφαλμάτωσης:

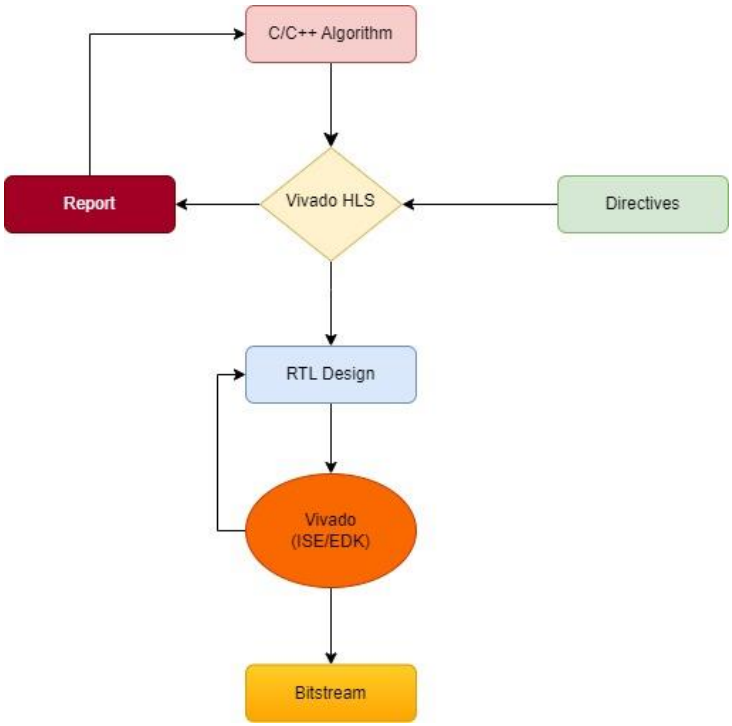
- **Προβολέας Κυματομορφών (Waveform Viewer):**
    - Αυτό το GUI εργαλείο επιτρέπει την παρακολούθηση όλων των σημάτων που χρησιμοποιούνται από ένα στοιχείο και την αντίστοιχη κυματομορφή τους.
    - Παρέχει τη δυνατότητα εισαγωγής δοκιμαστικών τιμών για την επιβεβαίωση της ορθής λειτουργίας του κώδικα.
- **Παρακολούθηση Σημάτων και Κυματομορφών:**
    - Ο χρήστης μπορεί να παρακολουθεί την εξέλιξη των τιμών των σημάτων με την πάροδο του χρόνου.
    - Διευκολύνει την εύκολη και γρήγορη αποσφαλμάτωση, επιτρέποντας τον εντοπισμό και τη διόρθωση λαθών που μπορεί να έχουν προκύψει σε οποιοδήποτε στάδιο της ανάπτυξης.

Η χρήση του ModelSim στην παρούσα διπλωματική εργασία παρείχε μια ξεκάθαρη εικόνα της συμπεριφοράς των λογικών κυκλωμάτων, επιτρέποντας τον εντοπισμό και τη διόρθωση πιθανών σφαλμάτων. Η εφαρμογή αυτή αποδείχθηκε απαραίτητη για την επιτυχή ανάπτυξη και υλοποίηση του περιφερειακού συστήματος.

2.2.2 Vitis HLS

Το Vivado HLS (High Level Synthesis) της Xilinx - AMD® είναι ένα εργαλείο σχεδίασης που επιτρέπει τη δημιουργία και υλοποίηση application-specific IPs χρησιμοποιώντας γλώσσες προγραμματισμού υψηλού επιπέδου, όπως η C και η C++. Παρέχει στους προγραμματιστές τη δυνατότητα να εργάζονται σε υψηλό επίπεδο αφαίρεσης χωρίς να θυσιάζουν την απόδοση των υλοποιήσεών τους, διευκολύνοντας την ανάπτυξη πολύπλοκων συστημάτων και αλγορίθμων σε FPGA. Με χαρακτηριστικά όπως φιλική διεπαφή χρήστη, δυνατότητες αυτόματης βελτιστοποίησης και υποστήριξη για ενσωμάτωση επιταχυντών υλικού, το Vivado HLS επιτρέπει τη γρήγορη και αποδοτική ανάπτυξη προηγμένων εφαρμογών σε ενσωματωμένα συστήματα, υπολογιστικές εφαρμογές και βιομηχανικούς τομείς [21].

Ο σχεδιασμός ενός component με τη χρήση HLS παρουσιάζεται στο ακόλουθο σχηματικό Εικόνα 6. Συγκεκριμένα, ο σχεδιαστής-προγραμματιστής πρέπει αρχικά να δημιουργήσει τον πηγαίο κώδικα σε γλώσσα υψηλού επιπέδου, όπως C ή C++. Το Vitis επιτρέπει στον προγραμματιστή να αξιολογήσει γρήγορα και με ακρίβεια τη λειτουργικότητα και την ορθότητα του αλγορίθμου, πραγματοποιώντας διάφορα δοκιμαστικά testbenches σε οποιοδήποτε στάδιο του σχεδιασμού. Η ενσωμάτωση του testbench είναι ιδιαίτερης σημασίας, καθώς επιτρέπει τον έλεγχο της ορθότητας του συστήματος και την εφαρμογή τεχνικών επιτάχυνσης της εκτέλεσης του κώδικα, δηλαδή των διαδικασιών σχεδίασης και υλοποίησης. Έτσι, η αξιολόγηση της λειτουργικότητας ενός συστήματος σχεδιασμένου σε γλώσσα υψηλού επιπέδου γίνεται πιο εύκολα και απλά, σε σύγκριση με μια υλοποίηση βασισμένη σε γλώσσα περιγραφής υλικού, η οποία απαιτεί εξωτερικά προγράμματα και εργαλεία προσομοίωσης σε πολύ χαμηλό επίπεδο, όπως το Modelsim που αναφέρθηκε προηγουμένως.



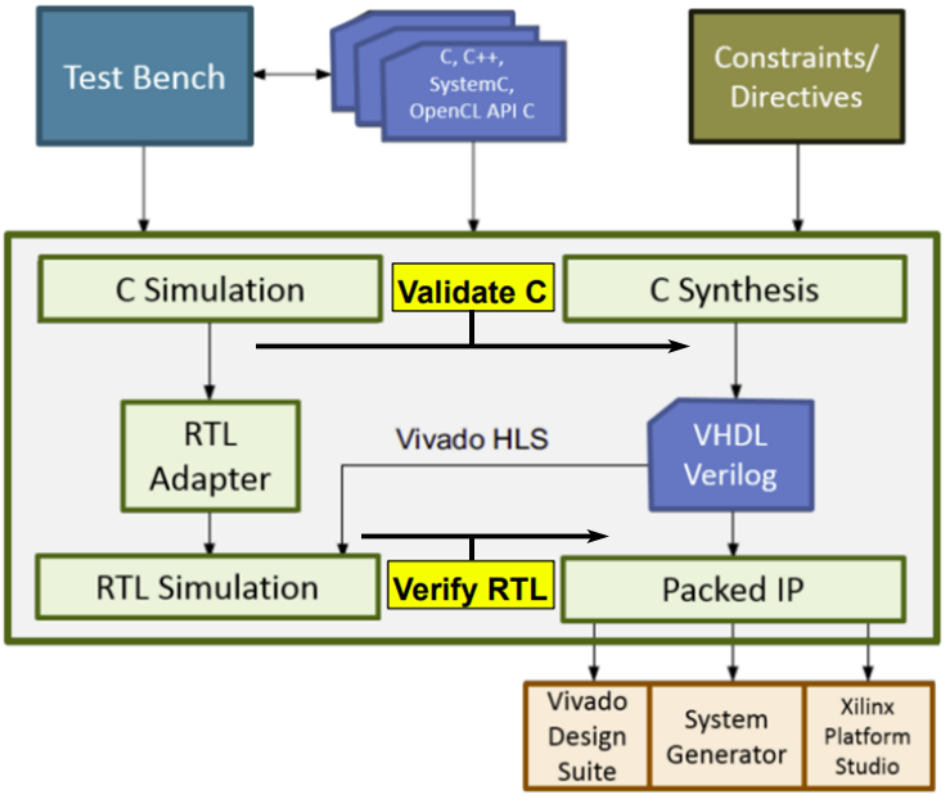
Εικόνα 6 Διάγραμμα ροής Vitis

Μετά την αρχική υλοποίηση του κώδικα, ακολουθεί η αυτόματη δημιουργία του αλγορίθμου σε επίπεδο RTL (Register Transfer Level), μια διαδικασία που αναφέρεται ως σύνθεση (synthesis) στο γραφικό περιβάλλον του Vivado HLS. Κατά τη σύνθεση RTL, πραγματοποιούνται έλεγχοι μέσω ενός συνόλου εντολών που ονομάζονται pragmas. Αυτές οι εντολές μπορεί να ορίζονται είτε από τον

σχεδιαστή-προγραμματιστή στον πηγαίο κώδικα C ή C++ με τη μορφή μεταγλωττιστή είτε σε εξωτερικό αρχείο που περιλαμβάνεται στο έργο [21].

Το HLS εφαρμόζει ορισμένες προεπιλεγμένες βελτιστοποιήσεις αυτόματα. Ωστόσο, για να επιτευχθεί η βέλτιστη απόδοση και η πλήρης αξιοποίηση του συστήματος, οι προγραμματιστές πρέπει να χρησιμοποιούν οδηγίες pragma. Η βελτιστοποίηση του τελικού προϊόντος είναι μια επαναλαμβανόμενη και απαιτητική διαδικασία δοκιμών, η οποία συχνά απαιτεί αναδιάρθρωση του κώδικα σε συνδυασμό με τη χρήση εντολών βελτιστοποίησης και επιτάχυνσης.

Στο τελικό στάδιο, δημιουργείται ένα κύκλωμα από λογικές πύλες με βάση το RTL, το οποίο στη συνέχεια παράγει το τελικό αρχείο bitstream. Το bitstream προορίζεται για τον προγραμματισμό μιας συγκεκριμένης προγραμματιζόμενης πλακέτας FPGA, όπως έχει καθοριστεί από τον προγραμματιστή κατά τη διαδικασία σχεδιασμού Εικόνα 7. Αξίζει να σημειωθεί ότι ο προγραμματιστής μπορεί να «συσκευάσει» το σχέδιό του σε διάφορες μορφές IP για μελλοντική χρήση, καθιστώντας τη διαδικασία επαναχρησιμοποιήσιμη και προσαρμόσιμη σε διαφορετικές εφαρμογές και έργα.



Εικόνα 7 Workflow Vivado HLS [4]

2.2.3 Vivado HLS

Ο μεταγλωττιστής υποστηρίζει ένα προγραμματιστικό περιβάλλον για ανάπτυξη εφαρμογών τόσο σε τυπικούς επεξεργαστές (open-source), όσο και σε πιο εξειδικευμένους επεξεργαστές που υποστηρίζονται από τις πλακέτες της εταιρίας AMD®. Πιο αναλυτικά, το πρόγραμμα Vivado HLS βασίζεται σε στοιχειώδεις τεχνολογίες άλλων μεταγλωττιστών επεξεργαστών για την ερμηνεία, την ανάλυση και την βελτιστοποίηση των πηγαίων κωδίκων προγραμμάτων σε C/C++. Σε αυτό το σημείο βέβαια θα πρέπει να τονιστεί ότι η κύρια διαφορά αναμεταξύ τους είναι ότι ο HLS

[4] Η εικόνα προέρχεται από το https://docs.amd.com/r/en-US/ug1399-vitis-hls .

δημιουργεί HDL, ενώ οι άλλοι compilers κώδικα μηχανής (machine code). Στοχεύοντας σε ένα συγκεκριμένο FPGA, ο μηχανικός λογισμικού δύναται να βελτιώσει και να βελτιστοποιήσει τον πηγαίο κώδικα, επιφέροντας καλύτερα αποτελέσματα στο σύστημα τόσο σε ισχύ και διάρκεια όσο και σε μειωμένη καθυστέρηση χωρίς να αντιμετωπίζει την συμφόρηση των επιδόσεων ενός ενιαίου χώρου μνήμης και να περιορίζεται από υπολογιστικούς πόρους, εφόσον του παρέχεται η δυνατότητα να δημιουργήσει την δική του αυτοτελή υλοποίηση οικοδομώντας τα δικά του όρια και περιορισμούς.

Όπως προαναφέραμε ο κώδικας εφαρμογών που είναι βασισμένος ο μεταγλωττιστής Vivado HLS εμπίπτει με τις ίδιες κατηγορίες με οποιονδήποτε άλλων μεταγλωττιστή επεξεργαστών που υπάρχει στην αγορά σήμερα [22]. Πιο συγκεκριμένα, πρέπει να αναφέρουμε, ότι η ανάλυση που διεκπεραιώνει ο Vivado HLS σε όλα τους πηγαίους κώδικες είναι από την άποψη:

- Λειτουργιών.
- Δηλώσεων υπό όρους.
- Βρόχων.
- και συναρτήσεων.

2.2.4 PYNQ

Το PYNQ™ αποτελεί ένα έργο ανοιχτού κώδικα [23] από την ευρέως γνωστή AMD® που διευκολύνει τη χρήση και την διαχείριση πλατφορμών στο πεδίο του Adaptive Computing. Ο όρος αναφέρεται στη δυνατότητα των υπολογιστικών συστημάτων να προσαρμόζουν τη λειτουργία και τη συμπεριφορά τους δυναμικά, ώστε να ανταποκρίνονται στις μεταβαλλόμενες συνθήκες και απαιτήσεις του περιβάλλοντος ή της εφαρμογής που εξυπηρετούν.

Το έργο αξιοποιεί την γλώσσα Python, τα σημειωματάρια Jupyter (Jupyter Notebooks) καθώς και τεράστιο οικοσύστημα με το μεγάλο εύρος των βιβλιοθηκών της Python. Οι σχεδιαστές-προγραμματιστές λογισμικού δύναται να αξιοποιήσουν τις δυνατότητες της προγραμματιστικής λογικής και των μικροεπεξεργαστών, για δημιουργία σύγχρονων ηλεκτρονικών συστημάτων που βρίσκουν εφαρμογή στις προγραμματιζόμενες πλακέτες που έχουν έρθει στο χώρο των προγραμματιζόμενων ολοκληρωμένων.

Το PYNQ μπορεί να αξιοποιηθεί για τη οικοδόμηση εφαρμογών υψηλών επιδόσεων με:

- παράλληλη εκτέλεση υλικού.
- επεξεργασία βίντεο που εμπεριέχει υψηλό ρυθμό καρέ.
- επιτάχυνση αλγορίθμων υλικού.
- επεξεργασία σήματος σε πραγματικό χρόνο.
- υψηλό εύρος ζώνης IO.

2.2.5 Κοινό που απευθύνεται το λογισμικό PYNQ

Το ανοιχτό λογισμικό PYNQ απευθύνεται σε προγραμματιστών και σχεδιαστών προγραμματιζόμενων και ενσωματωμένων συστημάτων, όπως για παράδειγμα:

- Προγραμματιστές λογισμικού που θέλουν να αξιοποιήσουν και να εντρυφήσουν στις δυνατότητες των πλατφόρμων Adaptive Computing χωρίς να έχουν οποιαδήποτε ανάγκη για εργαλεία σχεδίασης τύπου ASIC για τον σχεδιασμό του υλικού.

- Αρχιτέκτονες συστημάτων που αναζητούν μια εύκολη διεπαφή που βασίζεται σε λογισμικό και ένα περιβάλλον ταχείας δημιουργίας πρωτοτύπων και ανάπτυξης σε πλακέτες προγραμματιζόμενων Zynq, Alveo και AWS-F1.

- Τέλος, σε σχεδιαστές και αρχιτέκτονες υλικού που επιθυμούν να υλοποιήσουν εφαρμογές που έχουν προσαρμοστικότητα και απήχηση στο ευρύ καταναλωτικό κοινό.

2.2.6 Υποστηριζόμενες συσκευές και πλακέτες AMD®

Το PYNQ μπορεί να αξιοποιηθεί συγκεκριμένα με τις οικογένειες των FPGA Zynq™, Zynq UltraScale+™, Kria™, Zynq RFSoC, Alveo™ accelerator boards και AWS-F1 Εικόνα 8.

Εικόνα 8 Υποστηριζόμενες πλακέτες

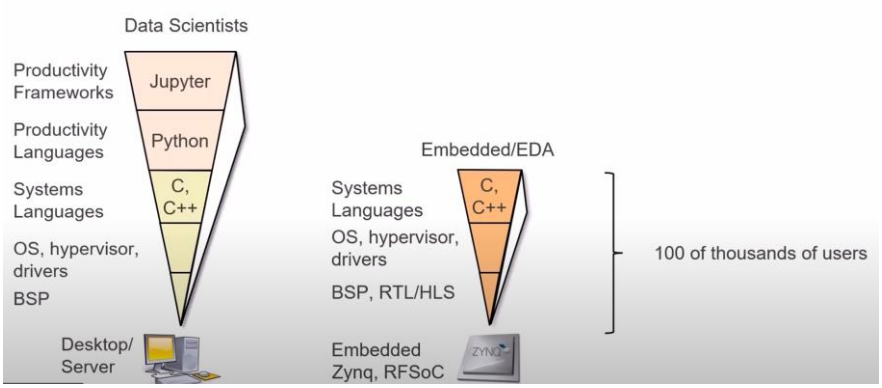
Το λογισμικό PYNQ μπορεί χρησιμοποιηθεί με δύο τρόπους: ως είδωλο Linux που γράφεται σε μια SD κάρτα με δυνατότητα εκκίνησης για μια πλακέτα της οικογένειας Zynq από το επίσημο αποθετήριο [24] , η οποία περιλαμβάνει το πακέτο pynq Python και άλλα πακέτα ανοικτού κώδικα, ή ως πακέτο Python για την Kria ή για έναν κεντρικό υπολογιστή Alveo ή AWS-F1. Περισσότερες πληροφορίες για το πακέτο, την εγκατάσταση και τις δυνατότητες του θα δούμε σε επόμενη ενότητα σε σχέση με την πλακέτα Kria.

2.2.7 Βασικές τεχνολογίες

Το Jupyter Notebook αποτελεί ένα διαδραστικό υπολογιστικό περιβάλλον με γραφικά προγραμμασμού βασισμένο στα προγράμματα περιήγησης. Παρέχει δυνατότητες για δημιουργία και συγγραφή εγγράφων σημειωματάριου Jupyter που δομούνται από ζωντανό πηγαίο κώδικα, διαδραστικά widgets, διαγράμματα, μαθηματικές εξισώσεις, επεξεργασμένο κείμενο, εικόνες και τέλος βίντεο Εικόνα 9. Ένα προγραμματιζόμενο σύστημα μπορεί εύκολα να προγραμματιστεί με την δυνατότητα της PYNQ στο Jupyter Notebook με την χρήση της γλώσσας προγραμματισμού Python. Κάνοντας χρήσης αυτού του υψηλού επιπέδου γλώσσας προγραμματισμού, οι προγραμματιστές λογισμικού και υλικού έχουν την δυνατότητα να βιβλιοθήκες υλικού και επικαλύψεις στην προγραμματιζόμενη λογική. Οι βιβλιοθήκες υλικού ή οι επικαλύψεις μπορούν

να επιταχύνουν το λογισμικό που εκτελείται σε μια πλακέτα Zynq ή Alveo και να προσαρμόσουν την πλατφόρμα υλικού και τις διεπαφές Εικόνα 10.

Εικόνα 9 Τεχνολογίες που αξιοποιήθηκαν



Εικόνα 10 Τυπικές χρήσεις του λογισμικού PYNQ[5]

2.2.8 Απαραίτητο λογισμικό

Το Jupyter Notebook μπορεί να εκτελεστεί σε σχεδόν οποιοδήποτε πρόγραμμα περιήγησης στο διαδίκτυο, παρέχοντας ένα ευέλικτο περιβάλλον για την ανάπτυξη κώδικα Εικόνα 11. Για τον προγραμματισμό του PYNQ με Python, απαιτείται μόνο ένα συμβατό πρόγραμμα περιήγησης. Για την επίτευξη υψηλότερων επιδόσεων, είναι δυνατή η χρήση γλωσσών προγραμματισμού όπως η C και η C++ σε συνδυασμό με την Python και το PYNQ. Το περιβάλλον ανάπτυξης λογισμικού AMD® Vitis είναι διαθέσιμο δωρεάν και παρέχει μια ολοκληρωμένη πλατφόρμα για τη δημιουργία και την ανάπτυξη εφαρμογών. Παράλληλα, υπάρχει δυνατότητα χρήσης εργαλείων ανάπτυξης λογισμικού από τρίτους κατασκευαστές, προσφέροντας ευελιξία και ευρεία υποστήριξη. Τέλος, νέες βιβλιοθήκες υλικού και επικαλύψεις μπορούν να δημιουργηθούν με τη χρήση τυποποιημένων εργαλείων σχεδιασμού υλικού της AMD®, καθώς και από τρίτους κατασκευαστές. Αυτό επιτρέπει την προσαρμογή και τη βελτιστοποίηση των εφαρμογών για συγκεκριμένες απαιτήσεις, μεγιστοποιώντας την απόδοση και την αποδοτικότητα των συστημάτων.

[5] Η εικόνα προέρχεται από το https://www.pynq.io/.

Εικόνα 11 Υποστηριζόμενοι διακομιστές πλοήγησης

2.2.9 Σύνοψη Δεύτερου Κεφαλαίου

Αρχικά, αναλύθηκε η βασική δομή και η λειτουργικότητα των FPGAs (Field-Programmable Gate Arrays), καθώς και η χρήση τους σε ποικίλες εφαρμογές που βρίσκουν εφαρμογή στην καθημερινότητά μας. Τα FPGAs είναι ιδιαίτερα ευέλικτα και επαναπρογραμματιζόμενα κυκλώματα που επιτρέπουν τη δυναμική αλλαγή της λειτουργικότητάς τους μετά την κατασκευή τους. Αυτό τα καθιστά ιδανικά για χρήση σε τομείς όπως τα δίκτυα επικοινωνίας, η επεξεργασία σήματος και οι ενσωματωμένες συστήματα. Επιπλέον, αναφέρθηκαν οι κύριες διαφορές μεταξύ FPGAs και των ολοκληρωμένων κυκλωμάτων ειδικών εφαρμογών (ASICs - Application-Specific Integrated Circuits). Συγκεκριμένα, τα FPGAs προσφέρουν μεγαλύτερη ευελιξία και ταχύτητα ανάπτυξης σε σχέση με τα ASICs, ενώ τα τελευταία προσφέρουν βελτιστοποιημένη απόδοση και χαμηλότερη κατανάλωση ενέργειας για συγκεκριμένες εφαρμογές. Η επιλογή μεταξύ των δύο εξαρτάται από τις συγκεκριμένες ανάγκες του έργου, το διαθέσιμο προϋπολογισμό, το χρόνο ανάπτυξης και τις απαιτήσεις απόδοσης. Τέλος, στην παρούσα διπλωματική εργασία, χρησιμοποιήθηκαν διάφορα προγράμματα και εργαλεία για την ανάπτυξη και αξιολόγηση των FPGAs. Αναλύθηκε η χρήση εργαλείων όπως τα Xilinx Vivado, το Vivado HLS, του Modelsim καθώς και του λογισμικού PYNQ.

Κεφάλαιο 3: Ανάλυση του αλγορίθμου

Σε αυτό το κεφάλαιο, θα εμβαθύνουμε στο θέμα της κρυπτογράφησης στην εποχή μας, παρουσιάζοντας μια οικογένεια αλγορίθμων, και εστιάζοντας στην παρουσίαση του αλγορίθμου κρυπτογράφησης που θα υλοποιηθεί από το περιφερειακό μας. Περιλαμβάνονται τα βήματα κατασκευής με αναλυτική περιγραφή, τα αποτελέσματα της κατασκευής, καθώς και ορισμένα τεστ για την αξιολόγηση της ορθής λειτουργίας του αλγορίθμου που βασίζεται το περιφερειακό.

3.1 Κρυπτογραφία

Ως κρυπτογραφία (cryptography) ορίζεται η εξερεύνηση τεχνικών που είναι βασισμένες σε δύσκολα επιλύσιμα μαθηματικά προβλήματα, με απώτερο σκοπό την διασφάλιση της ακεραιότητας και φερεγγυότητας των πληροφοριών. Αναλυτικότερα, πρέπει να υπογραμμίσουμε ότι η εφαρμογή της κρυπτογραφίας ονομάζεται κρυπτογράφηση. Κρυπτογράφηση, είναι ο μετασχηματισμός των δεδομένων, οτιδήποτε στοιχείων μπορεί να δώσει ο χρήστης, σε μια μορφή που είναι αδύνατον να αποκωδικοποιηθεί και να «σπάσει» χωρίς τη σωστή ακολουθία bit [25]. Πιο συγκεκριμένα, η ακολουθία αυτή καλείται “κλειδί” και η ύπαρξη της είναι στενά συνδεδεμένη με κάποιον κατάλληλο αλγόριθμο ή συνάρτηση κρυπτογράφησης. Υπάρχουν δύο κύριοι τύποι κρυπτογραφίας με κλειδί: η συμμετρική και η ασύμμετρη κρυπτογραφία. Η συμμετρική κρυπτογραφία, γνωστή και ως κρυπτογραφία με κοινό κλειδί (secret-key cryptography), χρησιμοποιεί το ίδιο κλειδί τόσο για την κρυπτογράφηση όσο και για την αποκρυπτογράφηση των δεδομένων. Παράλληλα, η ασύμμετρη κρυπτογραφία, γνωστή και ως κρυπτογραφία με δημόσιο κλειδί (public-key cryptography), χρησιμοποιεί δύο διαφορετικά αλλά σχετιζόμενα κλειδιά: ένα δημόσιο κλειδί για την κρυπτογράφηση των δεδομένων και ένα ιδιωτικό κλειδί για την αποκρυπτογράφηση. Έτσι, η αντίστροφη διαδικασία ορίζεται ως αποκρυπτογράφηση και έχει ως βασική προϋπόθεση την γνώση του κλειδιού. Στόχος της κρυπτογράφησης είναι η απόκρυψη των δεδομένων μας από όσους έχουν μη εξουσιοδοτημένη πρόσβαση σε αυτά. Επιπλέον, υπάρχουν και είδη κρυπτογραφίας που δεν χρησιμοποιούν κλειδιά, όπως ο αλγόριθμος SHA-256, ο οποίος είναι μια συνάρτηση κατακερματισμού (hash function) που μετασχηματίζει τα δεδομένα σε μια σταθερού μεγέθους ακολουθία χαρακτήρων, καθιστώντας την αρχική μορφή των δεδομένων μη αναγνώσιμη και μη αναστρέψιμη.

Όπως αναφέραμε και παραπάνω η κρυπτογράφηση καθώς και η αποκρυπτογράφηση σε ορισμένες περιπτώσεις έχουν την απαίτηση του κλειδιού για να επιτευχθεί η διαδικασία. Θα πρέπει να φανταστούμε το κλειδί αυτό σαν μυστικό κωδικό που μας παρέχει πρόσβαση ή την δυνατότητα παραλλαγής των δεδομένων μας. Σε αρκετές περιπτώσεις μηχανισμών γίνεται η χρήση του ίδιου κλειδιού και για τις δύο διαδικασίες, ενώ σε άλλες διαφέρουν αρκετά. Η χρήση της κρυπτογραφίας σήμερα επικεντρώνεται στην δημιουργία μηχανισμών ασφαλείας, όπως είναι για παράδειγμα η

ψηφιακή υπογραφή, η οποία συνδέεται άρρηκτα έναν χρήστη στον ψηφιακό χώρο, με ένα ή πολλά έγγραφα, προσδίδοντας του ένα μοναδικό ψηφιακό κλειδί και καθιστώντας, τον κάτοχό του εγγράφου [25]. Παράλληλα, υπάρχει και η ψηφιακή χρονοσφραγίδα (digital timestamp) που διασυνδέει ένα ψηφιακό έγγραφο με την στιγμή και ώρα που έγινε η δημιουργία του. Τέτοια συστήματα ασφαλείας και διασφάλισης ποιότητας, χρησιμοποιούνται στο ψηφιακό εμπόριο και τις ψηφιακές συναλλαγές στο διαδίκτυο του σήμερα.

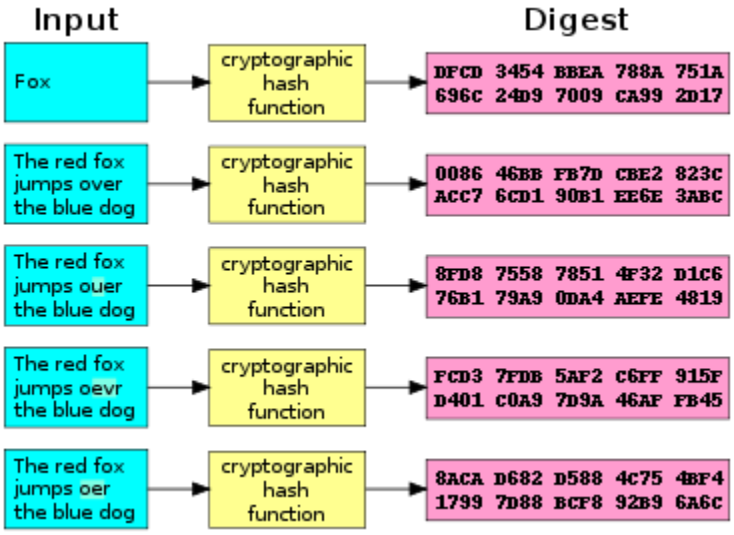
3.1.1 Συναρτήσεις κρυπτογράφησης κατακερματισμού

Ως συνάρτηση κρυπτογράφησης κατακερματισμού (Cryptographic Hash Function – CHF) ορίζεται ένας αλγόριθμος hashing που παρέχει ειδικές και εξειδικευμένες ιδιότητες και χαρακτηριστικά για μια κρυπτογραφική εφαρμογή ακολουθιών *n* bits. Αναλυτικότερα, κάθε συνάρτηση κρυπτογράφησης διέπεται από:

- Την πιθανότητα μιας συγκεκριμένης *n*-bit αποτελέσματος εξόδου (hashed τιμή - Digest) για μια τυχαία συμβολοσειρά εισόδου που από εδώ και παρακάτω θα το αποκαλούμε «μήνυμα» [26].

- Την δυσκολία εύρεσης μια συμβολοσειράς εξόδου που να ταιριάζει με την συμβολοσειρά εισόδου που έχει δεχτεί την κρυπτογράφηση Εικόνα 12. Οι συναρτήσεις κρυπτογράφησης χαρακτηρίζονται από τον όρο της αντοχής στην διασφάλιση της ασφάλειας του μηνύματος, για αυτό μπορούμε να πούμε ότι είναι και μονόδρομες διαδικασίες σε αρκετές περιπτώσεις.

- Τον γενικό κανόνα ότι δεν υπάρχει καμία πιθανότητα να βρούμε ένα ζεύγος τιμών που να δίνουν την ίδια τιμή hashing. Το γεγονός αυτό ενισχύει ότι δεν υπάρχον συγκρούσεις ανάμεσα στα αποτελέσματα.



Εικόνα 12 Παράδειγμα κρυπτογράφησης[6]

Οι κρυπτογραφικές συναρτήσεις κατακερματισμού, όπως αναφέραμε στην προηγούμενη υποενότητα, έχουν εφαρμογή στην ασφάλεια πληροφοριών για την προστασία των προσωπικών δεδομένων Εικόνα 13 και ιδιαίτερα σε: ψηφιακές υπογραφές, κώδικες αυθεντικοποίησης μηνυμάτων τύπου MAC και άλλες διάφορες μορφές επιβεβαίωσης χρήστη ή πρόσβασης

[6] Η εικόνα προέρχεται από το https://en.wikipedia.org/wiki/SHA-2 .

πληροφοριών [25]. Επιπροσθέτως, μπορούν να χρησιμοποιηθούν για την δημιουργία ευρετηρίου δεδομένων σε πίνακες, για ανίχνευση διπλότυπων δεδομένων, για την πρόσβαση δακτυλικών ψηφιακών αποτυπωμάτων, αλλά και για την αναγνώριση μοναδικών αρχείων παραγωγής κώδικα. Για αυτό ακριβώς το λόγο σε περιβάλλοντα ασφαλείας, οι κρυπτογραφημένες τιμές καλούνται και ως αποτυπώματα (fingerprints) ή αθροίσματα ελέγχου.

Εικόνα 13 Παράδειγμα κρυπτογράφησης για είσοδο κωδικών χρήστη σε servers

3.2 SHA-2

3.2.1 Οικογένεια SHA-2

Ο SHA-2 (Secure Hash Algorithm 2) είναι ένα σύνολο κρυπτογραφικών συναρτήσεων κατακερματισμού που σχεδιάστηκε από την Εθνική Υπηρεσία Ασφαλείας (NSA) των Ηνωμένων Πολιτειών και δημοσιεύθηκε για πρώτη φορά το 2001 [27].

Η οικογένεια αλγορίθμων SHA-2 εμπεριέχει σημαντικές αλλαγές και βελτιώσεις σε σχέση με τον προκάτοχό του, SHA-1. Η οικογένεια αλγορίθμων SHA-2 αποτελείται από έξι συναρτήσεις hashing που είναι για τιμές 224, 256, 384 ή 512 bits: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256. Πιο συγκεκριμένα, οι αλγόριθμοι SHA-256 και SHA-512 είναι νέες συναρτήσεις hashing που υπολογίζονται με οκτώ λέξεις των 32 και 64 bit αντίστοιχα. Χρησιμοποιούνται διαφορετικές σταθερές και μήκη μετατοπίσεων που θα δούμε σε επόμενη ενότητα, ωστόσο η κύρια δομή τους είναι πανομοιότυπη, με κύρια διαφορά τον αριθμό των επαναλήψεων για την εξαγωγή του τελικού αποτελέσματος.

Οι SHA-224 και SHA-384 είναι περικομμένες εκδόσεις των SHA-256 και SHA-512 αντίστοιχα, που υπολογίζονται με διαφορετικές αρχικές τιμές. Τα SHA-512/224 και SHA-512/256 είναι επίσης περικομμένες εκδόσεις του SHA-512, αλλά οι αρχικές τιμές παράγονται με τη μέθοδο που περιγράφεται στο Federal Information Processing Standards (FIPS) PUB 180-4 [27].

3.2.2 Εφαρμογές του προτύπου κατακερματισμού SHA-256

Οι συναρτήσεις hashing της οικογένειας SHA-2 υλοποιούνται σε ορισμένες αρκετά γνωστές και χρήσιμες εφαρμογές και πρωτόκολλα ασφαλείας, όπως τα SSL, PGP, SSH, TLS κτλ. Ωστόσο λόγω της υπολογιστικής πολυπλοκότητας των αλγορίθμων χαρακτηριστικό που πηγάζει από της πολλαπλές υπολογιστικές πράξεις, σήμερα έχουμε οδηγηθεί στην αναζήτηση λύσεων βασιζόμενες σε επιταχυντές υλικού για εφαρμογές ολοκληρωμένων κυκλωμάτων (ASICs ή FPGAs).

Αναλυτικότερα, θα περιγράψουμε κάποια παραδείγματα χρήσης ορισμένων αλγορίθμων με κύριο επίκεντρο τον SHA-256 που αποτελεί και την υλοποίηση του περιφερειακού στην παρούσα διπλωματική εργασία. Συγκεκριμένα, ο αλγόριθμος SHA-256 αποτελεί βασικό πυλώνα στην

αυθεντικοποίηση πακέτων λογισμικού Debian και παράλληλα σε συνδυασμό με τον SHA-512 προτείνεται για χρήση στο DNSSEC (DNS Security) [28]. Επίσης, βρίσκει εφαρμογή και σε λειτουργικά συστήματα όπως τα Unix και τα Linux που προωθούν την χρήση του για ασφαλή κατακερματισμό κωδικών πρόσβασης. Επιπροσθέτως, με την εξέλιξη του ηλεκτρονικού χρήματος σήμερα, πολλαπλά κρυπτονομίσματα που μέσα σε αυτά συμπεριλαμβάνεται και το γνωστό Bitcoin εκμεταλλεύονται τον αλγόριθμο του SHA-256 για την επαλήθευση συναλλαγών και τον υπολογισμό των αποδείξεων εργασίας.

3.2.3 Σύντομη ιστορική αναδρομή του SHA-256

Θα πρέπει να υπογραμμίσουμε ότι οι SHA-1 και οι SHA-2 είναι αλγόριθμοι hashing που έχουν γίνει βασική απαίτηση βάση νόμου στην κυβέρνηση των Ηνωμένων Πολιτειών Αμερικής, για ορισμένες εφαρμογές, οι οποίες απαιτούν την προστασία των ευαίσθητων δεδομένων από κακόβουλους χρήστες. Το παλιό πρωτόκολλο FIPS PUB 180-1 είχε καθιερώσει και την χρήση της προηγούμενης γενιάς αλγορίθμων SHA-1 από ιδιώτες και εμπόρους οποιονδήποτε οργανισμών, που αναζητούσαν ένα ασφαλές πρότυπο για την διασφάλιση των πληροφοριών και των δεδομένων τους [29].

Πίνακας 1 SHA-1 vs SHA-256

SHA-1 vs SHA-256	
Διάγραμμα σύγκρισης	
SHA-1	**SHA-256**
Ο SHA-1 είναι μία από τις πιο ευρέως χρησιμοποιούμενες και διαδεδομένες κρυπτογραφικές συναρτήσεις κατακερματισμού.	Ο SHA-256 είναι μια νεότερη, ασφαλέστερη κρυπτογραφική συνάρτηση κατακερματισμού, η οποία προτάθηκε το 2000 ως μια νέα γενιά συναρτήσεων SHA.
Χρησιμοποιείται συχνά από τις αρχές πιστοποιητικών SSL για την υπογραφή πιστοποιητικών.	Είναι μια συνάρτηση κατακερματισμού που χρησιμοποιείται συνήθως στο Blockchain.
Ο SHA-1 λαμβάνει μια είσοδο και παράγει μια τιμή κατακερματισμού 160 bit (20 byte), γνωστή ως message digest.	Ο αλγόριθμος SHA-256 παράγει μια τιμή κατακερματισμού 256-bit από συμπληρωμένα μπλοκ μηνυμάτων 512-bit.
Λόγω του μικρότερου μεγέθους bit, είναι πιο επιρρεπής σε επιθέσεις.	Ο SHA-256 υπολογίζει πάντα εσωτερικά ένα κατακερματισμό 256 bit, γεγονός που βελτιώνει σημαντικά την ασφάλεια.

Το 2010 οι ομοσπονδιακές υπηρεσίες μαζί με το NIST (National Institute of Standards and Technology) επέβαλλαν με μια κοινή προσπάθεια [25] σε όλο το πλανήτη να σταματήσει την χρήση της γενιάς SHA-1 και να προχωρήσουν περιοδικά και άμεσα στην επόμενη οικογένεια αλγορίθμων, η οποία είχε σαφώς πολλά περισσότερα πλεονεκτήματα με κύριο πλεονέκτημα την ανοχή στις συγκρούσεις που αναφέραμε σε προηγούμενη παράγραφο.

Σε αυτό το σημείο, θα πρέπει να υπογραμμίσουμε ότι η μετάβαση στην νέα γενιά αλγορίθμων κρυπτογράφησης δεν έγινε άμεσα. Παρά την σαφώς καλύτερη ασφάλεια από την προηγούμενη γενιά την SHA-1 υπήρχε πολύ μεγάλη έλλειψη υποστήριξης της οικογένειας SHA-2 από συστήματα με Windows XP SP2 ή παλαιότερα. Για να δοθεί λύση στο πρόβλημα αυτό, η ομάδα της Google Chrome μέσω ανακοίνωσης σε όλο τον κόσμο, έκανε γνωστό ένα σχέδιο για να κάνει το πρόγραμμα περιήγησης ιστού της να σταματήσει σταδιακά να υποστηρίζει πιστοποιητικά TLS που εξαρτώνται

από την οικογένεια αλγορίθμων κρυπτογράφησης SHA-1, μέσα σε περίοδο από τα τέλη του 2014 έως και τις αρχές του 2015. Παρόμοια αντιμετώπιση κράτησαν και εταιρίες όπως η Microsoft και η Mozilla [25]. Συγκεκριμένα, η Microsoft ανακοίνωσε ότι η μηχανή αναζήτησης Internet Explorer και ο Microsoft Edge θα σταματήσουν να υποστηρίζουν τα δημόσια πιστοποιητικά TLS που έχουν υπογραφεί με SHA-1 από τον Φεβρουάριο του 2017. Με την σειρά της, αλλά δυστυχώς αδέξια η Mozilla απενεργοποίησε το SHA-1 στις αρχές Ιανουαρίου 2016, αλλά αναγκάστηκε να το ενεργοποιήσει πάλι προσωρινά μέσω μιας ενημέρωσης του Firefox, μετά από προβλήματα με που ανέκαμψαν από τις διαδικτυακές διεπαφές χρήστη ορισμένων μοντέλων δρομολογητών και συσκευών ασφαλείας.

3.3 Ανάλυση SHA-256

Σε αυτή την υπό-ενότητα θα αναλύσουμε σχολαστικά τον αλγόριθμο πίσω από την συνάρτηση SHA-256 και θα δούμε αναλυτικά την υλοποίηση που έχει πραγματοποιηθεί στην παρούσα διπλωματική και στην οποία βασίζεται το περιφερειακό μας.

3.3.1 Βασικές πράξεις και σταθερές του αλγορίθμου SHA-256

Η SHA-256 είναι όπως προαναφέραμε μια συνάρτηση hashing χωρίς κλειδιά που έχει ως λειτουργία την επεξεργασία κάθε μηνύματος σε μπλοκ-κομμάτια των 512 bit, αξιοποιώντας 16 λέξεις των 32 bit Εικόνα 14. Η είσοδος του αλγορίθμου μπορεί να λάβει οποιοδήποτε μήνυμα αρκεί να είναι μικρότερο από 264 bits και ο αλγόριθμος μετά από τις πράξεις και τους μετασχηματισμούς να παράγει ένα αποτέλεσμα στην έξοδο μεγέθους 256 bit που στην βιβλιογραφία αναφέρεται ως message digest [26]. Ο αλγόριθμος για να κάνει hashing ένα μπλοκ μηνύματος απαιτούνται 64 γύροι επαναλήψεων όπου το μπλοκ υπόκεινται σε επεξεργασία, καθώς επίσης και 64 σταθερές τιμές για κάθε ένα γύρο επαναλήψεων. Οι βασικές λειτουργίες που εκτελούνται και θα δούμε παρακάτω είναι οι ακόλουθες:

- Οι λογικές πράξεις **AND, XOR και OR** αναπαρίστανται στην βιβλιογραφία του αλγορίθμου με τα σύμβολα **Λ, ⊕, |** .
- Οι **πράξεις συμπληρώματος κατά δύο** συμβολίζονται με το σύμβολο ¬.
- Η ακέραια πρόσθεση modulo 2^{32} που αναπαρίσταται με το σύμβολο **+**.
- Η συνένωση δύο δυαδικών λέξεων που έχει ως προσδιορισμό το σύμβολο **#**.
- **SHIFTRIGHT (x, c) = ((x) >> (c)),** χρησιμοποιείται για να αναπαραστήσει την πράξη δυαδικής μετατόπισης κατά δεξιά μιας δυαδικής λέξης x κατά ένα ακέραιο ποσό c bits, αποβάλλοντας τα δεξιότερα c bits και αντικαθιστώντας τα με μηδενικά στα αριστερά.
- **ROTATIONRIGHT (x, c) = (((x) >> (c)) | ((x) << (32 - (c)))),** είναι η πράξη περιστροφής προς τα δεξιά, επίσης γνωστή στην βιβλιογραφία ως κυκλική δεξιά μετατόπιση, όπου τα δεξιότερα c bit δεν απορρίπτονται. Αντίθετα συμπληρώνονται στην αριστερή πλευρά της δυαδικής λέξης.

Εικόνα 14 Βασικά χαρακτηριστικά του αλγορίθμου

Η επιτυχής εκτέλεση του αλγορίθμου, απαιτεί επίσης τη χρήση ορισμένων σταθερών που έχουν θεσπιστεί στο άρθρο με την παρουσίαση του αλγορίθμου το 2001 [30] και συνδέονται άρρηκτα μαζί του. Πιο συγκεκριμένα, πρόκειται για 64 δυαδικές λέξεις *W(64)* των 32 bit που μάλιστα έχουν ρίζα, δηλαδή πηγάζουν από το δεκαδικό μέρος των κυβικών ριζών των πρώτων 64 πρώτων αριθμών. Οι σταθερές αυτές τιμές που αξιοποιούνται για την υλοποίηση του αλγορίθμου SHA-256, οι οποίες έχουν την δεκαεξαδική τους μορφή φαίνονται στην Εικόνα 15.

```
static const uint32_t W[64] = {
        0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5,0x3956c25b,0x59f111f1,0x923f82a4,0xab1c5ed5,
        0xd807aa98,0x12835b01,0x243185be,0x550c7dc3,0x72be5d74,0x80deb1fe,0x9bdc06a7,0xc19bf174,
        0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240calcc,0x2de92c6f,0x4a7484aa,0x5cb0a9dc,0x76f988da,
        0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7,0xc6e00bf3,0xd5a79147,0x06ca6351,0x14292967,
        0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13,0x650a7354,0x766a0abb,0x81c2c92e,0x92722c85,
        0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3,0xd192e819,0xd6990624,0xf40e3585,0x106aa070,
        0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5,0x391c0cb3,0x4ed8aa4a,0x5b9cca4f,0x682e6ff3,
        0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208,0x90befffa,0xa4506ceb,0xbef9a3f7,0xc67178f2
};
```

Εικόνα 15 Σταθερές του αλγορίθμου SHA-256

3.3.2 Συναρτήσεις του αλγορίθμου SHA-256

Ο αλγόριθμος SHA-256, όπως και παρόμοιοι αλγόριθμοι της οικογένειας SHA εμπεριέχουν ορισμένες συγκεκριμένες λειτουργίες και συναρτήσεις. Πιο συγκεκριμένα, κάθε συνάρτηση του αλγορίθμου έχει ως είσοδο μια λέξη 32 bit και ως αποτέλεσμα μια λέξη 32 bit πάλι. Οι λέξεις στην βιβλιογραφία αναπαρίστανται σαν x, y και z. Επίσης, θα πρέπει να τονίσουμε ότι υπάρχουν 2 τριμερείς συναρτήσεις που δέχονται 3 εισόδους και 4 μοναδιαίες.

Οι παρακάτω συναρτήσεις χρησιμοποιούνται συνέχεια κατά ολόκληρη την διάρκεια του αλγορίθμου και ονομάζονται Choice, Majority, Sum 0, Sum 1, Sigma 0, Sigma 1 ή αλλιώς στην υλοποίηση μας CH,MAJ,EP0,EP1,SIG0,SIG1 Εικόνα 16.

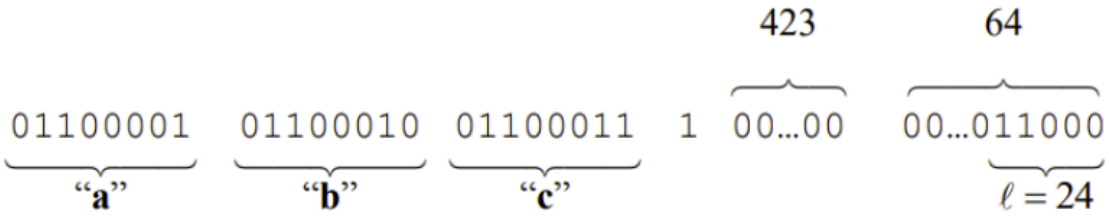
- **ROTATIONRIGHT (a, b)**: Ορίζει την κυκλική μετατόπιση δεξιά. Μετατοπίζει τα bits του a κατά b θέσεις προς τα δεξιά και τα bits που "περισσεύουν" τα μεταφέρει στην αρχή (αριστερά).
- **CH (x, y, z)**: Ορίζει τη λογική πράξη "Choice" (επιλογή). Επιλέγει bits από το y ή το z ανάλογα με τα bits του x.
- **MAJ (x, y, z)**: Ορίζει τη λογική πράξη "Majority" (πλειοψηφία). Για κάθε bit, επιστρέφει το bit που εμφανίζεται περισσότερο στα x, y, και z.
- **EP0 (x)**: Ορίζει τη συνάρτηση "Sum 0". Κάνει τρεις κυκλικές μετατοπίσεις δεξιά και παίρνει το XOR των αποτελεσμάτων.
- **EP1 (x)**: Ορίζει τη συνάρτηση "Sum 1". Κάνει τρεις κυκλικές μετατοπίσεις δεξιά και παίρνει το XOR των αποτελεσμάτων.
- **SIG0(x)**: Ορίζει τη συνάρτηση "Sigma 0". Κάνει δύο κυκλικές μετατοπίσεις δεξιά και μια κανονική μετατόπιση δεξιά, παίρνοντας το XOR των αποτελεσμάτων.
- **SIG1(x)**: Ορίζει τη συνάρτηση "Sigma 1". Κάνει δύο κυκλικές μετατοπίσεις δεξιά και μια κανονική μετατόπιση δεξιά, παίρνοντας το XOR των αποτελεσμάτων.

```
//defining the rotations and the algorithms praxis
#define ROTATIONRIGHT(a,b) (((a) >> (b)) | ((a) << (32-(b))))

#define CH(x,y,z) (((x) & (y)) ^ (~(x) & (z)))
#define MAJ(x,y,z) (((x) & (y)) ^ ((x) & (z)) ^ ((y) & (z)))
#define EP0(x) (ROTATIONRIGHT(x,2) ^ ROTATIONRIGHT(x,13) ^ ROTATIONRIGHT(x,22))
#define EP1(x) (ROTATIONRIGHT(x,6) ^ ROTATIONRIGHT(x,11) ^ ROTATIONRIGHT(x,25))
#define SIG0(x) (ROTATIONRIGHT(x,7) ^ ROTATIONRIGHT(x,18) ^ ((x) >> 3))
#define SIG1(x) (ROTATIONRIGHT(x,17) ^ ROTATIONRIGHT(x,19) ^ ((x) >> 10))
```

Εικόνα 16 Βασικές πράξεις του αλγορίθμου SHA-256

3.3.3 Διαδικασία Padding του αλγορίθμου SHA-256

Για να διασφαλιστεί ότι το μήνυμα είναι ακέραιο πολλαπλάσιο του 512 bit, το μήνυμα συμπληρώνεται με μια μορφή padding αν χρειαστεί Εικόνα 17. Συγκεκριμένα, συμπληρώνεται με μηδενικά μέχρι να έχει μήκος 448 bits. Τα τελευταία 64 bits έχουν σημασία για την κωδικοποίηση του μήκους του μηνύματος.



Εικόνα 17 Παράδειγμα εισόδου "abc" με padding

3.3.4 Δρομολόγηση μηνύματος του αλγορίθμου SHA-256

Το πρόγραμμα μηνυμάτων δημιουργείται από το padded μήνυμα και βρίσκει χρήση στον αλγόριθμο ως είσοδο για την τελική εξαγωγή της τιμής hashing. Μάλιστα, το πρόγραμμα μηνυμάτων αποτελείται από 64 λέξεις των 32 bit η κάθε μια. Συγκεκριμένα, οι πρώτες 16 λέξεις παράγονται διαιρώντας το αρχικό μήνυμα των 512 bit και την καταχώριση αυτών των τιμών σε καταχωρητές 32 bit.

Για να παράγουμε τις υπόλοιπες 48 λέξεις, χρησιμοποιούμε μερικές από τις λειτουργίες που παρουσιάσαμε νωρίτερα. Συγκεκριμένα, κάθε λέξη από W_{16} - W_{63} παράγεται από τον ακόλουθο τύπο Εξίσωση 1.

$$W[i] \coloneqq W[i-16] + s_0 + W[i-7] + s_1 \qquad (1)$$

3.3.5 Βασικός κορμός του αλγορίθμου SHA-256 (Συμπίεση)

Το τελευταίο βήμα του αλγορίθμου SHA-256 που είναι και το ποιο υπολογιστικά βαρύ του κομμάτι είναι γνωστό ως συμπίεση και θεωρείται ο κεντρικός πυλώνας του αλγορίθμου. Πρώτο βήμα είναι η αρχικοποίηση των συναρτήσεων κατακερματισμού. Για να γίνει αυτή η διαδικασία αξιοποιούνται 8 καταχωρητές-τιμές που ονομάζονται καταχωρητές κατάστασης Εικόνα 18 και έχουν ετικέτες a, b, c, d, e, f, g, h και οι οποίες παίρνουν τιμές που είναι σύμφωνες με το κλασματικό μέρος των τετραγωνικών ριζών των πρώτων 8 πρώτων αριθμών, όπως αναφέραμε παραπάνω. Οι τιμές αυτές σε δεκαεξαδική μορφή παραθέτονται παρακάτω:

```
(first 32 bits of the fractional parts of the square roots of the first 8 primes 2..19):
h0 := 0x6a09e667
h1 := 0xbb67ae85
h2 := 0x3c6ef372
h3 := 0xa54ff53a
h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd19
```

Εικόνα 18 Σταθερές Αλγορίθμου για αρχικοποίηση

Αφού φορτωθούν οι αρχικές τιμές hashing στους καταχωρητές μας, ο αλγόριθμος εκτελεί τις εξής πράξεις για 64 φορές Εξίσωση 2.

$$
\begin{aligned}
&for(i\ \alpha\pi\acute{o}\ 0\ \epsilon\acute{\omega}\varsigma\ 63)\{\\
&\quad S_1 = (e\ \textbf{rightrotate}\ 6)\ \text{xor}\ (e\ \textbf{rightrotate}\ 11)\ \text{xor}\ (e\ \textbf{rightrotate}\ 25)\\
&\quad ch = (e\ \textbf{and}\ f)\ \textbf{xor}\ \big((\ \textbf{not}\ e)\textbf{and}\ g\big)\\
&\quad temp_1 = h + s_1 + ch + m[i] + w[i]\\
&\quad s_0 = (a\ \textbf{rightrotate}\ 2)\ \textbf{xor}\ (a\ \textbf{rightrotate}\ 13)\text{xor}\ (a\ \textbf{rightrotate}\ 22)\\
&\quad maj = (a\ \textbf{and}\ b)\ \textbf{xor}\ (a\ \textbf{and}\ c)\ \textbf{xor}\ (b\ \textbf{and}\ c)\\
&\quad temp_2 = s_0 + maj\\
&\\
&\quad h = g\\
&\quad g = f\\
&\quad h = g\\
&\quad f = e\\
&\quad e = d + temp_1\\
&\quad d = c\\
&\quad c = b\\
&\quad b = a\\
&\quad a = temp_1 + temp_2\ \}
\end{aligned}
\qquad (2)
$$

Βλέπουμε ότι διαισθητικά ο αλγόριθμος μετατοπίζει όλες τις τιμές προς τα κάτω από το a στο h αφήνοντας πάντα την τελευταία και αξιοποιώντας τις προσωρινές τιμές αντικαθιστώντας κάποιες από αυτές τις τιμές με καινούργιες. Συγκεκριμένα, υπάρχει ένα πολύ ωραίο animation effect που δείχνει όλη την εκτέλεση του αλγορίθμου στο GitHub [31]. Έτσι, παρατηρούμε ότι για να υπολογίσουμε και να βρούμε τις 4 από τις 6 τιμές απαιτείται να χρησιμοποιήσουμε τις 6 συναρτήσεις του SHA-256 που αναφέραμε σε προηγούμενη υπό-ενότητα καθώς και τις σταθερές που προαναφέραμε επίσης.

Στο τελικό στάδιο μετά από 64 επαναλήψεις μας απομένουν 8 καταχωρητές που εμπεριέχουν τις τιμές του τελευταίου βήματος του αλγορίθμου μας. Το τελικό αποτέλεσμα του hashing του αλγορίθμου προκύπτει με την πρόσθεση των αρχικών τιμών κατακερματισμού σε αυτούς τους 8 καταχωρητές-τιμές. Τελικά, με τη συνένωση των λέξεων των 32-bit του καταχωρητή λαμβάνουμε την τελική έξοδο των 256-bit του της συνάρτησης η οποία ονομάζεται message digest Εικόνα 19.

```
Produce the final hash value (big-endian):
digest := hash := h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7
```

Εικόνα 19 Τελικός υπολογισμός

3.3.6 Περιγραφή ψευδοκώδικα του αλγορίθμου SHA-256

Στην υποενότητα αυτή περιγράφουμε τον ψευδοκώδικα του αλγορίθμου SHA-256 στο οποίο βασίζεται η υλοποίηση μας Εικόνα 20. Αρχικά, ορισμένες παρατηρήσεις σχετικά με τον αλγόριθμο και την υλοποίηση που θα ακολουθήσει στην επόμενη ενότητα:

- Όλες οι μεταβλητές που χρησιμοποιούνται αποτελούν ακέραιους αριθμούς των 32 bit μη προσημασμένους και η πρόσθεση υλοποιείται με modulo 2^{32}.
- Σε κάθε γύρο-επανάληψη, υπάρχει μια σταθερά W[i] που είναι αντιπροσωπευτική του αλγορίθμου και μια εγγραφή στον πίνακα προγραμματισμού μηνυμάτων m[i]. Η όλη διαδικασία είναι επαναληπτική για $0 \leq i \leq 63$.
- Η συνάρτηση συμπίεσης που είναι η βασική συνάρτηση του αλγορίθμου και ο κορμός του χρησιμοποιεί 8 μεταβλητές που από την βιβλιογραφία έχουν τις τιμές a έως h.
- Κατά την αποτύπωση των σταθερών σε αυτό τον ψευδοκώδικα και την υλοποίηση που ακολουθεί χρησιμοποιείται η Big-endian και επίσης κατά την ανάλυση των δεδομένων μπλοκ μηνυμάτων από bytes σε λέξεις. Έτσι για παράδειγμα έχουμε για πρώτη λέξη του μηνύματος εισόδου «abc» μετά την padding να είναι το ακόλουθο αποτέλεσμα➔0x61626380 Εικόνα 17.

Αρχικοποίηση τιμών Hashing:
(αποτελούν τα πρώτα 32 bit των κλασματικών μερών των τετραγωνικών ριζών των 8 πρώτων αριθμών 2..19):
h0 := 0x6a09e667
h1 := 0xbb67ae85
h2 := 0x3c6ef372
h3 := 0xa54ff53a
h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd1

Αρχικοποίηση του πίνακα σταθερών:
(αποτελείται από τα πρώτα 32 bits των κλασματικών μερών των κυβικών ριζών των πρώτων 64 πρώτων αριθμών 2..311):
W[0..63] := 0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5, 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174, 0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da, 0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967, 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85, 0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070, 0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2

Προεπεξεργασία (Padding):

i. *Ξεκίνα με το αρχικό μήνυμα μήκους L bits.*
ii. *Προσθήκη ενός μόνο bit «1».*
iii. *Προσθήκη K bit «0», όπου K είναι ο ελάχιστος αριθμός >= 0 τέτοιο ώστε (L + 1 + K + 64) να είναι πολλαπλάσιο του 512.*
iv. *Προσάρτηση του L ως ακέραιου big-endian αριθμού 64 bit, καθιστώντας το συνολικό μέγεθος του επεξεργασμένου μηνύματος να έχει μήκος πολλαπλάσιο των 512 bit.*
v. ***Right Rotate (κυκλική δεξιά μετατόπιση)**: Μετακινεί τα bits μιας λέξης προς τα δεξιά, με τα bits που "φεύγουν" από τη μία άκρη να επανεισάγονται στην άλλη άκρη.*
vi. ***Right Shift (δεξιά μετατόπιση)**: Μετακινεί τα bits μιας λέξης προς τα δεξιά, εισάγοντας μηδενικά (ή το πρόσημο για signed αριθμούς) στα πιο αριστερά bits που κενώνονται.*

Επεξεργασία του μηνύματος σε διαδοχικά τμήματα των 512 bit:

Απαιτείται ο τεμαχισμός του μηνύματος σε κομμάτια των 512 bit
για κάθε τεμάχιο:
{
Δημιουργία ενός πίνακα μηνύματος 64 καταχωρήσεων m[0..63] από λέξεις 32-bit
(Οι αρχικές τιμές στο w[0..63] δεν έχουν σημασία)
Αντιγραφή των τμημάτων στις πρώτες 16 λέξεις m[0..15] του πίνακα μηνυμάτων

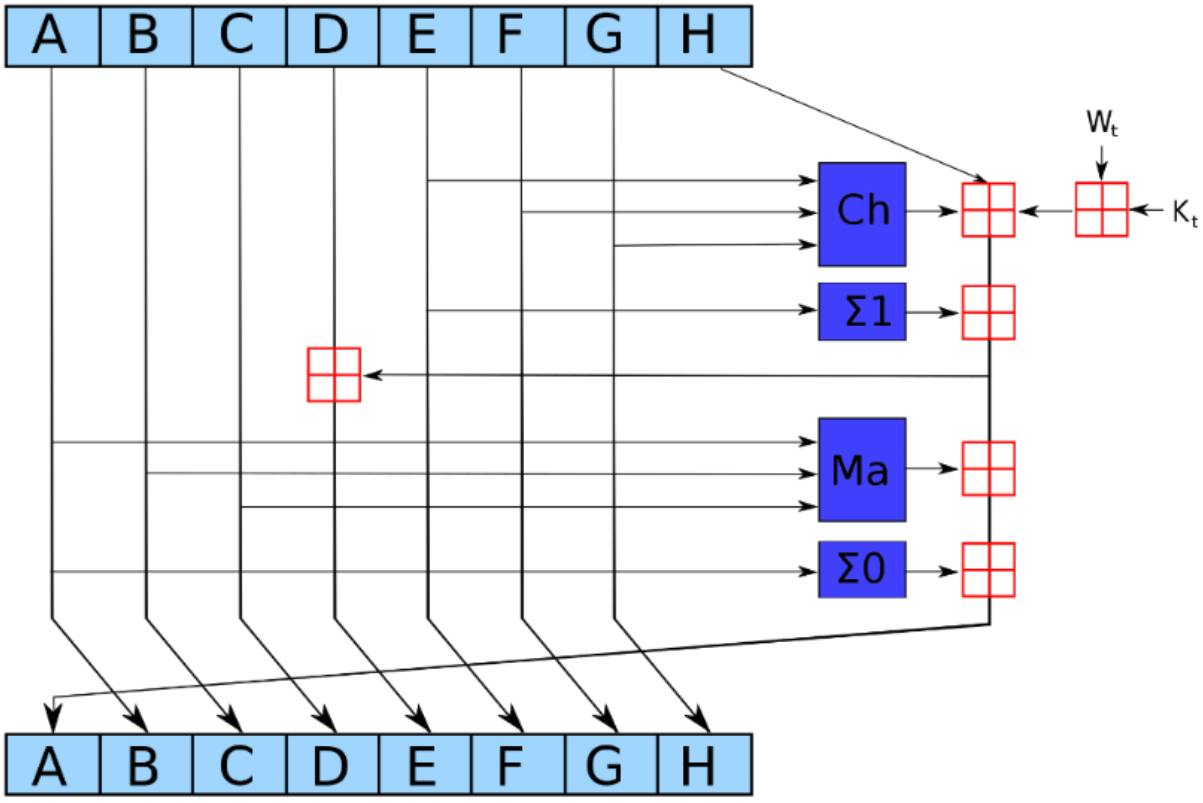
Επέκταση των πρώτων 16 λέξεων στις υπόλοιπες 48 λέξεις m[16..63] του πίνακα μηνυμάτων:
for i από 16 έως 63
{
(Εκτέλεση των παρακάτω πράξεων)
s0 := (m[i-15] rightrotate 7) xor (m [i-15] rightrotate 18) xor (m [i-15] rightshift 3)
s1 := (m[i-2] rightrotate 17) xor (m [i-2] rightrotate 19) xor (m [i-2] rightshift 10)
m[i] := m[i-16] + s0 + m[i-7] + s1
}
Αρχικοποίηση των μεταβλητών εργασίας a έως h στην παρούσα τιμή hashing:
a := h0
b := h1
c := h2
d := h3
e := h4
f := h5
g := h6
h := h7

Κύριος βρόχος λειτουργίας συμπίεσης (βασική πράξη):
for i από 0 έως 63
{
(Εκτέλεση των παρακάτω πράξεων και μετατοπισμός των τιμών προς τα κάτω)
S1 := (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)
ch := (e and f) xor ((not e) και g)
temp1 := h + S1 + ch + m[i] + W[i]
S0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)
maj := (a and b) xor (a και c) xor (b and c)
temp2 := S0 + maj

```
        h := g
        g := f
        f := e
        e := d + temp1
        d := c
        c := b
        b := a
        a := temp1 + temp2
    }
    Προσθήκη του συμπιεσμένου κομματιού στην παρούσα τιμή hashing:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
    h4 := h4 + e
    h5 := h5 + f
    h6 := h6 + g
    h7 := h7 + h
}
```

Παραγωγή της τελικής τιμής κατακερματισμού (big-endian):

digest := hash := h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7



Εικόνα 20 Μετατόπιση προς τα κάτω και πράξεις κάθε επανάληψης[7]

Το κόκκινο σχήμα ⊞ *είναι η πρόσθεση modulo* 2^{32} *για SHA-256, ή* 2^{64} *για SHA-512*

[7] Η εικόνα προέρχεται από το https://en.wikipedia.org/wiki/SHA-2 .

3.3.7 Περιγραφή της υλοποίησης μας του αλγορίθμου SHA-256

Στην υποενότητα αυτή θα περιγράψουμε τις συναρτήσεις της υλοποίησης μας του αλγορίθμου SHA-256. Στην παρακάτω εικόνα βλέπουμε τις σταθερές που χρησιμοποιούνται κατά την διάρκεια του αλγορίθμου, μια δομή τύπου struct που εμπεριέχει τις πληροφορίες για τα δεδομένα του αλγορίθμου, την προκαταβολική δήλωση των συναρτήσεων καθώς και μερικά define για βασικές μαθηματικές πράξεις που έχουμε περιγράψει προηγουμένως Εικόνα 21.

Στην συνέχεια, θα αναλύσουμε τις υπόλοιπες συναρτήσεις που έχουμε υλοποιήσει και εξυπηρετούν στην λειτουργία του αλγορίθμου. Πρώτη από αυτές αποτελεί η συνάρτηση sha256_initiliazation() Εικόνα 22 που έχει ως στόχο την δήλωση των σταθερών τιμών που αναφέραμε στην προηγούμενη υπό-ενότητα. Γενικά, ιδιαίτερη προσοχή γιατί σε κάθε συνάρτηση χρησιμοποιείται η δομή struct που έχουμε ορίσει παραπάνω.

Επόμενη συνάρτηση είναι η sha256_updating() που ο ρόλος της είναι να επεξεργάζεται δεδομένα εισόδου σε κομμάτια και να ενημερώνει ανάλογα την εσωτερική κατάσταση του πλαισίου SHA-256 (SHA256_struct). Γενικά, εμπεριέχει έναν δείκτη στη δομή SHA-256, η οποία περιέχει την τρέχουσα κατάσταση του υπολογισμού κατακερματισμού, έναν πίνακα με bytes που αντιπροσωπεύει τα δεδομένα εισόδου και τέλος το μήκος του πίνακα εισόδου. Για την ανίχνευση της επανάληψης, χρησιμοποιείται ένας μετρητής βρόγχου και υπάρχει μια δομή επανάληψης τύπου while που χρησιμοποιείται για την κάθε byte του πίνακα δεδομένων εισόδου.

```
//define macro
#define SHA256_BLOCK_SIZE 32  // SHA256 component has an output of 32 byte called digest

// defining the struct of sha256
typedef struct
{
    uint8_t data[64];
    uint32_t datalength;
    unsigned long long bitlength;
    uint32_t status[8];
} SHA256_struct;

//declaring the functions
//initialization function does the init process
//updating function is called to update the values in the algorithm
//output function is called in order to give the output of the algorithm
void sha256_initialization(SHA256_struct *sha256);
void sha256_updating(SHA256_struct *sha256, const uint8_t data[], size_t length);
void sha256_output(SHA256_struct *sha256, uint8_t hash[]);
void sha256_reverse_bytes(uint8_t *hashes, const uint32_t *status);
void sha256_append_length(SHA256_struct *sha256);
void sha256_padding(SHA256_struct *sha256);

#endif   // SHA256_H

//defining the rotations and the algorithms praxis
#define ROTATIONRIGHT(a,b) (((a) >> (b)) | ((a) << (32-(b))))

#define CH(x,y,z) (((x) & (y)) ^ (~(x) & (z)))
#define MAJ(x,y,z) (((x) & (y)) ^ ((x) & (z)) ^ ((y) & (z)))
#define EP0(x) (ROTATIONRIGHT(x,2) ^ ROTATIONRIGHT(x,13) ^ ROTATIONRIGHT(x,22))
#define EP1(x) (ROTATIONRIGHT(x,6) ^ ROTATIONRIGHT(x,11) ^ ROTATIONRIGHT(x,25))
#define SIG0(x) (ROTATIONRIGHT(x,7) ^ ROTATIONRIGHT(x,18) ^ ((x) >> 3))
#define SIG1(x) (ROTATIONRIGHT(x,17) ^ ROTATIONRIGHT(x,19) ^ ((x) >> 10))

//defining the matrix for the algorithm for the 64 loops
static const uint32_t W[64] = {
        0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5,0x3956c25b,0x59f111f1,0x923f82a4,0xab1c5ed5,
        0xd807aa98,0x12835b01,0x243185be,0x550c7dc3,0x72be5d74,0x80deb1fe,0x9bdc06a7,0xc19bf174,
        0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240calcc,0x2de92c6f,0x4a7484aa,0x5cb0a9dc,0x76f988da,
        0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7,0xc6e00bf3,0xd5a79147,0x06ca6351,0x14292967,
        0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13,0x650a7354,0x766a0abb,0x81c2c92e,0x92722c85,
        0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3,0xd192e819,0xd6990624,0xf40e3585,0x106aa070,
        0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5,0x391c0cb3,0x4ed8aa4a,0x5b9cca4f,0x682e6ff3,
        0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208,0x90befffa,0xa4506ceb,0xbef9a3f7,0xc67178f2
};
```

Εικόνα 21 Βασικές σταθερές του αλγορίθμου και δομές

Πιο συγκεκριμένα, κάθε byte δεδομένων εισόδου αποθηκεύεται στο buffer sha256->data στη θέση που υποδεικνύεται από το sha256->datalength. Το SHA-256 επεξεργάζεται δεδομένα σε κομμάτια των 512 bit (64 byte). Η λογική είναι ότι όταν φτάσει η δομή επανάληψης σε μια λέξη μήκους 64 byte σημαίνει ότι θα πρέπει να καλέσουμε την επόμενη συνάρτηση που θα αναλύσουμε παρακάτω (sha256_transformation), που έχει ως σκοπό την επεξεργασία των δεδομένων. Έπειτα, το συνολικό μήκος αυξάνεται κατά 512 bit αφού ισχύει ότι 64 bytes = 512 bits. Και στην συνέχεια το μέγεθος μηδενίζεται για να αρχίσει ο buffer πάλι την ίδια διαδικασία για το επόμενο κομμάτι εισόδου.

Γενικά, η συνάρτηση sha256_updating() έχει σαν είσοδο έναν πίνακα δεδομένων και τον επεξεργάζεται σε κομμάτια των 64 byte, ενημερώνοντας την κατάσταση SHA-256 για κάθε ένα κομμάτι. Εφόσον, συμπληρωθεί ένα κομμάτι των 64 bytes, καλεί την sha256_transformation για να ενημερώσει την κατάσταση hashing. Η συνάρτηση διατηρεί ένα buffer (sha256->data) και παρακολουθεί το συνολικό μήκος των δεδομένων που έχουν επεξεργαστεί μέχρι στιγμής σε bits (sha256->bitlength).

```
//function for initialization of the values of the algorithm
void sha256_initialization(SHA256_struct *sha256)
{
    //first init H 32bit words for the algorithm
    sha256->datalength = 0;
    sha256->bitlength = 0;
    sha256->status[0] = 0x6a09e667;
    sha256->status[1] = 0xbb67ae85;
    sha256->status[2] = 0x3c6ef372;
    sha256->status[3] = 0xa54ff53a;
    sha256->status[4] = 0x510e527f;
    sha256->status[5] = 0x9b05688c;
    sha256->status[6] = 0x1f83d9ab;
    sha256->status[7] = 0x5be0cd19;
}

//function called for updating the values of the algorithm
void sha256_updating(SHA256_struct *sha256, const uint8_t data[], size_t length)
{
    size_t i = 0;
//  #pragma HLS pipeline II=1
    while (i < length)
    {
        sha256->data[sha256->datalength] = data[i];
        sha256->datalength++;
        if (sha256->datalength == 64)
        {
            sha256_transformation(sha256, sha256->data);
            sha256->bitlength += 512;
            sha256->datalength = 0;
        }
        i++;
    }
}
```

Εικόνα 22 Αρχικοποίηση και τεμαχισμός της εισόδου δεδομένων

Έπειτα, βασικός δομικός πυλώνας της υλοποίησης μας είναι η συνάρτηση της συμπίεσης που είναι η sha256_transformation Εικόνα 23. Δεν θα την αναλύσουμε ιδιαίτερα εφόσον ακολουθεί τον αλγόριθμο της βιβλιογραφίας για την SHA-256 και απλά εμπεριέχει κατά κύριο λόγο τις πράξεις που λαμβάνουν χώρα κατά την επεξεργασία τον δεδομένων. Κύριο βήμα στην όλη διαδικασία είναι οι 64 επαναλήψεις με την χρήση μιας δομής while και η μετατόπιση προς τα κάτω του κάθε χαρακτήρα με ενδιάμεσες πράξεις, όπως περιγράψαμε στο κεφάλαιο του ψευδοκώδικα

παραπάνω. Στο τέλος, γίνεται η ανάθεση των τιμών σε 8 σταθερές που πάλι προαναφέραμε παραπάνω και είναι από το a έως το h.

Τελευταία συνάρτηση που χρησιμοποιείται στην δομή του αλγορίθμου και θα εξηγήσουμε είναι η sha256_output() Εικόνα 24, η οποία τρεις συναρτήσεις μια για το padding του υπόλοιπου του μυνήματος, μια για προσθήκη του μήκους του bit και μια για μετατροπή από little-endian σε big-endian. Η πρώτη συγκεκριμένα, εμπεριέχει το struct με την τρέχουσα κατάσταση του υπολογισμού και έναν πίνακα για την αποθήκευση της τελικής τιμής του κατακερματισμού μεγέθους 256 bit ή 32 byte. Η συνάρτηση αυτή είναι αρκετά σημαντική διότι διασφαλίζει ότι το συνολικό μήκος των δεδομένων σε bit θα είναι πολλαπλάσιο του 512, το οποίο είναι βασική απαίτηση του αλγορίθμου SHA-256. Για αυτό το λόγο επιβάλλεται και η συμπλήρωση των δεδομένων. Αναλυτικότερα, αν το τρέχον μήκος είναι μικρότερο των 56 bytes πρέπει να προσθέσει ένα byte 0x80 (το οποίο είναι το δυαδικό 10000000) και να γεμίσει τον buffer με 0x00 bytes μέχρι να φτάσει τα 56 bytes.

```
//transformation function is called everytime to make the transformation of the values
void sha256_transformation(SHA256_struct *sha256, const uint8_t data[])
{
    uint32_t a, b, c, d, e, f, g, h, i, j, t1, t2, m[64];

    for (i = 0; i < 16; ++i)
    {
        #pragma HLS unroll
        j = i * 4; // Update j in relation with i
        m[i] = (data[j] << 24) | (data[j + 1] << 16) | (data[j + 2] << 8) | (data[j + 3]);
    }
    for ( ; i < 64; ++i)
    {
        #pragma HLS unroll
        m[i] = SIG1(m[i - 2]) + m[i - 7] + SIG0(m[i - 15]) + m[i - 16];
    }

    a = sha256->status[0];
    b = sha256->status[1];
    c = sha256->status[2];
    d = sha256->status[3];
    e = sha256->status[4];
    f = sha256->status[5];
    g = sha256->status[6];
    h = sha256->status[7];

    i = 0;
    while (i < 64)
    {
        t1 = h + EP1(e) + CH(e, f, g) + W[i] + m[i];
        t2 = EP0(a) + MAJ(a, b, c);
        h = g;
        g = f;
        f = e;
        e = d + t1;
        d = c;
        c = b;
        b = a;
        a = t1 + t2;
        i++;
    }

    sha256->status[0] += a;
    sha256->status[1] += b;
    sha256->status[2] += c;
    sha256->status[3] += d;
    sha256->status[4] += e;
    sha256->status[5] += f;
    sha256->status[6] += g;
    sha256->status[7] += h;
}
```

Εικόνα 23 Συνάρτηση συμπίεσης στην υλοποίηση

Αν ο buffer δεν εμπεριέχει αρκετό χώρο για να φτάσει τα 56 bytes πρέπει να τον γεμίσουμε μέχρι τα 64 bytes με μηδενικά και να υποβληθεί σε επεξεργασία και στην συνέχεια να τον επαναφέρουμε. Η συγκεκριμένη διαδικασία διασφαλίζει ότι υπάρχουν 64 bytes έτοιμα να χρησιμοποιηθούν άμεσα για την τελική συμπλήρωση. Στη συνέχεια, οι άλλες δύο συναρτήσεις διασφαλίζουν ότι η τελική διαδικασία περιλαμβάνει τον υπολογισμό του συνολικού μήκους σε bit του μηνύματος, περιλαμβάνοντας επίσης τα δεδομένα που προστέθηκαν. Αυτό το μήκος bit προστίθεται στα τελευταία 8 bytes του buffer (ως big-endian). Έπειτα, το τελικό κομμάτι επεξεργάζεται και το τελικό αποτέλεσμα ή τελική τιμή αποθηκεύεται στον πίνακα sha256->status, ο οποίος είναι ένας πίνακας από 8 ακέραιους αριθμούς των 32 bit. Συμπερασματικά, η συνάρτηση sha256_output ολοκληρώνει τη διαδικασία του SHA-256 συμπληρώνοντας τα δεδομένα που έχουν απομείνει για να διασφαλίσει ότι το συνολικό μήκος του μηνύματος είναι ακέραιο πολλαπλάσιο των 512 bits. Προσθέτει το μήκος bit του μηνύματος, επεξεργάζεται τυχόν εναπομείναντα δεδομένα και στη συνέχεια μετατρέπει την εσωτερική κατάσταση κατακερματισμού στην τελική τιμή κατακερματισμού 256 bit (32 bytes), κατάλληλα διαμορφωμένη σε big-endian σειρά bytes.

```
void sha256_padding(SHA256_struct *sha256)
{
    uint32_t i = sha256->datalength;

    // Pad whatever data is left in the buffer
    sha256->data[i++] = 0x80; // Append a single '1' bit

    if (sha256->datalength < 56)
    {
        while (i < 56)
        {
            sha256->data[i++] = 0x00; // Pad with zeros
        }
    }
    else
    {
        while (i < 64)
        {
            sha256->data[i++] = 0x00; // Pad with zeros
        }
        sha256_transformation(sha256, sha256->data);
        memset(sha256->data, 0, 56);
    }
}

void sha256_append_length(SHA256_struct *sha256)
{
    // Append the bit length of the original message
    sha256->bitlength += sha256->datalength * 8;

    // Append the bit length in big endian format
    for (int i = 0; i < 8; ++i)
    {
        sha256->data[63 - i] = (sha256->bitlength >> (i * 8)) & 0xFF;
    }

    sha256_transformation(sha256, sha256->data);
}

void sha256_reverse_bytes(uint8_t *hashes, const uint32_t *status)
{
    // Reverse bytes to convert from big endian to little endian
    for (uint32_t i = 0; i < 32; ++i)
    {
        int statusIndex = i / 4;
        int byteIndex = i % 4;
        hashes[i] = (status[statusIndex] >> (24 - byteIndex * 8)) & 0xFF;
    }
}

void sha256_output(SHA256_struct *sha256, uint8_t hashes[])
{
    sha256_padding(sha256);       // Add padding
    sha256_append_length(sha256); // Append the bit length

    // Convert the hash state to the final hash (little endian to big endian)
    sha256_reverse_bytes(hashes, sha256->status);
}
```

Εικόνα 24 Συνάρτηση εξόδου του αποτελέσματος

3.3.8 Επιβεβαίωση ορθής λειτουργίας

Μια καλή τεχνική στην σχεδίαση περίπλοκων συστημάτων που εμπεριέχουν και λογισμικό αλλά και υλικό για προγραμματισμό είναι αρχικά να ασχολούμαστε καθαρά μόνο με το ένα από τα δύο κομμάτια, έτσι ώστε να μπορούμε να έχουμε τον έλεγχο σε κάθε επίπεδο του σχεδιασμού και να μπορούμε να σπάμε τα προβλήματα που μπορεί να προκύψουν κατά την σχεδίαση σε μικρότερες στοχευμένες ομάδες.

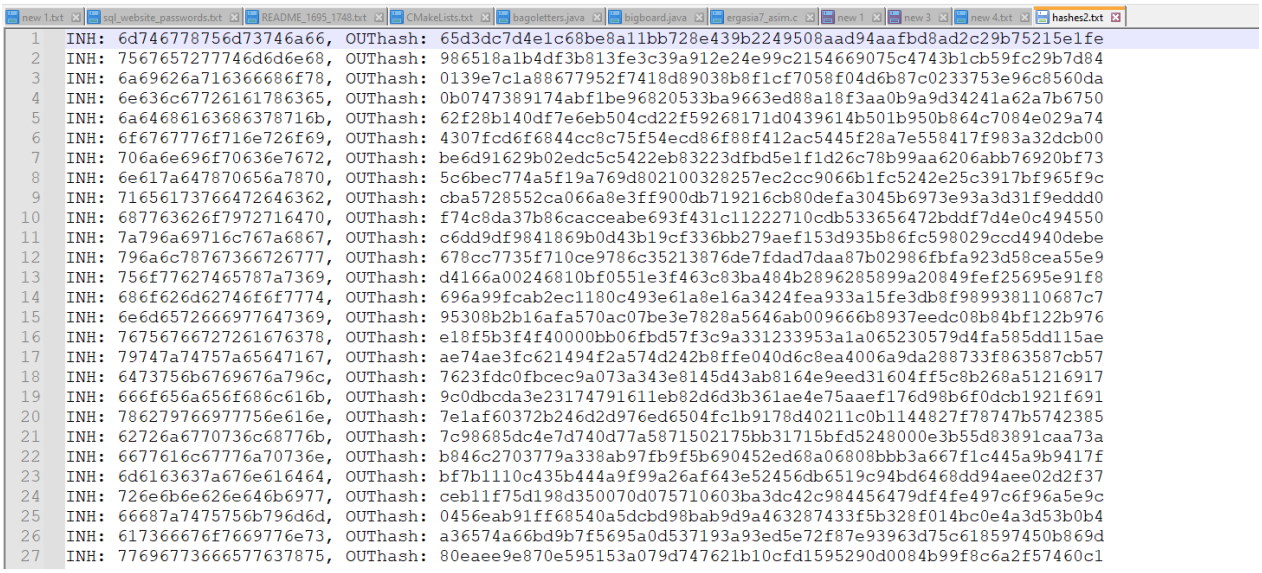
Ακολουθώντας το παραπάνω σκεπτικισμό αποφασίσαμε να επαληθεύσουμε την υλοποίηση μας σε C++ κάνοντας μια πρώτη σύγκριση για να δούμε αν λειτουργεί σε σχέση με ένα περιφερειακό που διατίθεται δωρεάν σε κάποια online πηγή. Έτσι, χρησιμοποιήθηκε για την επαλήθευση του κώδικα σε πρώτο στάδιο ένα περιφερειακό του GitHub [32], το οποίο είναι γραμμένο σε VHDL και αναπαριστά την βασική λειτουργία του αλγορίθμου SHA-256 που είναι η διαδικασία της συμπίεσης Εικόνα 25. Από το συγκεκριμένο περιφερειακό, που βρέθηκε σε ελεύθερη μορφή θα πρέπει να τονίσουμε ότι έλειπε το επίπεδο του αρχικού padding του μηνύματος και οπότε τα δοκιμαστικά τεστ έγιναν με την παροχή του padding από εμάς.

```
24
25    LIBRARY IEEE;
26    use IEEE.std_logic_1164.all;
27    use IEEE.std_logic_unsigned.all;
28
29    ENTITY sha256for1chunk is
30            Port ( reset    : in  STD_LOGIC;
31                           clock  : in  STD_LOGIC;
32                           --input side signals
33                           plain  : in  STD_LOGIC_VECTOR(511 downto 0);
34                           load : in  STD_LOGIC;
35                           empty : out STD_LOGIC;
36                           --output side signals
37                           digest       : out STD_LOGIC_VECTOR (255 downto 0);
38                           ready  : out STD_LOGIC);
39    end sha256for1chunk;
40
41    ARCHITECTURE Behavioral of sha256for1chunk is
42
43           type initT is array (0 to 7) of STD_LOGIC_VECTOR(31 downto 0);
44           constant init : initT := (
45                  x"6a09e667", x"bb67ae85", x"3c6ef372", x"a54ff53a", x"510e527f", x"9b05688c", x"1f83d9ab", x"5be0cd19");
46           type kT is array (0 to 63) of STD_LOGIC_VECTOR(31 downto 0);
47           constant k : kT := (
48                  x"428a2f98", x"71374491", x"b5c0fbcf", x"e9b5dba5", x"3956c25b", x"59f111f1", x"923f82a4", x"ab1c5ed5",
49                  x"d807aa98", x"12835b01", x"243185be", x"550c7dc3", x"72be5d74", x"80deb1fe", x"9bdc06a7", x"c19bf174",
50                  x"e49b69c1", x"efbe4786", x"0fc19dc6", x"240ca1cc", x"2de92c6f", x"4a7484aa", x"5cb0a9dc", x"76f988da",
51                  x"983e5152", x"a831c66d", x"b00327c8", x"bf597fc7", x"c6e00bf3", x"d5a79147", x"06ca6351", x"14292967",
52                  x"27b70a85", x"2e1b2138", x"4d2c6dfc", x"53380d13", x"650a7354", x"766a0abb", x"81c2c92e", x"92722c85",
53                  x"a2bfe8a1", x"a81a664b", x"c24b8b70", x"c76c51a3", x"d192e819", x"d6990624", x"f40e3585", x"106aa070",
54                  x"19a4c116", x"1e376c08", x"2748774c", x"34b0bcb5", x"391c0cb3", x"4ed8aa4a", x"5b9cca4f", x"682e6ff3",
55                  x"748f82ee", x"78a5636f", x"84c87814", x"8cc70208", x"90befffa", x"a4506ceb", x"bef9a3f7", x"c67178f2");
56           signal a,b,c,d,e,f,g,h,s0,s1,su0,su1,maj,ch,temp1,temp2 : STD_LOGIC_VECTOR(31 downto 0);
57           type wT is array (15 downto 0) of STD_LOGIC_VECTOR(31 downto 0);
58           signal w : wT;
59           signal wCNT, chunkCNT : STD_LOGIC_VECTOR(6 downto 0);
60           signal intEnable : STD_LOGIC;
61
62    BEGIN
```

Εικόνα 25 SHA-256 περιφερειακό σε VHDL

Η λογική για να γίνει το τεστ του αλγορίθμου ήταν η δημιουργία ενός δυναμικού αρχείου testbench που θα μπορούσαμε να δοκιμάσουμε το περιφερειακό της VHDL και να συγκρίνουμε τα αποτελέσματα με την υλοποίηση μας, ώστε να διαπιστώσουμε αν συμπίπτουν.

Αρχικά, τρέχουμε τον αλγόριθμο μας σε C για 100 αποτελέσματα για 3ψήφια τυχαία strings Εικόνα 27. Σε επόμενο βήμα τρέχουμε το αρχείο bar_generator.c με σκοπό να δώσουμε το padding και να αποθηκεύσουμε τα στοιχεία μας σε ένα .txt αρχείο με όνομα hashes2.txt Εικόνα 26 και σε επόμενο .txt με όνομα inputs.txt Εικόνα 28 τις εισόδους με padding.

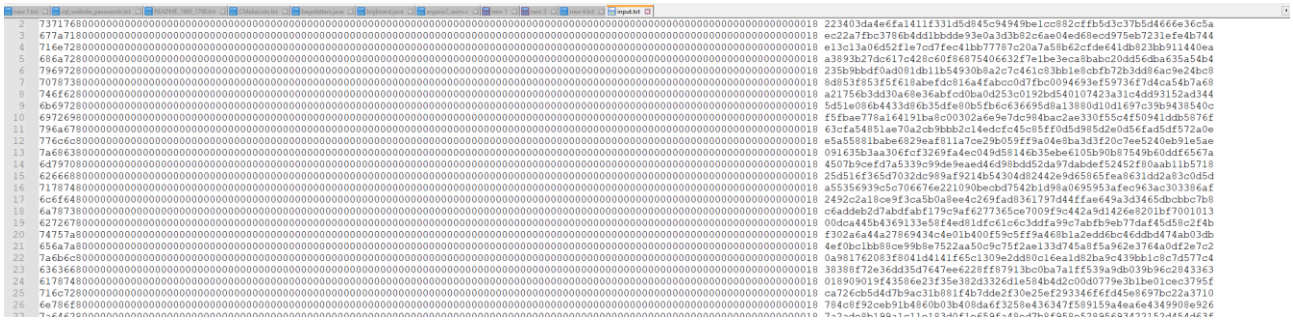


```
INH: 6d746778756d73746a66, OUThash: 65d3dc7d4e1c68be8a11bb728e439b2249508aad94aafbd8ad2c29b75215e1fe
INH: 7567657277746d6d6e68, OUThash: 986518a1b4df3b813fe3c39a912e24e99c2154669075c4743b1cb59fc29b7d84
INH: 6a69626a716366686f78, OUThash: 0139e7c1a88677952f7418d89038b8f1cf7058f04d6b87c0233753e96c8560da
INH: 6e636c67726161786365, OUThash: 0b0747389174abf1be96820533ba9663ed88a18f3aa0b9a9d34241a62a7b6750
INH: 6a64686163686378716b, OUThash: 62f28b140df7e6eb504cd22f59268171d0439614b501b950b864c7084e029a74
INH: 6f6767776f716e726f69, OUThash: 4307fcd6f6844cc8c75f54ecd86f88f412ac5445f28a7e558417f983a32dcb00
INH: 706a6e696f70636e7672, OUThash: be6d91629b02edc5c5422eb83223dfbd5e1f1d26c78b99aa6206abb76920bf73
INH: 6e617a647870656a7870, OUThash: 5c6bec774a5f19a769d802100328257ec2cc9066b1fc5242e25c3917bf965f9c
INH: 7165617376647264636362, OUThash: cba5728552ca066a8e3ff900db719216cb80defa3045b6973e93a3d31f9eddd0
INH: 687763626f7972716470, OUThash: f74c8da37b86cacceabe693f431c11222710cdb533656472bddf7d4e0c494550
INH: 7a796a69716c767a6867, OUThash: c6dd9df9841869b0d43b19cf336bb279aef153d935b86fc598029ccd4940debe
INH: 796a6c78767366726777, OUThash: 678cc7735f710ce9786c35213876de7fdad7daa87b02986fbfa923d58cea55e9
INH: 756f77627465787a7369, OUThash: d4166a00246810bf0551e3f463c83ba484b2896285899a20849fef25695e91f8
INH: 686f626d62746f6f7774, OUThash: 696a99fcab2ec1180c493e61a8e16a3424fea933a15fe3db8f989938110687c7
INH: 6e6d6572666977647369, OUThash: 95308b2b16afa570ac07be3e7828a5646ab009666b8937eedc08b84bf122b976
INH: 7675676672726167637378, OUThash: e18f5b3f4f40000bb06fbd57f3c9a331233953a1a065230579d4fa585dd115ae
INH: 79747a74757a65647167, OUThash: ae74ae3fc621494f2a574d242b8ffe040d6c8ea4006a9da288733f863587cb57
INH: 6473756b6769676a796c, OUThash: 7623fdc0fbcec9a073a343e8145d43ab8164e9eed31604ff5c8b268a51216917
INH: 666f656a656f686c616b, OUThash: 9c0dbcda3e23174791611eb82d6d3b361ae4e75aaef176d98b6f0dcb1921f691
INH: 7862797669777566e616e, OUThash: 7e1af60372b246d2d976ed6504fc1b9178d40211c0b1144827f78747b5742385
INH: 62726a6770736c68776b, OUThash: 7c98685dc4e7d740d77a5871502175bb31715bfd5248000e3b55d83891caa73a
INH: 6677616c67776a70736e, OUThash: b846c2703779a338ab97fb9f5b690452ed68a06808bbb3a667f1c445a9b9417f
INH: 6d6163637a676e616464, OUThash: bf7b1110c435b444a9f99a26af643e52456db6519c94bd6468dd94aee02d2f37
INH: 726e6b6e626e646b6977, OUThash: ceb11f75d198d350070d075710603ba3dc42c984456479df4fe497c6f96a5e9c
INH: 66687a7475756b796d6d, OUThash: 0456eab91ff68540a5dcbd98bab9d9a463287433f5b328f014bc0e4a3d53b0b4
INH: 617366676f7669776e73, OUThash: a36574a66bd9b7f5695a0d537193a93ed5e72f87e93963d75c618597450b869d
INH: 7769677366577637875, OUThash: 80eaee9e870e595153a079d747621b10cfd1595290d0084b99f8c6a2f57460c1
```

Εικόνα 26 Παραδείγματα αποτελεσμάτων Hashing

```c
srand( Seed: time( Time: NULL));

// We create a file pointer for writing the hashed values
FILE* fp = fopen( Filename: "hashes.txt",  Mode: "w");
if (fp == NULL)
{
    perror( ErrMsg: "Error opening file"); //check if file is open
    return 1;
}

// We generate and process 100 random strings
for (int i = 0; i < 100; ++i)
{
    //We generate a random string of length 10 ([11] for the size)
    char random_str[11];  // Including null terminator
    for (int j = 0; j < 10; ++j)
    {
        random_str[j] = 'a' + rand() % 26;  // Random lowercase letter
    }
    random_str[10] = '\0';  // Null terminator to terminate the file

    // Convert the random string to its hexadecimal representation
    char hex_str[21];  // Twice the length of the random string plus one for null terminator
    ascii_to_hex( ascii_str: random_str,  output: hex_str);

    // We calculate the SHA-256 hash of the random string we gave above
    SHA256_struct ctx; // define a struct for the  SHA-256
    uint8_t hash[32];  // SHA-256 produces a 32-byte hash, initialize the matrix to save it
    //call the functions to do the hashing
    sha256_initialization( sha256: &ctx);
    sha256_updating( sha256: &ctx,  data: (const uint8_t*)random_str,  length: strlen( Str: random_str));
    sha256_output( sha256: &ctx, hash);


    // Print the SHA-256 hash (digest) just to see the results
    printf( format: "String %d: %s\n", i + 1, random_str);
    printf( format: "Hexadecimal representation: %s\n", hex_str);
    printf( format: "SHA-256 Hash: ");
    print_hash(hash);
    printf( format: "\n");
    // Write the data to the file
    fprintf( stream: fp,  format: "INs: %s, INH: %s, OUThash: ", random_str, hex_str);
    for (int j = 0; j < 32; ++j)
    {  // SHA-256 produces a 32-byte hash
        fprintf( stream: fp,  format: "%02x", hash[j]);
    }
    fprintf( stream: fp,  format: "\n");
}

// Close the file
```

Εικόνα 27Κώδικας C για αποθήκευση αποτελεσμάτων σε αρχεία



Εικόνα 28 Προσαρμοσμένη είσοδος

Στην συνέχεια θα πρέπει να τρέξουμε ένα τρίτο αρχείο Εικόνα 29, ώστε να δημιουργήσουμε το δυναμικό αρχείο του testbench με τυχαία διανύσματα εισόδου που θα χρησιμοποιήσουμε στην προσομοίωση του Modelsim στο επόμενο κεφάλαιο για να δούμε εάν η υλοποίηση μας είναι επιτυχής.

testbench_generator.c

```c
#include <stdlib.h>

int main() {
    //we open the two files
    FILE *input_file = fopen("input.txt", "r");
    FILE *output_file = fopen("testbench.vhd", "w");

    if (input_file == NULL || output_file == NULL)
    {
        perror("Error opening files");
        return EXIT_FAILURE;
    }

    fprintf(output_file, "LIBRARY ieee;\n");
    fprintf(output_file, "USE ieee.std_logic_1164.ALL;\n\n");

    fprintf(output_file, "ENTITY sha256for1chunkTB IS\n");
    fprintf(output_file, "END sha256for1chunkTB;\n\n");

    fprintf(output_file, "ARCHITECTURE behavior OF sha256for1chunkTB IS\n\n");
    fprintf(output_file, "  -- Component Declaration for the Unit Under Test (UUT)\n\n");
    fprintf(output_file, "  COMPONENT sha256for1chunk\n");
    fprintf(output_file, "      Port ( reset   : in  STD_LOGIC;\n");
    fprintf(output_file, "                clock   : in  STD_LOGIC;\n");
    fprintf(output_file, "                plain   : in  STD_LOGIC_VECTOR(511 downto 0);\n");
    fprintf(output_file, "                load : in  STD_LOGIC;\n");
    fprintf(output_file, "                empty  : out STD_LOGIC;\n");
    fprintf(output_file, "                digest : out STD_LOGIC_VECTOR (255 downto 0);\n");
    fprintf(output_file, "                ready  : out STD_LOGIC);\n");
    fprintf(output_file, "  END COMPONENT;\n\n");

    fprintf(output_file, "  --Inputs\n");
    fprintf(output_file, "  signal reset : std_logic := '0';\n");
    fprintf(output_file, "  signal clock : std_logic := '0';\n");
    fprintf(output_file, "  signal plain : std_logic_vector(511 downto 0) := (others => '0');\n");
    fprintf(output_file, "  signal load : std_logic := '0';\n\n");

    fprintf(output_file, "  --Outputs\n");
    fprintf(output_file, "  signal digest : std_logic_vector(255 downto 0);\n");
    fprintf(output_file, "  signal ready, empty : std_logic;\n\n");

    fprintf(output_file, "  -- Clock period definitions\n");
    fprintf(output_file, "  constant clock_period : time := 10 ns;\n\n");

    fprintf(output_file, "  -- Done Signal\n");
    fprintf(output_file, "  signal sigdone : std_logic := '0';\n\n");

    fprintf(output_file, "-- Function vhdl_vec2int()\n");
    fprintf(output_file, "function vec2int (vec1: std_logic_vector) return integer is\n");
    fprintf(output_file, "variable retval: integer := 0;\n");
    fprintf(output_file, "alias vec: std_logic_vector(vec1'length-1 downto 0) is vec1;\n");
    fprintf(output_file, "begin\n");
    fprintf(output_file, "  for i in vec'high downto 1 loop\n");
    fprintf(output_file, "  if (vec(i)='1') then\n");
    fprintf(output_file, "      retval:=(retval+1)*2;\n");
    fprintf(output_file, "  else retval:=retval*2;\n");
    fprintf(output_file, "  end if;\n");
```

Εικόνα 29 Δυναμικό δοκιμαστικό πρόγραμμα

3.3.9 Σύνοψη Τρίτου Κεφαλαίου

Το τρίτο κεφάλαιο ασχολήθηκε με την κρυπτογραφία και τις κρυπτογραφικές συναρτήσεις κατακερματισμού, εστιάζοντας στη σύγκριση και την ιστορική εξέλιξη των αλγορίθμων SHA-1 και SHA-256. Παρουσιάστηκαν οι θεμελιώδεις έννοιες της κρυπτογράφησης και η σημασία της για την ασφάλεια των δεδομένων.

Αρχικά, δόθηκε μια εισαγωγή στην κρυπτογράφηση, εξηγώντας τον ρόλο της στην προστασία της εμπιστευτικότητας, της ακεραιότητας και της αυθεντικότητας των δεδομένων. Στη συνέχεια, συγκρίθηκαν οι αλγόριθμοι SHA-1 και SHA-2, επισημαίνοντας τις διαφορές στην ασφάλεια και την απόδοση. Ο SHA-1 θεωρήθηκε πλέον μη ασφαλής λόγω των ευπαθειών που είχαν ανακαλυφθεί, ενώ ο SHA-2, με τις διάφορες παραλλαγές του, προσέφερε αυξημένη ασφάλεια και ευρύτερη χρήση.

Ακολούθως, δόθηκε λεπτομερής επεξήγηση του αλγορίθμου SHA-256, μιας από τις πιο διαδεδομένες εκδοχές του SHA-2. Αναλύθηκαν τα κύρια χαρακτηριστικά του SHA-256, όπως η διαδικασία κατακερματισμού, η δομή των μπλοκ, οι μαθηματικές λειτουργίες και η ασφάλεια που προσέφερε.

Κεφάλαιο 4: Σχεδίαση και υλοποίηση

Στο κεφάλαιο αυτό θα αναλύσουμε τον τρόπο σχεδίασης και θα σκιαγραφήσουμε την υλοποίηση του συστήματος κρυπτογράφησης. Αρχικά, θα παρουσιάσουμε ορισμένα στοιχεία σχετικά με την πλακέτα FPGA που πραγματοποιήθηκε η υλοποίηση μας καθώς και μερικές πληροφορίες για την δομή του επεξεργαστή που χρησιμοποιήθηκε. Έπειτα, θα σκιαγραφήσουμε και θα εξερευνήσουμε την δομή του συστήματος στο πρόγραμμα Vivado και θα δούμε τα αποτελέσματα της υλοποίησης μας. Τέλος, θα παρουσιαστεί η εγκατάσταση του λειτουργικού Linux στην προγραμματιζομένη πλακέτα και θα παρουσιαστεί ο προγραμματισμός του επεξεργαστή με την χρήση μιας βιβλιοθήκης της Python.

4.1 FPGA Kria KR260 Robotics Starter Kit

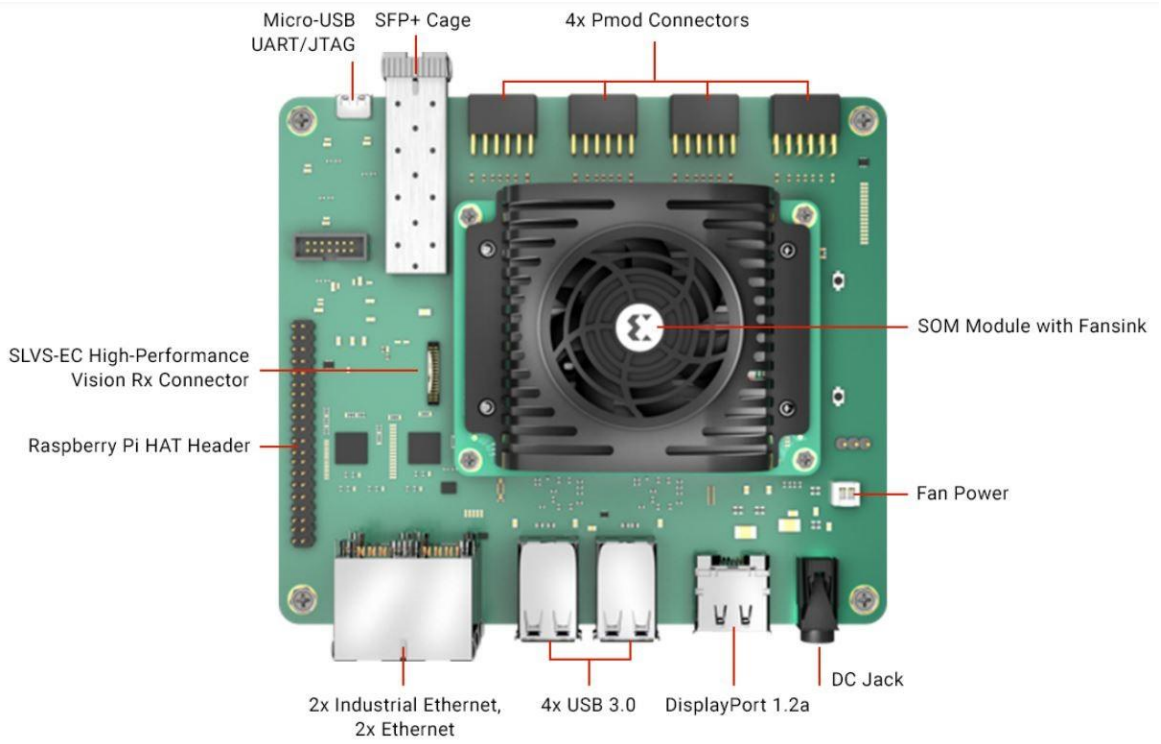
Η Kria KR260 Robotics Starter Kit Εικόνα 30 είναι η μια από τις πιο πρόσφατες πλατφόρμες ανάπτυξης που έχει στην διάθεση της η εταιρία της AMD®. Το Starter Kit είναι μια πλατφόρμα για την ανάπτυξη εφαρμογών ρομποτικής, μηχανικής όρασης, βιομηχανικής επικοινωνίας και ελέγχου [33]. Σαν κύριος δομικός πυλώνας έχει μια K26 SOM (System on Module) που συνδέεται σε μια robotics carrier card και παράλληλα εξοπλίζεται με μια ενεργή θερμική λύση ανεμιστήρα και ψήκτρας. Το SOM στο Starter Kit βασίζεται στην αρχιτεκτονική Zynq UltraScale+ MPSoC EV που συνδυάζεται με 4 GB μνήμης DDR4 αρκετά γρήγορων ταχυτήτων Εικόνα 31. Γενικά, η πλακέτα εξειδικεύεται στην κατασκευή εφαρμογών για επιτάχυνση υλικού και εμπεριέχει πακέτα με επιτάχυνση υλικού ROS 2 για τη ρομποτική με το Open Source Kria Robotics Stack (KRS) που διευκολύνει τους προγραμματιστές ROS 2 να κατασκευάσουν ρομπότ υψηλής απόδοσης που ανταποκρίνονται στις νέες απαιτήσεις της αγοράς.

Συγκεκριμένα, το SOM είναι πολύ συμπαγές και περιλαμβάνει κυρίως βασικά στοιχεία σε σχέση με άλλα πολύ μεγαλύτερα της αγοράς, όπως μια συσκευή πυριτίου με βάση το Zynq® UltraScale+™ MPSoC, μνήμη, μια μονάδα εκκίνησης και μια μονάδα ασφαλείας. Η κάρτα επιτρέπει διάφορες επιλογές διασύνδεσης, όπως πολλαπλές διασυνδέσεις Ethernet, συνδεσιμότητα SFP+, μια διασύνδεση αισθητήρα SLVS-EC και μια κάρτα microSD. Παράλληλα, όπως προαναφέραμε η πλακέτα περιλαμβάνει μια ψήκτρα με κάλυμμα και έναν ανεμιστήρα για να εξασφαλίσει την μείωση των θερμοκρασιών στο σύστημα επεξεργασίας.

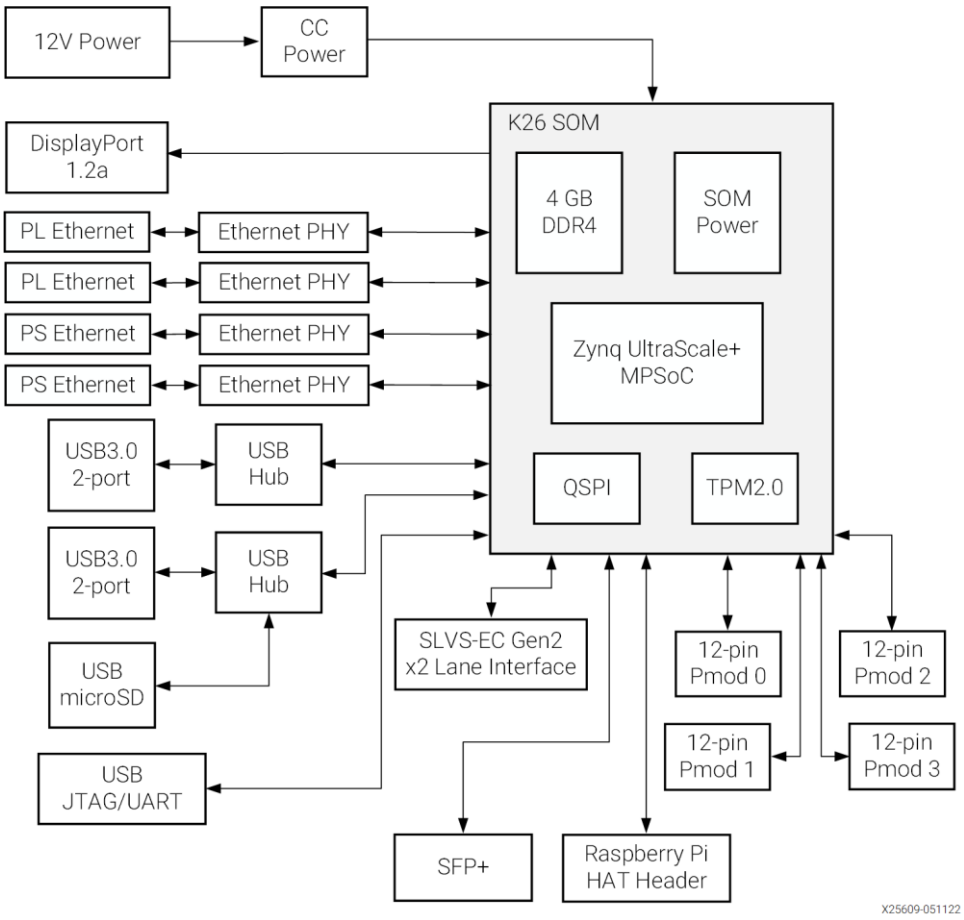
Σε αυτό το σημείο θα πρέπει να υπογραμμίσουμε ότι οι κύριες εφαρμογές που έχει ως στόχο η συγκεκριμένη πλακέτα είναι η κατασκευή και η σχεδίαση εφαρμογών που έχουν αντίκτυπο στον

αυτοματισμό των εργοστασίων, στην επικοινωνία, στον έλεγχο των ρομποτικών συστημάτων και στην μηχανική όραση.

Το κιτ εμπεριέχει ένα SOM βασισμένο στο Zynq UltraScale+ MPSoC με περιφερειακά που επιλέγονται από τον χρήστη και εστιάζουν στον τομέα της ρομποτικής, καθώς και ένα σύνολο προ-κατασκευασμένων επιταχυνόμενων εφαρμογών που ωστόσο μπορούν να δημιουργηθούν και νέες. Το KR260 και το βασικό K26 SOM υποστηρίζονται για πλήρη προσαρμογή από τον χρήστη μέσου του Vitis παρέχοντας την δυνατότητα για νέες εφαρμογές, των προσαρμόσιμων επικαλύψεων επιτάχυνσης και των αρχείων υλικού της πλακέτας των εργαλείων Vivado®. Ο ολοκληρωμένος συνδυασμός υλικού, πλατφόρμας και λογισμικού παρέχει μια γρήγορη out-of-box εμπειρία για τους προγραμματιστές [33], η οποία μπορεί στη συνέχεια να αξιοποιηθεί για τον σχεδιασμό προϊόντων που θα διευκολύνουν το ανθρώπινο σύνολο.



Εικόνα 30 Kria KR260 Robotics Starter Kit Image



Εικόνα 31 Kria KR260 Robotics Starter Kit διάγραμμα ροής δεδομένων[8]

Πίνακας 2 Kria KR260 Robotics Starter Kit Λεπτομέρειες προϊόντος

Προδιαγραφές	Περιγραφή
Λύση θερμικής ψύξης	Ενεργό
Βάρος	235g
Παράγοντας μορφής	SOM + Κάρτα φορέα
Διαστάσεις της KR260 Starter Kit	132 mm x 140mm x 36mm
Διαστάσεις του SOM με θερμική απαγωγή	60mm x 77mm x 27mm
Διαστάσεις της κάρτας φορέα	132 mm x 140mm x 34mm
Κύτταρα λογικής συστήματος	256k
Μπλοκ RAM μπλοκ	144
UltraRAM blocks	64
DSP τεμάχια	1.2k
Διεπαφή Ethernet	One PS Gb RGMII Ethernet One PS Gb SGMII Ethernet Δύο PL Gb Ethernet (RGMII) με υποστήριξη για χρονοευαίσθητη δικτύωση (TSN) και Ethernet για τεχνολογία αυτοματισμού ελέγχου (EtherCAT)

[8] Η εικόνα προέρχεται από το https://docs.amd.com/r/en-US/ds988-kr260-starter-kit/Product-Details.

Ενισχυμένος βυσματικός σύνδεσμος μικρού συντελεστή μορφής	Υποδοχή x1 SFP+ που υποστηρίζει 10GigE
Επεκτάσιμη σηματοδότηση χαμηλής τάσης με ενσωματωμένο ρολόι (SLVS-EC)	Διασύνδεση x1 SLVS-EC Gen2 x2 λωρίδες
Μνήμη DDR	512 Mb QSPI
Πρωτεύουσα μνήμη εκκίνησης	4 GB (4 x 512 Mb x 16 bit) [non-ECC] DDR4
Δευτερεύουσα μνήμη εκκίνησης	Zynq UltraScale+ MPSoC ρίζα εμπιστοσύνης υλικού (RoT) για υποστήριξη ασφαλούς εκκίνησης Infineon TPM2.0 για υποστήριξη μετρημένης εκκίνησης
Διασύνδεση τύπου Raspberry Pi hardware ενσωματωμένο στο πάνω (HATs)	x1
PMOD 12-pin διεπαφή	x4
USB3.0 διεπαφή	x2 USB 3.0/2.0 downstream (Host), το καθένα με δύο φυσικές θύρες χρήστη
DisplayPort 1.2a	x1 έξοδος βίντεο DisplayPort που υποστηρίζει 1920 x 1080 στα 60 Hz

Μερικές γενικές πληροφορίες σχετικά με την κατανάλωση ισχύος και ενέργειας καθώς και τις ανοχές της πλακέτας φαίνονται στο παρακάτω πίνακα.

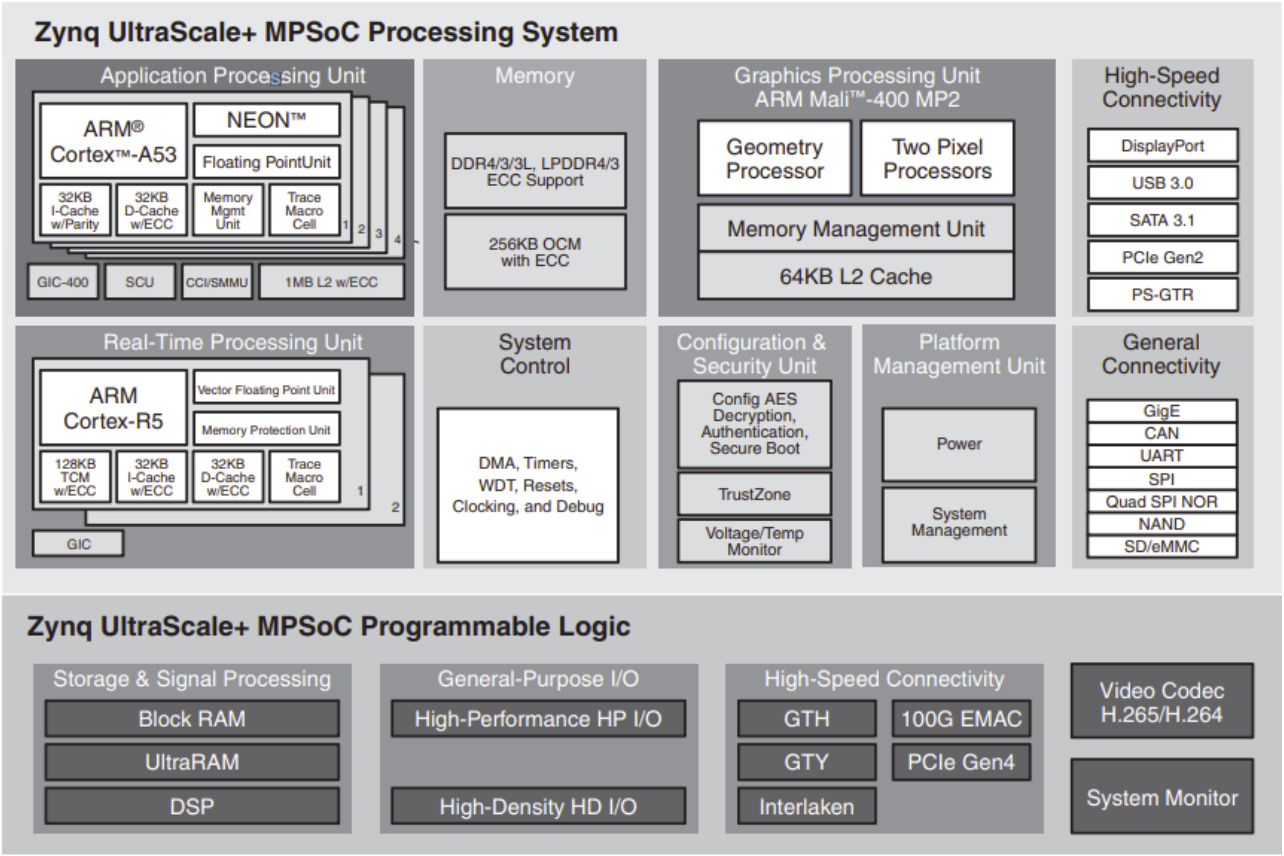
Πίνακας 3 Συνθήκες θερμοκρασίας λειτουργίας και αποθήκευσης

Specification	Condition
Θερμοκρασία λειτουργίας	0°C to 35°C
Θερμοκρασία αποθήκευσης	-40°C to 75°C
Υγρασία λειτουργίας, χωρίς συμπύκνωση	8% to 90%, και σημείο δρόσου -12°C
Υγρασία αποθήκευσης, χωρίς συμπύκνωση	5% to 95%

4.1.1 Zynq UltraScale+ MPSoC

Ο Zynq UltraScale+ MPSoC αποτελεί ένα SoC (System on Chip) Εικόνα 32 που εμπεριέχει πολλαπλές μηχανές επεξεργασίας, έναν κατάλογο από περιφερειακές συσκευές υψηλής ταχύτητας, προηγμένες δυνατότητες εισόδου και εξόδου καθώς και προγραμματιστική λογική μονάδα (PL - Programmable Logic) Εικόνα 32. Ο επεξεργαστικός πυρήνας του, περιλαμβάνει μια τετραπύρηνη APU (Accelerated Processing Unit) με βάση τον ARM® Cortex™ A53, μια διπύρηνη APU (Accelerated Processing Unit) με βάση τον ARM Cortex R5 RPU, μια μονάδα επεξεργασίας γραφικών Mali, μια μονάδα διαχείρισης πλατφόρμας και μια μονάδα κωδικοποιητή βίντεο (VCU) [34]. Ο Zynq UltraScale+ MPSoC προσφέρει ένα ευρύ φάσμα επιλογών διασύνδεσης, μπλοκ επεξεργασίας ψηφιακών σημάτων (DSP) και δυνατότητες προγραμματιζόμενης λογικής (PL), παρέχοντας έτσι εξαιρετική ευελιξία για να ανταποκρίνεται σε ποικίλες απαιτήσεις εφαρμογών. Οι προηγμένες δυνατότητες διασύνδεσης επιτρέπουν την εύκολη ενσωμάτωση με διάφορα συστήματα, ενώ τα μπλοκ DSP παρέχουν ισχυρές υπολογιστικές δυνατότητες για επεξεργασία σήματος σε πραγματικό χρόνο. Επιπλέον, η προγραμματιζόμενη λογική επιτρέπει την προσαρμογή και βελτιστοποίηση των λειτουργιών του συστήματος για ειδικές εφαρμογές, καθιστώντας το Zynq UltraScale+ MPSoC

κατάλληλο για χρήση σε ένα ευρύ φάσμα βιομηχανικών, επιστημονικών και καταναλωτικών εφαρμογών.



Εικόνα 32 Αρχιτεκτονική του επεξεργαστή Zynq UltraScale+ MPSoC[9]

Ο Zynq UltraScale+ MPSoC βασίζεται στην τεχνολογία υλοποίησης 16nm FinFET από την TSMC (Taiwan Semiconductor Manufacturing Company), με αποτέλεσμα μεγαλύτερες επιδόσεις και χαμηλότερη κατανάλωση ισχύος, επιτρέποντας το σχεδιασμό αποδοτικών ως προς την ισχύ συστημάτων πολυμέσων επόμενης γενιάς.

Ενσωματωμένη Προγραμματιστική Λογική - Programmable Logic (PL)

Οι πυρήνες ARM Cortex-A53, σε συνδυασμό με τη μονάδα μνήμης και τα πολλαπλά περιφερειακά του Zynq UltraScale+ MPSoC Εικόνα 33, διαδραματίζουν κρίσιμο ρόλο στη διαχείριση και τη σύλληψη δεδομένων από διάφορες πηγές [34]. Τα περιφερειακά του συστήματος, όπως οι διασυνδέσεις USB και Ethernet, μπορούν να χρησιμοποιηθούν για τη σύνδεση με συσκευές ροής βίντεο, όπως βιντεοκάμερες, κάμερες δικτύου και κάμερες web. Για παράδειγμα, οι μονάδες IP όπως οι SDI RX, HDMI RX και MIPI CSI μπορούν να χρησιμοποιηθούν για τη σύλληψη ακατέργαστου βίντεο από διάφορες πηγές.

Η Xilinx παρέχει μια εκτενή συλλογή από IPs διασύνδεσης οθόνης, διαθέσιμη στον κατάλογο IP του Xilinx Vivado. Οι χρήστες μπορούν να αδειοδοτήσουν αυτές τις IPs για την υλοποίηση διεπαφών οθόνης στο PL (Programmable Logic), επιτρέποντας τη δημιουργία προσαρμοσμένων και αποδοτικών λύσεων για την επεξεργασία και την προβολή βίντεο.

[9] Η εικόνα προέρχεται από το https://www.core-vision.nl/events/zynq-ultrascale-mpsoc-for-the-hardware-designer-21/?lang=en .

Any-to-Any συνδεσιμότητα με την χρήση του Vivado και SDK εργαλείων

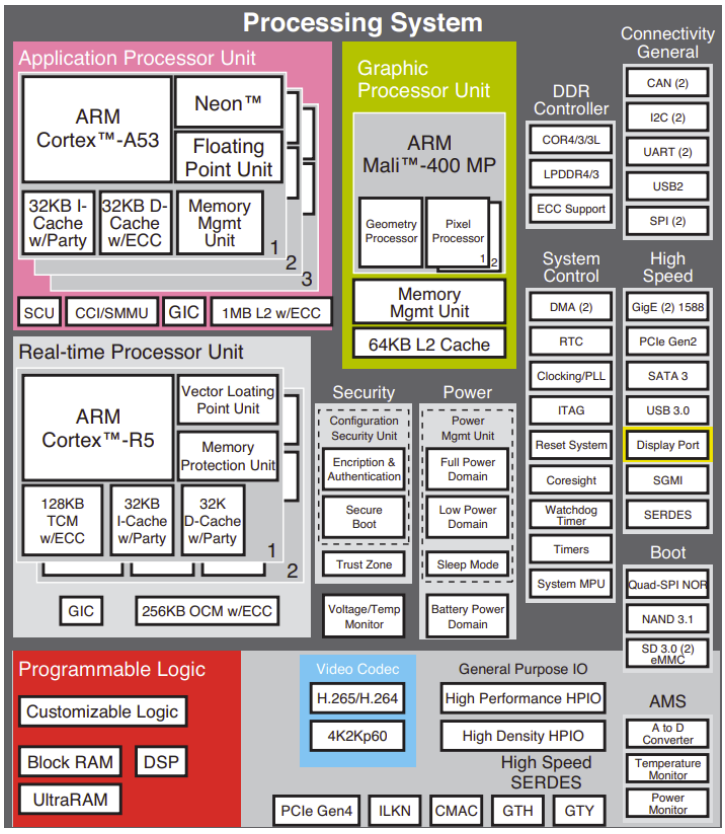
Για τη διαφοροποίηση του υλικού, πολλοί προγραμματιστές πλατφορμών χρησιμοποιούν το Programmable Logic (PL) για τη συνδεσιμότητα "any-to-any" [35]. Το Vivado Design Suite διευκολύνει τους σχεδιαστές συστημάτων στην ανάπτυξη των συστημάτων τους, παρέχοντας τις ακόλουθες σουίτες σχεδίασης:

- **Εργαλείο Ολοκληρωτή IP (IPI - IP Integrator)**: Χρησιμοποιεί μια προσέγγιση block-diagram, επιτρέποντας την ενσωμάτωση διαφόρων IPs για τη δημιουργία ενός ολοκληρωμένου συστήματος.

- **Οδηγός Διαμόρφωσης PS (PCW – Processing Configuration Wizard)**: Επιτρέπει στους χρήστες να διαμορφώσουν, να ενεργοποιήσουν ή να απενεργοποιήσουν τα περιφερειακά του Processing System (PS) και να υλοποιήσουν ρυθμίσεις για το ρολόι και τη μνήμη.

- **Κατάλογος IP**: Μέσω του IPI, τα IPs μπορούν να ενσωματωθούν σε ένα ευρύτερο σύστημα. Επιπλέον, οι χρήστες μπορούν να προσθέσουν και να χρησιμοποιήσουν τα δικά τους προσαρμοσμένα αποθετήρια IP στον κατάλογο IP.

Η χρήση του Vivado Design Suite προσφέρει σημαντικά πλεονεκτήματα στην ανάπτυξη προσαρμοσμένων συστημάτων, επιτρέποντας την ευέλικτη και αποτελεσματική ενσωμάτωση διαφόρων λειτουργικών μονάδων.



Εικόνα 33 Πλήρης δομική σύνθεση του Zynq UltraScale+ MPSoC[10]

[10] Η εικόνα προέρχεται από το https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-ultrascale-plus-mpsoc.html .

Ο Zynq UltraScale+ MPSoC αποτελεί την ιδανική λύση για ομάδες με περιορισμένους ή ανύπαρκτους πόρους υλικού, οι οποίες συχνά αντιμετωπίζουν δυσκολίες λόγω της απαραίτητης τεχνογνωσίας στην ανάπτυξη RTL (VHDL ή Verilog) στην αξιοποίηση των πλεονεκτημάτων μιας προγραμματιζόμενης συσκευής. Η Xilinx κυκλοφόρησε το εργαλείο SDSoC™, το οποίο προσφέρει μια οικεία εμπειρία ανάπτυξης εφαρμογών embedded C/C++/OpenCL, περιλαμβάνοντας ένα εύχρηστο Eclipse IDE και ένα ολοκληρωμένο περιβάλλον σχεδιασμού για ετερογενή ανάπτυξη στον Zynq UltraScale+ MPSoC [35].

Το εργαλείο SDSoC™ μειώνει τον χρόνο προγραμματισμού, παρέχοντας προφίλ σκιαγράφησης σε επίπεδο συστήματος, αυτοματοποιημένη επιτάχυνση λογισμικού, αυτοματοποιημένη συνδεσιμότητα συστήματος και βιβλιοθήκες. Επιπλέον, επιτρέπει στους προγραμματιστές να ορίζουν, να ενσωματώνουν και να επαληθεύουν λύσεις σε επίπεδο συστήματος γρήγορα και να παρέχουν στους τελικούς πελάτες ένα προσαρμοσμένο περιβάλλον προγραμματισμού.

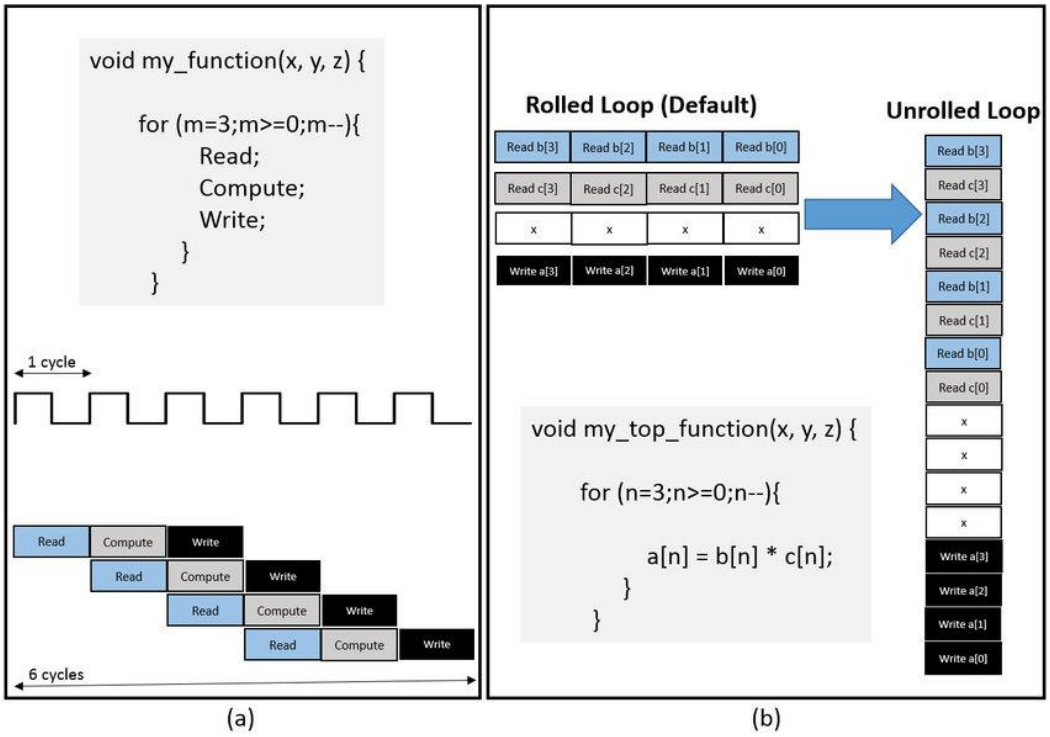
Οι πολύπλοκοι αλγόριθμοι για ανάλυση και επεξεργασία δεδομένων βίντεο, γραμμένοι σε υψηλού επιπέδου λογισμικό, μπορούν να επιταχυνθούν στο PL χρησιμοποιώντας το εργαλείο SDSoC™, συντομεύοντας έτσι τον κύκλο ανάπτυξης και αυξάνοντας την απόδοση. Αυτή η προσέγγιση προσφέρει σημαντικά πλεονεκτήματα σε όρους χρόνου ανάπτυξης και αποδοτικότητας, καθιστώντας τον Zynq UltraScale+ MPSoC μια εξαιρετικά ευέλικτη και αποτελεσματική επιλογή για εφαρμογές υψηλής απόδοσης.

4.2 Σχεδιασμός συστήματος στο επίπεδο του Υλικού

4.2.1 Υλοποίηση περιφερειακού κωδικοποίησης

Για να μεταβούμε από το επίπεδο του λογισμικού στο επίπεδο του υλικού απαιτούνται ορισμένες τροποποιήσεις στον κώδικα στο Vitis. Αρχικά, θα πρέπει λίγο να περιγράψουμε ορισμένα directives που μπορούν να αξιοποιηθούν για την βελτίωση του επιμέρους κώδικα και τις γενικής υλοποίησης μας. Γενικά, υπάρχουν συναρτήσεις της HLS που μπορούν να χρησιμοποιηθούν για να μειώσουμε των χρόνο εκτέλεσης ορισμένων επαναλήψεων ή για να μειώσουμε τον χώρο αποθήκευσης στην μνήμη, ώστε να βελτιωθεί το SoC που θα παράγει το πρόγραμμα της Xilinx. Εμείς στην συγκεκριμένη υλοποίηση χρησιμοποιήσαμε κυρίως την παρακάτω συνάρτηση:

#pragma HLS UNROLL factor= region skip_exit_check: Το συγκεκριμένο directive Εικόνα 34 που υπάγεται στις βιβλιοθήκες της HLS μπορεί να αξιοποιηθεί μέσα σε μια δομή επανάληψης και την μετασχηματίζει με σκοπό την δημιουργία πολλαπλών αντιγράφων με το περιεχόμενο της στο RTL που βγαίνει μετά το στάδιο της σύνθεσης [36]. Με την συγκεκριμένη μέθοδο, ορισμένες ή κάποιες επαναλήψεις εκτελούνται παράλληλα, επιταχύνοντας την διαδικασία εκτέλεσης του συγκεκριμένου κομματιού κώδικα. Η συγκεκριμένη συνάρτηση έχει και έναν παράγοντα factor ο οποίος αν δεν προσδιοριστεί η δομή επανάληψης επιβάλλεται να «ξεδιπλωθεί» ολόκληρη, αλλιώς σε διαφορετική περίπτωση «ξεδιπλώνονται» οι factor επαναλήψεις της. Επιπλέον, η παράμετρος region αποτελεί προαιρετική προσθήκη στην συνάρτηση, καθώς αναπτύσσει όλες τις λούπες που τοποθετούνται κάτω από τη θέση που έχει τοποθετηθεί η συγκεκριμένη μέθοδος pragma αφήνοντας την εξωτερική λούπα διπλωμένη (χρήση σε διπλή λούπα επανάληψης κυρίως). Η παράμετρος «skip_exit_check» αξιοποιείται όταν έχουμε μερικό unrolling.



Εικόνα 34 Παράδειγμα directive HLS unroll[11]

Σε αυτή την περίπτωση, πρέπει να προσδιοριστεί το factor και να είναι μικρότερο σε σχέση με τις πλήρεις επαναλήψεις της δομής επανάληψης, έτσι ώστε η συγκεκριμένη παράμετρος να λειτουργήσει σαν δομή ελέγχου για τερματισμό της λούπας επανάληψης ή αλλιώς όπως είναι ευρέως γνωστό σαν break. Αυτή η παράμετρος μπορεί να είναι πολύ χρήσιμη σε περιπτώσεις που θέλουμε να επιβάλλουμε κάποιο interrupt ή κάποιον έλεγχο σε μια λούπα για να εκτελεστεί λιγότερες φορές. Θα πρέπει σε αυτό το σημείο να τονίσουμε ότι εμείς χρησιμοποιήσαμε σε ορισμένες επαναλήψεις την συγκεκριμένη συνάρτηση pragma με σκοπό την επίτευξη καλύτερων χρόνων και πιο σταθερού αποτελέσματος σε θέμα ταχύτητας throuput. Αναλυτικότερα, ο βασικός περιορισμός της συγκεκριμένης δομής είναι ότι δεν μπορεί να αξιοποιηθεί σε δομές επανάληψης η οποία έχει πάρα πολλές επαναλήψεις δηλαδή πολύ μεγάλο αριθμό. Επιπλέον, δεν μπορεί να χρησιμοποιηθεί σε μεταβλητό αριθμό επαναλήψεων και επίσης αν χρησιμοποιηθεί το factor θα πρέπει να υπάρχει είναι πολλαπλάσιο του αριθμού των επαναλήψεων. Εναλλακτική λύση, αποτελεί η δομή directive PIPELINE που ωστόσο απαιτεί ο βρόχος επανάληψης να είναι παραλληλοποιήσιμος, δηλαδή να μην υπάρχουν εξαρτήσεις ανάμεσα στις μεταβλητές επανάληψης καθώς και την μετατροπή της λούπας σε δομή τύπου while σύμφωνα με το τεχνικό φυλλάδιο της Xilinx [37].

Ως τελευταία συνάρτηση Εικόνα 35 είναι ορισμένη η συνάρτηση του HLS που θα πρέπει να οριστεί μέσα στο περιβάλλον προγραμματισμού σαν Top function και θα μας επιτρέψει να μεταβούμε από το επίπεδο του λογισμικό στο υλικό. Αυτή είναι η συνάρτηση που αντιπροσωπεύει τον αλγόριθμο και γενικά το περιφερειακό, όταν θα βγει η RTL περιγραφή και θα φορτωθεί σε επόμενο χρόνο στο λειτουργικό πρόγραμμα Vivado, που θα δούμε παρακάτω. Εμπεριέχει κάποια βασικά χαρακτηριστικά του αλγορίθμου που θα χρησιμοποιηθούν σαν είσοδος στο περιφερειακό, όπως για παράδειγμα το μήκος της λέξης για hashing και ένας πίνακας για τα δεδομένα. Επιπλέον, ορίζονται ορισμένα πρωτόκολλα που θα αναλύσουμε σε επόμενη υποενότητα για την μεταφορά των δεδομένων από τον buffer του επεξεργαστή στο SoC που θα έχουμε δημιουργήσει και είναι το

[11] Η εικόνα προέρχεται από το https://www.researchgate.net/figure/a-Loop-Pipelining-and-b-Loop-Unrolling-These-operatives-are-used-for-achieving_fig6_334855119 .

axi και το axilite, ένα σειριακό και ένα παράλληλο πρωτόκολλο. Επίσης, ορίζονται όλες οι συναρτήσεις που προαναφέραμε στο προηγούμενο κεφάλαιο.

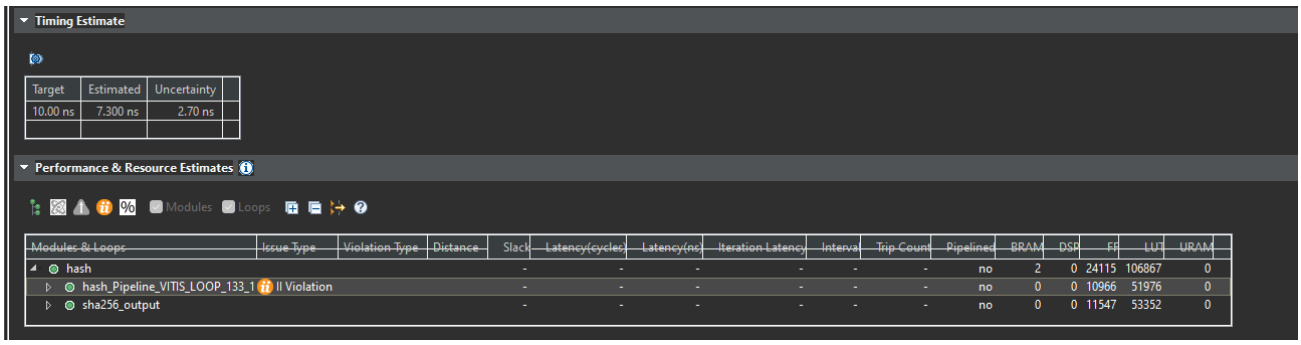
```
// code for the hls module
//uses the m_axi and s_axilite protocols to send the data
int hash(int text_length, uint8_t text_input[1024], uint8_t result[SHA256_BLOCK_SIZE]) {
#pragma HLS INTERFACE s_axilite port=text_length
#pragma HLS INTERFACE m_axi depth=1024 port=text_input
#pragma HLS INTERFACE s_axilite port=result
#pragma HLS INTERFACE s_axilite port=return

    SHA256_struct sha256;
    sha256_initialization(&sha256);
    sha256_updating(&sha256, text_input, text_length);
    sha256_output(&sha256, result);

    return true;
}
```

Εικόνα 35 Top Level Function HLS

Εφόσον έχουμε γράψει τον αλγόριθμο και έχουμε ορίσει τις μεταβλητές μας καθώς και την Top συνάρτηση που θα παράγει το περιφερειακό μας, εκτελούμε την διαδικασία της σύνθεσης – Synthesis που απαιτεί κάποια λεπτά για την δημιουργία της περιγραφής μας και την διασφάλισης της ποιότητας της. Εφόσον τελειώσει η διαδικασία, μπορούμε να δούμε ένα παράθυρο όπου αναπαρίστανται ορισμένα στοιχεία πολύ σημαντικά τόσο για την λειτουργία όσο και για την απόδοση του κώδικα μας.

Γενικά υπάρχουν ορισμένες τιμές που γίνονται κυρίως σε τελικό βήμα η αξιολόγηση του αλγορίθμου και του component Εικόνα 36 που βγάζει το πρόγραμμα Vitis [38]. Οι τρεις βασικές μετρήσεις είναι οι ακόλουθες:

- ✓ **Χρησιμοποίηση πόρων (Area):** Η συγκεκριμένη μετρική αφορά τους πόρους που καταναλώνονται για την υλοποίηση του σχεδίου κατά την σύνθεση του συγκεκριμένου μπλοκ κώδικα. Οι συνηθέστεροι πόροι που αναφέρονται στην παρούσα διπλωματική είναι η χρήση των διαθέσιμων LUT.
- ✓ **Χρόνος εκτέλεσης (Latency):** Ο χρόνος αυτός αναφέρεται από την στιγμή που θα ξεκινήσει η εκτέλεση μιας συνάρτησης μέχρι την στιγμή που θα παράγει όλες τις τιμές εξόδου της. Στο επίπεδο σχεδιασμού που αναφερόμαστε αλλά και γενικά στα FPGAs υπάρχει η μέτρηση του με την χρήση κύκλων ρολογιού ή αλλιώς clocks cycles.
- ✓ **Διάστημα επανεργοποίησης (Initiation Interval II):** Το διάστημα επανεργοποίησης του συγκεκριμένου στοιχείου, όπως αναφέρεται από την Xilinx είναι οι κύκλοι που απαιτούνται από την συνάρτηση επιτάχυνσης πυρήνα πάνω πάνω στο υλικό, προτού μπορέσει να δεχτεί νέα είσοδο τιμών. Η βελτίωση της συγκεκριμένης μετρικής μπορεί να επιτευχθεί με μεθόδους παραλληλοποίησης και μείωσης των επαναλήψεων εκτέλεσης μέσα στις ρουτίνες.



Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.300 ns	2.70 ns

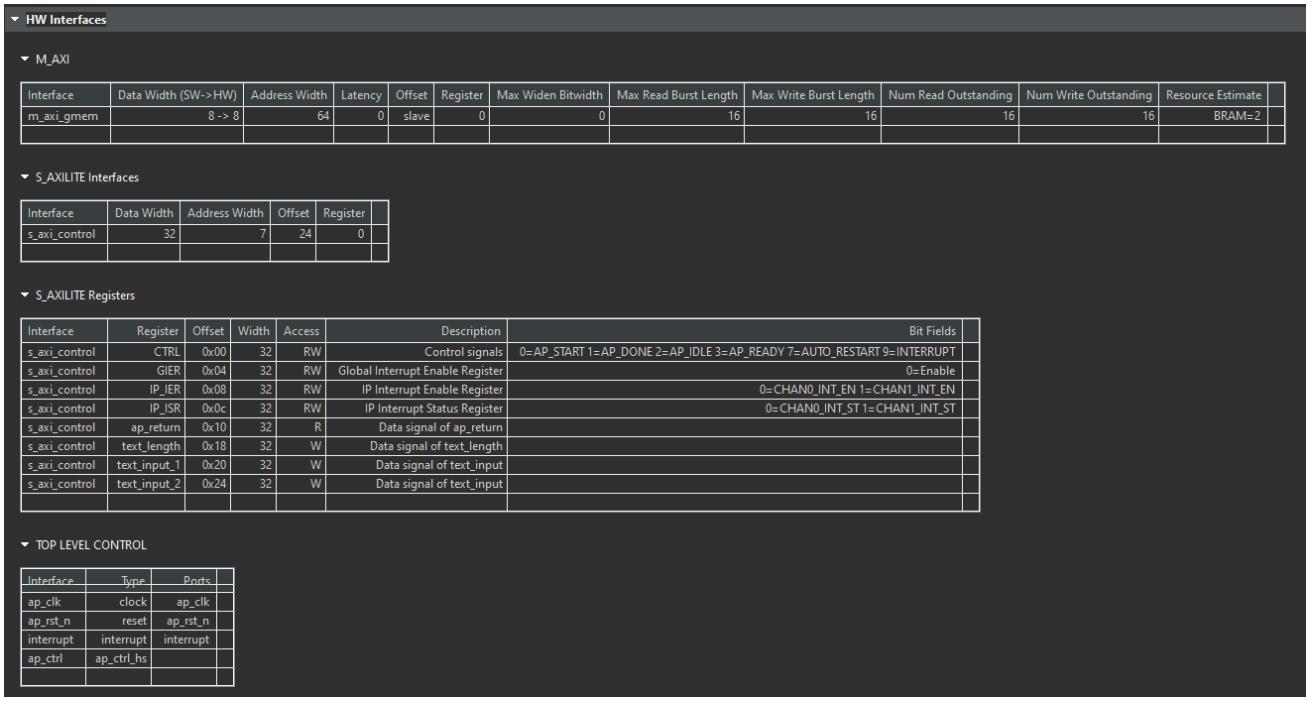
Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
hash				-	-	-	-	-	-	no	2	0	24115	106867	0
hash_Pipeline_VITIS_LOOP_133_1	II Violation			-	-	-	-	-	-	no	0	0	10966	51976	0
sha256_output				-	-	-	-	-	-	no	0	0	11547	53352	0

Εικόνα 36 Πίνακες αναζήτησης for SHA-256 Component

Επιπλέον, μπορούμε να αντλήσουμε πληροφορίες σχετικά με το πρωτόκολλα που χρησιμοποιήσαμε, όπως την θέση μνήμης που βρίσκονται ή το μέγεθος του bus που θα χρησιμοποιήσουμε ή για παράδειγμα αν θα πρέπει να λειτουργήσουν σαν slave ή master, ορολογίες που θα επεξηγήσουμε πιο αναλυτικά σε παρακάτω ενότητα. Επίσης, κάτι που είναι πολύ σημαντικό και θα μας βοηθήσει παρακάτω στην υλοποίηση είναι κάποιες συγκεκριμένες μεταβλητές και οι τιμές που παίρνουν με σκοπό τον έλεγχο των πρωτοκόλλων επικοινωνίας και αποστολής των δεδομένων, τιμές όπως για παράδειγμα το DONE=2 ή το START=1, που όπως υποδηλώνουν και οι ονομασίες τους αποτελούν σήματα για την έναρξη και τον τερματισμό της αποστολής των δεδομένων μας Εικόνα 37.



HW Interfaces

M_AXI

Interface	Data Width (SW->HW)	Address Width	Latency	Offset	Register	Max Widen Bitwidth	Max Read Burst Length	Max Write Burst Length	Num Read Outstanding	Num Write Outstanding	Resource Estimate
m_axi_gmem	8 -> 8	64	0	slave	0	0	16	16	16	16	BRAM=2

S_AXILITE Interfaces

Interface	Data Width	Address Width	Offset	Register
s_axi_control	32	7	24	0

S_AXILITE Registers

Interface	Register	Offset	Width	Access	Description	Bit Fields
s_axi_control	CTRL	0x00	32	RW	Control signals	0=AP_START 1=AP_DONE 2=AP_IDLE 3=AP_READY 7=AUTO_RESTART 9=INTERRUPT
s_axi_control	GIER	0x04	32	RW	Global Interrupt Enable Register	0=Enable
s_axi_control	IP_IER	0x08	32	RW	IP Interrupt Enable Register	0=CHAN0_INT_EN 1=CHAN1_INT_EN
s_axi_control	IP_ISR	0x0c	32	RW	IP Interrupt Status Register	0=CHAN0_INT_ST 1=CHAN1_INT_ST
s_axi_control	ap_return	0x10	32	R	Data signal of ap_return	
s_axi_control	text_length	0x18	32	W	Data signal of text_length	
s_axi_control	text_input_1	0x20	32	W	Data signal of text_input	
s_axi_control	text_input_2	0x24	32	W	Data signal of text_input	

TOP LEVEL CONTROL

Interface	Type	Ports
ap_clk	clock	ap_clk
ap_rst_n	reset	ap_rst_n
interrupt	interrupt	interrupt
ap_ctrl	ap_ctrl_hs	

Εικόνα 37 Μεταβλητές πρωτοκόλλων αποστολής δεδομένων

Επιπροσθέτως, έχουμε την δυνατότητα να ελέγξουμε και να δούμε τους τύπους των μεταβλητών στην Top function Εικόνα 38 που θα χρησιμοποιήσουμε παρακάτω για να κάνουμε πράξεις με τα δεδομένα, καθώς και τις θέσεις μνήμης που βρίσκονται οι διάφορες μεταβλητές σε διασύνδεση με τα πρωτόκολλα αποστολής μηνυμάτων.

SW I/O Information

Top Function Arguments

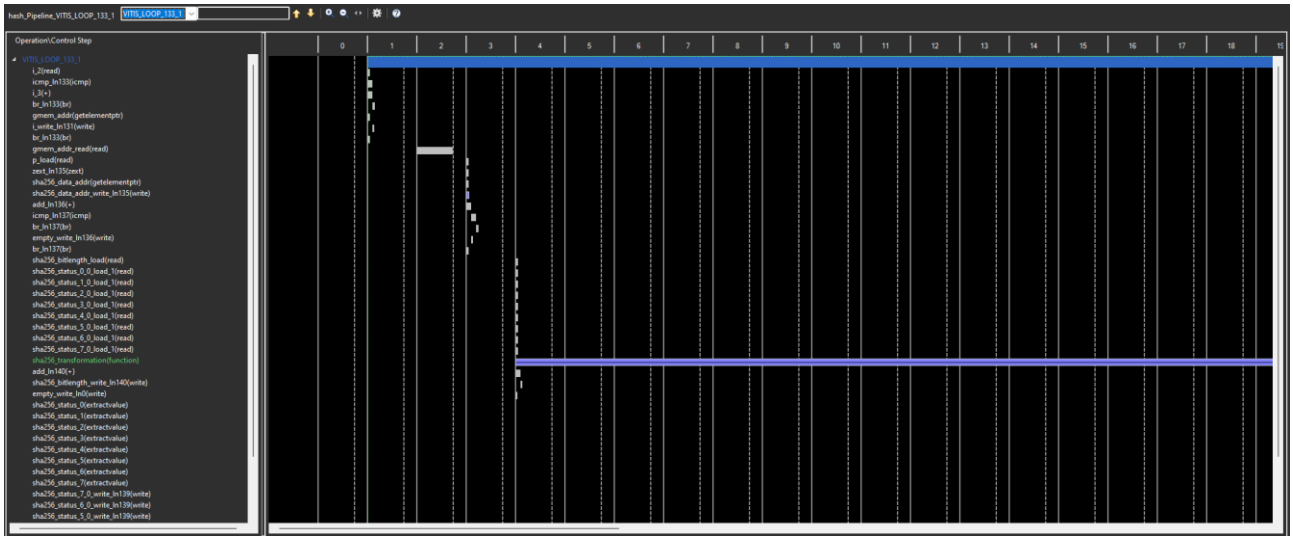
Argument	Direction	Datatype
text_length	in	int
text_input	in	unsigned char*
result	out	unsigned char*
return	out	int

SW-to-HW Mapping

Argument	HW Interface	HW Type	HW Usage	HW Info
text_length	s_axi_control	register		name=text_length offset=0x18 range=32
text_input	m_axi_gmem	interface		
text_input	s_axi_control	register	offset	name=text_input_1 offset=0x20 range=32
text_input	s_axi_control	register	offset	name=text_input_2 offset=0x24 range=32
result	s_axi_control	memory		name=result offset=64 range=32
return	s_axi_control	register		name=ap_return offset=0x10 range=32

Εικόνα 38 Τύποι μεταβλητών εισόδου εξόδου και θέσεις μνήμης

Τέλος, πέρα από τα παραπάνω στοιχεία χρειάζεται να τονίσουμε και ένα εργαλείο που υπάρχει εγκαταστημένο μέσα στο Vitis Εικόνα 39 στο οποίο φαίνεται παρακάτω ένα στιγμιότυπο της υλοποίησης μας και μας επιτρέπει να παρακολουθήσουμε τις πράξεις που γίνονται κατά την εκτέλεση σε χαμηλό επίπεδο ενώ μας παρουσιάζει και την διάρκεια τους σε κύκλους ρολογιού. Το συγκεκριμένο εργαλείο είναι πολύ σημαντικό καθώς μας επιτρέπει να δούμε καθυστερήσεις που μπορεί να συμβαίνουν σε πολύ χαμηλό επίπεδο να βελτιώσουμε κομμάτια της αλγοριθμικής λογικής καθώς και να παραλληλοποιήσουμε λειτουργίες και πράξεις που δεν εμπεριέχουν εξαρτήσεις και δεν θα μπορούσαμε εύκολα να αντιληφθούμε στο επίπεδο του πηγαίου κώδικα.



Εικόνα 39 Εργαλείο προσομοίωσης χαμηλού επιπέδου

4.2.2 Τεστ επαλήθευσης λειτουργίας στο Modelsim

Για να επαληθεύσουμε την λειτουργία του περιφερειακού φορτώνουμε το αρχείο της VHDL σε συνδυασμό με το testbench που έχουμε δημιουργήσει και μπορούμε να παρατηρήσουμε μέσω της assert αν τα αποτελέσματα των δύο υλοποιήσεων είναι ίδια. Το κομμάτι κώδικα στην Εικόνα 40 περιγράφει μια διαδικασία ελέγχου (testbench) σε γλώσσα VHDL, η οποία χρησιμοποιείται για την επαλήθευση της σωστής λειτουργίας της κρυπτογραφικής μονάδας. Ο κώδικας αυτός εκτελεί δύο

τεστ εισόδων και επαληθεύει τα αποτελέσματα (digest) συγκρίνοντάς τα με τις αναμενόμενες τιμές Εικόνα 40.

Τα επιμέρους βήματα είναι τα ακόλουθα:

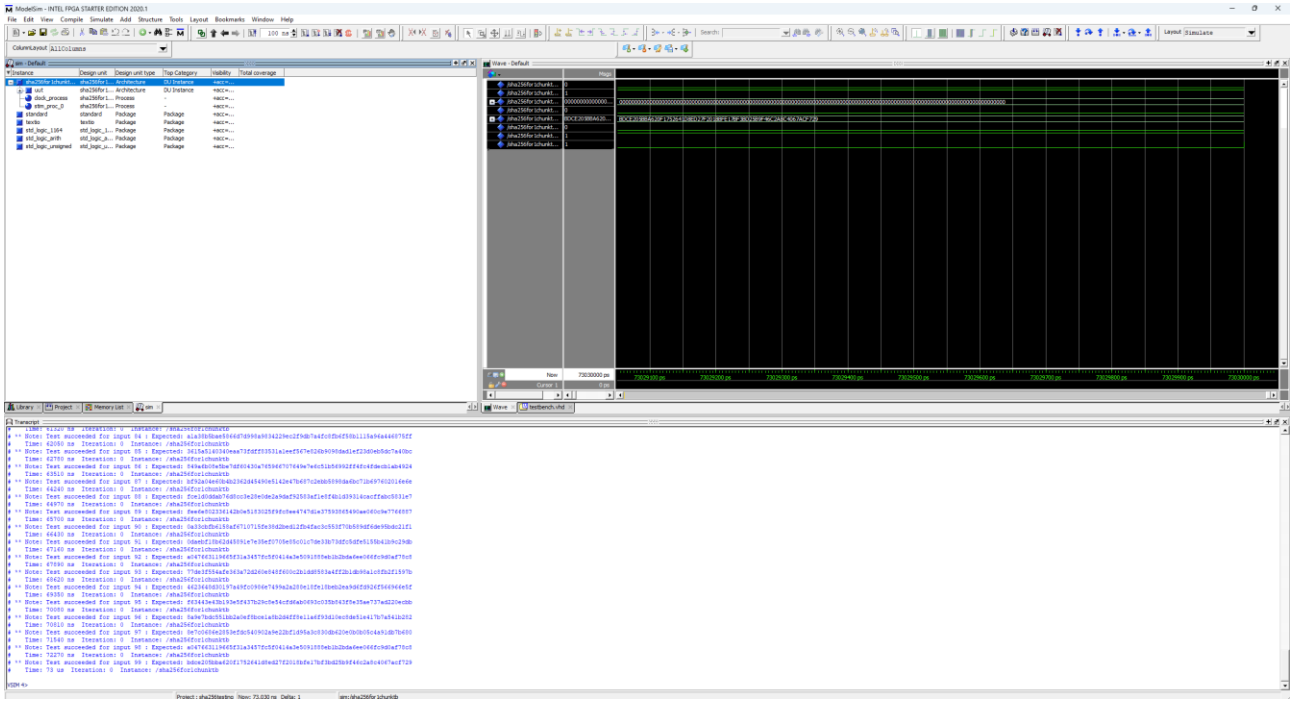
- Ορίζουμε το σήμα reset σε '1' για έναν κύκλο ρολογιού (clock period), και μετά σε '0' για άλλον έναν κύκλο ρολογιού, προκειμένου να γίνει reset στην κρυπτογραφική μονάδα.
- Ορίζουμε το σήμα load σε '1' για να υποδείξει στη μονάδα να φορτώσει τα δεδομένα εισόδου plain.
- Εισάγουμε την πρώτη είσοδο δεδομένων plain (ένα μεγάλο hexadecimal αριθμό).
- Αναμένουμε έναν κύκλο ρολογιού και μετά επαναφέρει το load σε '0'.
- Καθαρίζουμε τα δεδομένα εισόδου plain.
- Περιμένουμε για 70 κύκλους ρολογιού, δίνοντας χρόνο στη μονάδα να επεξεργαστεί τα δεδομένα και να παράγει το αποτέλεσμα.
- Ελέγχουμε αν το αποτέλεσμα digest της κρυπτογραφικής μονάδας είναι ίσο με την αναμενόμενη τιμή.
- Αν το αποτέλεσμα δεν είναι το αναμενόμενο, αναφέρουμε σφάλμα.
- Εάν το αποτέλεσμα είναι σωστό, αναφέρουμε επιτυχία.

Συνοπτικά, η διαδικασία ελέγχου που φαίνεται στην Εικόνα 40 και στην Εικόνα 41 είναι σε βήματα:

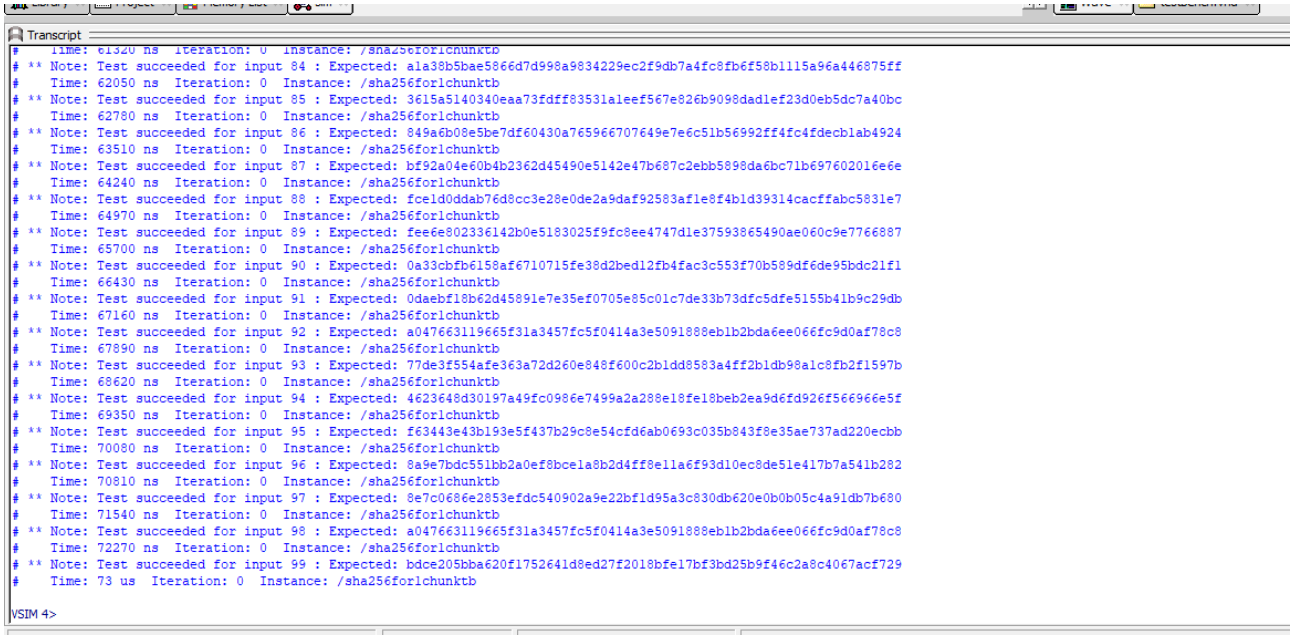
1. Κάνουμε reset στη μονάδα.
2. Φορτώνουμε δεδομένα εισόδου.
3. Περιμένουμε την επεξεργασία.
4. Ελέγχουμε το αποτέλεσμα συγκρίνοντάς το με την αναμενόμενη τιμή και αναφέρουμε επιτυχία Εικόνα 41 ή αποτυχία Εικόνα 42.

```
begin
reset <= '1';
wait for clock_period;
reset <= '0';
wait for clock_period;
load <= '1';
plain <= x"6c627880000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000018";
wait for clock_period;
load <= '0';
plain <= (others => '0');
wait for clock_period*70;
assert (digest = x"d3bf8202ad139083e2c24a18771b65a55919215d123eab85963f83c699b4749b")
 report "Test failed for input 0 : Expected: d3bf8202ad139083e2c24a18771b65a55919215d123eab85963f83c699b4749b" severity error;
assert (digest /= x"d3bf8202ad139083e2c24a18771b65a55919215d123eab85963f83c699b4749b")
 report "Test succeeded for input 0 : Expected: d3bf8202ad139083e2c24a18771b65a55919215d123eab85963f83c699b4749b" severity note;
reset <= '1';
wait for clock_period;
reset <= '0';
wait for clock_period;
load <= '1';
plain <= x"737176800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000018";
wait for clock_period;
load <= '0';
plain <= (others => '0');
wait for clock_period*70;
assert (digest = x"223403da4e6fa1411f331d5d845c94949be1cc882cffb5d3c37b5d4666e36c5a")
 report "Test failed for input 1 : Expected: 223403da4e6fa1411f331d5d845c94949be1cc882cffb5d3c37b5d4666e36c5a" severity error;
assert (digest /= x"223403da4e6fa1411f331d5d845c94949be1cc882cffb5d3c37b5d4666e36c5a")
 report "Test succeeded for input 1 : Expected: 223403da4e6fa1411f331d5d845c94949be1cc882cffb5d3c37b5d4666e36c5a" severity note;
```

Εικόνα 40 Παράδειγμα ελέγχου assert στον κώδικα VHDL



Εικόνα 41 Modelsim σύγκριση αποτελεσμάτων



Εικόνα 42 Παρακολούθηση αποτελεσμάτων μέσω της Assert

4.2.3 Πρωτόκολλα αποστολής δεδομένων από τον επεξεργαστή

Διαχείριση Πόρων M_AXI στο Vivado

Ο προσαρμογέας AXI Master (M_AXI) [39] διαδραματίζει κρίσιμο ρόλο στη μετατροπή των προσαρμοσμένων εντολών AXI από τον χρονοπρογραμματιστή HLS σε τυπικό πρωτόκολλο AXI AMBA® και στην αποστολή τους στην εξωτερική μνήμη. Η κατανάλωση πόρων της συσκευής αυτής εξαρτάται από το άθροισμα των πόρων για όλες τις μονάδες εγγραφής και ανάγνωσης.

Υπολογισμός Κατανάλωσης Πόρων

Η συνολική κατανάλωση πόρων του προσαρμογέα M_AXI περιλαμβάνει:

- **Μονάδες Εγγραφής:** Περιλαμβάνουν το μέγεθος της FIFO_wreq, buff_wdata και της FIFO_resp.
- **Μονάδες Ανάγνωσης:** Περιλαμβάνουν αντίστοιχες FIFO και buffers.

Ο γενικός τύπος για τον υπολογισμό του μεγέθους της FIFO είναι:

$$Μέγεθος\ FIFO = Πλάτος\ X\ Βάθος$$

Για παράδειγμα, η αποθήκευση FIFO 1KB μπορεί να διαμορφωθεί ως 32x32, 8x64 κ.λπ., ανάλογα με τις προδιαγραφές του σχεδίου.

Έλεγχος της Συμπεριφοράς Ριπής AXI4

Η βέλτιστη διασύνδεση AXI4 διασφαλίζει ότι η σχεδίαση δεν καθυστερεί ποτέ λόγω αναμονής πρόσβασης στο δίαυλο και ότι ο δίαυλος δεν καθυστερεί περιμένοντας τη σχεδίαση να διαβάσει ή να γράψει δεδομένα. Για την επίτευξη αυτής της βέλτιστης διασύνδεσης, μπορούμε να χρησιμοποιήσουμε συγκεκριμένα pragma ή οδηγίες INTERFACE για τον καθορισμό της συμπεριφοράς της πρόσβασης σε ριπές.

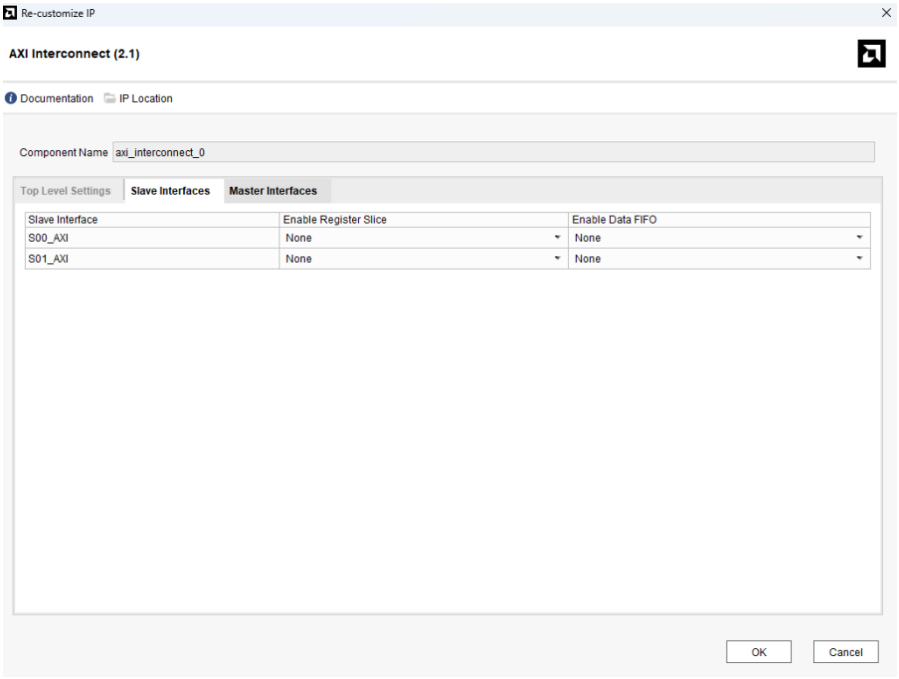
Προσαρμογή των διεπαφών AXI4 Master στο IP Integrator

Κατά την ενσωμάτωση μιας σχεδίασης HLS RTL που χρησιμοποιεί μια κύρια διασύνδεση AXI4 σε μια σχεδίαση στο Vivado IP integrator, μπορούμε να προσαρμόσουμε το μπλοκ σύμφωνα με τις ανάγκες μας. Η διαδικασία προσαρμογής περιλαμβάνει τα ακόλουθα βήματα:

1. **Επιλογή Μπλοκ HLS:**
    - Από το διάγραμμα μπλοκ στο IP integrator, επιλέγουμε το μπλοκ HLS.
    - Κάνουμε δεξί κλικ και επιλέγουμε “Customize Block” (Προσαρμογή μπλοκ).
2. **Ρυθμίσεις προσαρμογής:**
    - Στο παράθυρο διαλόγου Re-Customize IP, μπορούμε να προσαρμόσουμε διάφορες ρυθμίσεις που σχετίζονται με τη διασύνδεση AXI4.
    -

Παραδείγματα Ρυθμίσεων στο Re-Customize IP

Στο παράθυρο διαλόγου Re-Customize IP μπορούμε να ρυθμίσουμε παραμέτρους όπως το πλάτος δεδομένων, τον τύπο και το μέγεθος των FIFO, καθώς και τις συμπεριφορές ριπής για τη βελτιστοποίηση της αποδοτικότητας της διασύνδεσης AXI4 Εικόνα 43. Αυτές οι προσαρμογές επιτρέπουν τη βέλτιστη απόδοση και τη διαλειτουργικότητα μεταξύ των διαφόρων μπλοκ του συστήματος.



Εικόνα 43 Παράθυρο αναπροσαρμογής του στοιχείου axi_interconnect

Διαχείριση Διεπαφών AXI4-Lite για HLS IP στο Vivado και Vitis HLS

Στην ανάπτυξη και υλοποίηση ενός HLS IP ή πυρήνα, χρησιμοποιούμε τη διεπαφή Slave AXI4-Lite (s_axilite) για την επικοινωνία μεταξύ του επεξεργαστή και του πυρήνα. Αυτή η διεπαφή λειτουργεί ως δίαυλος συστήματος και επιτρέπει στον κεντρικό υπολογιστή ή τον ενσωματωμένο επεξεργαστή να εκκινεί, να σταματά και να διαβάζει ή να γράφει δεδομένα στον πυρήνα.

Χρήση της Διεπαφής s_axilite

Γενική Περιγραφή

Η διεπαφή s_axilite Εικόνα 44 υλοποιείται ως προσαρμογέας που καταγράφει τα δεδομένα που επικοινωνούνται από τον κεντρικό υπολογιστή σε καταχωρητές στον προσαρμογέα. Αυτή η διεπαφή διευκολύνει την αλληλεπίδραση μεταξύ του επεξεργαστή και του IP πυρήνα, επιτρέποντας την άμεση πρόσβαση και έλεγχο του πυρήνα μέσω εντολών ανάγνωσης και εγγραφής [40].

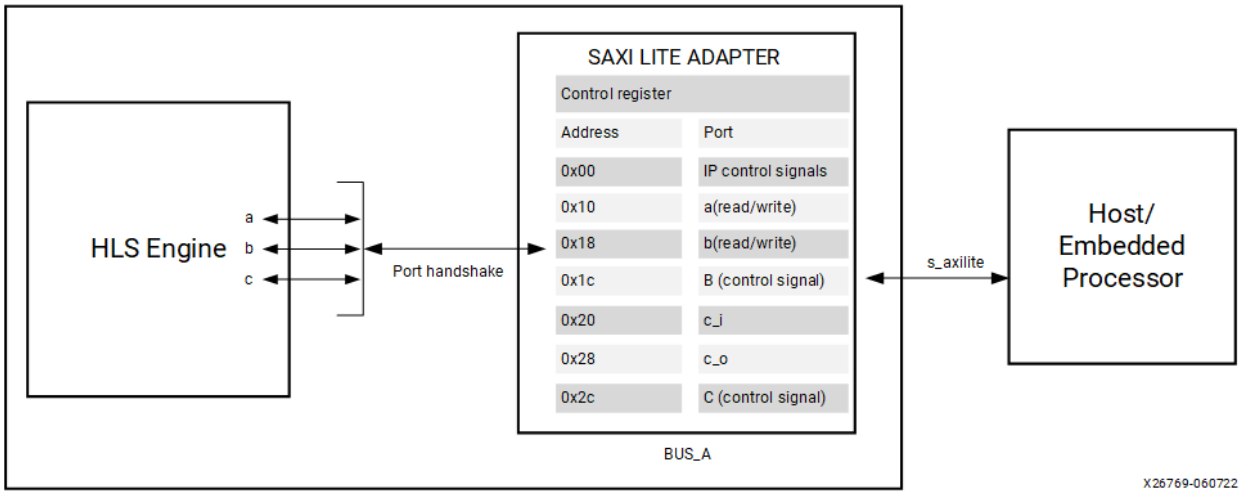
Στη Ροή του Vitis Kernel

1. **Ανάθεση Δεικτών σε s_axilite:**
   - Όταν συνθέτουμε τη σχεδίαση στο Vitis HLS, μπορούμε να διαβάζουμε ή να γράφουμε από έναν δείκτη σε μια κλιμακωτή τιμή όταν αυτός ανατίθεται σε μια διεπαφή s_axilite.
   - Από προεπιλογή, οι δείκτες εκχωρούνται σε διασυνδέσεις m_axi. Για να τους εκχωρήσουμε χειροκίνητα στη s_axilite, χρησιμοποιούμε το pragma ή την οδηγία INTERFACE.
2. **Δέσμη (Bundle):**
   - Για τον προσαρμογέα s_axilite στη ροή του Vitis Kernel, το εργαλείο δημιουργεί μια ενιαία διασύνδεση s_axilite που εξυπηρετεί ολόκληρη τη σχεδίαση.
3. **Μετατοπίσεις (Offset):**

- Το εργαλείο επιλέγει αυτόματα τις μετατοπίσεις για τη διασύνδεση, αποφεύγοντας την ανάγκη καθορισμού μετατοπίσεων χειροκίνητα.

Στη Ροή του Vivado IP

1. **Χρήση Διεπαφής s_axilite:**
   - Από προεπιλογή, η ροή του Vivado IP δεν χρησιμοποιεί τη διεπαφή s_axilite.
   - Για να χρησιμοποιήσουμε το s_axilite ως κανάλι επικοινωνίας για τα κλιμακωτά ορίσματα, τους δείκτες σε κλιμακωτές τιμές, τη μετατόπιση στη διεύθυνση του δείκτη m_axi και τον τύπο επιστροφής της συνάρτησης, πρέπει να καθορίσουμε χειροκίνητα το pragma ή την οδηγία INTERFACE.
2. **Δέσμη (Bundle):**
   - Αυτή η ροή υποστηρίζει πολλαπλές διασυνδέσεις s_axilite, καθορισμένες από το bundle.
3. **Μετατοπίσεις (Offset):**
   - Από προεπιλογή, τα ορίσματα τοποθετούνται σε διαδοχική σειρά ξεκινώντας από το 0x10 στο χάρτη καταχωρητών ελέγχου.



Εικόνα 44 Συνδεσιμότητα HLS μηχανής, πρωτοκόλλου και επεξεργαστή[12]

Πίνακας 4 Θεσεις μνήμης ελέγχου m_axi και s_axilite

Address	Description
0x00	Σήματα ελέγχου
0x04	Παγκόσμιος καταχωρητής ενεργοποίησης διακοπής
0x08	IP Interrupt Enable Register (ανάγνωση/εγγραφή)
0x0c	IP Interrupt Status Register (ανάγνωση /TOW)
0x10	Μετρητής αυτόματης επανεκκίνησης (Εγγραφή- υπάρχει μόνο με καταμετρημένη αυτόματη επανεκκίνηση)
0x14	Εγγραφή γραμματοκιβωτίου εισόδου (ανάγνωση/εγγραφή- υπάρχει μόνο όταν το γραμματοκιβώτιο εισόδου είναι ενεργοποιημένο)
0x18	Ανάγνωση γραμματοκιβωτίου εξόδου (ανάγνωση/εγγραφή- υπάρχει μόνο όταν το γραμματοκιβώτιο εξόδου είναι ενεργοποιημένο)

[12] Η εικόνα προέρχεται από το https://docs.amd.com/r/en-US/ug1399-vitis-hls/AXI4-Master-Interface .

4.2.4 Περιγραφή υλοποίησης στο λειτουργικό Vivado

Στο Vivado, η αρχιτεκτονική master-slave χρησιμοποιείται για την επικοινωνία μεταξύ των διαφόρων στοιχείων του συστήματος, και το πρωτόκολλο AXI (Advanced eXtensible Interface) είναι ευρέως διαδεδομένο για αυτήν την επικοινωνία. Το AXI Interconnect Εικόνα 45 είναι το στοιχείο που επιτρέπει τη σύνδεση πολλαπλών master και slave συσκευών, διασφαλίζοντας την ομαλή μεταφορά δεδομένων μεταξύ τους [41].

Αρχιτεκτονική Master-Slave και AXI Interconnect

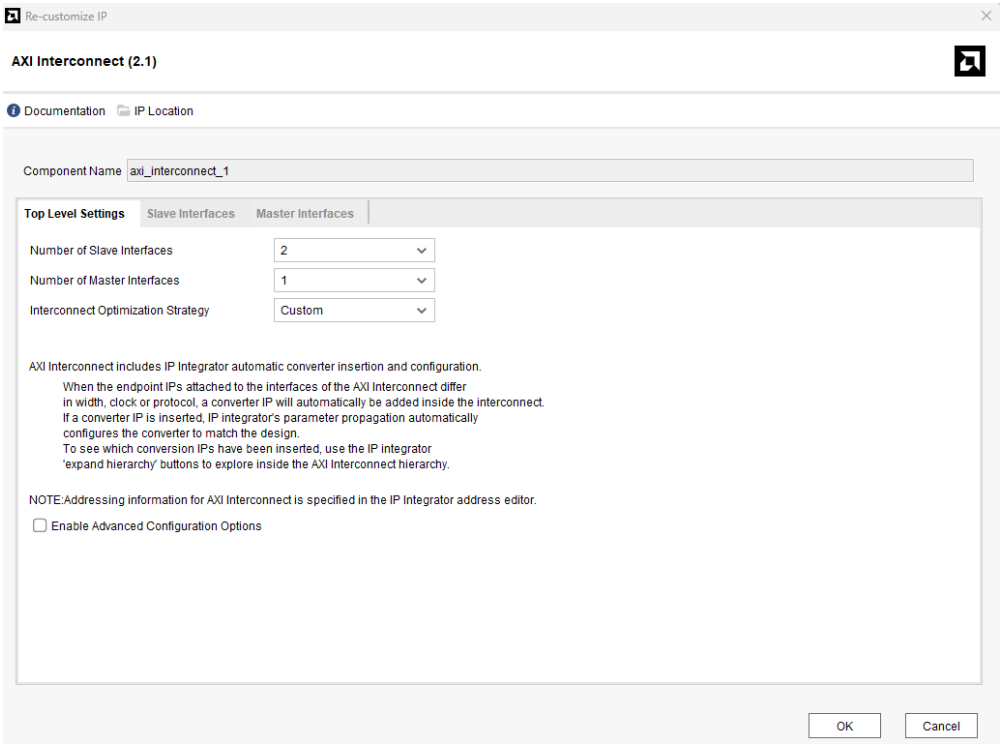
Βασικές Έννοιες

1. **Master Device (Συσκευή Master):**
    - Είναι η συσκευή που ξεκινά τις μεταφορές δεδομένων. Το master μπορεί να διαβάζει ή να γράφει δεδομένα από ή προς μία ή περισσότερες συσκευές slave. Συνήθως, οι επεξεργαστές και οι DMA controllers είναι συσκευές master.
2. **Slave Device (Συσκευή Slave):**
    - Είναι η συσκευή που ανταποκρίνεται στις αιτήσεις του master. Οι συσκευές slave μπορεί να είναι μνήμες, περιφερειακά ή άλλες ειδικές μονάδες που παρέχουν δεδομένα στο master ή λαμβάνουν δεδομένα από αυτό.

Λειτουργία AXI Interconnect

Το AXI Interconnect επιτρέπει τη σύνδεση πολλών master και slave συσκευών, διευκολύνοντας την επικοινωνία μεταξύ τους μέσω του πρωτοκόλλου AXI. Ακολουθούν οι βασικές λειτουργίες του:

1. **Διαχείριση Αιτήσεων και Απαντήσεων:**
    - Το AXI Interconnect λαμβάνει αιτήσεις από τις συσκευές master και τις προωθεί στις κατάλληλες συσκευές slave. Επίσης, λαμβάνει τις απαντήσεις από τις συσκευές slave και τις επιστρέφει στα αντίστοιχα master.
2. **Δρομολόγηση Δεδομένων:**
    - Δρομολογεί τα δεδομένα ανάλογα με τις διευθύνσεις και τις αιτήσεις, εξασφαλίζοντας ότι κάθε αίτηση διαβάσματος ή γραψίματος φτάνει στη σωστή συσκευή slave.
3. **Διαχείριση Συγχρονισμού:**
    - Εξασφαλίζει ότι τα δεδομένα διαβιβάζονται συγχρονισμένα, χρησιμοποιώντας κοινά ρολόγια και διασυνδέσεις.



Εικόνα 45 AXI Interconnect παράθυρο προσαρμογής

Για την υλοποίηση στο περιβάλλον ανάπτυξης Vivado, απαιτείται η δημιουργία ενός σχηματικού διαγράμματος. Ακολουθώντας τα παρακάτω βήματα, δημιουργούμε ένα έργο για την πλακέτα πλατφόρμα Kria KR260 Robotics Starter Kit:

Δημιουργία πρότζεκτ στο Vivado

1. **Έναρξη νέου πρότζεκτ:**
    - Ξεκινάμε το Vivado και δημιουργούμε ένα νέο πρότζεκτ, επιλέγοντας την πλακέτα Kria KR260 Robotics Starter Kit ως στόχο υλοποίησης.
2. **Εισαγωγή του αρχείου IP:**
    - Μεταφορτώνουμε και αποσυμπιέζουμε το αρχείο .zip που δημιουργήθηκε από το Vitis HLS.
    - Τοποθετούμε το περιεχόμενο στο IP Repository του πρότζεκτ μας ακολουθώντας τα εξής βήματα:
        - Μεταβαίνουμε στη Διαχείριση Έργου (Project Manager).
        - Επιλέγουμε Ρυθμίσεις (Settings).
        - Πλοηγούμαστε στην καρτέλα IP και προσθέτουμε το αποθετήριο στην ενότητα IP Repository.

Δημιουργία Διαγράμματος Μπλοκ

3. **IP Integrator:**

- Ανοίγουμε τον IP Integrator και επιλέγουμε “Create Block Diagram” για να δημιουργήσουμε ένα νέο διάγραμμα μπλοκ.

4. **Προσθήκη Μπλοκ:**
   - Προσθέτουμε τα ακόλουθα μπλοκ στο διάγραμμα:
     - **Zynq UltraScale+ MPSoC**: Το κύριο επεξεργαστικό σύστημα (PS) Εικόνα 46.
     - **Hash**: Το IP που δημιουργήθηκε από το Vitis HLS Εικόνα 47.
     - **AXI Interconnect**: Για τη διασύνδεση με το δίαυλο m_axi του IP μας.
5. **Ρύθμιση Zynq UltraScale+ MPSoC:**
   - Κάνουμε διπλό κλικ στο μπλοκ Zynq UltraScale+ MPSoC για να ανοίξουμε τον διάλογο ρυθμίσεων.
   - Ενεργοποιούμε το **AXI HP0 FPD (High Performance Port 0)**.
   - Ελέγχουμε ότι το πλάτος δεδομένων είναι 32 bit, το οποίο πρέπει να ταιριάζει με αυτό που συντέθηκε στο Vitis HLS.

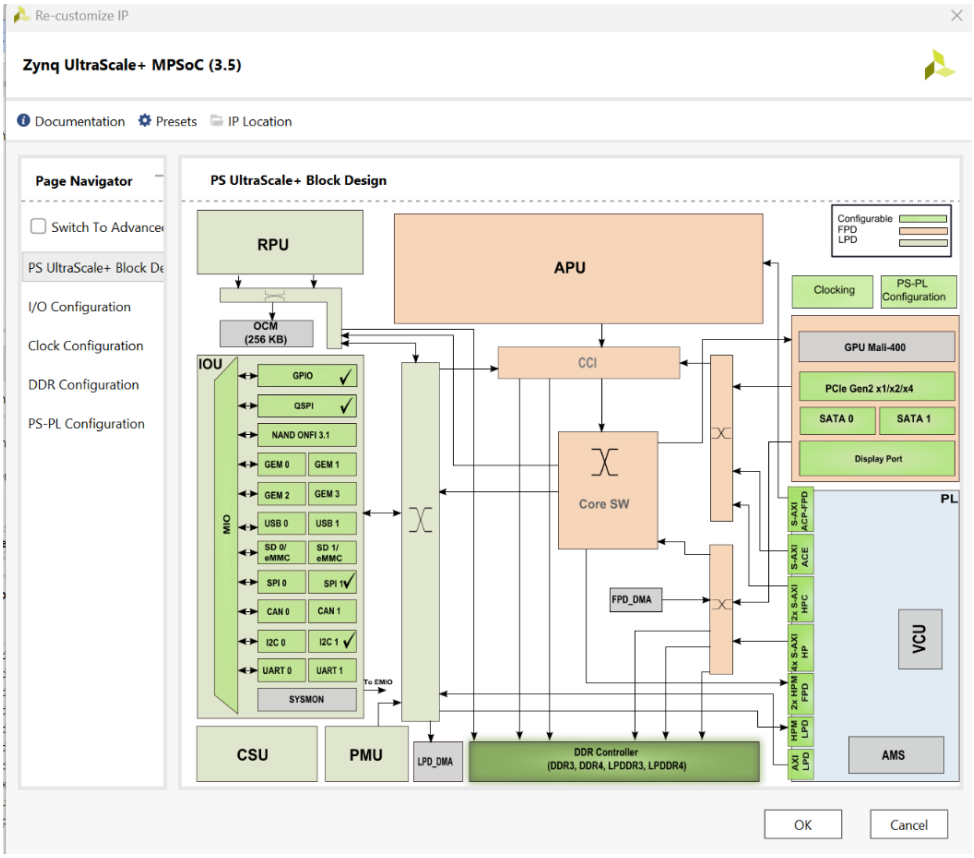
Ρύθμιση διαύλων και μεταβλητών

6. **Διασύνδεση μικρών μεταβλητών:**
   - Για τις μικρές μεταβλητές όπως το text_length και το result, επιλέγουμε το s_axilite, το οποίο είναι ένα σειριακό πρωτόκολλο κατάλληλο για μικρές μεταβλητές. Αυτό διευκολύνει την πρόσβαση μέσω του PYNQ σε μελλοντικό στάδιο.
7. **Διασύνδεση μεγάλων ρυθμιστικών μνήμης:**
   - Για μεγάλες ρυθμιστικές μνήμες όπως η text_input[1024], επιλέγουμε το m_axi, ένα παράλληλο πρωτόκολλο. Αν και απαιτεί περισσότερη λογική και διασυνδέσεις, εξασφαλίζει ταχύτατες μεταφορές δεδομένων.

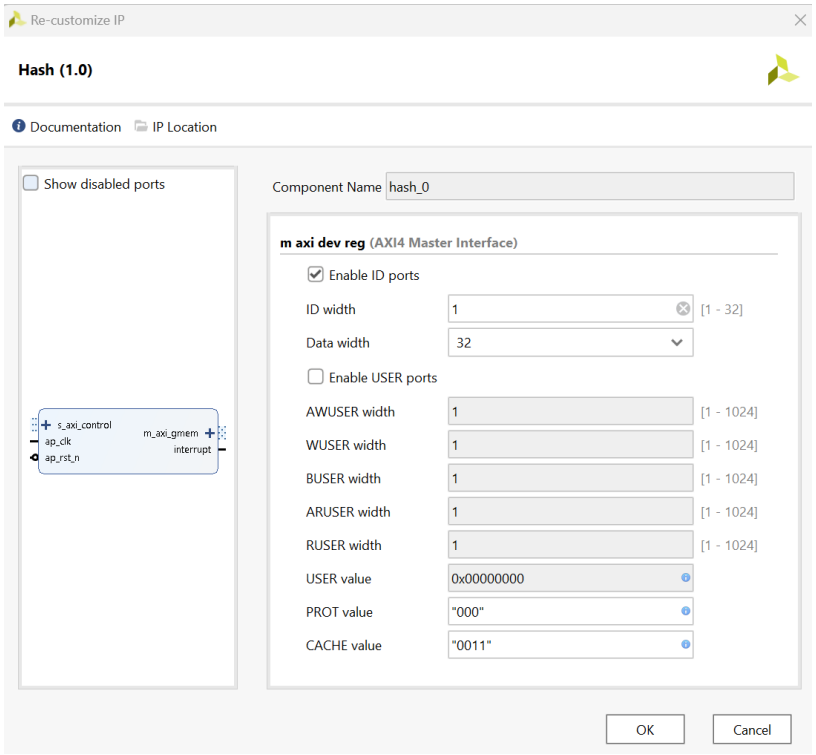
Αυτόματες συνδέσεις και σύνθεση συστήματος

8. **Αυτόματες Συνδέσεις:**
   - Χρησιμοποιούμε το εργαλείο “make autoconnections” για να αυτοματοποιήσουμε τις περισσότερες συνδέσεις στο σύστημα, όπως την ενσωμάτωση κοινού ρολογιού σε όλα τα στοιχεία και την προσαρμογή της προγραμματιστικής λογικής της επεξεργαστικής μονάδας Εικόνα 48.
9. **Σύνθεση Συστήματος:**
   - Εκτελούμε τη σύνθεση του συστήματος, η οποία διαρκεί αρκετή ώρα. Κατά τη διάρκεια αυτής της διαδικασίας, τοποθετούνται πολλαπλά pins τόσο του επεξεργαστή όσο και του περιφερειακού. Επίσης, πραγματοποιείται η μετάφραση του περιφερειακού σε γλώσσα που είναι επεξεργάσιμη από το λογισμικό του Vivado Εικόνα 49.

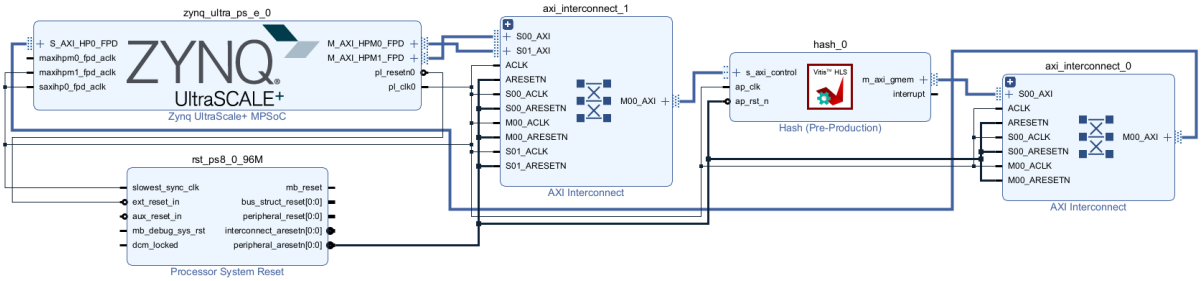
Ακολουθώντας αυτά τα βήματα, διαμορφώνουμε ένα λειτουργικό Εικόνα 50 και αποδοτικό Εικόνα 51 σύστημα στο Vivado για την πλατφόρμα Kria KR260 Robotics Starter Kit, εξασφαλίζοντας τη σωστή λειτουργία και απόδοση του σχεδίου μας Εικόνα 52.



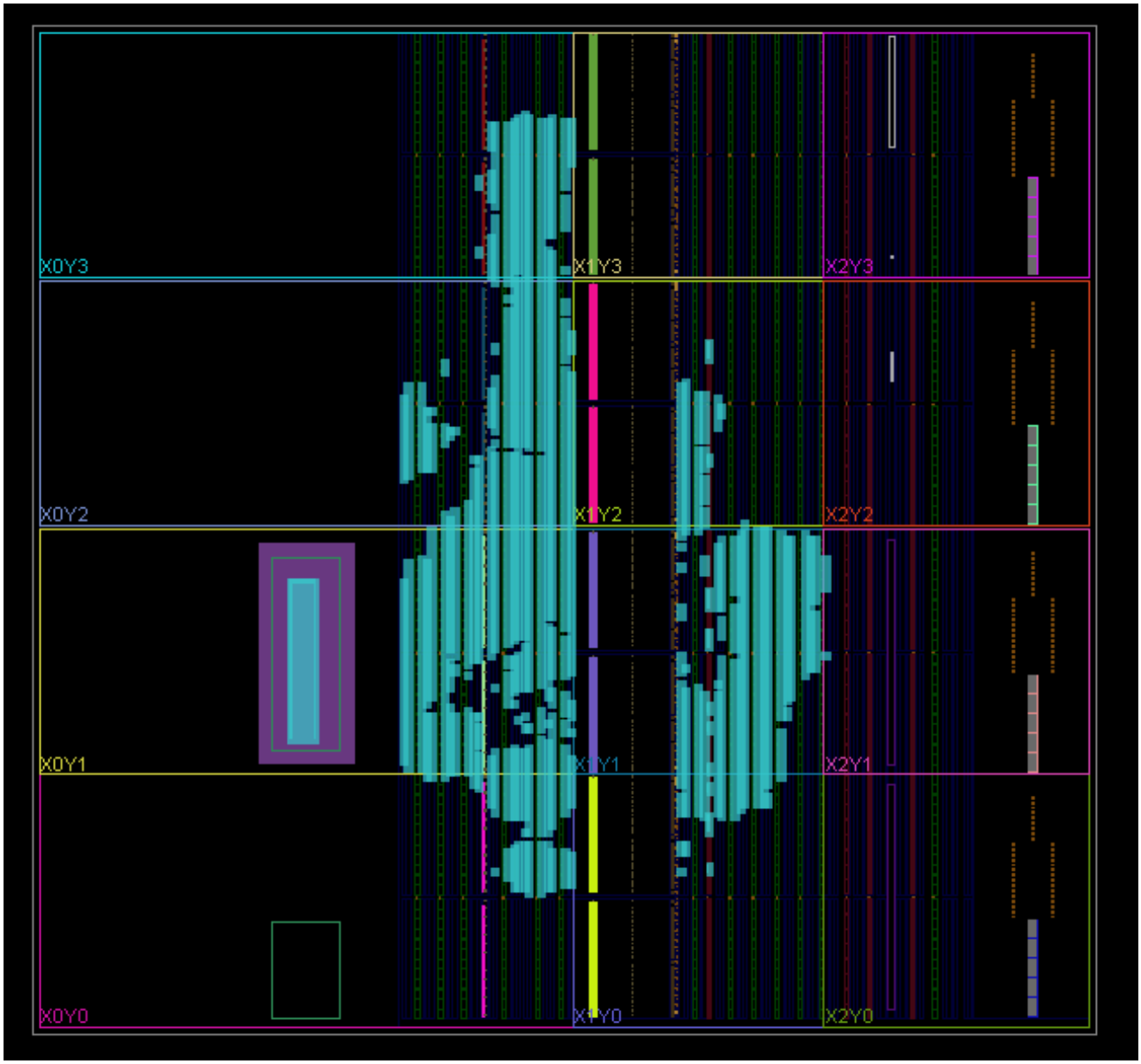
Εικόνα 46 Δομή επεξεργαστή στο Vivado



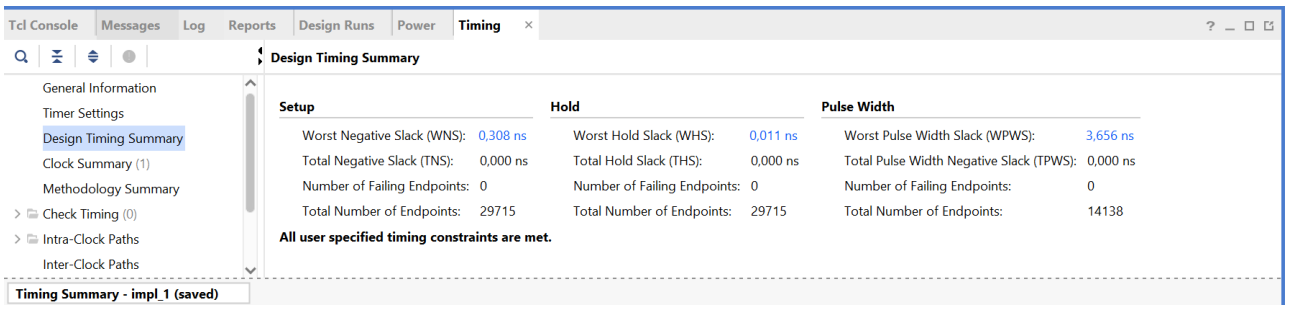
Εικόνα 47 Παράθυρο προσαρμογής περιφερειακού



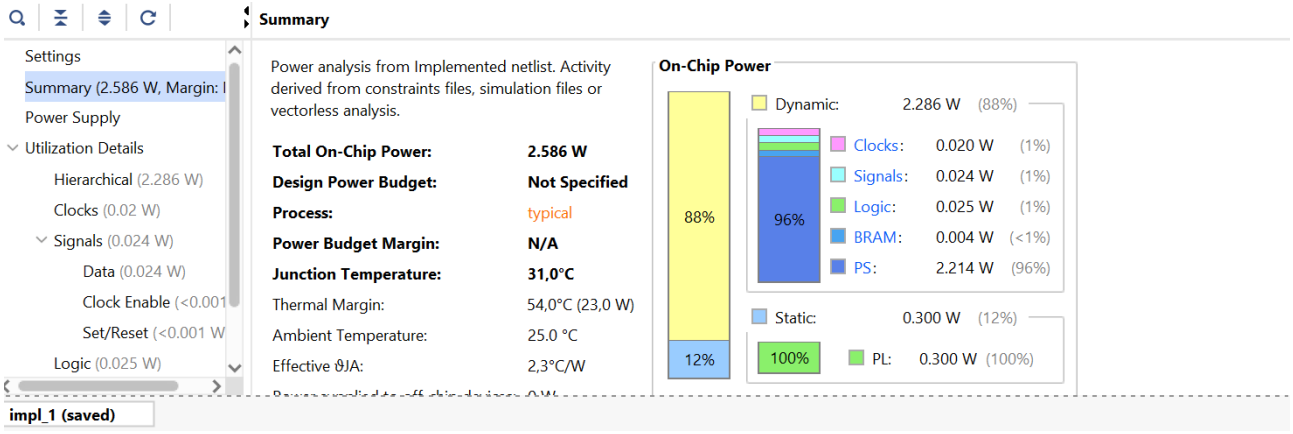
Εικόνα 48 Σύστημα Κρυπτογράφησης



Εικόνα 49 Υλοποιημένο σύστημα με δρομολόγηση των pins στην πλακέτα



Εικόνα 50 Σύνοψη χρονισμού σχεδιασμού



Εικόνα 51 Σύνοψη κατανάλωσης ενέργειας

LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy
0	**0**	**0**	**0**	**0**	**6/16/24, 2:13 PM**	**00:00:29**	**Vivado Synthesis Defaults (Vivado Synthesis 2023)**
20274	13858	7	0	0	6/16/24, 2:13 PM	00:10:31	Vivado Implementation Defaults (Vivado Implementation 2023)

Εικόνα 52 Στοιχεία υλοποίησης

4.3 Υλοποίηση και εφαρμογή συστήματος στο επίπεδο της Kria

4.3.1 Ubuntu Image

Για την εγκατάσταση του Linux Image που χρησιμοποιήσαμε, ακολουθήθηκαν ορισμένες οδηγίες που παρέχονται από τον κατασκευαστή και την εταιρία της AMD® [42].

Βήμα 1. Εγκατάσταση του Image στην κάρτα SD (Ubuntu)

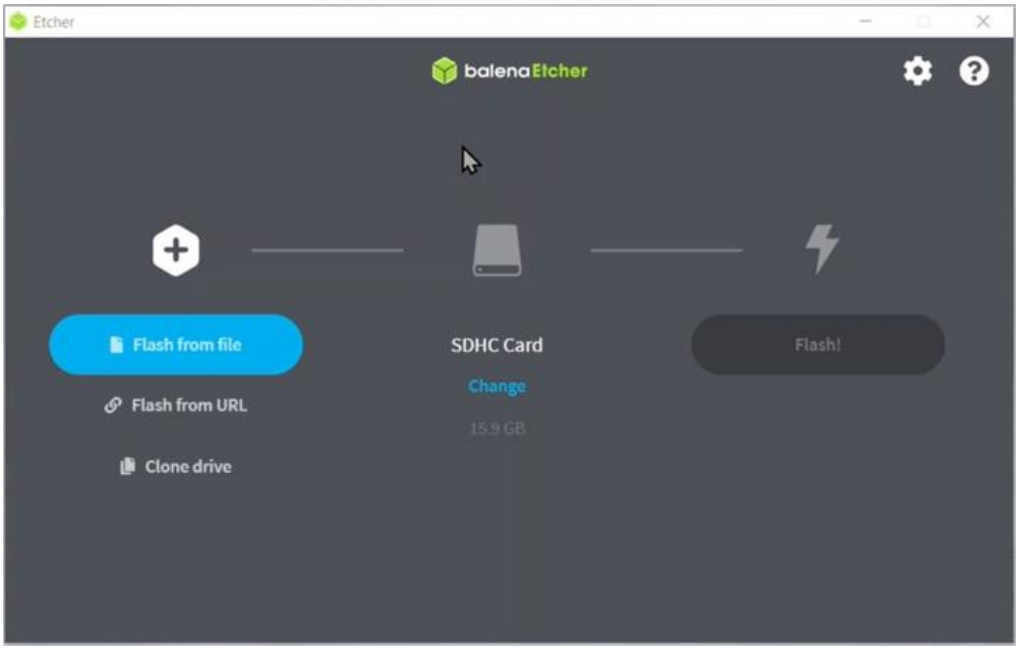
Το συγκεκριμένο FPGA διαθέτει μια κύρια και μια δευτερεύουσα συσκευή εκκίνησης, δίνοντας την δυνατότητα στο χρήστη είτε να χρησιμοποιήσει το έτοιμο λογισμικό είτε να εγκαταστήσει κάποια διαφορετική εικόνα πάνω στην πλακέτα. Θα πρέπει να υπογραμμίσουμε ότι η πρωτεύουσα συσκευή εκκίνησης είναι μια μνήμη QSPI που βρίσκεται στο SOM (System On Memory), η οποία είναι προ-προγραμματισμένη (προ-φορτωμένη εικόνα QSPI) στο εργοστάσιο. Η δευτερεύουσα συσκευή εκκίνησης είναι μια διεπαφή κάρτας microSD στην κάρτα φορέα και σε περίπτωση που δεν εντοπιστεί από το σύστημα η προγραμματιζόμενη πλακέτα κάνει εκκίνηση χρησιμοποιώντας το λογισμικό που είναι προ εγκατεστημένο από το εργοστάσιο.

Για τη ρύθμιση της κάρτας microSD, κατεβάσαμε την πιο πρόσφατη εικόνα Linux της κάρτας SD και στη συνέχεια την γράψαμε χρησιμοποιώντας ένα εργαλείο αναλαμπής εικόνας.

1. Πραγματοποιήθηκε λήψη της εικόνας που είναι συμβατή με την Kria KR260 και αποθηκεύτηκε την στον υπολογιστή μας [43].

2. Πραγματοποιήθηκε λήψη του Balena Etcher (διατίθεται σε εκδόσεις για Windows, Linux και macOS) Εικόνα 53.

3. Ακολουθήσαν οι οδηγίες του εργαλείου και επιλέχτηκε η εικόνα που κατεβάσαμε για να τη μεταφέρουμε στην κάρτα microSD.



Εικόνα 53 Γραφικό περιβάλλον Balena Etcher

Βήμα 2. Σύνδεση των περιφερειακών για δοκιμαστική λειτουργία στην Kria(Ubuntu)

Ακολουθήσαμε τις βασικές συνδέσεις για την AMD® Kria KR260 Εικόνα 54:

1. Τοποθετήσαμε την κάρτα microSD που περιέχει την εικόνα εκκίνησης στην υποδοχή κάρτας microSD (J11) του Starter Kit.

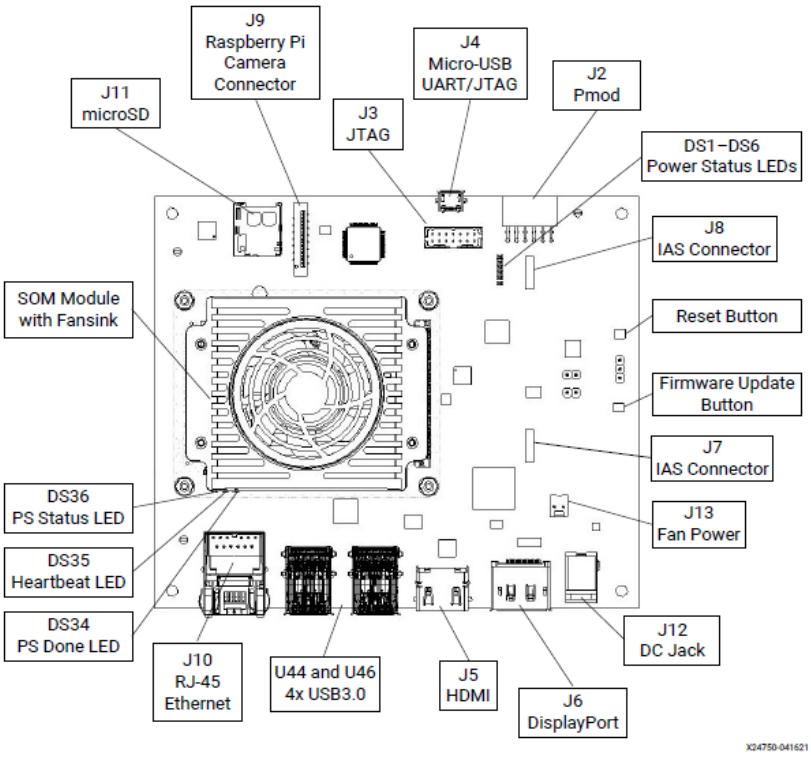
2. Προμηθευτήκαμε το καλώδιο USB-A σε micro-B (γνωστό και ως καλώδιο micro-USB), το οποίο υποστηρίζει τη μεταφορά δεδομένων. Συνδέσαμε το άκρο micro-B στο J4 του Starter Kit.

3. Συνδέσαμε το πληκτρολόγιο/ποντίκι USB στις θύρες USB (U44 και U46).

4. Συνδεθήκαμε σε μια οθόνη/οθόνη με τη βοήθεια ενός καλωδίου DisplayPort.

5. Συνδέσαμε το καλώδιο Ethernet σε μία από τις θύρες PS ETH (J10D που είναι η επάνω δεξιά θύρα) για την απαιτούμενη πρόσβαση στο διαδίκτυο με εργοστασιακά φορτωμένο υλικολογισμικό.

6. Πήραμε το τροφοδοτικό και το συνδέσαμε στην υποδοχή DC (J12) του Starter Kit.



Εικόνα 54 Kria KR260 Robotics Starter Kit συνδέσεις ακροδεκτών[13]

Βήμα 3. Εκκίνηση του Starter Kit (Ubuntu)

Γενικά, εφόσον έχουμε πραγματοποιήσει τις απαραίτητες συνδέσεις που περιγράψαμε στο προηγούμενο βήμα υπάρχουν δύο τρόποι για συνδεθούμε και να εισέλθουμε στις λειτουργίες τις πλακέτας. Ο "παραδοσιακός" τρόπος που είναι η σύνδεση μέσω σειριακής θύρας και δεύτερον η χρήση γραφικού περιβάλλοντος επιφάνειας εργασίας GNOME που απαιτεί την σύνδεση πληκτρολογίου, ποντικιού και οθόνης, όπως περιγράψαμε στα προηγούμενα βήματα.

Εμείς δοκιμάσαμε και τα δύο περιβάλλοντα, ωστόσο θα πρέπει να τονίσουμε ότι μέσω της σειριακής θύρας USB-UART παρέχονται παραπάνω δυνατότητες, εφόσον σε κάποιες εφαρμογές γενικά δεν μπορούμε να έχουμε πρόσβαση από την διεπαφή.

Για να έχουμε πρόσβαση στις λειτουργίες και στις βιβλιοθήκες που θα χρειαστούμε στην εκπόνηση της παρούσας διπλωματικής στην πρώτη εκκίνηση απαιτείται η εγκατάσταση ορισμένων βιβλιοθηκών και η ρύθμιση της πλακέτας στο περιβάλλον των Ubuntu Linux.

i. Πρώτη εντολή που θα πρέπει να τρέξουμε στο bash είναι:

sudo apt update

Η εντολή "sudo apt upgrade" είναι μια εντολή του λειτουργικού συστήματος Linux, που χρησιμοποιείται για την αναβάθμιση των πακέτων που είναι εγκατεστημένα στο σύστημα. Το apt είναι ο διαχειριστής πακέτων που χρησιμοποιείται στις διανομές Linux βασισμένες στο Debian, όπως το Ubuntu. Αναλυτικότερα:

[13] Η εικόνα προέρχεται από το https://www.mouser.com/pdfDocs/22.pdf .

- **sudo**: Η εντολή sudo (superuser do) εκτελεί την επόμενη εντολή με δικαιώματα υπερχρήστη (root). Αυτό απαιτείται γιατί η αναβάθμιση πακέτων συστήματος επηρεάζει το σύνολο του λειτουργικού και χρειάζεται διοικητικά δικαιώματα.

- **apt**: Το apt είναι ο διαχειριστής πακέτων που χρησιμοποιείται για την εγκατάσταση, αναβάθμιση και αφαίρεση πακέτων λογισμικού.

- **upgrade**: Η εντολή upgrade χρησιμοποιείται για να αναβαθμίσει όλα τα εγκατεστημένα πακέτα στην τελευταία διαθέσιμη έκδοση. Κατά την εκτέλεση αυτής της εντολής, το apt θα κατεβάσει και θα εγκαταστήσει τις νεότερες εκδόσεις των πακέτων που είναι ήδη εγκατεστημένα στο σύστημα.

- Αυτή η εντολή ενημερώνει τη λίστα των διαθέσιμων πακέτων και των εκδόσεών τους, αλλά δεν εγκαθιστά ή αναβαθμίζει κανένα πακέτο.

- Κατεβάζει τις τελευταίες πληροφορίες πακέτων από τα αποθετήρια που αναφέρονται στο αρχείο /etc/apt/sources.list και ενημερώνει τον τοπικό δείκτη πακέτων με αυτές τις πληροφορίες.

- **Εκτελούμε αυτή την εντολή πρώτα** για να διασφαλίσουμε ότι έχουμε τις πιο πρόσφατες πληροφορίες σχετικά με τις διαθέσιμες ενημερώσεις.

ii. Επόμενη εντολή είναι η:

$$sudo\ apt\ upgrade$$

- Αυτή η εντολή αναβαθμίζει όλα τα εγκατεστημένα πακέτα στις τελευταίες εκδόσεις τους, βάσει των πληροφοριών που ελήφθησαν με την sudo apt update.

- Εγκαθιστά τις πιο νέες εκδόσεις όλων των πακέτων που είναι ήδη εγκατεστημένα στο σύστημα, χωρίς να αφαιρεί ή να εγκαθιστά άλλα πακέτα.

iii. Έπειτα εκτελούμε την εντολή:

$$sudo\ snap\ install\ xlnx-config\ --classic\ xlnx-config.sysinit$$

Όλες οι σχετικές πληροφορίες για την εντολή βρίσκονται στο λινκ: https://xilinx-wiki.atlassian.net/wiki/spaces/A/overview

Η εντολή “sudo snap install xlnx-config –classic” και το αρχείο “xlnx-config.sysinit” είναι μέρος μιας διαδικασίας εγκατάστασης και διαμόρφωσης ενός εργαλείου ή πακέτου λογισμικού που ονομάζεται “xlnx-config”, το οποίο σχετίζεται με εργαλεία της Xilinx, για FPGA ή άλλες προγραμματιζόμενες λογικές συσκευές. Το “xlnx-config.sysinit” είναι ένα αρχείο ή σενάριο (script) που εκτελείται κατά την εκκίνηση (initialization) του συστήματος ή του εργαλείου

iv. Μια εντολή που μπορούμε να τρέξουμε σε αυτό το βήμα για να βεβαιωθούμε ότι έχουμε πραγματοποιήσει σωστά την διαδικασία είναι η :

$$sudo\ xlnx-config\ --xmutil\ boardid\ -b\ som$$

Η συγκεκριμένη εντολή έχει ως σκοπό την διαμόρφωση και την επαλήθευση της λειτουργίας της συσκευής Xilinx και της συσκευής που περιλαμβάνεται σε μια System-on-Module (SOM).

v. Τελευταίο, βήμα αποτελεί η εγκατάσταση του λογισμικού PYNQ που γίνεται με τις παρακάτω εντολές:

$$git\ clone\ https://github.com/Xilinx/Kria-PYNQ$$

$$cd\ Kria-PYNQ$$

$$sudo\ bash\ install.sh$$

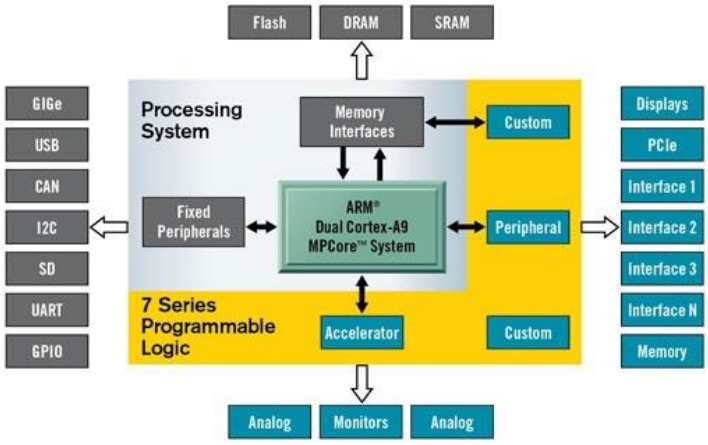
1. **Αντιγραφή του αποθετηρίου (repository) από το GitHub**: Χρησιμοποιώντας την εντολή git clone, κατεβάζουμε τον κώδικα του Kria-PYNQ από το GitHub στον τοπικό σας υπολογιστή [44].

2. **Μετακίνηση στον κατάλογο Kria-PYNQ**: Αφού κατεβάσουμε το αποθετήριο, μετακινούμε στον κατάλογο Kria-PYNQ χρησιμοποιώντας την εντολή cd.

3. **Εκτέλεση του script εγκατάστασης**: Χρησιμοποιώντας την εντολή sudo bash install.sh, εκτελείτε το script εγκατάστασης που περιέχεται στο αποθετήριο. Αυτή η διαδικασία μπορεί να διαρκέσει έως και 30 λεπτά, καθώς ενδέχεται να περιλαμβάνει την εγκατάσταση πολλών εξαρτήσεων και ρυθμίσεων. Επιπλέον, θα πρέπει σε αυτό το σημείο να τρέξουμε ένα από τα συγκεκριμένα τρία αρχεία .sh που υπάρχουν στο github. Εμείς τρέξαμε για εγκατάσταση το αντίστοιχο της πλακέτας μας.

4.3.2 PYNQ Overlays

Η συσκευή AMD®-Xilinx® Zynq® All Programmable αποτελεί ένα σύστημα σε ολοκληρωμένο κύκλωμα (SoC) που βασίζεται σε έναν διπύρηνο επεξεργαστή ARM® Cortex®-A9 Εικόνα 55, ο οποίος αναφέρεται ως Processing System (PS), ενσωματωμένο με ένα FPGA fabric, το οποίο αναφέρεται ως Programmable Logic (PL). Το υποσύστημα PS περιλαμβάνει έναν αριθμό αποκλειστικών περιφερειακών, όπως ελεγκτές μνήμης, USB, UART, IIC, και SPI, ενώ μπορεί να επεκταθεί περαιτέρω με πρόσθετο υλικό IP σε ένα PL Overlay [45].



Εικόνα 55 Προγραμματιστική Μονάδα Βιβλιοθήκης[14]

[14] Η εικόνα προέρχεται από το https://pynq.readthedocs.io/en/latest/pynq_overlays.html .

Οι επικαλύψεις (Overlays) ή βιβλιοθήκες υλικού είναι προγραμματιζόμενα/διαμορφώσιμα σχέδια FPGA που επεκτείνουν την εφαρμογή του χρήστη από το σύστημα επεξεργασίας του Zynq στην προγραμματιζόμενη λογική. Αυτές οι επικαλύψεις μπορούν να χρησιμοποιηθούν για την επιτάχυνση μιας εφαρμογής λογισμικού ή για την προσαρμογή της πλατφόρμας υλικού σε συγκεκριμένες απαιτήσεις εφαρμογής.

Χρήση Overlays στο PYNQ

Το PYNQ παρέχει μια διεπαφή Python, η οποία επιτρέπει τον έλεγχο των επικαλύψεων στο PL από την Python που εκτελείται στο PS. Ο σχεδιασμός FPGA είναι μια εξειδικευμένη εργασία που απαιτεί γνώσεις και τεχνογνωσία μηχανικού υλικού. Οι επικαλύψεις του PYNQ δημιουργούνται από εξειδικευμένους σχεδιαστές υλικού και ενσωματώνονται με το API Python του PYNQ, καθιστώντας τις προσβάσιμες από προγραμματιστές λογισμικού [45].

Οφέλη και δυνατότητες

1. **Επιτάχυνση εφαρμογών**:
    - Οι επικαλύψεις μπορούν να επιταχύνουν απαιτητικές εφαρμογές λογισμικού, μεταφέροντας επεξεργαστικές λειτουργίες στο PL.
    - Αυτό επιτρέπει την εκτέλεση υπολογιστικά εντατικών διεργασιών με υψηλότερη απόδοση και αποδοτικότητα.
2. **Προσαρμογή πλατφόρμας υλικού**:
    - Οι επικαλύψεις επιτρέπουν την προσαρμογή της πλατφόρμας υλικού στις συγκεκριμένες ανάγκες μιας εφαρμογής.
    - Παρέχουν ευελιξία στη σχεδίαση και ανάπτυξη συστημάτων, επιτρέποντας την εύκολη προσαρμογή σε διαφορετικές εφαρμογές και απαιτήσεις.
3. **Εύκολη Χρήση από Προγραμματιστές Λογισμικού**:
    - Με τη χρήση του API Python, οι προγραμματιστές λογισμικού μπορούν να προγραμματίζουν και να ελέγχουν εξειδικευμένες επικαλύψεις υλικού χωρίς να χρειάζονται ειδικές γνώσεις σχεδιασμού FPGA.
    - Αυτό είναι ανάλογο με τις βιβλιοθήκες λογισμικού που δημιουργούνται από ειδικούς προγραμματιστές και χρησιμοποιούνται από άλλους προγραμματιστές σε επίπεδο εφαρμογής.

Φόρτωση ενός Overlay (επικάλυψης)

Από προεπιλογή, μια επικάλυψη (bitstream) που ονομάζεται base φορτώνεται στο Programmable Logic (PL) κατά την εκκίνηση. Η επικάλυψη βάσης μπορεί να θεωρηθεί ως ένα σχέδιο αναφοράς για μια πλακέτα. Νέες επικαλύψεις μπορούν να εγκατασταθούν ή να αντιγραφούν στην πλακέτα και να φορτωθούν στο PL κατά τη διάρκεια της εκτέλεσης του συστήματος.

Σύνθεση Μιας Επικάλυψης

Μια επικάλυψη συνήθως περιλαμβάνει τα εξής στοιχεία:

- **Ένα ρεύμα bit (bitstream)** για τη διαμόρφωση του υφάσματος της FPGA.
- **Ένα αρχείο HWH** σχεδιασμού Vivado, το οποίο καθορίζει το διαθέσιμο IP.
- **Python API**, το οποίο ενεργοποιεί τις διεπαφές των IPs.

Χρήση της Κλάσης PYNQ Overlay

Η κλάση PYNQ Overlay μπορεί να χρησιμοποιηθεί για τη φόρτωση μιας επικάλυψης. Μια επικάλυψη ενσαρκώνεται καθορίζοντας το όνομα του αρχείου bitstream. Η διαδικασία ενσωμάτωσης της επικάλυψης περιλαμβάνει επίσης τη φόρτωση του bitstream από προεπιλογή και την ανάλυση του αρχείου HWH.

Διαδικασία Φόρτωσης Επικάλυψης

Η φόρτωση μιας επικάλυψης περιλαμβάνει τα ακόλουθα βήματα:

1. **Καθορισμός ονόματος Bitstream**:
    - Ο χρήστης καθορίζει το όνομα του αρχείου bitstream που επιθυμεί να φορτώσει.
2. **Λήψη και ανάλυση**:
    - Η κλάση PYNQ Overlay αναλαμβάνει την λήψη του bitstream και την ανάλυση του αντίστοιχου αρχείου HWH.
3. **Ενεργοποίηση διεπαφών IPs**:
    - Τα IPs που περιλαμβάνονται στην επικάλυψη εκτίθενται μέσω του Python API, επιτρέποντας τη χρήση τους από προγραμματιστές λογισμικού.

Βιβλιοθήκες PYNQ

Τα τυπικά ενσωματωμένα συστήματα υποστηρίζουν έναν σταθερό συνδυασμό περιφερειακών, όπως SPI, IIC, UART, Video, και USB, ενώ συνήθως διαθέτουν περιορισμένο αριθμό ακίδων GPIO (General Purpose Input/Output). Στα συστήματα αυτά, οι GPIO ελέγχονται από την κύρια CPU, η οποία διαχειρίζεται το υπόλοιπο σύστημα, με αποτέλεσμα η απόδοση των GPIO να είναι συχνά περιορισμένη.

Αντίθετα, οι πλατφόρμες AMD®-Xilinx® διαθέτουν πολύ περισσότερες διαθέσιμες ακίδες IO από ένα τυπικό ενσωματωμένο σύστημα [45]. Ειδικοί ελεγκτές υλικού και πρόσθετοι επεξεργαστές soft real-time μπορούν να ενσωματωθούν στην προγραμματιζόμενη λογική (PL), επιτρέποντας υψηλότερη απόδοση στις διεπαφές σε σχέση με άλλα ενσωματωμένα συστήματα.

Η θύρα USB και άλλες τυποποιημένες διεπαφές μπορούν να χρησιμοποιηθούν για τη σύνδεση έτοιμων USB και άλλων περιφερειακών συσκευών στο Zynq PS, όπου μπορούν να ελεγχθούν από την Python/Linux. Η εικόνα του PYNQ περιλαμβάνει επί του παρόντος προγράμματα οδήγησης για τις πιο συχνά χρησιμοποιούμενες κάμερες USB, περιφερειακά WiFi και άλλες τυπικές συσκευές USB.

Χρήση Περιφερειακών και Υποστήριξη από το PYNQ

Το PYNQ λειτουργεί σε Linux και, για τις πλατφόρμες Zynq και Zynq Ultrascale+, χρησιμοποιούνται τα εξής περιφερειακά PS από προεπιλογή:

- **Κάρτα SD**: Χρησιμοποιείται για την εκκίνηση του συστήματος και τη φιλοξενία του συστήματος αρχείων Linux.
- **UART**: Παρέχει πρόσβαση σε τερματικό Linux.
- **USB**: Χρησιμοποιείται για σύνδεση έτοιμων USB περιφερειακών συσκευών.
- **Ethernet**: Μπορεί να χρησιμοποιηθεί για τη σύνδεση με το σημειωματάριο Jupyter ή για Ethernet over USB Gadget στο Zynq Ultrascale+.

Η πλατφόρμα PYNQ περιλαμβάνει προγράμματα οδήγησης για τις πιο συχνά χρησιμοποιούμενες κάμερες USB, περιφερειακά WiFi και άλλες τυπικές συσκευές USB.

Υποστήριξη Περιφερειακών μέσω του Vivado και του PYNQ API

Το Vivado περιλαμβάνει μια βιβλιοθήκη υλικού IP που μπορεί να χρησιμοποιηθεί για τη σύνδεση σε ένα ευρύ φάσμα προτύπων και πρωτοκόλλων διασύνδεσης. Το PYNQ παρέχει ένα Python API για έναν αριθμό κοινών περιφερειακών συσκευών, όπως:

- **Video (HDMI In and Out).**
- **Συσκευές GPIO (Buttons, Switches, LEDs).**
- **Αισθητήρες και ενεργοποιητές.**

Το API του PYNQ μπορεί επίσης να επεκταθεί για την υποστήριξη πρόσθετων IP.

Διασύνδεση Εξωτερικών Περιφερειακών

Οι πλατφόρμες Zynq [23] διαθέτουν συνήθως μία ή περισσότερες κεφαλίδες ή διεπαφές που επιτρέπουν τη σύνδεση εξωτερικών περιφερειακών ή την απευθείας σύνδεση με τις ακίδες PL του Zynq. Έτοιμα περιφερειακά μπορούν να συνδεθούν στις διεπαφές Pmod και Arduino, ενώ άλλα περιφερειακά μπορούν να συνδεθούν μέσω προσαρμογέων ή με breadboard.

Σημειώνεται ότι, για να χρησιμοποιηθεί ένα περιφερειακό, εκτός από τη φυσική σύνδεση στις ακίδες Zynq PL, απαιτείται η ενσωμάτωση ενός ελεγκτή στην επικάλυψη και η παροχή ενός οδηγού λογισμικού.

Υποσύστημα PynqMicroBlaze και Χαμηλού Επιπέδου Έλεγχος Επικάλυψης

Οι βιβλιοθήκες PYNQ παρέχουν υποστήριξη για το υποσύστημα PynqMicroBlaze, επιτρέποντας τη φόρτωση προ-μεταγλωττισμένων εφαρμογών και τη δημιουργία και μεταγλώττιση νέων εφαρμογών από το Jupyter.

Το PYNQ παρέχει επίσης υποστήριξη για έλεγχο χαμηλού επιπέδου μιας επικάλυψης, περιλαμβάνοντας:

- Ανάγνωση/εγγραφή IO με χαρτογράφηση μνήμης.

- Κατανομή μνήμης για χρήση από έναν κύριο PL.
- Έλεγχος και διαχείριση επικάλυψης (φόρτωση επικάλυψης, ανάγνωση IP σε επικάλυψη).
- Έλεγχος χαμηλού επιπέδου του PL (φόρτωση bitstream).

Βασικό κλάση που χρησιμοποιήθηκε στην παρούσα διπλωματική εργασία είναι η MMIO.

MMIO

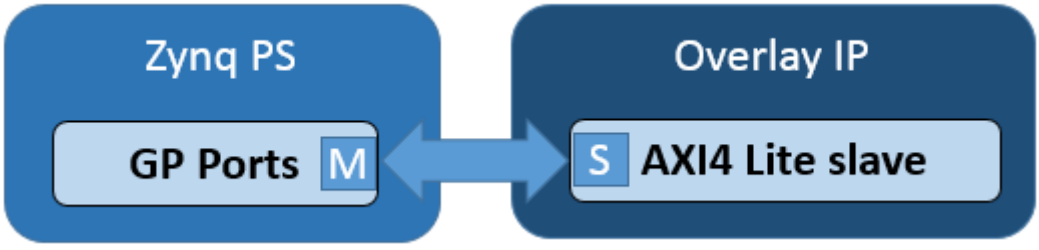
Η κλάση MMIO επιτρέπει σε ένα αντικείμενο της Python [45] να έχει πρόσβαση σε διευθύνσεις στη μνήμη του συστήματος που είναι χαρτογραφημένες, διευκολύνοντας την πρόσβαση στους καταχωρητές και στο χώρο διευθύνσεων των περιφερειακών συσκευών στο Programmable Logic (PL).

Θύρες AXI GP

Σε μια επικάλυψη, τα περιφερειακά που συνδέονται στις θύρες γενικού σκοπού AXI (AXI General Purpose Ports) έχουν τους καταχωρητές ή το χώρο διευθύνσεών τους χαρτογραφημένους στη χαρτογράφηση της μνήμης του συστήματος Εικόνα 56. Με το PYNQ, ο καταχωρητής ή ο χώρος διευθύνσεων ενός περιφερειακού μπορεί να προσπελαστεί από την Python χρησιμοποιώντας την κλάση MMIO.

Πλεονεκτήματα της Χρήσης της Κλάσης MMIO

Το MMIO παρέχει έναν απλό αλλά ισχυρό τρόπο πρόσβασης και ελέγχου περιφερειακών συσκευών. Για απλά περιφερειακά με μικρό αριθμό προσβάσεων στη μνήμη ή όπου η απόδοση δεν είναι κρίσιμη, η χρήση της κλάσης MMIO είναι συνήθως επαρκής για τους περισσότερους προγραμματιστές. Οι λειτουργίες ανάγνωσης και εγγραφής στη μνήμη επιτρέπουν την άμεση αλληλεπίδραση με τους καταχωρητές των περιφερειακών, καθιστώντας τη διαδικασία ελέγχου απλή και αποτελεσματική.



Εικόνα 56 Επικοινωνία Zynq PS με Overlay IP[15]

Χρήση της κλάσης PYNQ DMA για Υψηλές Αποδόσεις

Ωστόσο, σε περιπτώσεις όπου η απόδοση είναι κρίσιμη ή απαιτείται η μεταφορά μεγάλων ποσοτήτων δεδομένων μεταξύ του Processing System (PS) και του PL, η χρήση των διασυνδέσεων Zynq HP (High Performance) με DMA IP (Direct Memory Access Intellectual Property) και της κλάσης PYNQ DMA μπορεί να είναι πιο κατάλληλη. Οι διασυνδέσεις αυτές επιτρέπουν τη γρήγορη και

[15] Η εικόνα προέρχεται από το https://pynq.readthedocs.io/en/latest/pynq_libraries/mmio.html .

αποδοτική μεταφορά δεδομένων, ελαχιστοποιώντας την καθυστέρηση και την κατανάλωση πόρων της κύριας CPU.

4.3.3 PYNQ προγραμματισμός στο περιβάλλον του Jupyter Notebooks

Για τον προγραμματισμό του επεξεργαστή στην πλατφόρμα ανάπτυξης, είναι απαραίτητη η φόρτωση του overlay και η δέσμευση συγκεκριμένων μεταβλητών τόσο για το περιφερειακό όσο και για τις διευθύνσεις μνήμης. Όπως απεικονίζεται στην Εικόνα 57, η διαδικασία αυτή επιτρέπει την αρχικοποίηση των απαραίτητων συνιστωσών για την εκτέλεση του προγράμματος.

Δημιουργία Συνάρτησης Κρυπτογράφησης

Η υλοποίηση της συνάρτησης κρυπτογράφησης περιλαμβάνει τα εξής βήματα:

1. **Δέσμευση Θέσεων Μνήμης:** Αρχικά, δεσμεύουμε θέσεις μνήμης για το μέγεθος των δεδομένων που θα κρυπτογραφηθούν καθώς και για τα ίδια τα δεδομένα. Αυτή η δέσμευση επιτρέπει την κατάλληλη προετοιμασία της μνήμης για την υποδοχή των δεδομένων προς επεξεργασία.

2. **Επανάληψη Αποστολής Δεδομένων:** Η συνάρτηση κρυπτογράφησης περιλαμβάνει μια επαναληπτική διαδικασία όπου ο επεξεργαστής στέλνει συνεχώς δεδομένα, με την έναρξη να σηματοδοτείται από το σήμα ενεργοποίησης (AP_START=1). Η διαδικασία αυτή συνεχίζεται έως ότου ολοκληρωθεί η επεξεργασία των δεδομένων, η οποία υποδεικνύεται από την κατάσταση AP_DONE=0.

3. **Επιστροφή Κρυπτογραφημένης Τιμής:** Με την ολοκλήρωση της επεξεργασίας, η κρυπτογραφημένη τιμή επιστρέφεται σε δεκαεξαδική μορφή. Αυτό επιτρέπει την εύκολη ανάγνωση και χρήση της κρυπτογραφημένης πληροφορίας.

Χρήση της Βιβλιοθήκης hashlib

Επιπροσθέτως, υπάρχει η υλοποίηση μιας δεύτερης συνάρτησης κρυπτογράφησης που χρησιμοποιούμε τη βιβλιοθήκη hashlib της Python, η οποία περιλαμβάνει την ίδια συνάρτηση υλοποιημένη σε Python. Η χρήση της hashlib παρέχει μία εναλλακτική και αξιόπιστη μέθοδο κρυπτογράφησης, η οποία μπορεί να χρησιμοποιηθεί για την επικύρωση και σύγκριση των αποτελεσμάτων της υλοποίησης του hardware Εικόνα 58.

First we must configure the hardaware. We programm the bitstream overlay with the instructions in its page

In [38]:
```
from pynq import Overlay, allocate

# Loading the overlay as described in the doctumentation
overlay = Overlay('news.bit')

#  the IP core
hash_ip = overlay.hash_0

# variable to approach the memory-mapped I/O interface for the IP core
mmio_interface = hash_ip.mmio

# In order to have access to the register map for the IP core
reg_map = hash_ip.register_map

# Have access to the register classes
reg_classes = reg_map._register_classes
```

Then, we must do the memory mapping for our project. We print the available registers and then we allocate a matrix for PL to have access to the text input from hls code on the PS. Also, we need to map memory location for the PS to access results on the PL

In [39]:
```
# Print variables locations and names in order to see what variables we will use
# We will use text_input1 in the process
for registers_names, register in registers.items():
    print(registers_names, register)

# We register the location  for the (axilite)
# We calculate the result address and size in terms of 32-bit words (//4 to do implement this)
result_addr = reg_map.Memory_result.address // 4
result_word_size = reg_map.Memory_result.width // 4
result_values = mmio_interface.array[result_addr : result_addr + result_word_size]

# We allocate a buffer for the input matrix (size 1024 bits as defined in vitis)
input_length = 1024
input_buffer = allocate(shape=(input_length,), dtype='u1', cacheable=False)  # unsigned 8-bit integers

# We save the device address of the input buffer to the IP core register
reg_map.text_input_1.text_input = input_buffer.device_address

# We initialize the input buffer with zeros
input_buffer[:] = 0
```

```
CTRL (<class 'pynq.registers.RegisterCTRL'>, 0, 32, None, None, 'read-write')
GIER (<class 'pynq.registers.RegisterGIER'>, 4, 32, None, None, 'read-write')
IP_IER (<class 'pynq.registers.RegisterIP_IER'>, 8, 32, None, None, 'read-write')
IP_ISR (<class 'pynq.registers.RegisterIP_ISR'>, 12, 32, None, None, 'read-write')
ap_return (<class 'pynq.registers.Registerap_return'>, 16, 32, None, None, 'read-only')
text_length (<class 'pynq.registers.Registertext_length'>, 24, 32, None, None, 'write-only')
text_input_1 (<class 'pynq.registers.Registertext_input_1'>, 32, 32, None, None, 'write-only')
text_input_2 (<class 'pynq.registers.Registertext_input_2'>, 36, 32, None, None, 'write-only')
Memory_result (<class 'pynq.registers.RegisterMemory_result'>, 64, 32, None, None, 'read-write')
```

Εικόνα 57 Φόρτωση του Overlay και δέσμευση μεταβλητών

Next we need to write the hashing functions

In [36]:
```
import hashlib
import numpy as np
from pynq import allocate
import secrets

# Define Built-in function in hashlib library for the SHA-256 algorithm
def sha256_builtin(input_bytes):
    return hashlib.sha256(input_bytes).hexdigest()

# Define the function for our hardware implemented hardware system
def sha256_hardware(input_bytes):
    #We must set the length of the input text (first input in the Component)
    reg_map.text_length = len(input_bytes)

    #We write the input bytes to the input buffer (Second input)
    input_buffer[:len(input_bytes)] = np.frombuffer(input_bytes, dtype=np.uint8)

    #We provide the START signal to start the SHA-256 calculation process (vitis component)
    reg_map.CTRL.AP_START = 1

    # We control with the DONE when the hardware computation is done
    while reg_map.CTRL.AP_DONE == 0:
        pass

    # We return the result as a hexadecimal string value
    result = bytearray(result_values)
```

Εικόνα 58 Ορισμός συνάρτησης κρυπτογράφησης

4.3.4 Σύνοψη Τέταρτου Κεφαλαίου

Στο κεφάλαιο αυτό, παρουσιάστηκε η μετάβαση από την ανάπτυξη κώδικα στην υλοποίηση του περιφερειακού SHA-256 στο περιβάλλον του Vivado. Ειδικότερα, αναλύθηκε η διαδικασία που ακολουθήθηκε για τη σχεδίαση και την εφαρμογή του αλγορίθμου SHA-256 σε επίπεδο υλικού, καθώς και τα βήματα για την επιτυχή ενσωμάτωση και παραμετροποίηση στο Vivado.

Το κεφάλαιο ολοκληρώθηκε με την παρουσίαση της υλοποίησης ενός περιφερειακού συστήματος που βασίστηκε στον αλγόριθμο SHA-256. Περιγράφηκαν η αρχιτεκτονική, οι σχεδιαστικές αποφάσεις, οι τεχνικές προκλήσεις που αντιμετωπίστηκαν και οι λύσεις που υιοθετήθηκαν για τη βελτιστοποίηση της απόδοσης και της ασφάλειας του συστήματος. Παρουσιάστηκαν επίσης τα αποτελέσματα των δοκιμών και των αξιολογήσεων, καταδεικνύοντας την αποτελεσματικότητα και την αξιοπιστία της υλοποίησης.

Ακολούθως, περιεγράφηκε η εγκατάσταση του Linux Image στο σύστημα και ο προγραμματισμός του μέσω της βιβλιοθήκης PYNQ. Η βιβλιοθήκη αυτή παρείχε ένα εύχρηστο περιβάλλον για την ανάπτυξη εφαρμογών που εκμεταλλεύονταν τις δυνατότητες του FPGA, επιτρέποντας την αποτελεσματική αλληλεπίδραση μεταξύ λογισμικού και υλικού.

Σημαντικό μέρος του κεφαλαίου αφιερώθηκε στην εξέταση του Kria KR260 Robotics Starter Kit, περιγράφοντας τον επεξεργαστή και τις δυνατότητές του. Παρέχονταν λεπτομέρειες για την αρχιτεκτονική του συστήματος και τις λειτουργίες που το καθιστούσαν κατάλληλο για ρομποτικές εφαρμογές.

Τέλος, έγινε αναφορά στη βιβλιοθήκη της Python, εξηγώντας πώς χρησιμοποιήθηκε για τον προγραμματισμό και την αυτοματοποίηση του συστήματος. Παρουσιάστηκαν πρακτικά παραδείγματα και κώδικας που διευκόλυναν την ανάπτυξη εφαρμογών με τη χρήση του PYNQ και του Kria KR260, καθιστώντας σαφές πώς η συνδυασμένη χρήση αυτών των εργαλείων οδήγησε σε αποδοτική και καινοτόμο ανάπτυξη συστημάτων.

Κεφάλαιο 5: Συμπεράσματα – Επίλογος

Στο παρόν σημείο, παρουσιάζουμε τα αποτελέσματα της μελέτης μας και εξετάζουμε την απόδοση του προτεινόμενου συστήματος σε σύγκριση με παρόμοιες εφαρμογές στον τομέα μας. Η ανασκόπηση αυτή περιλαμβάνει επίσης μια σύντομη ανασύνθεση του ερευνητικού περιεχομένου μας, ενώ αναλύουμε προοπτικές για μελλοντικές επεκτάσεις και βελτιώσεις της προτεινόμενης μεθοδολογίας.

5.1 Αποτελέσματα συστήματος κρυπτογράφησης

Όπως φαίνεται στην Εικόνα 59, πραγματοποιήθηκε έλεγχος της ακρίβειας του συστήματος μέσω της χρήσης της εντολής assert για τη σύγκριση των αποτελεσμάτων της δημιουργηθείσας συνάρτησης με αυτά της ενσωματωμένης βιβλιοθήκης hashlib. Ο συγκεκριμένος έλεγχος είναι κρίσιμος για τη διασφάλιση της ορθότητας και της ακρίβειας των μαθηματικών και αλγοριθμικών προσεγγίσεων που υιοθετούνται στο πλαίσιο της μεθοδολογίας μας.

```
#We can now write testbenches to check the functions results and in next step calculate the time
#outputs of the algorithms

# Test with specific strings(strings of our choice) ('abc', 'dasigenis')
print("Hardware SHA-256 for 'abc':", sha256_hardware(b'abc'))
print("Built-in SHA-256 for 'abc':", sha256_builtin(b'abc'))
print("Hardware SHA-256 for 'dasigenis':", sha256_hardware(b'dasigenis'))
print("Built-in SHA-256 for 'dasigenis':", sha256_builtin(b'dasigenis'))

# We run tests for test for random lengthjs of bytes and check with assert if the result match
for length in range(513):
    random_bytes = secrets.token_bytes(length)
    hls_result = sha256_hardware(random_bytes)
    builtin_result = sha256_builtin(random_bytes)
    assert hls_result == builtin_result, f"Mismatch at length {length}: {hls_result} != {builtin_result}"

print("All testbenches are completed and successfully passed.")

# We perform timing measurement for performance comparison
# We use 1000 loops and 5 repeats for each timing measurement (5000 test results)
hls_timing = %timeit -n 1000 -r 5 -o sha256_hardware(secrets.token_bytes(1024))
builtin_timing = %timeit -n 1000 -r 5 -o sha256_builtin(secrets.token_bytes(1024))

print('Performance ->', hls_timing.average / builtin_timing.average)
```

```
Hardware SHA-256 for 'abc': ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
Built-in SHA-256 for 'abc': ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
Hardware SHA-256 for 'dasigenis': c323f6292f19c491ea48224134ebadc44f65b0bc160f4b301d0bee17b95997ed
Built-in SHA-256 for 'dasigenis': c323f6292f19c491ea48224134ebadc44f65b0bc160f4b301d0bee17b95997ed
All testbenches are completed and successfully passed.
244 µs ± 1.26 µs per loop (mean ± std. dev. of 5 runs, 1,000 loops each)
37.4 µs ± 324 ns per loop (mean ± std. dev. of 5 runs, 1,000 loops each)
Performance -> 6.5166674666125735
```

Εικόνα 59 assert στο επίπεδο της Python

Μέσω της εντολής assert επιβεβαιώνουμε ότι το αποτέλεσμα της συνάρτησης που υλοποιούμε συμφωνεί ακριβώς με το αναμενόμενο αποτέλεσμα που παράγει η hashlib, επιβεβαιώνοντας την ορθότητα και την επιτυχία της υλοποίησής μας. Αυτή η διαδικασία αποτελεί κρίσιμο βήμα για την επιβεβαίωση της λειτουργικότητας του συστήματος και την αποφυγή πιθανών σφαλμάτων στην υλοποίηση των κρυπτογραφικών λειτουργιών μας.

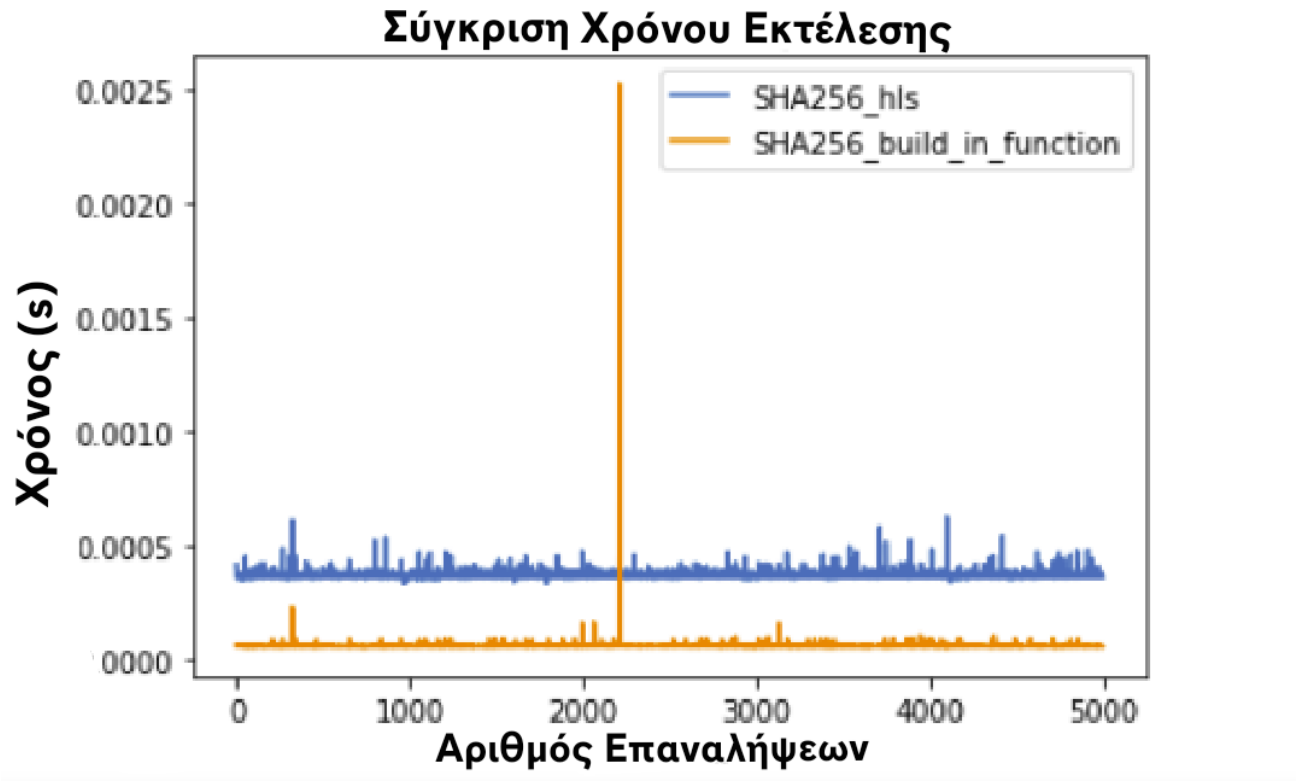
Όπως φαίνεται στις Εικόνα 60 χρησιμοποιήσαμε τη βιβλιοθήκη Python "matplotlib" για να πραγματοποιήσουμε χρονικές μετρήσεις και να αξιολογήσουμε την απόδοση του συστήματος μας σε σχέση με ένα πλήρως βελτιστοποιημένο πρόγραμμα χρήσης της βιβλιοθήκης Python. Η διεξαγωγή αυτών των μετρήσεων είναι καίριας σημασίας για την αξιολόγηση της απόδοσης και της επεξεργαστικής ικανότητας του συστήματος μας κατά την επεξεργασία μεγάλου όγκου δεδομένων.

Όπως παρατηρούμε στην Εικόνα 61, το σύστημα επιτυγχάνει ικανοποιητικές ταχύτητες που πλησιάζουν σημαντικά τις επιδόσεις ενός πλήρως βελτιστοποιημένου προγράμματος. Αυτή η αναφορά των χρονικών μετρήσεων ενισχύει την εμπιστοσύνη στην αποτελεσματικότητα και την απόδοση της αναπτυγμένης μας λύσης, παρά τις διαφορές στη βελτιστοποίηση που μπορεί να παρατηρηθούν μεταξύ τους.

Make a diagramm for timing

In [37]:
```
import matplotlib.pyplot as plt

#We define lists to store times in order to make the diagramms
timings_hls = []
timings_build_in = []

def measure_time(func, *args, **kwargs):
    result = %timeit -n 1 -r 1 -o func(*args, **kwargs)
    return result.average

def wrapper_SHA256_hls(token_value):
    return SHA256_hls(token_value)

def wrapper_SHA256_build_in(token_value):
    return SHA256_build_in(token_value)

# Run both functions for 5000 results and save the execution results in the matrices that we defined above
for _ in range(5000):

    random_strings = secrets.token_bytes(1024)

    result_hls = measure_time(wrapper_SHA256_hls, random_strings)
    timings_hls.append(result_hls)

    result_build_in = measure_time(wrapper_SHA256_build_in, random_strings)
    timings_build_in.append(result_build_in)

# We plot the timing results for both functions
plt.plot(timings_hls, label='SHA256_hls')
plt.plot(timings_build_in, label='SHA256_build_in_function')
plt.xlabel('Iteration')
plt.ylabel('Time (s)')
plt.title('Execution Time Comparison')
plt.legend()
plt.show()
```

Εικόνα 60 Συναρτήσεις για σχεδιασμό γραφικών παραστάσεων



Εικόνα 61 Γραφική παράσταση σύγκρισης αποτελεσμάτων

Σε σύγκριση με τον [7], ο οποίος προηγουμένως είχε υλοποιήσει τον ίδιο αλγόριθμο στο Modelsim, η υλοποίηση μας καταδεικνύει σημαντικά βελτιωμένες επιδόσεις, ιδίως όσον αφορά την ταχύτητα και τον ρυθμό επεξεργασίας. Σύμφωνα με τα αποτελέσματα των χρονικών μετρήσεων μας, η εκτέλεση μιας πράξης και η παραγωγή ενός αποτελέσματος στην υλοποίησή τους απαιτεί περίπου 120ns, σε αντίθεση με τα 5000 αποτελέσματα που απαιτούνται στην δική μας. Αυτή η διαφορά καταδεικνύει τη σημαντική αύξηση της απόδοσης του συστήματος μας σε σχέση με τις προηγούμενες υλοποιήσεις.

Επιπρόσθετα, η υλοποίηση μας παρουσιάζει χαμηλότερη καθυστέρηση σε κάθε κύκλο ρολογιού σε σύγκριση με το [46], γεγονός που συμβάλλει στη βελτίωση της συνολικής απόδοσης του υπολογισμού της hashed τιμής. Αυτό το στοιχείο επιβεβαιώνει την αποτελεσματικότητα της σχεδιαστικής μας επιλογής και την ικανότητά της να παρέχει γρηγορότερες χρονικές αποκρίσεις σε κάθε επεξεργαστική ενέργεια.

Τέλος, η υλοποίηση μας επιτυγχάνει υψηλότερο throughput σε σύγκριση με το [47], επιτρέποντας την επεξεργασία μεγαλύτερης ποσότητας δεδομένων με μεγαλύτερη ταχύτητα και αποτελεσματικότητα. Αυτό το χαρακτηριστικό καταδεικνύει την ικανότητα της υλοποίησής μας να υποστηρίζει απαιτητικές εφαρμογές και να παρέχει υψηλή απόδοση σε πραγματικές συνθήκες λειτουργίας. Ωστόσο, παρατηρήθηκε ότι απαιτείται σημαντικός αριθμός πόρων για την υλοποίησή του σε σύγκριση με παρόμοιες εφαρμογές, κάτι που θα πρέπει να ληφθεί υπόψη κατά την αξιολόγηση της καταλληλότητας για συγκεκριμένες εφαρμογές.

5.2 Συμπεράσματα

Η κρυπτογραφία αποτελεί κρίσιμο πεδίο έρευνας και ανάπτυξης στην εποχή της ψηφιακής ασφάλειας, με τη χρήση προηγμένων τεχνικών όπως ο αλγόριθμος SHA-256 να αποτελεί πυλώνα της ασφάλειας δεδομένων. Η ενσωμάτωση της κρυπτογραφίας σε υλικό FPGA έχει αποδειχθεί ότι προσφέρει υψηλή επιδόσεις και αποτελεσματικότητα, ειδικά όταν απαιτείται γρήγορη επεξεργασία και ασφάλεια δεδομένων σε πραγματικό χρόνο.

Στο πλαίσιο αυτής της διπλωματικής εργασίας, αναπτύχθηκε ένα σύστημα περιφερειακού υλικού που ενσωματώνει τον αλγόριθμο κρυπτογράφησης SHA-256 και συνδέεται με έναν επεξεργαστή τύπου ARM. Το σύστημα αυτό είναι σχεδιασμένο να παρέχει γρήγορη και αξιόπιστη επεξεργασία κρυπτογραφημένων δεδομένων, αξιοποιώντας τη συνδυασμένη ισχύ του FPGA για υψηλή απόδοση και του ARM επεξεργαστή για ευελιξία και διαχείριση γενικού σκοπού των λειτουργιών.

Το σύστημα δοκιμάστηκε επιτυχώς στην πλατφόρμα ρομποτικής εκκίνησης Kria KR260, όπου απέδειξε την ικανότητά του να επεξεργάζεται μεγάλες ποσότητες δεδομένων με χαμηλό χρόνο απόκρισης. Οι αναλύσεις και οι μετρήσεις απόδοσης κατέδειξαν ότι η υλοποίησή μας προσφέρει σημαντική βελτίωση σε σχέση με προηγούμενες εφαρμογές, επιτυγχάνοντας υψηλό throughput και αποδοτικότητα στην κρυπτογράφηση δεδομένων. Ωστόσο, παρατηρήθηκε ότι απαιτείται σημαντικός αριθμός πόρων για την υλοποίησή του σε σύγκριση με παρόμοιες εφαρμογές.

Συνολικά, η εργασία αυτή ανοίγει νέους ορίζοντες για την εφαρμογή της κρυπτογραφίας σε ενσωματωμένα συστήματα και ρομποτικές εφαρμογές, προσφέροντας ένα πρακτικό παράδειγμα σύνθεσης υλικού και λογισμικού για απαιτητικές ανάγκες ασφαλείας και απόδοσης.

5.3 Μελλοντικές βελτιώσεις

Για μελλοντικές βελτιώσεις της υλοποίησής μας, προτείνουμε τις ακόλουθες ενέργειες:

1. **Βελτιστοποίηση Πόρων:** Αν και η υλοποίησή μας προσφέρει σημαντική απόδοση, απαιτεί σημαντικό αριθμό πόρων. Μια μελλοντική βελτίωση θα μπορούσε να επικεντρωθεί στη μείωση του απαιτούμενου αποθέματος πόρων, είτε μέσω βελτιώσεων στον αλγόριθμο είτε μέσω πιο αποδοτικών τεχνικών σχεδίασης.

2. **Αλλαγή Μεταβλητών στο Περιφερειακό της C++:** Προτείνεται η αλλαγή των μεταβλητών στο περιφερειακό της C++ και η χρήση μεταβλητών που είναι βελτιστοποιημένες για High-Level Synthesis (HLS). Αυτό θα μπορούσε να μειώσει τον αριθμό των απαιτούμενων πόρων και να βελτιώσει την απόδοση.

3. **Παράλληλη Επεξεργασία:** Η παράλληλη επεξεργασία όλων των κομματιών του κώδικα σε επίπεδο C++ είναι μια άλλη σημαντική βελτίωση. Μέσω της παράλληλης επεξεργασίας, μπορούμε να επιτύχουμε σημαντική αύξηση της αποδοτικότητας και της ταχύτητας εκτέλεσης.

4. **Επανασχεδιασμός στο επίπεδο του Vivado:** Προτείνεται ένας επανασχεδιασμός του συστήματος σε επίπεδο Vivado. Αυτό θα επιτρέψει την καλύτερη κατανομή των πόρων και την επίτευξη υψηλότερης απόδοσης. Με την ανασχεδίαση, μπορούμε να εντοπίσουμε και

να διορθώσουμε τυχόν αστοχίες στη διαχείριση πόρων και να βελτιώσουμε τη συνολική αποδοτικότητα του συστήματος.

5. **Βελτιστοποίηση στο επίπεδο της Python:** Επίσης, προτείνεται η βελτιστοποίηση του κώδικα Python που χρησιμοποιείται για τον προγραμματισμό του overlay, ειδικά στη δέσμευση μνήμης για τις μεταβλητές. Με βελτιώσεις σε αυτό το επίπεδο, μπορούμε να διασφαλίσουμε την αποδοτική χρήση της μνήμης και να μειώσουμε τη συνολική κατανάλωση πόρων.

Αυτές οι βελτιώσεις θα ενισχύσουν σημαντικά την αποδοτικότητα, τη λειτουργικότητα και την απόδοση του συστήματος, καθιστώντας το ακόμα πιο ανταγωνιστικό και αποδοτικό για μελλοντικές εφαρμογές.

Βιβλιογραφία

[1] Wayne Wolf, *Σχεδιασμός ψηφιακών συστημάτων σε FPGAs*. 2014. Accessed: Jun. 20, 2024. [Online]. https://newtech-pub.com/wp-content/uploads/2014/03/FPGAKefalaio1.pdf

[2] "Xilinx FPGA vs Altera FPGA: What Is the Difference?" Accessed: Jun. 15, 2024. [Online]. Available: https://www.nextpcb.com/blog/xilinx-fpga-vs-altera-fpga

[3] "altera_fpga.pdf." Accessed: Jun. 15, 2024. [Online]. Available: https://www.apel.ee.upatras.gr/dic/notes/altera_fpga.pdf

[4] Δ. Ηλιάδης-Αποστολίδης, *Homomorphic Encryption με τη βιβλιοθήκη Microsoft SEAL και επιτάχυνση με FPGA*. 2021. [Online]. https://ikee.lib.auth.gr/record/336388/files/Iliadis-Apostolidis%20Dimosthenis.pdf?version=1

[5] A. Sideris, T. Sanida, M. V. Sanida, M. Dossis, and M. Dasygenis, "Accelerate Processing of Image with the Keccak-512 Algorithm on Cryptoprocessor," in *2023 8th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, Aug. 2023, pp. 1–4. doi: 10.1109/SEEDA-CECNSM61561.2023.10470528.

[6] M. Kammoun, M. Elleuchi, M. Abid, and M. S. BenSaleh, "FPGA-based implementation of the SHA-256 hash algorithm," in *2020 IEEE International Conference on Design & Test of Integrated Micro & Nano-Systems (DTS)*, Jun. 2020, pp. 1–6. doi: 10.1109/DTS48731.2020.9196134.

[7] Ν. Σ. Μεταξάκης, "Σχεδιασμός αλγορίθμου SHA-256 στην γλώσσα VHDL και υλοποίησή του σε FPGA," Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης, 2022. Accessed: May 28, 2024. [Online]. Available: https://ikee.lib.auth.gr/record/345355

[8] "AMD KR260 Robotics Starter Kit," AMD. Accessed: May 25, 2024. [Online]. Available: https://www.amd.com/en/products/system-on-modules/kria/k26/kr260-robotics-starter-kit.html

[9] "UltraScale MPSoC Architecture." Accessed: Jun. 15, 2024. [Online]. Available: https://www.xilinx.com/products/technology/ultrascale-mpsoc.html

[10] "What is an FPGA." Accessed: Apr. 28, 2024. [Online]. Available: https://www.latticesemi.com/en/What-is-an-FPGA

[11] AMD, "Introduction to FPGA Design with Vivado High-Level Synthesis," 2019.

[12] C. services *et al.*, "FPGA or Structured ASIC: Which Is Right for You? Intel," Intel. Accessed: Jun. 15, 2024. Available: https://www.intel.com/content/www/us/en/products/programmable/fpga-vs-structured-asic.html

[13] M. E. de Lima and D. J. Kinniment, "Sea-of-gates architecture," *Microelectron. J.*, vol. 26, no. 5, pp. 431–440, Jul. 1995, doi: 10.1016/0026-2692(95)98945-N.

[14] “AMD Technical Information Portal.” Accessed: Jun. 18, 2024. [Online]. Available: https://docs.amd.com/v/u/en-US/ug998-vivado-intro-fpga-design-hls
[15] W. Green, “Beginning FPGA Graphics,” Project F. Accessed: Jun. 15, 2024. [Online]. Available: https://projectf.io/posts/fpga-graphics/
[16] “Difference between Verilog and VHDL - Naukri Code 360.” Accessed: Jun. 15, 2024. [Online]. Available: https://www.naukri.com/code360/library/difference-between-verilog-and-vhdl
[17] “ModelSim,” *Wikipedia*. Mar. 17, 2023. Accessed: Jun. 15, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=ModelSim&oldid=1145117528
[18] “Intel Quartus Prime,” *Wikipedia*. May 09, 2024. Accessed: Jun. 15, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Intel_Quartus_Prime&oldid=1223029127
[19] “What is an FPGA? Field Programmable Gate Array,” AMD. Accessed: Jun. 15, 2024. [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html
[20] “FPGA Video and Vision Solutions - Intel® FPGA,” Intel. Accessed: Jun. 15, 2024. [Online]. Available: https://www.intel.com/content/www/us/en/smart-video/products/programmable/overview.html
[21] “Vitis HLS.” Accessed: Jun. 15, 2024. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-hls.html
[22] “Navigating Content by Design Process • Vitis High-Level Synthesis User Guide (UG1399) • Reader • AMD Technical Information Portal.” Accessed: Jun. 15, 2024. [Online]. Available: https://docs.amd.com/r/en-US/ug1399-vitis-hls/Navigating-Content-by-Design-Process
[23] “Xilinx/PYNQ.” Xilinx, Jun. 14, 2024. Accessed: Jun. 15, 2024. [Online]. Available: https://github.com/Xilinx/PYNQ
[24] “Downloads,” AMD. Accessed: Jun. 15, 2024. [Online]. Available: https://www.xilinx.com/support/download.html
[25] “SHA-2,” *Wikipedia*. Apr. 26, 2024. Accessed: May 08, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=SHA-2&oldid=1220891789
[26] A. W. Appel, “Verification of a Cryptographic Primitive: SHA-256,” *ACM Trans. Program. Lang. Syst.*, 2015.
[27] “Announcing Approval of Federal Information Processing Standard (FIPS) 180-2, Secure Hash Standard; a Revision of FIPS 180-1,” Federal Register. Accessed: Jun. 15, 2024. [Online]. Available: https://www.federalregister.gov/documents/2002/08/26/02-21599/announcing-approval-of-federal-information-processing-standard-fips-180-2-secure-hash-standard-a
[28] H. Gilbert and H. Handschuh, “Security Analysis of SHA-256 and Sisters,” in *Selected Areas in Cryptography*, M. Matsui and R. J. Zuccherato, Eds., Berlin, Heidelberg: Springer, 2004, pp. 175–193. doi: 10.1007/978-3-540-24654-1_13.
[29] J. Mehta, “SHA1 Vs. SHA256 - What’s the Difference Between?,” CheapSSLWeb.com Blog. Accessed: Jun. 15, 2024. [Online]. Available: https://cheapsslweb.com/blog/sha1-vs-sha256/
[30] W. Penard and T. van Werkhoven, “On the Secure Hash Algorithm family,” *blog.infocruncher.com*, Available: https://blog.infocruncher.com/resources/ethereum-whitepaper-annotated/On%20the%20Secure%20Hash%20Algorithm%20family%20(2008).pdf
[31] Greg, GitHub Repository “in3rsha/sha256-animation.” May 27, 2024. Accessed: May 27, 2024. [Online]. Available: https://github.com/in3rsha/sha256-animation
[32] B. V. Fekete, GitHub Repository “fbv81bp/SHA2-256_in_VHDL: VHDL Secure Hash Algorithm 2 cores,” SHA2-256_in_VHDL. Accessed: Jun. 15, 2024. [Online]. Available: https://github.com/fbv81bp/SHA2-256_in_VHDL
[33] K. Kv, “Kria KV260 Vision AI Starter Kit Data Sheet,” 2022.
[34] “Zynq UltraScale+ MPSoC,” AMD. Accessed: May 27, 2024. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html
[35] “Zynq UltraScale+ MPSoC Data Sheet: Overview (DS891),” 2022.

[36] Κ. Μπαρκάτσας, “Υλοποίηση ML-AI σε high-end FPGA,” Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης, 2022. [Online]. Available: https://ikee.lib.auth.gr/record/342735/?ln=el
[37] “Directives • UltraFast Design Methodology Guide for FPGAs and SoCs (UG949) • Reader • AMD Technical Information Portal.” Accessed: Jun. 15, 2024. [Online]. Available: https://docs.amd.com/r/en-US/ug949-vivado-design-methodology/Directives
[38] Κ. Π. Αξιώτης, “Υλοποίηση αλγορίθμου Wavelets σε FPGA με τη χρήση τεχνικών High-Level Synthesis,” Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης, 2020. [Online]. Available: https://ikee.lib.auth.gr/record/315328?ln=el
[39] “M_AXI Resources • Vitis High-Level Synthesis User Guide (UG1399) • Reader • AMD Technical Information Portal.” Accessed: Jun. 15, 2024. [Online]. Available: https://docs.amd.com/r/en-US/ug1399-vitis-hls/M_AXI-Resources
[40] “S_AXILITE Example • Vitis High-Level Synthesis User Guide (UG1399) • Reader • AMD Technical Information Portal.” Accessed: Jun. 15, 2024. [Online]. Available: https://docs.amd.com/r/en-US/ug1399-vitis-hls/S_AXILITE-Example
[41] “AXI Interconnect,” AMD. Accessed: Jun. 15, 2024. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi_interconnect.html
[42] “Setting up the SD Card Image,” AMD. Accessed: Jun. 15, 2024. [Online]. Available: https://www.amd.com/en/products/system-on-modules/kria/k26/kv260-vision-starter-kit/getting-started-ubuntu/setting-up-the-sd-card-image.html
[43] “Install Ubuntu on AMD,” Ubuntu. Accessed: Jun. 15, 2024. [Online]. Available: https://ubuntu.com/download/amd
[44] “Xilinx/kria-vitis-platforms.” Xilinx, Jun. 10, 2024. Accessed: Jun. 19, 2024. [Online]. Available: https://github.com/Xilinx/kria-vitis-platforms
[45] “PYNQ,” PYNQ. Accessed: Apr. 30, 2024. [Online]. Available: http://www.pynq.io/
[46] “IS-2_report.pdf.” Accessed: Jun. 14, 2024. [Online]. Available: https://archive.alvb.in/svn/VHDL/SHA256/reference/IS-2_report.pdf
[47] R. García, I. Algredo-Badillo, M. Morales-Sandoval, C. Feregrino, and R. Cumplido, “A compact FPGA-based processor for the Secure Hash Algorithm SHA-256,” *Comput. Electr. Eng.*, vol. 40, Jan. 2013, doi: 10.1016/j.compeleceng.2013.11.014.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

κ.λπ.	και λοιπά
AI	Artificial intelligence
ASIC	Application-specific integrated circuit
ASSP	Application Specific Standard Products
CHF	Cryptographic Hash Function
DMA	Direct Memory Access
FF	Flip-Flop
FIFO	First in first out
FPGAs	Field Programmable Gate Arrays
HDL	Hardware Description Language
HLS	High Level Synthesis
HW	Hardware
IC	Integrated Circuits
I/O	Inputs/Outputs
IP	Intellectual Property
IPI	IP Integrator
LUT	Look-up Tables
MCU	Microcontroller Unit
NIST	National Institute of Standards and Technology
PCW	Processing Configuration Wizard
PL	Programmable Logic
PLL	Phase-Locked Loops
PS	Processing System
SoC	System on chip
SOM	System on Module
TSMC	Taiwan Semiconductor Manufacturing Comp.

Απόδοση Ξενόγλωσσων Όρων

Αλγόριθμος	Algorithm
Αποτυπώματα	Fingerprints
Αφέντης	Master
Δεδομένα	Data
Δημόσιο	Public
Εικόνα	Image
Εκκίνηση	Boot
Ελεύθερης πρόσβασης	Open-source
Ενσωματωμένα	Embedded
Επανάληψη βρόγχου	Loop
Ιδιωτικό	Private
Καθυστέρηση	Latency
Κλειδί	Key
Κουτί	Box
Κρυπτογράφηση	Hashing
Κυματομορφή	Waveform
Λογισμικό	Software
Μήνυμα	Message
Μετατόπιση	Shift
Πλακέτα Αξιολόγησης	Evaluation Board
Προτυποποίηση	Prototyping
Ροή εργασίας	Workflow
Ρομποτική	Robotics
Σημειωματάριο	Notebooks
Σκλάβος	Slave
Συμπλήρωση	Padding
Σύνθεση	Synthesis
Συσκευή	Device
Σύστημα σε τσιπ	SoC
Τεχνητή Νοημοσύνη	AI
Υλικό	Hardware
Υπογράμμιση Σύνταξης	Syntax Highlighting
Ψηφιακή σφραγίδα	Digital Timestamp