



ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

**ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ
ΤΕΧΝΟΛΟΓΙΑ ΥΛΙΚΟΥ ΚΑΙ ΛΟΓΙΣΜΙΚΟΥ: ΣΧΕΔΙΑΣΗ ΚΑΙ ΑΝΑΠΤΥΞΗ
ΔΙΑΧΥΤΩΝ ΣΥΣΤΗΜΑΤΩΝ ΥΠΟΛΟΓΙΣΜΟΥ**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

HOU-CS-PGP-2026-06

Σχεδιασμός και Ανάπτυξη Συστήματος Ασφάλειας, σε Υλικό

ΠΑΓΙΑΤΗΣ ΣΠΥΡΙΔΩΝ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΣΚΛΑΒΟΣ ΝΙΚΟΛΑΟΣ

ΠΑΤΡΑ 2026



HELLENIC OPEN UNIVERSITY SCHOOL OF SCIENCES AND TECHNOLOGY





Διπλωματική Εργασία *HOU-CS- PGP-2026-06*

Σχεδιασμός και Ανάπτυξη Συστήματος Ασφάλειας, σε Υλικό

Παγιάτης Σπυρίδων



© ΕΑΠ, 2026

Η παρούσα διατριβή, η οποία εκπονήθηκε στα πλαίσια του ΜΠΣ ΣΔΥ, και τα λοιπά αποτελέσματα της αντίστοιχης Διπλωματικής Εργασίας (ΔΕ) αποτελούν συνιδιοκτησία του ΕΑΠ και του φοιτητή, ο καθένας από τους οποίους έχει το δικαίωμα ανεξάρτητης χρήσης και αναπαραγωγής τους (στο σύνολο ή τμηματικά) για διδακτικούς και ερευνητικούς σκοπούς, σε κάθε περίπτωση αναφέροντας τον τίτλο, το συγγραφέα και το ΕΑΠ όπου εκπονήθηκε η ΔΕ καθώς και τον επιβλέποντα και την επιτροπή κρίσης.



Σχεδιασμός και Ανάπτυξη Συστήματος Ασφάλειας, σε Υλικό

Παγιάτης Σπυρίδων

Επιβλέπων
Α' καθηγητής

Σκλάβος Νικόλαος
Καθηγητής Πανεπιστημίου Πατρών

Επιβλέπων
Β' καθηγητής

Νικοπολιτίδης Πέτρος
Καθηγητής Πανεπιστημίου Θεσ/κης ΑΠΘ

Περίληψη

Στην εποχή που διανύουμε έχει ήδη ξεκινήσει η ραγδαία ανάπτυξη της διασύνδεσης ετερογενών συστημάτων. Η τάση αυτή ενισχύθηκε με την έλευση του Internet of Things (IoT) και την εξέλιξη των ασύρματων επικοινωνιών 5G/6G.

Η ανάγκη λοιπόν για αυθεντικοποίηση και προστασία των δεδομένων έχει γίνει επιτακτική, έτσι ώστε να διασφαλιστεί η ιδιωτικότητα και η προσωπική ασφάλεια των χρηστών. Ωστόσο, ένα βασικό πρόβλημα που ανακύπτει είναι ότι οι νέες τεχνολογίες εστιάζουν στην ενεργειακή αποδοτικότητα, γιατί η πλειονότητα αυτών είναι αυτόνομες και ασύρματες. Παράλληλα, ακόμη και στα ενσύρματα συστήματα, η ανάγκη για ενεργειακή βελτιστοποίηση εντείνεται λόγω της πράσινης μετάβασης.

Σε αυτό το πλαίσιο, οι παραδοσιακές κρυπτογραφικές λύσεις δεν επαρκούν, δημιουργώντας την ανάγκη για κρυπτογραφία χαμηλής κατανάλωσης ενέργειας, περιορισμένη χρήση υπολογιστικών πόρων και μικρής επιφάνειας υλοποίησης. Τα χαρακτηριστικά αυτά επιτρέπουν την ενσωμάτωσή τους πάνω στις αυτόνομες ασύρματες συσκευές. Η διπλωματική εργασία εστιάζει πάνω στην ελαφροβαρή κρυπτογραφία (lightweight cryptography) και στον αλγόριθμο ASCON ο οποίος προτάθηκε το 2014. Πιο συγκεκριμένα επικεντρώνεται στην έκδοση AEAD128, σύμφωνα με το πρότυπο SP 800-232 το οποίο δημοσιεύτηκε τον Αύγουστο του 2025.

Η υλοποίηση του αλγορίθμου έγινε σε Artix-7 FPGA, χρησιμοποιώντας τη γλώσσα περιγραφής υλικού VHDL και το περιβάλλον ανάπτυξης Xilinx Vivado. Ακολουθήθηκε αρχιτεκτονική μιας μετάθεσης (permutation) ανά κύκλο ρολογιού, έτσι ώστε να μειωθεί η ενεργειακή κατανάλωση όπως και η επιφάνεια υλοποίησης, καθιστώντας την προσέγγιση κατάλληλη για εφαρμογές περιορισμένων πόρων. Τέλος γίνεται αξιολόγηση της υλοποίησης ως προς την ταχύτητα, την κατανάλωση ενέργειας και τη δεσμευμένη επιφάνεια υλικού και τα αποτελέσματα συγκρίνονται με παρόμοιες ερευνητικές εργασίες στη βιβλιογραφία.



Λέξεις-κλειδιά: AEAD128, Ascon, Lightweight cryptography, IoT , AMD, Vivado, VHDL

Περιεχόμενο: Κείμενο, γραφήματα, εικόνες, πρόγραμμα σε VHDL, project στο Vivado, πίνακες, διαγράμματα, αναλύσεις

«Η εκπαίδευση δεν αλλάζει τον κόσμο. Αλλάζει τους ανθρώπους που θα αλλάξουν τον κόσμο»

Paulo Freire



Abstract

In the current era, the interconnection of systems has been rapidly expanding, driven largely by the emergence of the Internet of Things (IoT) and the evolution of wireless communication technologies such as 5G and 6G. As a result, the need for secure authentication and data protection has become increasingly critical in order to safeguard privacy and user security.

However, a key challenge arises from the fact that modern technologies place strong emphasis on energy efficiency, since the majority of these systems are autonomous and wireless. Even in wired systems, energy efficiency remains a priority due to the ongoing transition toward greener technologies. Consequently, traditional cryptographic solutions are no longer sufficient, leading to the need for lightweight cryptographic approaches that require low energy consumption, minimal resources, and small integration area, making them suitable for integration into autonomous wireless devices.

This work focuses on lightweight cryptography, specifically on the ASCON algorithm, which was introduced in 2014. More precisely, we concentrate on ASCON-AEAD128, its latest version standardized as SP 800-232 and published in August 2025.

The implementation was carried out on an Artix-7 FPGA using the VHDL hardware description language within the Xilinx Vivado environment. A one-permutation per clock cycle architecture was adopted in order to reduce both energy consumption and hardware area, making the design well-suited for the intended application. Finally, the implementation is evaluated in terms of performance, power consumption, and area, and is compared with other research works.

Design and development of a hardware security system

Pagiatis Spiros

Supervisor Professor

Sklavos Nikolaos

Professor, University of Patras

Co-Supervisor Professor

Nikopolitidis Petros

Professor, Aristotle University AUTH



Περιεχόμενα

ΠΕΡΙΛΗΨΗ.....	4
ABSTRACT	6
1. ΕΙΣΑΓΩΓΗ.....	11
1.1 Η ανάγκη της κρυπτογραφίας σε ενσωματωμένα συστήματα.....	11
1.2 Στόχοι και αντικείμενο της εργασίας.....	12
1.3 Μεθοδολογική προσέγγιση	13
1.4 Χαρακτηριστικά του συστήματος υλοποίησης.....	14
1.5 Δομή της διπλωματικής εργασίας.....	14
1.6 Συσχέτιση με τους στόχους βιώσιμης ανάπτυξης του ΟΗΕ	15
2. ΕΡΕΥΝΗΤΙΚΗ ΑΝΑΣΚΟΠΗΣΗ	16
2.1 Κρυπτογραφία.....	16
2.1.1 Ελαφροβαρής Κρυπτογραφία ή <i>Lightweight Cryptography</i>	17
2.2 ASCON	18
2.2.1 Η δομή του ASCON.....	20
2.2.2 Ο αλγόριθμος AEAD128	22
2.2.3 Ο αλγόριθμος Hash256.....	25
2.3 FPGAs	26
2.3.1 Ανάγκη χρήσης FPGAs	27
2.3.2 Artix 7	27
3. ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΚΑΙ ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑ ΣΥΣΤΗΜΑΤΟΣ.....	30
3.1 Register Transfer Level	30
3.2 Ανάλυση και λειτουργικότητα μονάδων.....	33
4. ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΟΣ ΣΤΟ VIVADO.....	36
4.1 System Synthesis	36
4.2 System Implementation.....	38
5. ΑΞΙΟΛΟΓΗΣΗ	41
5.1 Έλεγχος ορθής λειτουργίας.....	41
5.1.1 Vivado simulation	41
5.1.2 Δημιουργία δεδομένων ελέγχου	45
5.1.3 Έλεγχος ενδιάμεσων τιμών.....	46
5.2 Μετρήσεις	49
5.2.1 Επιφάνεια υλοποίησης συστήματος	49
5.2.2 Κατανάλωση ενέργειας συστήματος.....	50
5.2.3 Throughput	52
5.3 Σύγκριση αποτελεσμάτων.....	53
6. ΣΥΜΠΕΡΑΣΜΑΤΑ	60
7. ΑΝΑΦΟΡΕΣ.....	61



Λίστα Εικόνων

Εικόνα 1: Προκλήσεις κρυπτογραφίας.....	12
Εικόνα 2: Lightweight Cryptography Diagram [12]	17
Εικόνα 3: AEAD Encryption Diagram	19
Εικόνα 4: ASCON Round Constant (RC).....	20
Εικόνα 5: ASCON SBox Logic.....	21
Εικόνα 6: ASCON SBox LUT & Circuit.....	21
Εικόνα 7: ASCON Diffusion Layer Diagram	21
Εικόνα 8: ASCON Diffusion Layer Logic.....	22
Εικόνα 9: ASCON AEAD128 Encryption Architecture	23
Εικόνα 10: ASCON AEAD128 Decryption Architecture.....	23
Εικόνα 11: ASCON AEAD128 Pseudocode.....	24
Εικόνα 12: ASCON Hash256 Architecture.....	25
Εικόνα 13: ASCON Hash256 Pseudocode	26
Εικόνα 14: Current System Utilization	29
Εικόνα 15: Αρχιτεκτονική συστήματος σε επίπεδο καταχωρητών (RTL).....	30
Εικόνα 16: FSMState_reg bus.....	31
Εικόνα 17: FSMCommand_reg bus.....	32
Εικόνα 18: busType_reg bus.....	32
Εικόνα 19: round_reg bus	32
Εικόνα 20: Current System Block Diagram.....	33
Εικόνα 21: System Dataflow Design	36
Εικόνα 22: Synthesis System Utilization Summary.....	36
Εικόνα 23: Synthesis Power Consumption Summary.....	38
Εικόνα 24: Implementation System Utilization Summary.....	39
Εικόνα 25: Implementation Power Consumption Summary.....	39
Εικόνα 26: Testbench Input Data.....	41
Εικόνα 27: System Key Input	42
Εικόνα 28: System Nonce Input.....	42
Εικόνα 29: System Associate Data Input	43
Εικόνα 30: System Message Input	43
Εικόνα 31: System Ciphertext Output.....	43
Εικόνα 32: System Tag Output	44
Εικόνα 33: Hash256 MSB Output (255-128).....	44
Εικόνα 34: Hash256 LSB Output (127-0).....	44
Εικόνα 35: Vivado Simulation Log.....	45
Εικόνα 36: pyAscon Data Out	45
Εικόνα 37: pyAscon/Vivado Results Compare.....	46
Εικόνα 38: pyAscon print debug example	47
Εικόνα 39: Vivado Waveform Before Init	48
Εικόνα 40: Vivado Waveform After Init	48
Εικόνα 41: Σύνοψη implementation συστήματος	49
Εικόνα 42: Σύνοψη πρωτευόντων δομικών στοιχείων υλοποίησης.....	49
Εικόνα 43: Περιβάλλον εργασίας του Artix7	50
Εικόνα 44: Αναφορά κατανάλωσης συστήματος στα 200MHz.....	50
Εικόνα 45: Κατανάλωση I/O blocks στα 200MHz	51



Εικόνα 46: Συνολική αναφορά κατανάλωσης στα 100MHz.....	51
Εικόνα 47: Vivado Simulation Waveform	52
Εικόνα 48: Message Process Time.....	53
Εικόνα 49: Σύγκριση Throughput του σχεδιασμού με παρόμοιες εργασίες	57
Εικόνα 50: Σύγκριση επιφάνειας υλοποίησης του σχεδιασμού με παρόμοιες εργασίες.....	58
Εικόνα 51: Σύγκριση συχνότητας ρολογιού του σχεδιασμού με παρόμοιες εργασίες.....	58
Εικόνα 52: Σύγκριση κατανάλωσης ισχύος του σχεδιασμού με παρόμοιες εργασίες	58
Εικόνα 53: Σύγκριση TP & Area του σχεδιασμού με παρόμοιες εργασίες.....	59
Εικόνα 54: Σύγκριση Mbps/LUTs του σχεδιασμού με παρόμοιες εργασίες.....	59
Εικόνα 55: Σύγκριση Κατανάλωσης & Συχνότητας του σχεδιασμού με παρόμοιες εργασίες	59

Λίστα Πινάκων

Πίνακας 1: Σύγκριση ενσωματωμένων συστημάτων	12
Πίνακας 2: Χαρακτηριστικά Ascon version 1.2	18
Πίνακας 3: ASCON Initialization Vectors Parameters	18
Πίνακας 4: ASCON AEAD128 Initialization Vectors Values.....	19
Πίνακας 5: Παραμετροποίηση τιμών διανυσμάτων (IV).....	19
Πίνακας 6: Υπολογισμός τιμών διανυσμάτων (IV).....	19
Πίνακας 7: Σύγκριση FPGAs με άλλες λύσεις κρυπτογραφικών συστημάτων	27
Πίνακας 8: Series 7 FPGAs Characteristics.....	28
Πίνακας 9: Artix7 Devices Characteristics.....	28
Πίνακας 10: FSMState_reg.....	31
Πίνακας 11: FSMCommand_reg States.....	31
Πίνακας 12: busType_reg States	32
Πίνακας 13: Synthesis Timing Summary	37
Πίνακας 14: Synthesis Hold Slacks (Intra clock paths).....	37
Πίνακας 15: Implementation Timing Summary	38
Πίνακας 16: Implementation Hold Slacks (intra clock paths)	39
Πίνακας 17: System Implementation I/O pins Characteristics	40
Πίνακας 18: Compare pyAscon/Vivado States.....	48
Πίνακας 19: Πόροι συστήματος υλοποίησης (Current).....	53
Πίνακας 20: Πόροι υλοποίησης της βιβλιογραφικής αναφοράς [8]	53
Πίνακας 21: Πόροι υλοποίησης της βιβλιογραφικής αναφοράς [26]	55
Πίνακας 22: Πόροι υλοποίησης της βιβλιογραφικής αναφοράς [27]	56
Πίνακας 23: Πόροι υλοποίησης της βιβλιογραφικής αναφοράς [28]	56
Πίνακας 24: Πόροι υλοποίησης της βιβλιογραφικής αναφοράς [29]	57



Συντομογραφίες και Ακρωνύμια

IoT	Internet of Things
FPGA	Field Programming Gate Array
VHDL	Verry high speed Hardware Description Language
ASIC	Application Specific Integrated Circuits
MCU	Micro Controller Unit
NIST	National Institute of Standards and Technology
SPN	Substitution Permutation Networks
AEAD	Authenticated Encryption with Associated Data
FSM	Finite State Machine
LWC	Light Weight Cryptography
AD	Associate Data
Plaintext	Μήνυμα
Ciphertext	Κρυπτογραφημένο κείμενο
MSB	Most Significant Bit
LSB	Least Significant Bit
RTL	Register Transfer Level
HDL	Hardware Description Language
HLS	High Level Synthesis
S	State των 320bits
s(i)	Η λέξη i των 64bits του state S
KAT	Known Answer Tests
IV	Initialization Vector
CLB	Configurable Logic Block
LUT	Look Up Table
FF	Flip Flop



1. Εισαγωγή

Η εποχή που διανύουμε χαρακτηρίζεται από τη ραγδαία εξέλιξη της ανάπτυξης των ασύρματων τεχνολογιών και την ευρεία διασύνδεση όλων των συσκευών και μηχανών αναμεταξύ τους. Αυτές οι διασυνδέσεις καλύπτουν ένα ευρύ φάσμα, ξεκινώντας από το οικιακό περιβάλλον (π.χ., πόρτες, παράθυρα, στόρια, φώτα, θέρμανση, νερό, ηλεκτρικό), επεκτεινόμενες στις μεταφορές (π.χ., αυτόνομη οδήγηση, έλεγχος κυκλοφορίας, έλεγχος οχημάτων, αποστολή τοποθεσίας, εγγύτητα οχημάτων, έλεγχος και βελτίωση κυκλοφορίας, εύρεση parking) και φτάνοντας ως και την βιομηχανία (π.χ., επιτήρηση και λειτουργία μηχανών, βελτίωση παραγωγικότητας, ασφάλεια προσωπικού, ασφάλεια υποδομών).

Παράλληλα οι φορέσιμες συσκευές (wearables) όπως τα έξυπνα ρολόγια, έξυπνα γυαλιά, βιομετρικοί αισθητήρες (δακτυλίδια, βραχιόλια), και φορετές συσκευές εντοπισμού να ενσωματώνονται όλο και περισσότερο στην καθημερινότητα. Ιδιαίτερη θέση κατέχουν τα κινητά τηλέφωνα, τα οποία συγκεντρώνουν πολλές από τις παραπάνω λειτουργίες και επιπλέον, λειτουργούν ως μέσα ταυτοποίησης, επιτρέποντας την απομακρυσμένη αλληλεπίδραση και εκτέλεση ενεργειών.

Όπως δείχνουν τα παραπάνω, οι ασύρματες επικοινωνίες και η διασύνδεση των διάφορων φορέσιμων και φορητών συσκευών, γίνεται επιτακτική και αναγκαία. Κατά συνέπεια, η διασφάλιση της ιδιωτικότητας και η προστασία των προσωπικών δεδομένων αποτελούν κρίσιμες απαιτήσεις, ιδιαίτερα υπό περιορισμούς κατανάλωσης ισχύος και επιφάνειας υλοποίησης, ώστε τα συστήματα να μπορούν να ενσωματωθούν σε μικροσκοπικές συσκευές με περιορισμένους πόρους.

Σε αυτό το σημείο υστερούν οι παραδοσιακές μορφές κρυπτογραφίας και έχει αναπτυχθεί μια νέα κατηγορία, με τον όρο Lightweight Cryptography (ελαφροβαρής κρυπτογραφία).

1.1 Η ανάγκη της κρυπτογραφίας σε ενσωματωμένα συστήματα

Το βασικό πρόβλημα της παραδοσιακής κρυπτογραφίας (π.χ., αλγόριθμοι AES, RSA) [1] είναι ότι χρειάζονται σημαντικούς υπολογιστικούς και ενεργειακούς πόρους, ενώ έχουν και αυξημένες απαιτήσεις μνήμης. Αυτό έχει ως αποτέλεσμα μειωμένη συνολική αποδοτικότητα με αρκετά υψηλό κόστος [2]. Συνεπώς, στην ασύρματη ασφάλεια IoT συσκευών με περιορισμένους πόρους, βρίσκει εφαρμογή η Lightweight Cryptography [3], σκοπός της οποίας είναι να λειτουργεί με την μικρότερη κατανάλωση ενέργειας και σε αρκετά μικρή επιφάνεια υλοποίησης. Η ελαφροβαρής κρυπτογραφία επιτυγχάνει την αποδοτικότητά της μέσω χρήσης απλών μαθηματικών πράξεων (όπως προσθέσεις, ολισθήσεις και λογικές πράξεις xor, or, and, not), μικρότερων δομών, μειωμένου αριθμού επαναλήψεων (rounds) και γενικότερης βελτιστοποίησης για ενσωματωμένα συστήματα.



Τα κύρια ενσωματωμένα συστήματα για την υλοποίηση τέτοιων συστημάτων είναι τα FPGA, ASIC [4] και MCU [5], όπου το καθένα από αυτά έχει τις ιδιαιτερότητες που συνοψίζονται στον Πίνακα 1.

Πίνακας 1: Σύγκριση ενσωματωμένων συστημάτων

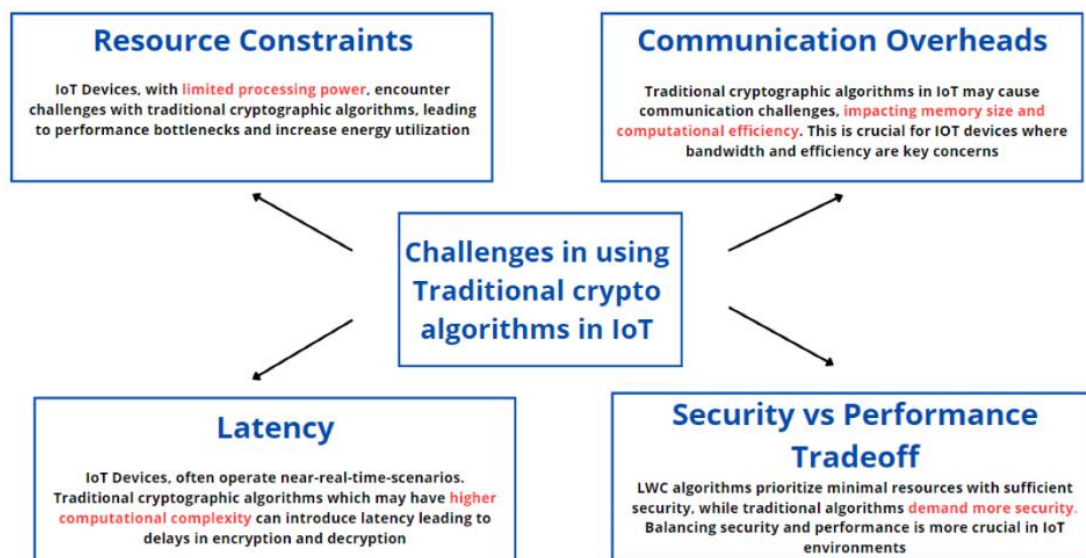
	FPGA	ASIC	MCU
Κόστος Αγοράς	Μέτριο/Μεγάλο	Μέτριο	Μικρό
Κόστος Υλοποίησης	Μικρό	Τεράστιο	Μικρό
Ταχύτητα	Μέτρια/Μεγάλη	Μεγάλη	Μικρή
Ευελιξία	Μεγάλη	Ελάχιστη	Τεράστια
Κατανάλωση	Μέτρια	Ελάχιστη	Μικρή/Μέτρια

Στην παρούσα εργασία επιλέχθηκε η χρήση FPGA, λόγω της ευνοϊκής σχέσης απόδοσης προς κόστος.

1.2 Στόχοι και αντικείμενο της εργασίας

Στόχος της εργασίας είναι η υλοποίηση ενός αποδοτικού αυτόνομου συστήματος ασφάλειας σε επίπεδο υλικού, βασισμένο στις σύγχρονες εξελίξεις της ελαφροβαρούς κρυπτογραφίας. Αναλυτικότερα επιλέγεται ο αλγόριθμος ASCON [6] ο οποίος παρέχει την βέλτιστη απόδοση σε σχέση με την κατανάλωση πόρων και επιπέδου ασφάλειας, καθιστώντας τον κατάλληλο για εφαρμογές ενσωματωμένων συστημάτων και συσκευών περιορισμένων πόρων και ενέργειας.

Στο παρακάτω σχήμα (Εικόνα 1) φαίνονται οι προκλήσεις που καλούνται να αντιμετωπίσουν οι νέοι αλγόριθμοι κρυπτογραφίας [6].



Εικόνα 1: Προκλήσεις κρυπτογραφίας



Πιο συγκεκριμένα, αντικείμενο μελέτης του αλγορίθμου ASCON αποτελεί η τελευταία έκδοση του, που ονομάζεται AEAD128, η οποία διαχειρίζεται τα δεδομένα σε little endianness (για να είναι αποδοτικότερη σε ενσωματωμένα συστήματα μειωμένων πόρων) και προτυποποιείται από το NIST μέσω του προτύπου SP 800-232 έκδοσης Αυγούστου 2025 [7]. Η υλοποίηση πραγματοποιείται στο περιβάλλον Vivado της εταιρίας Xilinx με χρήση της γλώσσας περιγραφής υλικού VHDL, και ως πλατφόρμα υλοποίησης χρησιμοποιείται το Artix 7 της AMD. Ο σχεδιασμός στοχεύει στην εξισορρόπηση ανάμεσα στην απόδοση και στην κατανάλωση πόρων. Για τον σκοπό αυτό επιλέχθηκε μια αρχιτεκτονική περιορισμένου throughput (μιας μετάθεσης ανά κύκλο ρολογιού), αλλά αυξημένης ενεργειακής απόδοσης και μειωμένης επιφάνειας υλοποίησης.

Επιπλέον, η σχεδίαση ακολουθεί αρθρωτή (modular) δομή, επιτρέποντας την ευκολότερη κατανόηση, τον εντοπισμό σφαλμάτων και τη μελλοντική επέκταση ή βελτιστοποίηση του συστήματος.

Στο τέλος με τα δεδομένα που έχουν συλλεχθεί από τις διάφορες αναλύσεις και μετρήσεις που έγιναν στο περιβάλλον Vivado, πραγματοποιείται αξιολόγηση και σύγκριση με άλλες ερευνητικές εργασίες που έχουν δημοσιευθεί, προκειμένου να εξαχθούν συμπεράσματα σχετικά με τα πλεονεκτήματα και τα σημεία που χρήζουν περαιτέρω βελτίωσης.

1.3 Μεθοδολογική προσέγγιση

Η μεθοδολογία που ακολουθήθηκε για την υλοποίηση της εργασίας ακολούθησε τα παρακάτω στάδια. Καταρχάς ξεκίνησε με εκτενή μελέτη της σχετικής βιβλιογραφίας πάνω στους αλγορίθμους ασφαλείας μειωμένων πόρων στα ενσωματωμένα συστήματα [1] [2] [3] [6]. Στη συνέχεια πραγματοποιήθηκε αξιολόγηση στο είδος του ενσωματωμένου συστήματος που θα ήταν ιδανικό για την υλοποίηση του αλγορίθμου [4] [5] [8].

Στην συνέχεια μέσω συνεννόησης και έγκρισης από τον επιβλέποντα καθηγητή κ. Σκλάβο, έγινε η επιλογή του ASCON AEAD128 ως αλγόριθμου κρυπτογραφίας, και του FPGA Artix 7 της AMD λόγω της βέλτιστης απόδοσης σε σχέση με το κόστος, την κατανάλωση και την ταχύτητα που θα ήταν δυνατόν να επιτευχθεί.

Πραγματοποιήθηκε ανάλυση των λειτουργικών και υλικών απαιτήσεων όπως και περιορισμών που τυχόν προκύψουν, και μελετήθηκαν υλοποιήσεις σε γλώσσα C που υπάρχουν στο repository του ASCON στο github [9]. Επιπλέον, έγινε χρήση του εργαλείου pyAscon [10] σε Python, το οποίο επέτρεψε την παρακολούθηση της εσωτερικής κατάστασης του αλγορίθμου και διευκόλυνε σημαντικά τη διαδικασία κατανόησης και αποσφαλμάτωσης βήμα προς βήμα.



Ακολούθως μελετήθηκε το περιβάλλον Vivado και υλοποιήθηκαν παραδείγματα σε VHDL και προσομοίωση τους, με σκοπό την εξοικείωση με το περιβάλλον ανάπτυξης.

Με το πέρας της υλοποίησης έγινε τελικός έλεγχος καλής λειτουργίας μέσω fixed test vectors των 128 bits, και μέσω του simulation ελέγχθηκαν, εντοπίστηκαν και αντιμετωπίστηκαν προβλήματα χρονισμού (timing violations) που είχαν προκύψει. Τέλος έγινε η synthesis και το implementation στο Vivado, όπου εξάχθηκαν δεδομένα μέγιστης συχνότητας, κατανάλωσης, κρίσιμων μονοπατιών, χρήσης πόρων, διαγραμμάτων κτλ. και αξιολογήθηκαν συγκριτικά με τη σχετική βιβλιογραφία.

1.4 Χαρακτηριστικά του συστήματος υλοποίησης

Κάνοντας μια σύνοψη σύμφωνα με τα προαναφερόμενα του Κεφαλαίου 1, υλοποιείται ένα σύστημα κρυπτογραφίας που εστιάζει στην απλότητα, στην χαμηλή κατανάλωση και στην επεκτασιμότητα αλλά και στην μείωση της επιφάνειας υλοποίησης. Για να επιτευχθεί αυτό ακολουθήθηκε λογική modular σχεδίασης με διακριτές μονάδες, όπου γίνεται μια μετάθεση ανά κύκλο ρολογιού. Αυτό φυσικά επηρεάζει την ταχύτητα του, διατηρώντας παράλληλα ένα απλούστερο σύστημα, χωρίς περίπλοκες τεχνικές παραλληλίας, γεγονός που το καθιστά κατάλληλο για το πλήθος των απλών συσκευών που δεν εστιάζουν αποκλειστικά στο throughput, αλλά στην ενεργειακή απόδοση και στην μικρή επιφάνεια υλοποίησης.

Επιπλέον ενσωματώθηκε στην υλοποίηση του AEAD128 (δανείστηκε κοινά modules) και μια άλλη διαδικασία του αλγορίθμου ASCON, που είναι η συνάρτηση κατακερματισμού Hash256. Η λειτουργία της συγκεκριμένης function, είναι ότι παίρνει ένα σύνολο δεδομένων μεταβλητού μήκους (στην παρούσα υλοποίηση έχει περιοριστεί στα 128 bits, αλλά μπορεί να αναβαθμιστεί και να πάρει αόριστο μήκος), και παράγει μια συμβολοσειρά των 256 bits. Το συγκεκριμένο string μπορεί να οριστεί ως ταυτότητα των συγκεκριμένων δεδομένων (ακόμη και ένα bit να αλλάξει από τα δεδομένα εισόδου, προκύπτει μια εντελώς διαφορετική έξοδος). Αυτό επιτυγχάνεται καθώς με τα 256 bits της συνάρτησης προκύπτουν 2^{256} πιθανές έξοδοι. Με αποτέλεσμα να δημιουργείται ένας τεράστιος αριθμός συνδυασμών, καθιστώντας δύσκολο (αν όχι αδύνατο), να προκύψει το ίδιο Hash από διαφορετικά δεδομένα.

1.5 Δομή της διπλωματικής εργασίας

Η δομή που ακολουθήθηκε εστιάζει σε διακριτά κεφάλαια όπου αναλύονται, ερευνώνται και υλοποιούνται, οι επιμέρους ενότητες της μελέτης. Πιο συγκεκριμένα η εργασία υλοποιείται σε 6 κεφάλαια και ακολουθούν την παρακάτω δομή:

- Στο 1^ο [Κεφάλαιο](#) γίνεται μια εισαγωγή στο πρόβλημα, και αναλύονται οι λόγοι που υπάρχει η επιτακτική ανάγκη της ύπαρξης κρυπτογραφίας. Στην συνέχεια παρουσιάζονται οι στόχοι και το αντικείμενο της εργασίας, η μεθοδολογική προσέγγιση που



ακολουθήθηκε, και γίνεται μια πρώτη παρουσίαση των χαρακτηριστικών του συστήματος υλοποίησης.

- Το **2^ο Κεφάλαιο** επικεντρώνεται στο θεωρητικό υπόβαθρο της εργασίας. Παρουσιάζεται η ελαφροβαρής κρυπτογραφία και αναλύεται ο αλγόριθμος αυθεντικοποιημένης κρυπτογράφησης με συσχετιζόμενα δεδομένα ASCON AEAD128 και μια συνάρτηση κατακερματισμού αυτού, η Hash256. Στο τέλος παρατίθεται μια συνοπτική εισαγωγή στα FPGA, στα πλεονεκτήματά τους, καθώς και στα χαρακτηριστικά του Artix 7 και συγκεκριμένα του XC7A75T που χρησιμοποιήθηκε.
- Στο **3^ο Κεφάλαιο** αναλύεται η αρχιτεκτονική του συστήματος και επεξηγείται η λειτουργία των γραμμένων σε VHDL υποπρογραμμάτων (ή διακριτών δομών).
- Στο **4^ο Κεφάλαιο** παρουσιάζονται τα στάδια σύνθεσης (synthesis) και υλοποίησης (implementation) στο περιβάλλον Vivado, και αναλύονται τα αποτελέσματα που προκύπτουν από τα εργαλεία σχεδίασης, όπως χρονισμός, χρήση πόρων και άλλα χαρακτηριστικά του συστήματος.
- Στο **5^ο Κεφάλαιο** υπάρχει η αξιολόγηση του συστήματος. Αρχικά εξετάζεται η ορθότητα της λειτουργίας του, παρουσιάζονται οι μετρήσεις που συλλέχθηκαν (απόδοση, κατανάλωση, επιφάνεια), και στη συνέχεια γίνεται σύγκριση με αντίστοιχες δημοσιευμένες ερευνητικές εργασίες.
- Τέλος, στο **6^ο Κεφάλαιο** παραθέτονται τα συμπεράσματα που εξάχθηκαν, καθώς και προτάσεις για μελλοντική έρευνα και πιθανές επεκτάσεις του συστήματος.

1.6 Συσχέτιση με τους στόχους βιώσιμης ανάπτυξης του ΟΗΕ

Πρωταρχικά η συγκεκριμένη εργασία συνδέεται με τον στόχο βιώσιμης ανάπτυξης 9 (Βιομηχανία, Καινοτομία και Υποδομές). Η σύνδεση με τον συγκεκριμένο στόχο προκύπτει καθώς εστιάζει στην μελέτη και υλοποίηση ενός σύγχρονου και καινοτόμου συστήματος αυθεντικοποιημένης κρυπτογράφησης με συσχετιζόμενα δεδομένα. Το οποίο μπορεί να συμβάλει στην ανάπτυξη ασφαλών ψηφιακών επικοινωνιών στο διαδίκτυο των πραγμάτων (IoT), άλλα και στα δίκτυα νέας γενιάς.

Δευτερευόντως, συνδέεται με τον στόχο βιώσιμης ανάπτυξης 16 (Ειρήνη, Δικαιοσύνη και Ισχυροί Θεσμοί). Η επίτευξή του επιτυγχάνεται καθώς η προστασία των δεδομένων και η ασφάλεια των πληροφοριακών συστημάτων, αποτελούν απόλυτες και ισχυρές προϋποθέσεις για την δικαιοσύνη και τους ισχυρούς θεσμούς σε μια δημοκρατική κοινωνία.

Συνεπώς η παρούσα εργασία συμβάλει στην προώθηση καινοτόμων τεχνολογικών λύσεων, με απόλυτη έμφαση στην ασφάλεια. Υποστηρίζοντας άμεσα τους στόχους βιώσιμης ανάπτυξης, που σχετίζονται με την καινοτομία και την προστασία.



2. Ερευνητική ανασκόπηση

2.1 Κρυπτογραφία

Κρυπτογραφία είναι ο επιστημονικός κλάδος που ασχολείται με την προστασία της πληροφορίας, έτσι ώστε μη εξουσιοδοτημένοι χρήστες να μην έχουν πρόσβαση σε αυτήν. Πιο συγκεκριμένα οι βασικοί της στόχοι είναι [11] [3] [1]:

- **Εμπιστευτικότητα.** Μόνο εξουσιοδοτημένοι χρήστες μπορούν να έχουν πρόσβαση στα δεδομένα. Αυτό επιτυγχάνεται μετατρέποντας τα σε μια μορφή που ακόμη και να υποκλαπούν, δεν μπορεί να αποκτηθεί πρόσβαση σε αυτά.
- **Ακεραιότητα.** Προστατεύεται η αλλοίωση των δεδομένων. Μέσω συναρτήσεων κατακερματισμού, ο παραλήπτης είναι σίγουρος ότι το μήνυμα που έχει λάβει είναι αυτό το οποίο έχει στείλει ο συγκεκριμένος αποστολέας.
- **Αυθεντικοποίηση.** Διασφαλίζεται η ταυτότητα των δυο μερών (αποστολέας και παραλήπτης) μέσω κλειδιών, πιστοποιητικών, ψηφιακών υπογραφών, βιομετρικών δεδομένων κ.α.
- **Μη αποποίηση ευθύνης.** Από την στιγμή που τα μέρη είναι ταυτοποιημένα, τότε η κάθε ενέργεια τους καταγράφεται και έτσι δεν μπορεί ο αποστολέας να αρνηθεί την ιδιοκτησία των απεσταλμένων δεδομένων του.

Τα βασικά είδη στα οποία χωρίζεται η κρυπτογραφία είναι [11] [3] [1]:

- **Συμμετρική κρυπτογραφία.** Χρησιμοποιείται το ίδιο κλειδί από τον αποστολέα, παραλήπτη για την κρυπτογράφηση/αποκρυπτογράφηση. Είναι γρήγορη και αποδοτική, κάτι την οποία την κάνει κατάλληλη για συστήματα περιορισμένων πόρων.
- **Ασύμμετρη κρυπτογραφία.** Χρησιμοποιούνται δυο κλειδιά συνδεδεμένα απόλυτα μεταξύ τους, ένα δημόσιο γνωστό σε όλους το οποίο κρυπτογραφεί το μήνυμα, και ένα ιδιωτικό γνωστό μόνο στον παραλήπτη που αποκρυπτογραφεί το μήνυμα. Δεν είναι τόσο γρήγορη και αποδοτική (χρειάζεται περισσότερους πόρους), αλλά παρέχει μεγαλύτερη ασφάλεια σε σχέση με την συμμετρική.
- **Συναρτήσεις κατακερματισμού.** Δεν χρησιμοποιείται καθόλου κλειδί, αλλά μέσω μιας συμβολοσειράς κάποιων bits, δημιουργείται μια μοναδική ταυτότητα για τα συγκεκριμένα δεδομένα. Το χαρακτηριστικό της είναι η μονόδρομη λειτουργία της (δηλαδή δεν παρέχεται η δυνατότητα αποκρυπτογράφησης της).

Στην παρούσα εργασία εστιάζουμε στην συμμετρική κρυπτογραφία και ειδικότερα στην ελαφροβαρή (που επί το πλείστον υλοποιείται μέσω συμμετρικής κρυπτογραφίας), αλλά και σε μια συνάρτηση κατακερματισμού που επιτυγχάνεται μέσω αυτής.

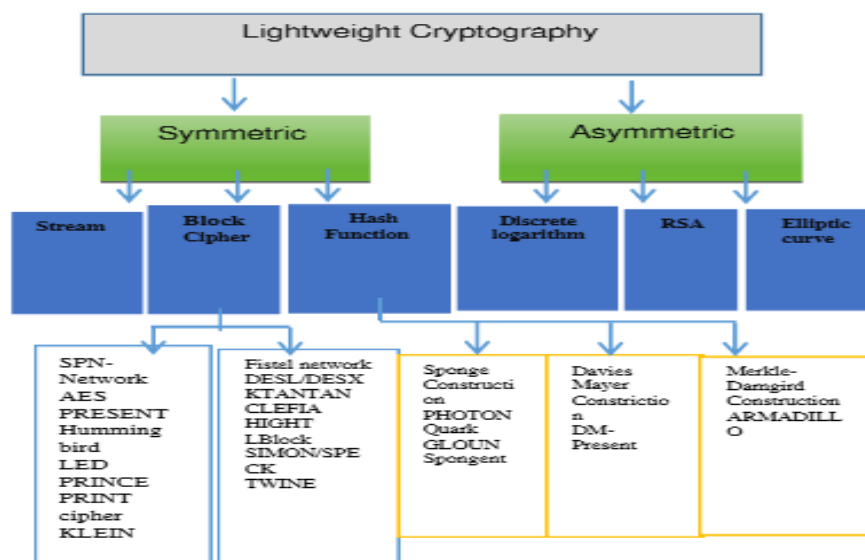
2.1.1 Ελαφροβαρής Κρυπτογραφία ή *Lightweight Cryptography*

Η ελαφροβαρής κρυπτογραφία ή *Lightweight Cryptography* είναι μια ειδική κατηγορία η οποία επικεντρώνεται σε συστήματα με περιορισμένους πόρους ενέργειας, αποθήκευσης, επιφάνειας υλοποίησης και υπολογιστικής ισχύος. Κατάλληλη για IoT και για ασύρματες επικοινωνίες συσκευών μειωμένων πόρων. Η αποδοτικότητα των αλγορίθμων LWC επιτυγχάνεται μέσω της απαίτησης μικρής μνήμης RAM και ROM, μειωμένης υπολογιστικής πολυπλοκότητας και χαμηλής κατανάλωσης ισχύος [3].

Σαν αποτέλεσμα των παραπάνω συχνά υπάρχουν συμβιβασμοί μεταξύ απόδοσης και ασφάλειας, και κάποιοι αλγόριθμοι LWC μπορεί να παρουσιάζουν μεγαλύτερη ευπάθεια από τους συμβατικούς (αυτό ενισχύεται από την πληθώρα των διαφορετικών αλγορίθμων και την έλλειψη ενιαίων προτύπων) [11]. Σημαντικό είναι ότι η LWC δεν θεωρείται εγγενώς λιγότερο ασφαλής, απλά απαιτείται κατάλληλη εξισορρόπηση μεταξύ απόδοσης και ασφάλειας, κάνοντας την κατάλληλη επιλογή η οποία θα είναι συμβατή με τις ανάγκες της εκάστοτε εφαρμογής [2] [3].

Οι βασικές κατηγορίες LWC είναι [12] (Εικόνα 2):

- **Block Ciphers.** Κρυπτογραφούν τα δεδομένα σε fixed length words (πχ. 128 bits) και χρησιμοποιούν επαναλαμβανόμενους γύρους κάνοντας απλές πράξεις (xor, not, shift).
- **Stream Ciphers.** Κρυπτογραφούν τα δεδομένα bit-bit ή byte-byte χρησιμοποιώντας ένα ψευδοτυχαίο keystream το οποίο αλληλοεπιδρά με το plain text (μήνυμα).
- **Hash Functions.** Παράγουν μια μοναδική σύνοψη (Hash digest) σταθερού μήκους, η οποία χρησιμοποιείται για την επαλήθευση της ακεραιότητας των δεδομένων.
- **AEAD.** Είναι ένα νέο είδος αλγορίθμων που συνδυάζουν αυθεντικοποίηση με κρυπτογράφηση. Ο ASCON ανήκει σε αυτήν την κατηγορία και θα αναλυθεί στην επόμενη ενότητα.



Εικόνα 2: *Lightweight Cryptography* Diagram [12]



2.2 ASCON

Ο αλγόριθμος ASCON παρουσιάστηκε για πρώτη φορά στο πλαίσιο του διαγωνισμού CAESAR το 2014 [13] με δημιουργούς του τους Christoph Dobraunig, Florian Mendel, Maria Eichlseder, και Martin Schl  ffer. Το 2023 στο NIST Lightweight Cryptography competition [14], επιλέχθηκε ως νέο πρότυπο ελαφροβαρούς κρυπτογραφίας για εφαρμογές αυθεντικοποίησης και Hashing.

Το πρώτο αρχικό επίσημο προσχέδιο δημοσιεύτηκε τον Νοέμβριο του 2024 και πήρε την τελική επίσημη μορφή του τον Αύγουστο του 2025, με τελευταία έκδοση του προτύπου την NIST SP 800-232 [7] που ο αλγόριθμος προτυποποιήθηκε ως AEAD128. Αξίζει να σημειωθεί ότι οι παλαιότερες εκδόσεις (πχ. v1.2) του ASCON χρησιμοποιούσαν διαφορετική ονοματολογία (Πίνακας 2), όπως Ascon-128 και Ascon-128a [15].

Πίνακας 2: Χαρακτηριστικά Ascon version 1.2

Parameters for recommended authenticated encryption schemes.							
Name	Algorithms	Bit size of				Rounds	
		key	nonce	tag	data block	p^a	p^b
ASCON-128	$\mathcal{E}, \mathcal{D}_{128,64,12,6}$	128	128	128	64	12	6
ASCON-128a	$\mathcal{E}, \mathcal{D}_{128,128,12,8}$	128	128	128	128	12	8

Παρόλο που η έκδοση Ascon-128a με την AEAD128 έχουν ίδια χαρακτηριστικά ως προς το μέγεθος κλειδιού (key), nonce, tag και αριθμό rounds, δεν παράγουν το ίδιο αποτέλεσμα εξόδου. Αυτό συμβαίνει γιατί αλλάζουν τα IV (initialization vectors) όπως και το bit ordering (από big endianness αλλάζει σε little endianness). Πιο συγκεκριμένα τα IV της τελευταίας version υπολογίζονται σύμφωνα με τις παραμέτρους που παρουσιάζονται στον Πίνακα 3 [7], και δίνουν τα αποτελέσματα που φαίνονται στον Πίνακα 4.

Πίνακας 3: ASCON Initialization Vectors Parameters

Ascon variants	v (8 bits)	a (4 bits)	b (4 bits)	t (16 bits)	$r/8$ (8 bits)
Ascon-AEAD128	1	12	8	128	16
Ascon-Hash256	2	12	12	256	8
Ascon-XOF128	3	12	12	0	8
Ascon-CXOF128	4	12	12	0	8



Πίνακας 4: ASCON AEAD128 Initialization Vectors Values

Ascon variants	Initial value
Ascon-AEAD128	0x00001000808c0001
Ascon-Hash256	0x0000080100cc0002
Ascon-XOF128	0x0000080000cc0003
Ascon-CXOF128	0x0000080000cc0004

Τα IV (Initialization Vectors) έχουν μήκος 64 bits και σχηματίζονται ως

$$IV = u \parallel 0^8 \parallel a \parallel b \parallel t \parallel r/8 \parallel 0^{16}.$$

Κατά συνέπεια ο υπολογισμός των IV από τα AEAD128, Hash256 γίνεται σύμφωνα με τις παραμέτρους του Πίνακα 3, στο πλαίσιο του οποίου λαμβάνονται οι τιμές του Πίνακα 5. Αποτέλεσμα αυτού είναι η δημιουργία των IV του πίνακα 6, τα οποία συμφωνούν με αυτά του αλγορίθμου που φαίνονται στον Πίνακα 4.

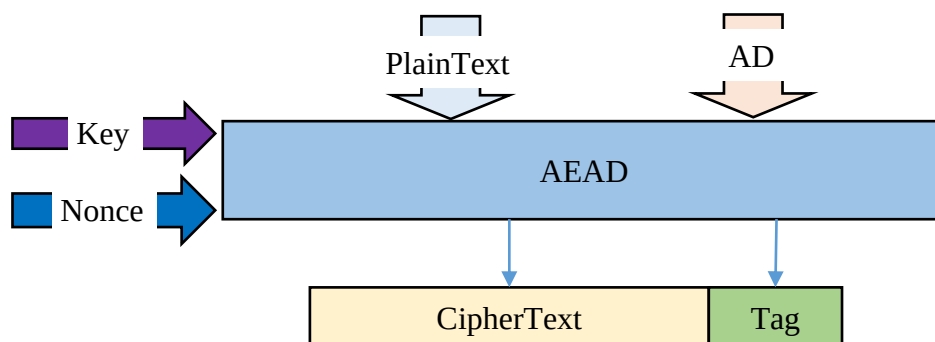
Πίνακας 5: Παραμετροποίηση τιμών διανυσμάτων (IV)

	<i>u</i> (8 bits)	<i>a</i> (4 bits)	<i>b</i> (4 bits)	<i>t</i> (16 bits)	<i>r/8</i> (8 bits)
AEAD128	1 = 0x01	12 = 0xC	8 = 0x8	128 = 0x0080	16 = 0x10
Hash256	2 = 0x02	12 = 0xC	12 = 0xC	256 = 0x0100	8 = 0x08

Πίνακας 6: Υπολογισμός τιμών διανυσμάτων (IV)

	<i>u</i>	0^8	<i>a</i>	<i>b</i>	<i>t</i>	<i>r/8</i>	0^{16}
AEAD128	0x01	0x00	0xC	0x8	0x0080	0x10	0x0000
Endianness	0x0000	0x10	0x0080	0x8	0xC	0x00	0x01
	AEAD128 IV = 0x00001000808C0001						
Hash256	0x02	0x00	0xC	0xC	0x0100	0x08	0x0000
Endianness	0x0000	0x08	0x0100	0xC	0xC	0x00	0x02
	Hash256 IV = 0x0000080100CC0002						

Επιπλέον ο ASCON είναι ένας AEAD αλγόριθμος, οπότε παρέχει ταυτόχρονα authentication και encryption όπως φαίνεται στην Εικόνα 3.



Εικόνα 3: AEAD Encryption Diagram



Αυτό που επιτυγχάνει ο αλγόριθμος αυθεντικοποιημένης κρυπτογράφησης με συσχετιζόμενα δεδομένα (AEAD) είναι ότι διασφαλίζει την εμπιστευτικότητα αλλά και την ακεραιότητα δεδομένων.

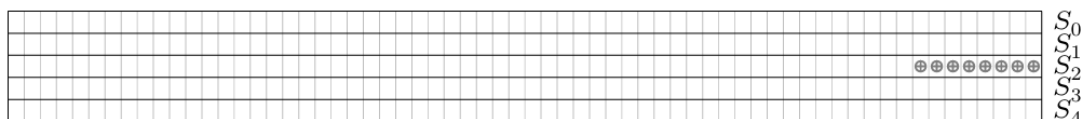
Αυτό πραγματοποιείται λαμβάνοντας τα key, nonce, plaintext, associate data και βγάζοντας σαν έξοδο το κωδικοποιημένο κείμενο (cipher text) και το tag (όπου επικολλιέται στο τέλος του μηνύματος). Το tag είναι το αποτέλεσμα μιας Hash function σταθερού μήκους των 128 bits η οποία επιβεβαιώνει την ακεραιότητα του μηνύματος. Επίσης τα AD (associate data) δεν κωδικοποιούνται παρά μόνο αυθεντικοποιούνται μέσω του tag. Επιπροσθέτως για να υπάρξει ασφαλής λειτουργία, το nonce πρέπει να αλλάζει σε κάθε νέο block μηνυμάτων (plain texts). Σε αντίθεση λοιπόν με τις συμβατικές προσεγγίσεις οι AEAD πραγματοποιούν τις δυο αυτές διαδικασίες, χρησιμοποιώντας όμως τον ίδιο πυρήνα και παρόμοιες εντολές. Συνεπώς μειώνεται κατά πολύ η πολυπλοκότητα και η επιφάνεια υλοποίησης και βελτιώνεται η απόδοση του (παρουσιάζεται πιο αναλυτικά στα παρακάτω κεφάλαια).

2.2.1 Η δομή του ASCON

Ο αλγόριθμος αποτελείται από μια κατάσταση (state) των 320 bits όπου ενημερώνεται σε κάθε βήμα του. Το state των 320 bits οργανώνεται σε 5 λέξεις των 64 bits. Η καρδιά του ASCON είναι η μεταθετική συνάρτηση (permutation) η οποία απορροφά τα δεδομένα (μέσω SPN [16]), και πραγματοποιεί τρεις διαδικασίες [7] σε κάθε κύκλο της.

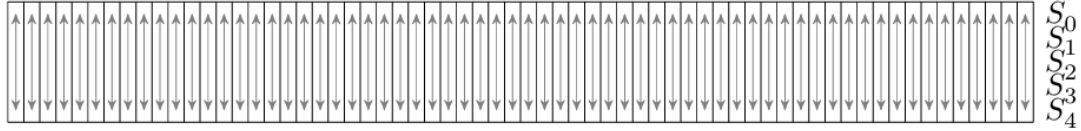
- **Round Constant ή RC (πρόσθεση σταθεράς).** Είναι το πρώτο μέρος της permutation, σε κάθε κύκλο ρολογιού γίνεται xor στα 8 LSB του $s(2)$, με μια διαφορετική σταθερά. Η σταθερά επιλέγεται σύμφωνα με τον τύπο $c_i = \text{const}_{16-\text{rnd}+i}$ όπου $\text{rnd} \leq 16$ και $0 \leq i \leq \text{rnd}-1$, άρα για τους 12 γύρους ξεκινάει από την $16-12=4^{\text{η}}$ και φτάνει μέχρι και την $15^{\text{η}}$ σταθερά, ενώ για τους 8 γύρους ξεκινάει από την $16-8=8^{\text{η}}$ και φτάνει μέχρι και την $15^{\text{η}}$ (Εικόνα 4).

i	const_i	i	const_i
0	0x000000000000003c	8	0x00000000000000b4
1	0x000000000000002d	9	0x00000000000000a5
2	0x000000000000001e	10	0x0000000000000096
3	0x000000000000000f	11	0x0000000000000087
4	0x00000000000000f0	12	0x0000000000000078
5	0x00000000000000e1	13	0x0000000000000069
6	0x00000000000000d2	14	0x000000000000005a
7	0x00000000000000c3	15	0x000000000000004b



Εικόνα 4: ASCON Round Constant (RC)

- **Substitution Layer ή SBox (μη γραμμικό επίπεδο αντικατάστασης).** Εδώ γίνεται η εισαγωγή μη γραμμικότητας στο σύστημα σε επίπεδο bit. Όπως φαίνεται στην Εικόνα 5 πραγματοποιούνται 64 παράλληλες πράξεις των 5 bit η κάθε μια, σύμφωνα με την τοπολογία που φαίνεται στην ίδια εικόνα.



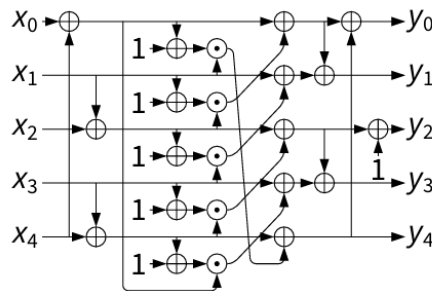
$$\begin{aligned} y_0 &= x_4x_1 \oplus x_3 \oplus x_2x_1 \oplus x_2 \oplus x_1x_0 \oplus x_1 \oplus x_0, \\ y_1 &= x_4 \oplus x_3x_2 \oplus x_3x_1 \oplus x_3 \oplus x_2x_1 \oplus x_2 \oplus x_1 \oplus x_0, \\ y_2 &= x_4x_3 \oplus x_4 \oplus x_2 \oplus x_1 \oplus 1, \\ y_3 &= x_4x_0 \oplus x_4 \oplus x_3x_0 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0, \\ y_4 &= x_4x_1 \oplus x_4 \oplus x_3 \oplus x_1x_0 \oplus x_1. \end{aligned}$$

Εικόνα 5: ASCON SBox Logic

Για την απλοποίηση του σχεδιασμού μπορεί να χρησιμοποιηθεί ένας Look Up Table (LUT), όπως αυτός που φαίνεται στο πάνω μέρος της Εικόνα 6. Ωστόσο, η χρήση LUT κάνει τον σχεδιασμό πιο επιρρεπή στα side channel attacks και για αυτό αποφεύγεται. Επιπροσθέτως στην ίδια εικόνα (Εικόνα 6) υπάρχει και το λογικό κύκλωμα αναπαράστασης του SBox και πως γίνονται οι πράξεις σε αυτό.

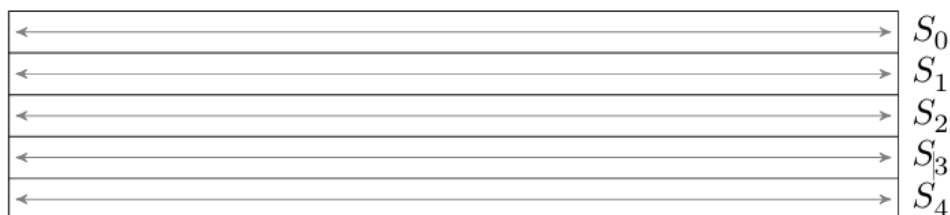
x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$SBox(x)$	4	b	1f	14	1a	15	9	2	1b	5	8	12	1d	3	6	1c
x	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
$SBox(x)$	1e	13	7	e	0	d	11	18	10	c	1	19	16	a	f	17

Note that 5-bit inputs are represented in hexadecimal (e.g., $x=1$ corresponds to $(0,0,0,0,1)$).



Εικόνα 6: ASCON SBox LUT & Circuit

- **Diffusion Layer (γραμμικό επίπεδο διάχυσης).** Είναι ο γραμμικός μετασχηματισμός ο οποίος πραγματοποιεί διάχυση σε κάθε λέξη των 64 bits στο state των 320 bits.



Εικόνα 7: ASCON Diffusion Layer Diagram



Η διάχυση γίνεται με πράξεις xor και δεξιές κυκλικές μεταθέσεις ανά λέξη των 64 bits, σύμφωνα με την Εικόνα 8.

$$\begin{aligned}\Sigma_0(S_0) &= S_0 \oplus (S_0 \ggg 19) \oplus (S_0 \ggg 28) \\ \Sigma_1(S_1) &= S_1 \oplus (S_1 \ggg 61) \oplus (S_1 \ggg 39) \\ \Sigma_2(S_2) &= S_2 \oplus (S_2 \ggg 1) \oplus (S_2 \ggg 6) \\ \Sigma_3(S_3) &= S_3 \oplus (S_3 \ggg 10) \oplus (S_3 \ggg 17) \\ \Sigma_4(S_4) &= S_4 \oplus (S_4 \ggg 7) \oplus (S_4 \ggg 41)\end{aligned}$$

Εικόνα 8: ASCON Diffusion Layer Logic

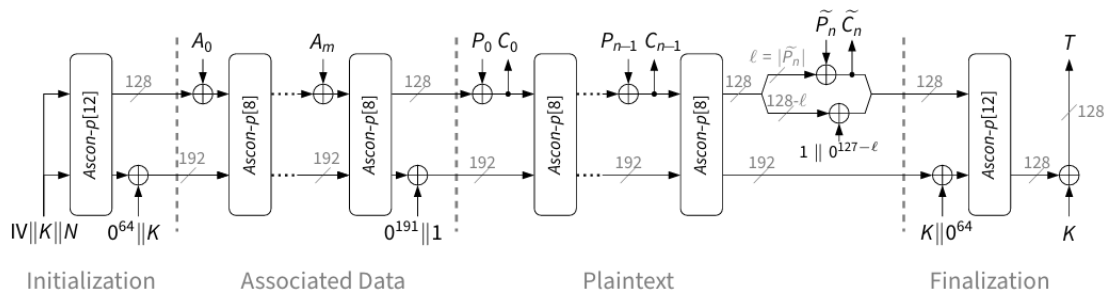
Parsing & Padding. Κατά την είσοδο των δεδομένων στο σύστημα (message και AD), για να δουλέψουν σωστά όλες οι παράμετροι χωρίζονται σε blocks από bits (σε αυτήν την περίπτωση των 128). Για αυτό τον σκοπό το Parse διαχωρίζει τα δεδομένα σε ακέραια πολλαπλάσια των 128 bits (π.χ. για δεδομένα μεγέθους 1280 bits, τότε το **parsing** θα δημιουργήσει 10 blocks των 128 bits).

Όταν τα δεδομένα δεν αποτελούν ακέραιο πολλαπλάσιο των 128 bits ή όταν το μήνυμα έχει μήκος μικρότερο των 128 bits, εφαρμόζεται η διαδικασία **padding**. Σε αυτήν την περίπτωση προστίθεται ένα “1” στο τέλος των δεδομένων και στην συνέχεια τοποθετούνται όσα “0” χρειάζονται έτσι ώστε να δημιουργηθεί ένα block των 128 bit.

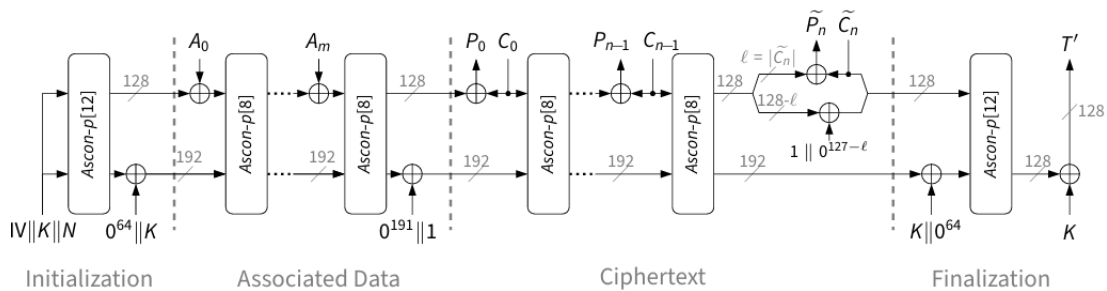
2.2.2 Ο αλγόριθμος AEAD128

Όπως αναφέρθηκε και στην αρχή του Κεφαλαίου 2.2 ο αλγόριθμος AEAD128 έχει ακριβώς τα ίδια χαρακτηριστικά (δυαδικό μήκος των key, nonce, message, tag και ίδια rounds p^a , p^b) με τον Ascon-128a. Στις Εικόνες 9,10 φαίνονται τα block διαγράμματα της κωδικοποίησης/ αποκωδικοποίησης. Το αξιοσημείωτο σε αυτά είναι ότι είναι ακριβώς ίδια, και στο μόνο που διαφέρουν είναι η κατεύθυνση προς το σύστημα των Plaintext και Ciphertext [7].

Η παραπάνω προσέγγιση αποτελεί ένα ακόμη χαρακτηριστικό που καθιστά τον ASCON ιδιαίτερα αποδοτικό, καθώς συμβάλλει στη μείωση τόσο της επιφάνειας υλοποίησης όσο και της ενεργειακής κατανάλωσης του συστήματος. Αυτό επιτυγχάνεται επειδή χρησιμοποιείται ο ίδιος πυρήνας λειτουργίας για όλες τις βασικές διαδικασίες του αλγορίθμου, αποφεύγοντας την ανάγκη ύπαρξης ξεχωριστών κυκλωμάτων και πολύπλοκων μεταγωγών μεταξύ διαφορετικών μονάδων επεξεργασίας.



Εικόνα 9: ASCON AEAD128 Encryption Architecture



Εικόνα 10: ASCON AEAD128 Decryption Architecture

Η λειτουργία του αλγορίθμου χωρίζεται σε τέσσερις διακριτές φάσεις:

- **Initialization (αρχικοποίηση).** Το IV (των 64 bit), το key (των 128 bit) και το nonce (των 128bit) τοποθετούνται το ένα δίπλα στο άλλο με την προαναφερόμενη σειρά και προκύπτει το state των 320 bits (64+128+128). Στην συνέχεια το state εισέρχεται 12 φορές στην permutation function και τα 128 MSB της state περνούν στο επόμενο στάδιο. Παράλληλα εκτελείται xor του Key με τα 128 LSB (της state), με το αποτέλεσμα να περνάει επίσης στο επόμενο στάδιο (προκύπτει το ίδιο αποτέλεσμα με το εάν εκτελούνταν xor με τα 128+64=192 LSB. Καθώς όποια λογική κατάσταση γίνει xor με το 0, θα παραμείνει ίδια).
- **Associated Data (συσχετιζόμενα δεδομένα).** Αρχικά πραγματοποιείται xor στα 128 MSB του state με τα AD (των 128 bits) και εισέρχεται το state στην μονάδα μετάθεσης 8 φορές. Ακολούθως όσα άλλα AD παραμένουν (αν υπάρχουν) εισέρχονται σειριακά. Τέλος εκτελείται xor του MSB από το state, με το “1” (αναφέρθηκε παραπάνω γιατί γίνεται μόνο στο ένα bit).
- **Ciphertext (κρυπτογράφημα).** Πραγματοποιείται xor των 128 MSB του state με το μήνυμα και η έξοδος αυτής της πράξης δίνει το κρυπτογράφημα. Στην συνέχεια περνάει το state από την μονάδα μετάθεσης 8 φορές (εδώ αν υπάρχουν άλλα block μηνυμάτων, περνάνε σειριακά ακολουθώντας ξανά την διαδικασία του συγκεκριμένου σταδίου από την αρχή). Στο τέλος της διαδικασίας ελέγχεται το μήκος του τελευταίου μηνύματος και γίνεται xor μόνο με τα MSB του state που αντιστοιχούν στο μήκος του μηνύματος, και η έξοδος του κρυπτογραφήματος έχει το ίδιο length με το μήνυμα. Για τη μετάβαση στο επόμενο στάδιο γίνεται σε αυτό parsing και padding.



- **Finalization (οριστικοποίηση).** Αυτό είναι το τελευταίο στάδιο όπου γίνεται xor στα bits (191 downto 64) του state με το key. Εισέρχεται στην μονάδα μετάθεσης 12 φορές και στο αποτέλεσμα γίνεται xor στα 128 LSB του state με το key. Η προαναφερόμενη έξοδος είναι το tag, και με αυτήν την συμβολοσειρά των 128 bit ελέγχεται η ακεραιότητα του μηνύματος κατά την αποκωδικοποίηση, ή εκτελείται αυθεντικοποίηση στην κωδικοποίηση.

Στην Εικόνα 11 παρατίθεται ο ψευδοκώδικας του AEAD128.

Algorithm 3 $\text{Ascon-AEAD128}.\text{enc}(K, N, A, P)$

Input: 128-bit key K , 128-bit nonce N , associated data A , plaintext P

Output: ciphertext C , 128-bit tag T

$IV \leftarrow 0x00001000808c0001$

▷ Initialization

$S \leftarrow IV \parallel K \parallel N$

$S \leftarrow \text{Ascon-p}[12](S)$

$S \leftarrow S \oplus (0^{192} \parallel K)$

if $|A| > 0$ **then**

▷ Processing associated data

$A_0, \dots, A_{m-1}, \widetilde{A}_m \leftarrow \text{parse}(A, 128)$

$A_m \leftarrow \text{pad}(\widetilde{A}_m, 128)$

for $i = 0$ **to** m **do**

$S \leftarrow \text{Ascon-p}[8]((S_{[0:127]} \oplus A_i) \parallel S_{[128:319]})$

end for

end if

$S \leftarrow S \oplus (0^{319} \parallel 1)$

$P_0, \dots, P_{n-1}, \widetilde{P}_n \leftarrow \text{parse}(P, 128)$

▷ Processing plaintext

$\ell \leftarrow |\widetilde{P}_n|$

for $i = 0$ **to** $n - 1$ **do**

$S_{[0:127]} \leftarrow S_{[0:127]} \oplus P_i$

$C_i \leftarrow S_{[0:127]}$

$S \leftarrow \text{Ascon-p}[8](S)$

end for

$S_{[0:127]} \leftarrow S_{[0:127]} \oplus \text{pad}(\widetilde{P}_n, 128)$

$\widetilde{C}_n \leftarrow S_{[0:\ell-1]}$

$C \leftarrow C_0 \parallel \dots \parallel C_{n-1} \parallel \widetilde{C}_n$

$S \leftarrow \text{Ascon-p}[12](S \oplus (0^{128} \parallel K \parallel 0^{64}))$

▷ Finalization

$T \leftarrow S_{[192:319]} \oplus K$

return C, T

Εικόνα 11: ASCON AEAD128 Pseudocode

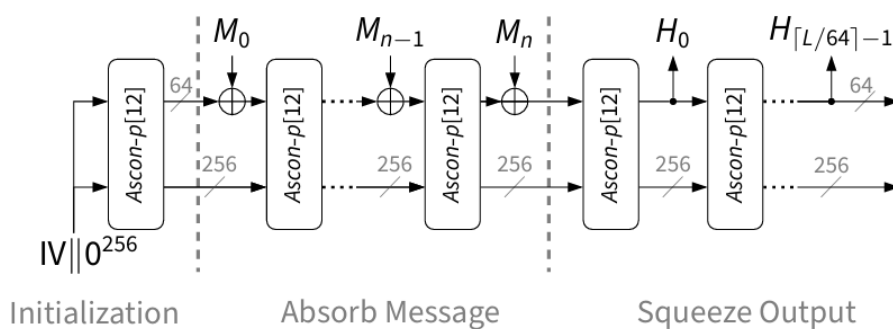
2.2.3 Ο αλγόριθμος Hash256

Στην παρούσα εργασία εξετάζεται επίσης η συνάρτηση κατακερματισμού Hash256 του ASCON [7], της οποίας το block διάγραμμα παρουσιάζεται στην Εικόνα 12.

Η λειτουργία της διεκπεραιώνεται από τις ακόλουθες φάσεις:

- **Initialization (αρχικοποίηση).** Συνδυάζεται το IV (των 64bit) με 256 “0” για να δημιουργηθεί το state των 320 bits. Στην συνέχεια περνάει από την μονάδα μετάθεσης 12 φορές.
- **Absorb Message (απορρόφηση μηνύματος).** Εδώ το block του μηνύματος (rate) είναι 64 bits. Αρχικά πραγματοποιείται xor στα 64 MSB του state με αυτό και περνάει από την μονάδα μετάθεσης 12 φορές. Η διαδικασία συνεχίζει επαναληπτικά για όσο υπάρχουν blocks δεδομένων (εδώ υλοποιείται και πάλι padding/parsing αλλά αυτήν την φορά με data length των 64 bits).
- **Squeeze Output (οριστικοποίηση εξόδου).** Το state εισέρχεται 12 φορές στην μονάδα μετάθεσης, στην συνέχεια από το state που δημιουργήθηκε παίρνονται τα 64 MSB και ορίζονται σαν τα 64 MSB της κατάστασης των 256 bits της Hash256. Αυτό θα επαναληφθεί άλλες τρεις φορές για να συμπληρωθούν και τα υπόλοιπα 256-64=192 bits του Hash output. Με το που ολοκληρωθεί η διαδικασία δημιουργείται το Hash των 256 bits.

$$\text{Hash256} = H0 \parallel H1 \parallel H2 \parallel H3$$



Εικόνα 12: ASCON Hash256 Architecture

Στην Εικόνα 13 παρατίθεται ο ψευδοκώδικας της Hash256. Αν υπάρξει ανάγκη βελτιστοποίηση της απόδοσης μπορεί να αντικατασταθεί το Initialization (γιατί πάντα υπολογίζεται το ίδιο αποτέλεσμα) με μια σταθερά. Ωστόσο, μια τέτοια τροποποίηση θα οδηγούσε σε απόκλιση από το επίσημο πρότυπο του αλγορίθμου.



Algorithm 5 Ascon-Hash256(M)

Input: Bitstring $M \in \{0,1\}^*$

Output: Digest $H \in \{0,1\}^{256}$

```

 $IV \leftarrow 0x00000080100cc0002$  ▷ Initialization
 $S \leftarrow \text{Ascon-p}[12](IV \parallel 0^{256})$ 

 $M_0, \dots, M_{n-1}, \widetilde{M}_n \leftarrow \text{parse}(M, 64)$  ▷ Absorbing
 $M_n \leftarrow \text{pad}(\widetilde{M}_n, 64)$ 
for  $i = 0$  to  $n - 1$  do
     $S_{[0:63]} \leftarrow S_{[0:63]} \oplus M_i$ 
     $S \leftarrow \text{Ascon-p}[12](S)$ 
end for
 $S_{[0:63]} \leftarrow S_{[0:63]} \oplus M_n$ 

 $S \leftarrow \text{Ascon-p}[12](S)$  ▷ Squeezing
for  $i = 0$  to  $2$  do
     $H_i \leftarrow S_{[0:63]}$ 
     $S \leftarrow \text{Ascon-p}[12](S)$ 
end for
 $H_3 \leftarrow S_{[0:63]}$ 

 $H \leftarrow H_0 \parallel H_1 \parallel H_2 \parallel H_3$ 
return  $H$ 

```

Εικόνα 13: ASCON Hash256 Pseudocode

2.3 FPGAs

Το Field Programming Gate Array είναι ένα ολοκληρωμένο κύκλωμα το οποίο παρέχει την δυνατότητα προγραμματιζόμενης διαμόρφωσης της εσωτερικής δομής του [17].

Ο σχεδιασμός πραγματοποιείται μέσω γλωσσών περιγραφής υλικού (Hardware Description Languages – HDL), όπως η VHDL, επιτρέποντας την υλοποίηση εξειδικευμένου hardware ανάλογα με τις απαιτήσεις της εφαρμογής. Σε αντίθεση με τους μικροελεγκτές (MCU), όπου οι εντολές εκτελούνται κυρίως σειριακά, τα FPGA επιτρέπουν την παράλληλη εκτέλεση πολλαπλών διεργασιών ανάλογα με την αρχιτεκτονική του σχεδιασμού.

Η βασική δομή του FPGA αποτελείται από:

- **CLBs - Configurable Logic Blocks (Διαμορφώσιμα Λογικά Blocks).** Περιέχουν LUTs (Look Up Tables), FFs (Flip Flops), MUXs (Multiplexers) κ.α.
- **Rooting & Interconnections (δρομολόγηση & διασυνδέσεις).** Προσφέρει το δίκτυο και τις διασυνδέσεις μεταξύ των CLBs.
- **I/O Blocks (μονάδες εισόδου εξόδου).** Εξασφαλίζει την διασύνδεση του με τις εξωτερικές περιφερειακές συσκευές.



Παρά τα σημαντικά πλεονεκτήματά τους, τα FPGA απαιτούν σχεδίαση σε αρκετά χαμηλό επίπεδο αφαίρεσης, γεγονός που αυξάνει την πολυπλοκότητα ανάπτυξης. Για να ξεπεραστεί το πρόβλημα έχουν δημιουργηθεί περιβάλλοντα HLS (High Level Synthesis) όπου μέσω γλωσσών υψηλού επιπέδου (C, C++) μπορούν να προστεθούν/προσαρμοστούν διάφορα modules σε υψηλό επίπεδο αφαίρεσης (υπάρχει βέβαια πιθανότητα μείωσης της απόδοσης και φυσικά περιορίζεται ο έλεγχος χαμηλού επιπέδου).

2.3.1 Ανάγκη χρήσης FPGAs

Η ραγδαία ανάπτυξη των ασύρματων επικοινωνιών και του IoT έχει δημιουργήσει την ανάγκη ύπαρξης ισχυρής ασφάλειας (μέσω κρυπτογραφίας) αλλά και μεγάλου throughput, με μειωμένη κατανάλωση ισχύος και μικρή επιφάνεια υλοποίησης για τη διασφάλιση φορητών και φορετών συσκευών. Τα FPGA είναι ιδιαίτερα κατάλληλα σε αυτές τις υλοποιήσεις [18] γιατί προσφέρουν ασφάλεια σε επίπεδο υλικολογισμικού, μεγάλη ταχύτητα με μικρό latency, επεκτασιμότητα, ευελιξία όπως και ενεργειακή απόδοση, χρησιμοποιώντας παράλληλα μικρότερη δεσμευμένη επιφάνεια υλικού (λόγω συγκεκριμένης αρχιτεκτονικής). Παρότι τα ASIC υπερτερούν σε τομείς όπως η ταχύτητα, η ενεργειακή απόδοση και η επιφάνεια υλοποίησης, παρουσιάζουν σημαντικά μειονεκτήματα. Αυτά είναι το υψηλό κόστος κατασκευής τους, ο μεγάλος χρόνος υλοποίησης και η περιορισμένη επεκτασιμότητα όπως και ευελιξία τους.

Στον Πίνακα 7 συνοψίζονται τα χαρακτηριστικά των FPGAs σε σύγκριση με άλλες λύσεις υλοποίησης κρυπτογραφικών συστημάτων.

Πίνακας 7: Σύγκριση FPGAs με άλλες λύσεις κρυπτογραφικών συστημάτων

Feature	FPGA-Based Solution	Microcontroller-Based Solution	Software-Based Encryption
Security Level	High (Hardware)	Moderate (Hardware/Software)	Moderate (Software)
Resource Utilization	Low	Moderate	High
Latency	Low (<50 ms)	Moderate (50–100 ms)	High (>100 ms)
Scalability	High	Moderate	Low

2.3.2 Artix 7

Στην παρούσα εργασία σαν ενσωματωμένο σύστημα χρησιμοποιείται ένα FPGA της σειράς Artix 7 της πρώην Xilinx [19], που από τον Αύγουστο του 2023 ανήκει αποκλειστικά στην AMD [20]. Η σειρά 7 FPGA περιλαμβάνει διαφορετικές οικογένειες συσκευών με διαφορετικά χαρακτηριστικά απόδοσης, κατανάλωσης και κόστους (datasheet [21]) που συνοψίζονται στον Πίνακα 8.



Πίνακας 8: Series 7 FPGAs Characteristics

Max. Capability	Spartan-7	Artix-7	Kintex-7	Virtex-7
Logic Cells	102K	215K	478K	1,955K
Block RAM ⁽¹⁾	4.2 Mb	13 Mb	34 Mb	68 Mb
DSP Slices	160	740	1,920	3,600
DSP Performance ⁽²⁾	176 GMAC/s	929 GMAC/s	2,845 GMAC/s	5,335 GMAC/s
MicroBlaze CPU ⁽³⁾	260 DMIPs	303 DMIPs	438 DMIPs	441 DMIPs
Transceivers	–	16	32	96
Transceiver Speed	–	6.6 Gb/s	12.5 Gb/s	28.05 Gb/s
Serial Bandwidth	–	211 Gb/s	800 Gb/s	2,784 Gb/s
PCIe Interface	–	x4 Gen2	x8 Gen2	x8 Gen3
Memory Interface	800 Mb/s	1,066 Mb/s	1,866 Mb/s	1,866 Mb/s
I/O Pins	400	500	500	1,200
I/O Voltage	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V
Package Options	Low-Cost, Wire-Bond	Low-Cost, Wire-Bond, Bare-Die Flip-Chip	Bare-Die Flip-Chip and High-Performance Flip-Chip	Highest Performance Flip-Chip

Η σειρά Artix 7 όμως, έχει μια καλή αναλογία απόδοσης προς κόστος, γιατί προσφέρει χαμηλή κατανάλωση, υψηλή απόδοση και ευελιξία. Αυτό την κάνει κατάλληλη για IoT συσκευές και ενσωματωμένα συστήματα.

Κατά την ανάπτυξη της παρούσας υλοποίησης έμφαση δόθηκε κυρίως στη λειτουργική ολοκλήρωση του συστήματος και λιγότερο στη βελτιστοποίηση της απόδοσης. Από το timing report στο Vivado, προέκυψε ότι η μέγιστη συχνότητα λειτουργίας που μπορούσε να επιτευχθεί θα υπερέβαινε ελαφρώς τα 200Mhz. Επίσης το design απαιτούσε 128 pins για είσοδο, 128 pins για έξοδο και 18 pins για τα σήματα ελέγχου, συνεπώς η ανάγκη για I/O pins είναι $128+128+18 = 274$.

Ελέγχοντας στην συνέχεια το datasheet DC and AC Switching Characteristics [22] διαπιστώθηκε ότι το μικρότερο speed grade που είναι το -1, μπορούσε να φτάσει αυτήν την ταχύτητα. Ελέγχοντας το documentation AMD product line [23], και εστιάζοντας στις απαιτήσεις του design, υπήρχαν τρεις υποψήφιες συσκευές, όπως φαίνονται στον Πίνακα 9.

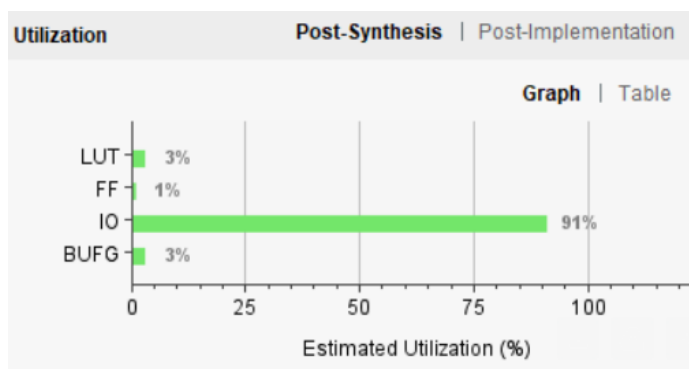
Πίνακας 9: Artix7 Devices Characteristics

	XC7A75T	XC7A100T	XC7A200T
Logic Cells	75,520	101,440	215,360
DSP Slices	180	240	740
Memory	3,780	4,860	13,140
GTP 6.6 Gb/s Transceivers	8	8	16
I/O Pins	300	300	500

Μετά από δοκιμές και των τριών σε grade -3 (που είναι το ταχύτερο) και σε grade -1 (η μικρότερη), παρατηρήθηκαν ελάχιστες διαφορές. Συνεπώς επιλέχθηκε αυτό με το μικρότερο grade ταχύτητας με part number: XC7A75TFGG676-1, το οποίο ικανοποιεί τις ελάχιστες



ανάγκες του design , καθώς διαθέτει 300 I/O pins έναντι των 274 που απαιτούνται, ενώ παράλληλα οι διαθέσιμοι πόροι (LUTs, Flip-Flops και BUFGs) αποδείχθηκαν αρκετοί σύμφωνα με τα reports του Vivado. Συμπερασματικά, η επιλογή της συγκεκριμένης συσκευής κρίθηκε και ως η οικονομικότερη λόγω των αρκετά λιγότερων I/O ports, της μικρότερης ταχύτητας και των λιγότερων λογικών στοιχείων και μνήμης. Η χρήση των πόρων της συσκευής από τον σχεδιασμό της συγκεκριμένης υλοποίησης παρουσιάζεται στην Εικόνα 14.

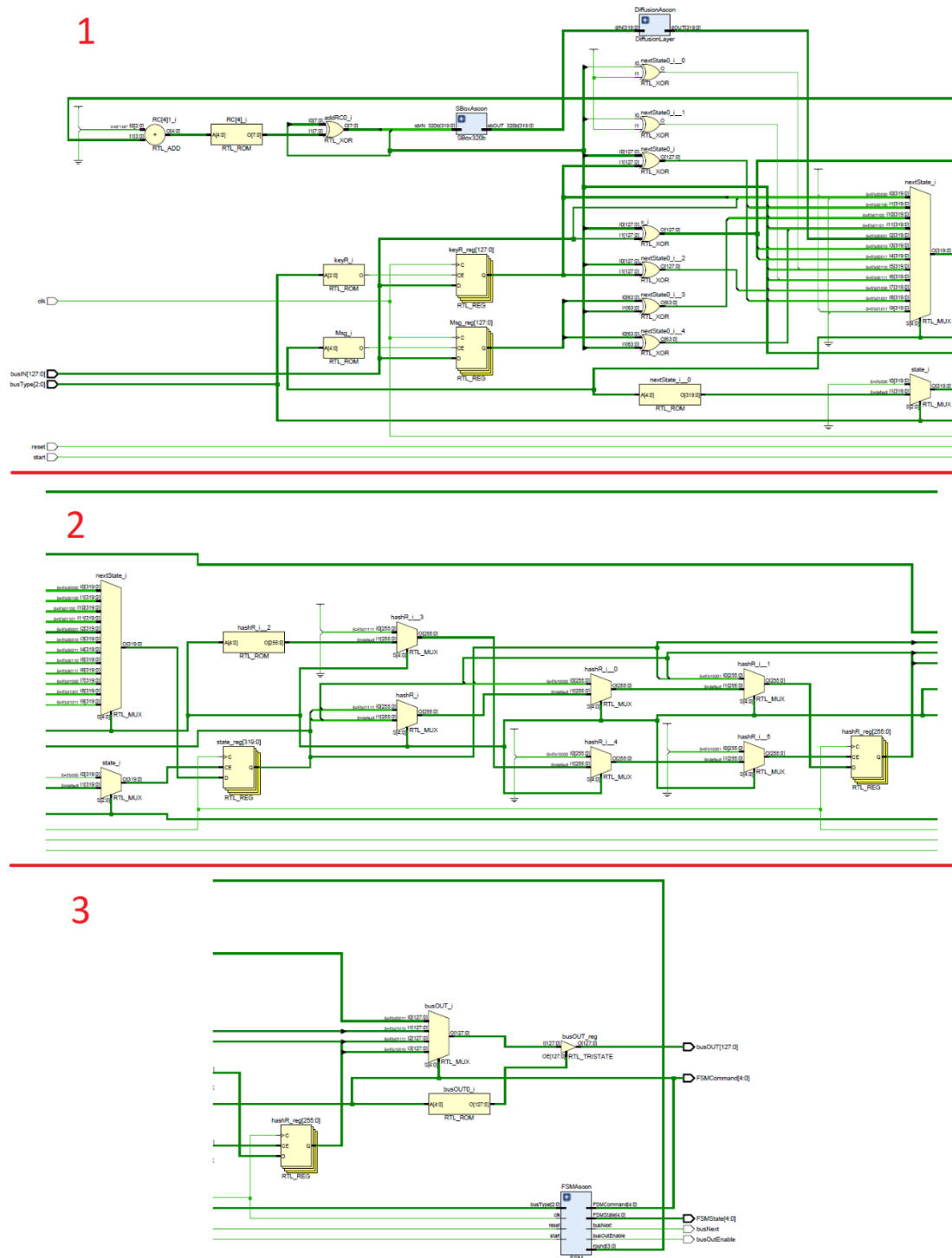


Εικόνα 14: Current System Utilization

3. Αρχιτεκτονική και λειτουργικότητα συστήματος

3.1 Register Transfer Level

Η αρχιτεκτονική του συστήματος σε επίπεδο καταχωρητών αποτυπώνεται στην Εικόνα 15. Εκεί ενσωματώνονται τρεις συνεχόμενες εικόνες που περιλαμβάνουν όλο το σύστημα, εξαιτίας της συμπεκνωμένης πληροφορίας όμως, είναι δύσκολη η ανάγνωσή της. Κατά συνέπεια για αναλυτικότερες πληροφορίες κατευθυνόμαστε στο project του Vivado.



Εικόνα 15: Αρχιτεκτονική συστήματος σε επίπεδο καταχωρητών (RTL)



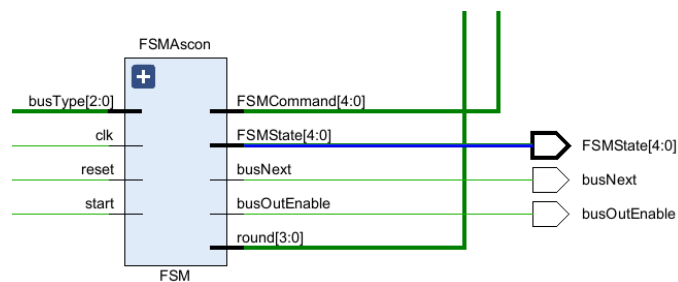
Στην συνέχεια παρουσιάζονται οι βασικοί καταχωρητές ελέγχου, οι οποίοι υλοποιούν την λογική λειτουργία του συστήματος και περιλαμβάνονται στο υποπρόγραμμα **FSM.vhd**.

Στον Πίνακα 10 και στην Εικόνα 16 παρουσιάζεται ο FSMState_reg, διαθέτει 24 καταστάσεις και ορίζεται ως FSMState[4:0] με δυαδικό μήκος ($\log_2 24 = 4.6$) των 5 bits.

Στον Πίνακα 11 παρουσιάζεται ο FSMCommand_reg, είναι των 5 bits λόγω των 20 καταστάσεων, και στην Εικόνα 17 απεικονίζεται η διαδρομή που ακολουθεί ο δίαυλος του (μπλε γραμμή).

Πίνακας 10: FSMState_reg

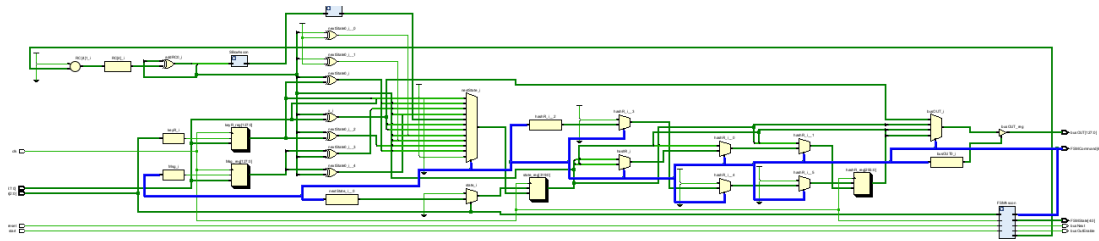
State	Encoding
idle	00000
initround	00001
initover	00010
initfinalstage	00011
absorbfinalcheck	00100
perround	00101
absorbover	00110
absorbend	00111
perround2	01000
tagfin	01001
hashinit	01010
hashround	01011
hashmsg	01100
hashmsgfin	01101
hashsh1	01110
hashsh2	01111
hashsh3	10000
hashsh4	10001
finall	10010
last	10011
hashroundfin	10100
encodeover	10101
encodeoverfinal	10110
tag	10111



Εικόνα 16: FSMState_reg bus

Πίνακας 11: FSMCommand_reg States

Init	applyRound	applyAD
applyEnc	applyKeyI	applyKeyF
applyOne	applyOneAt0	applyKeyTag
applyKeyTagFin	applyTag	applyHashInit
applyHashMsg_1	applyHashMsg_2	getHash0
getHash1	getHash2	getHash3
printLast	NOP	



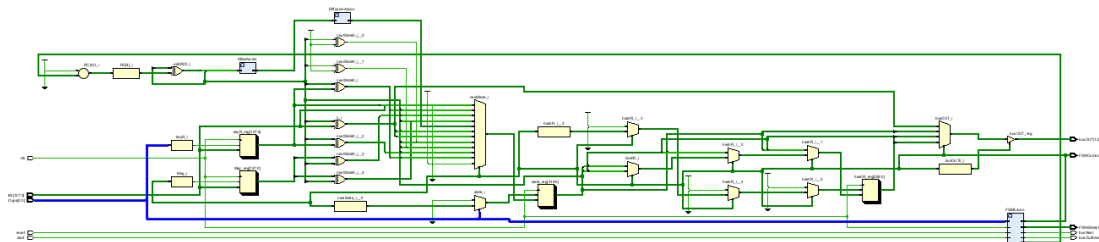
Εικόνα 17: FSMCommand_reg bus

Στον Πίνακα 12 παρουσιάζεται ο busType_reg, που είναι των τριών bits λόγω των έξι καταστάσεων, και στην Εικόνα 18 απεικονίζεται η διαδρομή που ακολουθεί ο δίαυλος του (μπλε γραμμή).

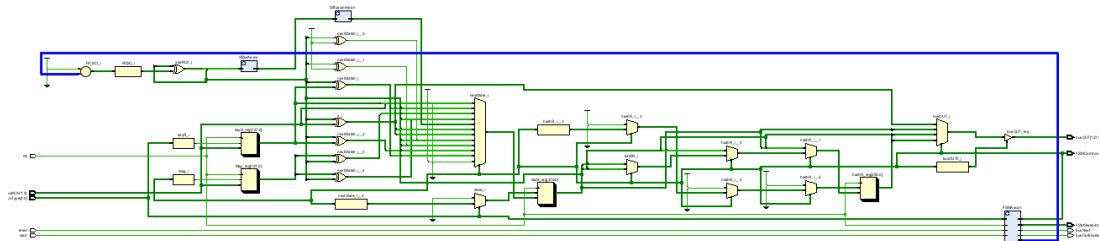
Στην Εικόνα 19 παρουσιάζεται ο round_reg ο οποίος είναι των τεσσάρων bits (μετράει από το 0 μέχρι το 12), στην ίδια εικόνα απεικονίζεται και η διαδρομή του δίαυλου του (μπλε γραμμή).

Πίνακας 12: busType_reg States

Key	Nonce	AData
Message	Tag	Hash256



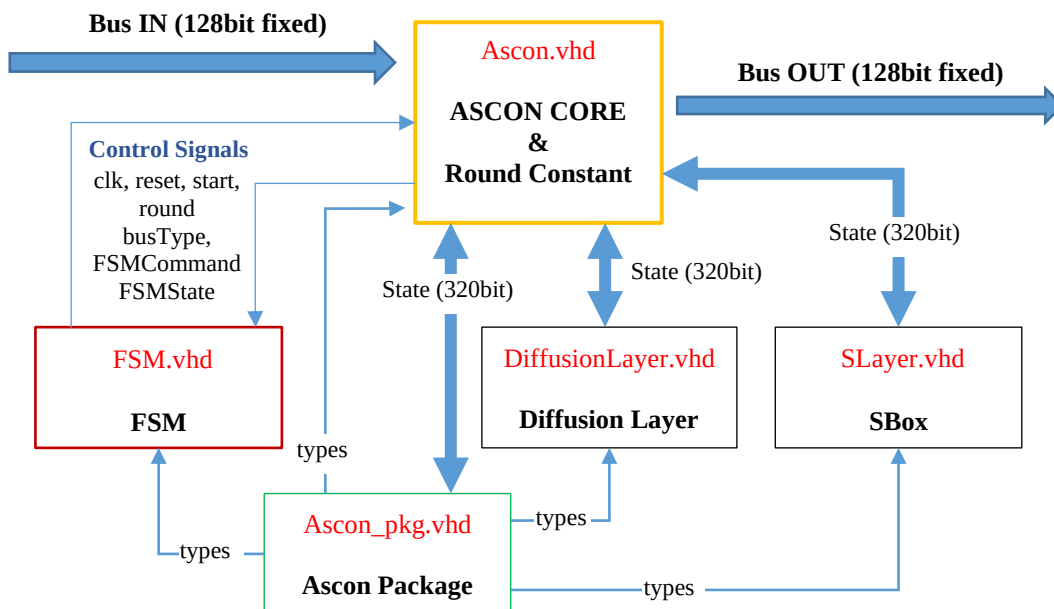
Εικόνα 18: busType_reg bus



Εικόνα 19: round_reg bus

Όπως αναφέρθηκε και στην αρχή της ενότητας, οι πληροφορίες που μπορούν να αντληθούν από τις εικόνες των καταχωρητών είναι περιορισμένες λόγω της συμπτυκνωμένης πληροφορίας. Επομένως, για περαιτέρω πληροφόρηση μεταφερόμαστε στο Vivado, εδώ έχουν παρατεθεί για να εμφανίσουν τα μήκη και τις πολυπλοκότητες των διαδρομών (για τυχόν μελλοντική βελτιστοποίηση), όπως και τις καταστάσεις που περιέχουν.

Με βάση την ανάλυση που προηγήθηκε παρουσιάζεται η αρχιτεκτονική του συστήματος, και στην Εικόνα 20 παρατίθεται το block διάγραμμα του προτεινόμενου συστήματος. Στο διάγραμμα απεικονίζονται οι επιμέρους λειτουργικές μονάδες, καθώς και οι μεταξύ τους διασυνδέσεις. Η λειτουργία και ο ρόλος κάθε μονάδας αναλύονται εκτενέστερα στην [Ενότητα 3.2](#) (Ανάλυση και Λειτουργικότητα Μονάδων).



Εικόνα 20: Current System Block Diagram

3.2 Ανάλυση και λειτουργικότητα μονάδων

Η πρώτη μονάδα του συστήματος είναι το πακέτο **Ascon_pkg.vhd** στο οποίο έχουν δηλωθεί όλες οι κοινές και ευρέως χρησιμοποιούμενες διαδικασίες. Για τη διευκόλυνση της ανάπτυξης και της αναγνωσιμότητας του κώδικα, έχουν οριστεί νέοι τύποι δεδομένων, υπότυποι, σταθερές και συναρτήσεις. Η λειτουργία των νέων τύπων έχει αναφερθεί στην [Ενότητα 3.1](#) (Register Transfer Level), οι υπότυποι χρησιμοποιούνται κυρίως για λόγους ευκολότερης ανάγνωσης και κατανόησης του κώδικα, ενώ οι σταθερές εξυπηρετούν προκαθορισμένες μη επεξεργάσιμες τιμές. Υπάρχουν επίσης οι συναρτήσεις που αντιστρέφουν το endianness από big σε little ή από little σε big, σε λέξεις των 64, 128 και 320 bits. Η επιλογή δημιουργίας τριών ξεχωριστών συναρτήσεων έγινε με στόχο τη βελτιστοποίηση της σύνθεσης του κυκλώματος. Μια γενικευμένη συνάρτηση με μεταβλητές παραμέτρους θα οδηγούσε σε αυξημένη επιφάνεια υλοποίησης, καθώς το συντιθέμενο κύκλωμα θα περιλάμβανε επιπλέον λογική που τελικά δεν θα χρησιμοποιούνταν. Πέρα από αυτά, υλοποιήθηκε και μια συνάρτηση που πραγματοποιεί δεξιά περιστροφή σε λέξεις των 64bits, κατά προκαθορισμένο αριθμό θέσεων.



Η μονάδα **SLayer.vhd** αποτελείται από δυο υποπρογράμματα, τα οποία συνδυάζονται μέσω port mapping.

Το πρώτο είναι το **SBox5b** το οποίο μεταφέρει την λογική της Substitution Layer και πιο συγκεκριμένα τις λογικές πράξεις που φαίνονται στην Εικόνα 5: ASCON SBox Logic) σε λέξεις μήκους των 5 bits. Το δεύτερο υποπρόγραμμα **SBox320b**, δημιουργεί μέσω port mapping και generate statements 64 τέτοιες διαδικασίες, οι οποίες τρέχουν παράλληλα και ενημερώνουν το state.

Η **DiffusionLayer.vhd** κάνει χορ περιστρέφοντας σε δυο διαφορετικές διατάξεις το $s(i)$ του state S . Πιο συγκεκριμένα, οι πέντε λέξεις των 64 bits του state επεξεργάζονται με διαφορετικούς συνδυασμούς περιστροφών, σύμφωνα με τις λογικές πράξεις που παρουσιάζονται στην Εικόνα 8: ASCON Diffusion Layer Logic). Για την υλοποίηση των περιστροφών χρησιμοποιείται η συνάρτηση δεξιάς περιστροφής που έχει οριστεί στο **Ascon_pkg.vhd**.

Η μονάδα **FSM.vhd** υλοποιεί μια Μηχανή Πεπερασμένων Καταστάσεων. Ευθύνη της FSM είναι ανά κύκλο ρολογιού να ελέγχει και να εκτελεί όλο τον αλγόριθμο του ASCON AEAD128 και Hash256. Αυτό επιτυγχάνεται μέσω των καταχωρητών που έχουν ήδη αναφερθεί στην [Ενότητα 3.1](#) (Register Transfer Level) και είναι οι **FSMState_reg**, **FSMCommand_reg**, **busType_reg**, **round_reg** όπως και των **check_reg** που είναι υποτύπου **hashChecking** των τεσσάρων bits, και των **busNext_reg**, **flag_reg** που είναι του ενός bit.

Η FSM αναλαμβάνει τη φόρτωση των nonce, associated data και message όταν απαιτείται, ενώ παράλληλα αποστέλλει εντολές προς τον πυρήνα **Ascon.vhd** μέσω του καταχωρητή **FSMCommand_reg**. Το μυστικό κλειδί (key) φορτώνεται αρχικά απευθείας από τη μονάδα **Ascon.vhd**. Επομένως, αν εξεταστεί ο κώδικας σε συνδυασμό με τις κυματομορφές της προσομοίωσης που παρουσιάζονται στην [Ενότητα 5.1](#) (Έλεγχος Ορθής Λειτουργίας), γίνεται άμεσα κατανοητή η λειτουργία του συστήματος, καθώς οι βασικοί καταχωρητές εμφανίζονται σε μορφή συμβολοσειρών, αποτυπώνοντας με σαφήνεια τις καταστάσεις και τις μεταβάσεις του αλγορίθμου.

Ο πυρήνας του συστήματος που είναι η μονάδα **Ascon.vhd**. Αρχικά φορτώνει το μυστικό κλειδί (key) στο σύστημα και αναμένει τις εντολές που θα της σταλούν από την **FSM.vhd** μέσω του **FSMCommand_reg**.

Αυτές είναι οι: Init, applyRound, applyAD, applyEnc, applyKeyI, applyKeyF, applyOne, applyOneAt0, applyKeyTag, applyKeyTagFin, applyTag, applyHashInit, applyHashMsg_1, applyHashMsg_2, getHash0, getHash1, getHash2, getHash3, printLast, NOP.

Κύρια λειτουργία της μονάδας αποτελεί η υλοποίηση της συνάρτησης μετάθεσης (permutation), η οποία εκτελείται σε έναν κύκλο ρολογιού. Η διαδικασία αυτή υλοποιείται μέσω τριών επιμέρους φάσεων οι οποίες λειτουργούν παράλληλα:



- Εκτελείται η σταθερά γύρου (RC ή Round Constant) με την εντολή:

```
addRC <= state(319 downto 136) & (state(135 downto 128) xor RC(4 + round)) & state(127 downto 0);
```

Γίνεται xor των 8 LSB (135 downto 128) του s(2) με την σταθερά που έχει ήδη οριστεί στο **Ascon_pkg.vhd**. Η παραπάνω εντολή τρέχει συνέχεια και ανανεώνει τον καταχωρητή addRC ανά κύκλο ρολογιού.

- Παράλληλα μέσω port mapping εκτελείται η **SLayer.vhd** με την εντολή:

```
port map(sbIN_320b => addRC, sbOUT_320b => SBoxState);
```

Σαν είσοδο δέχεται τον καταχωρητή addRC, εκτελείται η substitution layer και παράγει ως έξοδο το SBoxState.

- Για να ολοκληρωθεί μια μετάθεση (permutation) δίνεται σαν παράλληλη είσοδος ο καταχωρητής SBoxState στην diffusion layer, μέσω της εντολής:

```
port map(dIN => SBoxState, dOUT => perEND);
```

Και ορίζεται σαν έξοδος ο register perEND, που με την εντολή applyRound του FSM,

```
with FSMCommand select  
  nextState <= ..... (λοιπές εντολές)  
               perEND when applyRound,
```

εφαρμόζεται μια ολοκληρωμένη μετάθεση (permutation) σε έναν κύκλο ρολογιού.

Τελευταίο είναι το αρχείο χρονικών περιορισμών **timingChecking.xdc**. Χρησιμοποιείται για τον ορισμό της ταχύτητας του συστήματος, ώστε να εξαχθούν ρεαλιστικά δεδομένα χρονισμού και απόδοσης. Ποιο συγκεκριμένα ορίζεται ως:

```
create_clock -period 5.000 [get_ports clk]
```

Η εντολή αυτή δημιουργεί ένα ρολόι (clock) με περίοδο 5 ns στον ακροδέκτη clk του συστήματος, το οποίο αντιστοιχεί σε συχνότητα:

$$f = \frac{1}{T} \quad \text{άρα} \quad f = \frac{1}{5 \cdot 10^{-9}} = 0.2 \cdot 10^9 = \mathbf{200MHz}$$

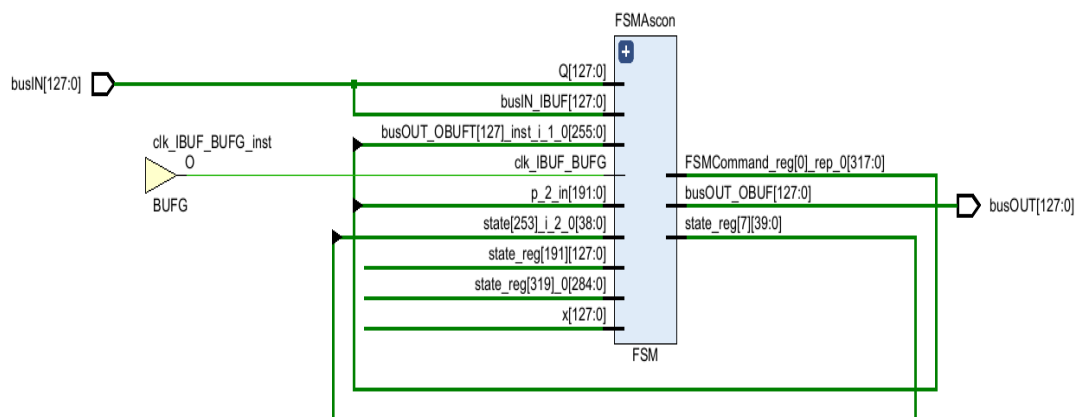


4. Υλοποίηση συστήματος στο Vivado

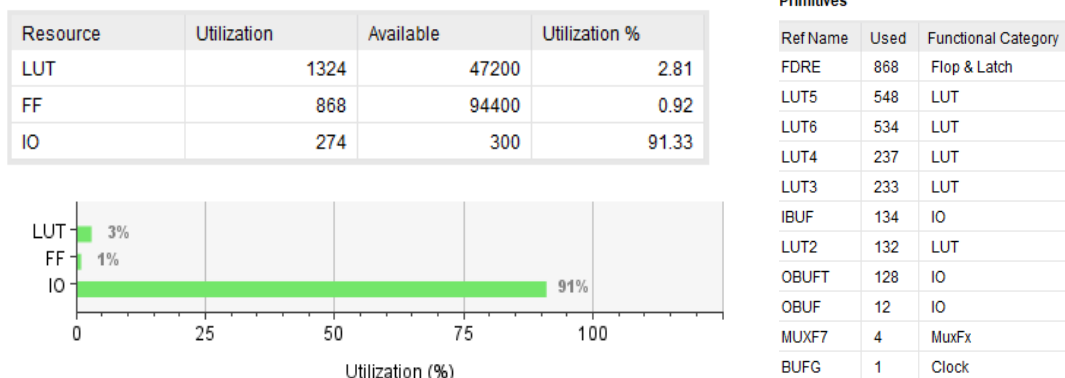
Όλα τα αρχεία της υλοποίησης έχουν αναρτηθεί στο παρόν [repository \[24\]](#) του GitHub.

4.1 System Synthesis

Στην Εικόνα 21 παρουσιάζεται το dataflow του συστήματος και οι διασυνδέσεις του, στην συνέχεια η Εικόνα 22 εμφανίζει τη σύνοψη χρήσης και τους πόρους που καταναλώνονται από αυτό.



Εικόνα 21: System Dataflow Design



Εικόνα 22: Synthesis System Utilization Summary

Το timing summary που δίνεται από το Vivado, χωρίζεται σε τρία διακριτά πεδία:

- **Setup.** Ενημερώνει για τον χρόνο που φτάνουν τα δεδομένα στους καταχωρητές, πριν την επόμενη ακμή του clock (για να μην υπάρξει πρόβλημα, είναι απαραίτητη θετική τιμή).



- **Hold.** Ενημερώνει για τον χρόνο που πρέπει να παραμένουν σταθερά τα δεδομένα στους καταχωρητές, πριν την επόμενη ακμή του clock (για να μην υπάρξει πρόβλημα, είναι απαραίτητη θετική τιμή).
- **Pulse Width.** Ενημερώνει για τον χρόνο που πρέπει να παραμένουν σταθερά τα σήματα, σε Low ή High κατάσταση (για να μην υπάρξει πρόβλημα, είναι απαραίτητη θετική τιμή).

Στον Πίνακα 13 φαίνεται η σύνοψη χρονισμού του συστήματος προς υλοποίηση, εκεί παρατηρείται το μικρό περιθώριο του Worst Hold Slack το οποίο είναι 147 ps. Συνεπώς αν σε μελλοντική αναβάθμιση χρειαστεί να αυξηθεί η ταχύτητα, θα υπάρξει ο παρόν περιορισμός.

Πίνακας 13: Synthesis Timing Summary

Clock: clk

Statistics

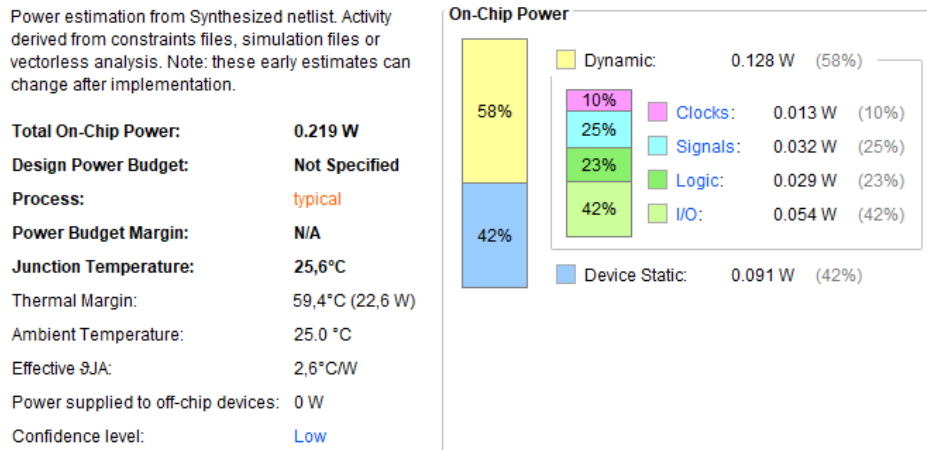
Type	Worst Slack	Total Violation	Failing Endpoints	Total Endpoints
Setup	0,905 ns	0,000 ns	0	1346
Hold	0,147 ns	0,000 ns	0	1346
Pulse Width	2,000 ns	0,000 ns	0	869

Για να αρθεί ο περιορισμός του μικρού Hold time, μπορεί να πραγματοποιηθεί troubleshooting στα intra clock paths (Πίνακας 14), έτσι ώστε να διαπιστωθεί σε ποιους καταχωρητές και μονοπάτια παρουσιάζεται η καθυστέρηση.

Πίνακας 14: Synthesis Hold Slacks (Intra clock paths)

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Sc
Path 11	0.147	1	2	7	FSMAscon/flag_reg/C	FSMAscon/flag_reg/D	0.391	0.245	0.146	0.0	cli
Path 12	0.169	1	2	48	FSMAscon/FSM_s...tate_reg[3]/C	FSMAscon/round_reg[1]/D	0.413	0.248	0.165	0.0	cli
Path 13	0.170	1	2	68	FSMAscon/FSM_s...tate_reg[1]/C	FSMAscon/busNext_reg/D	0.414	0.245	0.169	0.0	cli
Path 14	0.171	0	1	11	state_reg[56]/C	hashR_reg[0]/D	0.307	0.147	0.160	0.0	cli
Path 15	0.171	0	1	11	state_reg[28]/C	hashR_reg[100]/D	0.307	0.147	0.160	0.0	cli
Path 16	0.171	0	1	11	state_reg[29]/C	hashR_reg[101]/D	0.307	0.147	0.160	0.0	cli
Path 17	0.171	0	1	11	state_reg[30]/C	hashR_reg[102]/D	0.307	0.147	0.160	0.0	cli
Path 18	0.171	0	1	11	state_reg[31]/C	hashR_reg[103]/D	0.307	0.147	0.160	0.0	cli
Path 19	0.171	0	1	11	state_reg[21]/C	hashR_reg[109]/D	0.307	0.147	0.160	0.0	cli
Path 20	0.171	0	1	11	state_reg[50]/C	hashR_reg[10]/D	0.307	0.147	0.160	0.0	cli

Σαν τελευταία αναφορά της σύνθεσης προβάλλεται η κατανάλωση ενέργειας (Εικόνα 23). Ο σχεδιασμός έχει συνολική κατανάλωση 219 mW, με 58% δυναμική και 42% στατική (η στατική έχει μεγάλο ποσοστό γιατί είναι αρκετά μικρή η δυναμική). Σε περίπτωση που πρέπει να μειωθεί περεταίρω η συνολική κατανάλωση, μπορεί να γίνει χρονισμός του συστήματος στα 100MHz αντί των 200MHz (θα υπάρξει μια μείωση της δυναμικής περίπου στο 50%).



Εικόνα 23: Synthesis Power Consumption Summary

4.2 System Implementation

Η synthesis (σύνθεση) μεταφράζει τον κώδικα απλά σε λογική, συνεπώς τα αποτελέσματα της Ενότητας 4.1 (System Synthesis) δεν είναι απόλυτα ακριβή. Συμπληρωματικά λοιπόν θα παρατεθούν εκ νέου και τα στοιχεία από το implementation (υλοποίηση), όπου το Vivado τοποθετεί αυτήν την λογική σε συγκεκριμένο FPGA (στην παρόν στο xc7a75tfgg676-1). Αρχικά θα αντιπαρατεθούν τα στοιχεία του Πίνακα 13 (Synthesis Timing Summary), με αυτά του Πίνακα 15 (Implementation Timing Summary). Εστιάζοντας στο Worst Hold Slack που διαπιστώνονται χρονικές αποκλίσεις ($0.152 - 0.147 = 0.005$) της τάξης των 5 ps.

Πίνακας 15: Implementation Timing Summary

latch_loops (0)

Intra-Clock Paths

clk

Setup 0,402 ns (10)

Hold 0,152 ns (10)

Pulse Width 2,000 ns (30)

Inter-Clock Paths

Other Path Groups

User Ignored Paths

Unconstrained Paths

Statistics

Type	Worst Slack	Total Violation	Failing Endpoints	Total Endpoints
Setup	0,402 ns	0,000 ns	0	1356
Hold	0,152 ns	0,000 ns	0	1356
Pulse Width	2,000 ns	0,000 ns	0	874

Αξίζει να σημειωθεί λοιπόν ότι οι διαφορές είναι αρκετά μικρές, αλλά υπάρχουν. Επίσης εάν η αξιολόγηση βασιζόταν αποκλειστικά στις αναφορές της σύνθεσης, το τελικό σύστημα θα παρουσίαζε αποκλίσεις από την πραγματική συμπεριφορά της υλοποίησης. Αυτό γίνεται πιο εμφανές συγκρίνοντας τον Πίνακα 14 (Synthesis Hold Slacks (Intra clock paths)) με τα αντίστοιχα δεδομένα του Πίνακα 16 (Implementation Hold Slacks (intra clock paths)).

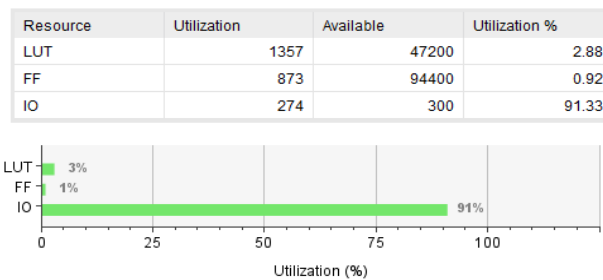
Παρατηρούνται σαφείς διαφοροποιήσεις στα χρονικά χαρακτηριστικά, όπου συνεπάγεται δημιουργία διαφορετικών κρίσιμων διαδρομών μεταξύ των κυκλωμάτων.



Πίνακας 16: Implementation Hold Slacks (intra clock paths)

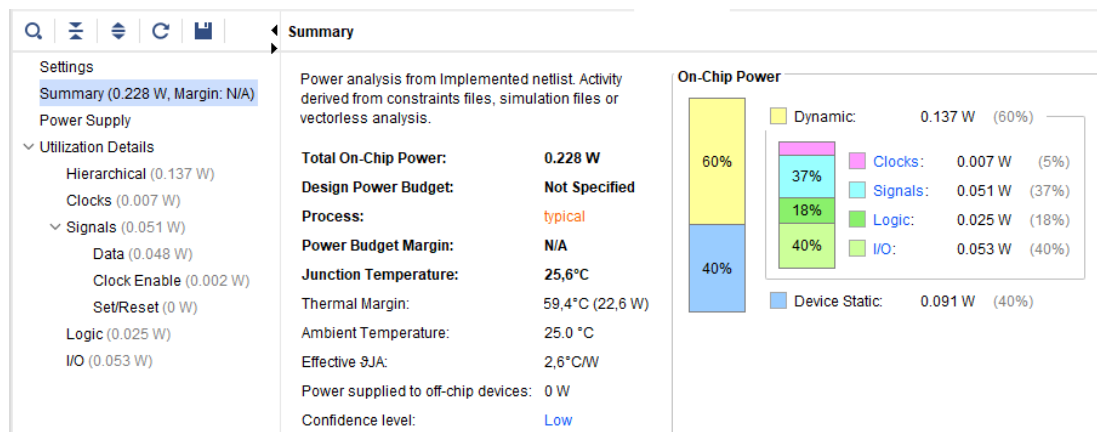
Timing													
Intra-Clock Paths - clk - Hold													
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	
Path 11	0.152	1	1	3	keyR_reg[107]C	state_reg[17]D	0.311	0.186	0.125	0.0	clk	clk	
Path 12	0.160	1	1	3	keyR_reg[111]C	state_reg[175]D	0.295	0.186	0.109	0.0	clk	clk	
Path 13	0.167	1	1	4	keyR_reg[26]C	state_reg[90]D	0.301	0.186	0.115	0.0	clk	clk	
Path 14	0.192	1	1	4	keyR_reg[60]C	state_reg[124]D	0.321	0.186	0.135	0.0	clk	clk	
Path 15	0.194	1	1	3	keyR_reg[96]C	state_reg[160]D	0.302	0.186	0.116	0.0	clk	clk	
Path 16	0.199	0	1	11	state_reg[8]C	hashR_reg[112]D	0.289	0.164	0.125	0.0	clk	clk	
Path 17	0.200	0	1	11	state_reg[29]C	hashR_reg[229]D	0.308	0.141	0.165	0.0	clk	clk	
Path 18	0.203	0	1	11	state_reg[60]C	hashR_reg[196]D	0.307	0.141	0.166	0.0	clk	clk	
Path 19	0.206	1	1	4	keyR_reg[50]C	state_reg[114]D	0.312	0.186	0.126	0.0	clk	clk	
Path 20	0.208	2	2	1	Msg_reg[23]C	state_reg[23]D	0.341	0.231	0.110	0.0	clk	clk	

Επιπροσθέτως παρουσιάζονται οι μικρές μεταβολές στους χρησιμοποιούμενους πόρους του FPGA (Εικόνα 24, Implementation System Utilization Summary), συγκριτικά με την προηγούμενη αναφορά της Εικόνας 22 (Synthesis System Utilization Summary). Παράλληλα, εξετάζονται οι διαφορές στην κατανάλωση ισχύος (Εικόνα 25, Implementation Power Summary), σε σχέση με τα προηγούμενα αποτελέσματα της Εικόνας 23 (Synthesis Power Summary), όπου και εκεί παρατηρείται μια μικρή απόκλιση μεταξύ θεωρητικής και πραγματικής κατανάλωσης.



Name	Slice LUTs (47200)	Slice Registers (94400)	F7 Muxes (31700)	Slice (15850)	LUT as Logic (47200)	Bonded IOB (300)	BUFGCTRL (32)
ASCON	1357	873	4	423	1357	274	1
FSMAscn (FSM)	993	41	4	377	993	0	0

Εικόνα 24: Implementation System Utilization Summary



Εικόνα 25: Implementation Power Consumption Summary



Συμπληρωματικά, παρουσιάζεται και η αντιστοίχιση ορισμένων από τους 274 ακροδέκτες του συστήματος (Πίνακας 17), που είναι απαραίτητοι για τη διασύνδεση της υλοποίησης με εξωτερικές συσκευές. Ενδεικτικά, κάποιες από τις στήλες που αναφέρονται είναι οι:

- Αριθμός και ονομασία του υλικού ακροδέκτη (1^η και 3^η στήλη).
- Ονομασία που του έχει δοθεί από τον προγραμματιστή (2^η στήλη).
- Τύπος χρήσης ακροδέκτη (4^η στήλη).
- I/O Bank στην οποία ανήκει (5^η στήλη).
- Ικανότητα διαχείρισης ρεύματος (6^η στήλη).
- Τερματισμός και εμπέδηση (7^η στήλη).

Πίνακας 17: System Implementation I/O pins Characteristics

Pin	Signal Name	Pin Name	Use	IO Bank	Drive (mA)	Off-Chip Termination
A1		VCCO_35	VCCO	35		
A2	busOUT[87]	IO_L20N_T3_35	TRISTATE	35	12	FP_VTT_50
A3	busOUT[88]	IO_L20P_T3_35	TRISTATE	35	12	FP_VTT_50
A4	busOUT[95]	IO_L16N_T2_35	TRISTATE	35	12	FP_VTT_50
A5	busOUT[97]	IO_L15N_T2_DQS_35	TRISTATE	35	12	FP_VTT_50
A17	busIN[32]	IO_L9P_T1_DQS_16	INPUT	16		NONE
A18	busIN[31]	IO_L9N_T1_DQS_16	INPUT	16		NONE
A19	busIN[29]	IO_L10N_T1_16	INPUT	16		NONE
A20	busIN[19]	IO_L15N_T2_DQS_16	INPUT	16		NONE
A22	busIN[15]	IO_L17N_T2_16	INPUT	16		NONE
A23	busIN[8]	IO_L21P_T3_DQS_16	INPUT	16		NONE
A24	busIN[7]	IO_L21N_T3_DQS_16	INPUT	16		NONE
A25	busIN[9]	IO_L20N_T3_16	INPUT	16		NONE
AA2	FSMState[0]	IO_L8P_T1_13	OUTPUT	13	12	FP_VTT_50
AA2	FSMCommand[4]	IO_L8N_T1_13	OUTPUT	13	12	FP_VTT_50
AA2	FSMState[2]	IO_L7P_T1_13	OUTPUT	13	12	FP_VTT_50
AA2	busNext	IO_L5N_T0_13	OUTPUT	13	12	FP_VTT_50
AB2	FSMCommand[3]	IO_L9P_T1_DQS_13	OUTPUT	13	12	FP_VTT_50
AB2	FSMState[1]	IO_L7N_T1_13	OUTPUT	13	12	FP_VTT_50
AB2	reset	IO_L3P_T0_DQS_13	INPUT	13		NONE
AC2	FSMCommand[2]	IO_L9N_T1_DQS_13	OUTPUT	13	12	FP_VTT_50
AC2	busType[2]	IO_L3N_T0_DQS_13	INPUT	13		NONE
B1	busOUT[85]	IO_L21N_T3_DQS_35	TRISTATE	35	12	FP_VTT_50
B2	busOUT[89]	IO_L19N_T3_VREF_35	TRISTATE	35	12	FP_VTT_50
B4	busOUT[96]	IO_L16P_T2_35	TRISTATE	35	12	FP_VTT_50
B5	busOUT[98]	IO_L15P_T2_DQS_35	TRISTATE	35	12	FP_VTT_50
B17	busIN[35]	IO_L7N_T1_16	INPUT	16		NONE
B19	busIN[30]	IO_L10P_T1_16	INPUT	16		NONE
B20	busIN[20]	IO_L15P_T2_DQS_16	INPUT	16		NONE
B21	busIN[17]	IO_L16N_T2_16	INPUT	16		NONE
B22	busIN[16]	IO_L17P_T2_16	INPUT	16		NONE
B24	busIN[3]	IO_L23N_T3_16	INPUT	16		NONE
B25	busIN[10]	IO_L20P_T3_16	INPUT	16		NONE
B26	busIN[5]	IO_L22N_T3_16	INPUT	16		NONE
C1	busOUT[86]	IO_L21P_T3_DQS_35	TRISTATE	35	12	FP_VTT_50
C2	busOUT[90]	IO_L19P_T3_35	TRISTATE	35	12	FP_VTT_50
C3	busOUT[93]	IO_L17N_T2_35	TRISTATE	35	12	FP_VTT_50
C4	busOUT[99]	IO_L14N_T2_SRCC_35	TRISTATE	35	12	FP_VTT_50
C17	busIN[36]	IO_L7P_T1_16	INPUT	16		NONE
C18	busIN[25]	IO_L12N_T1_MRCC_16	INPUT	16		NONE
C19	busIN[23]	IO_L13N_T2_MRCC_16	INPUT	16		NONE
C21	busIN[18]	IO_L16P_T2_16	INPUT	16		NONE
C22	busIN[12]	IO_L19P_T3_16	INPUT	16		NONE
C23	busIN[11]	IO_L19N_T3_VREF_16	INPUT	16		NONE
C24	busIN[4]	IO_L23P_T3_16	INPUT	16		NONE
C26	busIN[6]	IO_L22P_T3_16	INPUT	16		NONE
D1	busOUT[81]	IO_L23N_T3_35	TRISTATE	35	12	FP_VTT_50



5. Αξιολόγηση

5.1 Έλεγχος ορθής λειτουργίας

Στο παρόν κεφάλαιο παρουσιάζεται η διαδικασία επαλήθευσης και ελέγχου ορθής λειτουργίας της υλοποίησης. Αρχικά, αναλύεται το αρχείο **testbench.vhd**, το οποίο χρησιμοποιήθηκε για την προσομοίωση και την επαλήθευση της λειτουργίας του συστήματος στο περιβάλλον Vivado, μέσω της σύγκρισης των παραγόμενων αποτελεσμάτων με προκαθορισμένα δεδομένα εισόδου και εξόδου. Στη συνέχεια, παρουσιάζεται η διαδικασία παραγωγής έγκυρων διανυσμάτων ελέγχου (test vectors) με χρήση της εφαρμογής **pyAscon.py**, η οποία είναι γραμμένη σε Python. Τα παραγόμενα διανύσματα χρησιμοποιήθηκαν ως δεδομένα αναφοράς για την αξιολόγηση της υλοποίησης και την αποσφαλμάτωσή της. Ακολούθως, περιγράφεται η διαδικασία παρακολούθησης και επαλήθευσης των ενδιάμεσων τιμών της εσωτερικής κατάστασης (state) σε κάθε στάδιο λειτουργίας του αλγορίθμου, με σκοπό τον εντοπισμό και τη διόρθωση πιθανών αποκλίσεων από την αναμενόμενη συμπεριφορά. Τέλος, λαμβάνονται όλες οι απαραίτητες μετρήσεις και συγκρίνονται, όπως και αξιολογούνται με άλλες δημοσιευμένες ερευνητικές εργασίες.

5.1.1 Vivado simulation

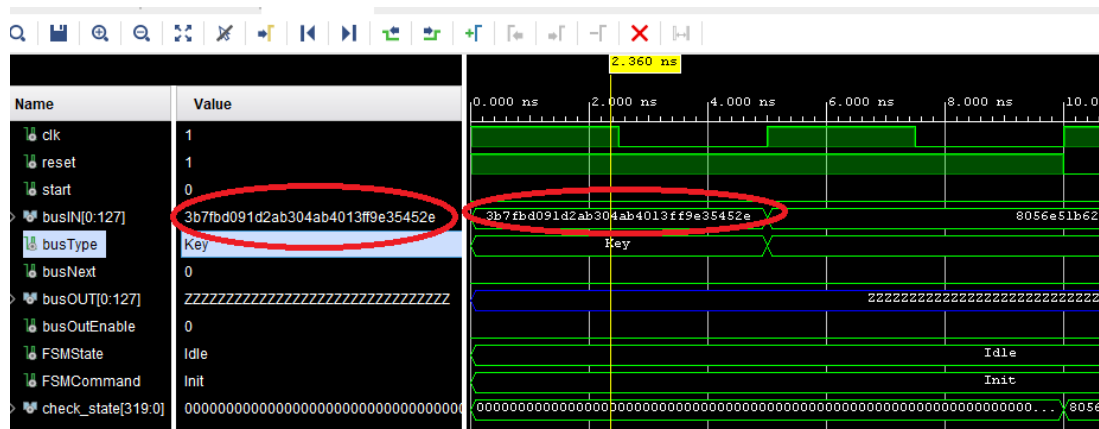
Για την επαλήθευση της ορθής λειτουργίας του αλγορίθμου ASCON AEAD128, πραγματοποιείται προσομοίωση μέσω του **testbench.vhd**, με δεδομένα εισόδου αυτά της Εικόνας 26. Η εφαρμογή τους στο σύστημα γίνεται αφού πρώτα αλλάξει η σειρά των bytes στην κάθε λέξη των 128 bits, κατά συνέπεια πραγματοποιείται μια μετατροπή από big endian σε little endian μέσω της συνάρτησης `swap_endian`.

Στην Εικόνα 27 παρουσιάζεται η εισαγωγή του μυστικού κλειδιού (`busType = Key`) και η μετατροπή του `endianness` που πραγματοποιήθηκε.

Από [2e45359eff1340ab04b32a1d09bd7f3b](#) μετατρέπεται σε [3b7fbd091d2ab304ab4013ff9e35452e](#).

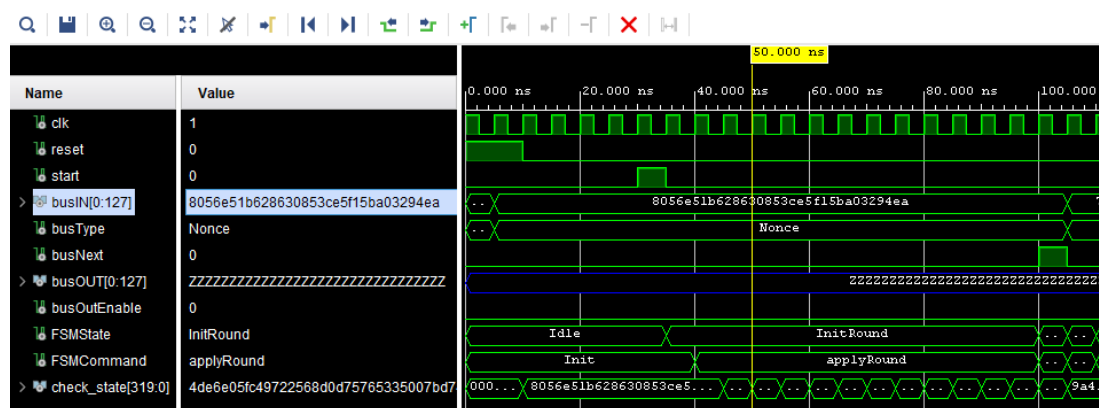
```
-- test 1
busIN <= swap_endian(x"2e45359eff1340ab04b32a1d09bd7f3b"); --testA
busType <= Key;
wait until (rising_edge(Clk)); -- and (busNext = '1');
busIN <= swap_endian(x"ea9432a05bf1e53c853086621be55680"); --testA
busType <= Nonce;
wait until (rising_edge(Clk) and (busNext = '1'));
busIN <= swap_endian(x"54686973207465787420697320617373"); --testA
busType <= AData;
wait until (rising_edge(Clk) and (busNext = '1'));
busIN <= (127=>'1', others=>'0');
busType <= AData;
wait until (rising_edge(Clk) and (busNext = '1'));
busIN <= swap_endian(x"05319cf7ffec692672cf2722d8df7f27"); --testA
busType <= Message;
```

Εικόνα 26: Testbench Input Data



Στην Εικόνα 28 εμφανίζεται η εισαγωγή του nonce (busType = Nonce) και η μετατροπή του endianness που πραγματοποιήθηκε.

Από `ea9432a05bf1e53c853086621be55680` γίνεται `8056e51b628630853ce5f15ba03294ea`.

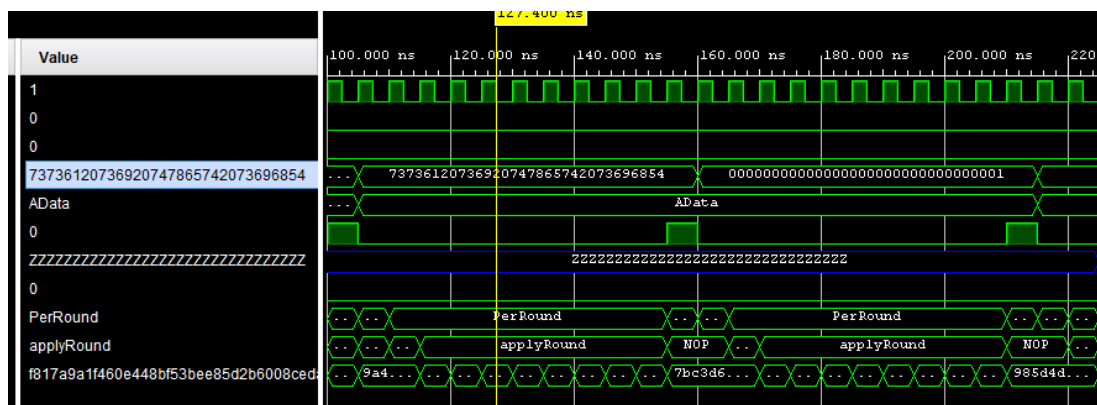


Στην Εικόνα 29 πραγματοποιείται η είσοδος των associate data στα οποία γίνεται και padding (λόγω του ότι ο αλγόριθμος αναφέρει $A_m \leftarrow \text{pad}(A_m, 128)$ και το σύστημα υλοποιείται με fixed length words των 128 bits).

Συνεπώς παρατηρώντας το busIN την χρονική στιγμή που το busType ορίζεται ως AData, στα αριστερά του υπάρχουν τα associate data με την μετατροπή του endianness

54686973207465787420697320617373 → 73736120736920747865742073696854.

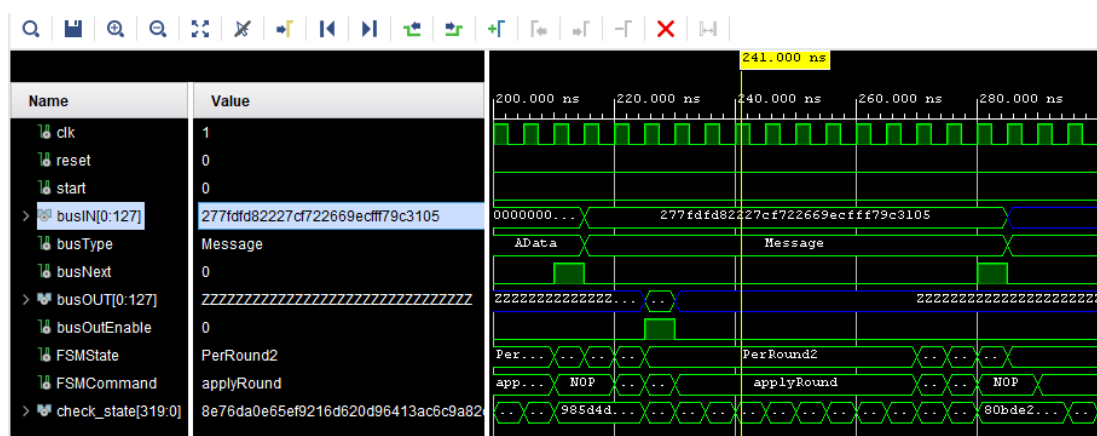
Στα δεξιά ακολουθεί το padding με τιμή 0x10000000000000000000000000000000 (στο Vivado το ποιο αριστερό bit είναι το LSB, για αυτό στο simulation φαίνεται ως 0x01).



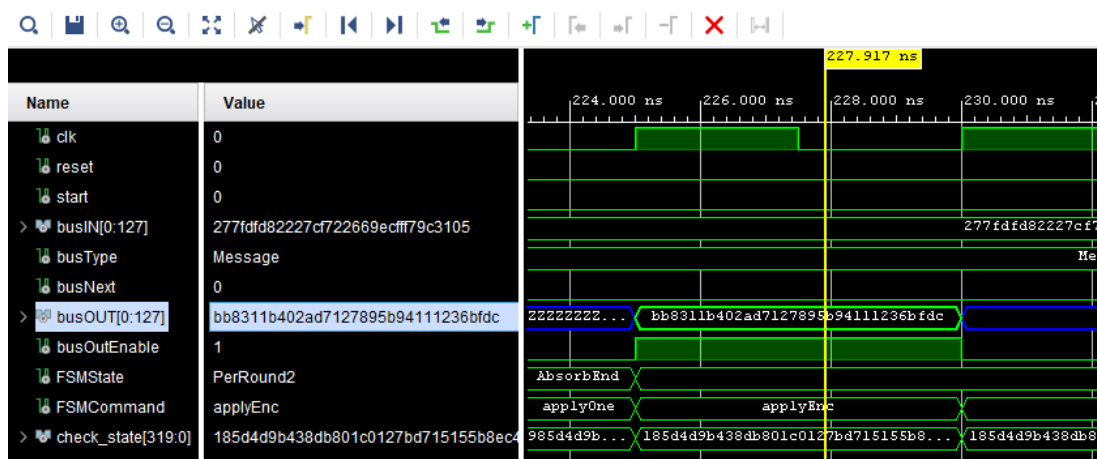
Εικόνα 29: System Associate Data Input

Τελευταίο εισάγεται το μήνυμα (busType = Message) που παρουσιάζεται στην Εικόνα 30, εμφανίζεται επίσης η μετατροπή του endianness, που η αρχική τιμή 05319cf7ffec692672cf7222d8df7f27 γίνεται 277dfd82227cf722669ecfff79c3105.

Τα παραπάνω δεδομένα εισόδου (Key, Nonce, AD, Message), μετά την επεξεργασία τους παράγουν το κρυπτογράφημα στον καταχωρητή εξόδου των 128 bits busOUT (Εικόνα 31), και παραμένουν εκεί για όση ώρα παραμένει σε λογική κατάσταση High το busOutEnable.



Εικόνα 30: System Message Input

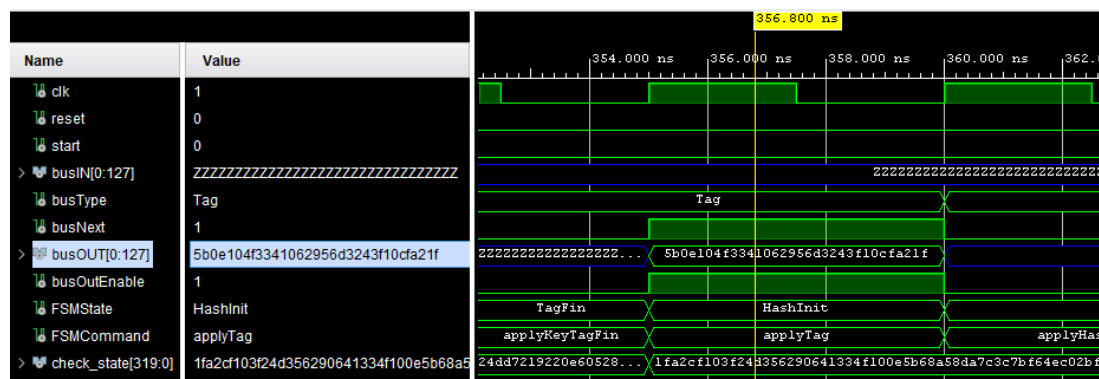


Εικόνα 31: System Ciphertext Output

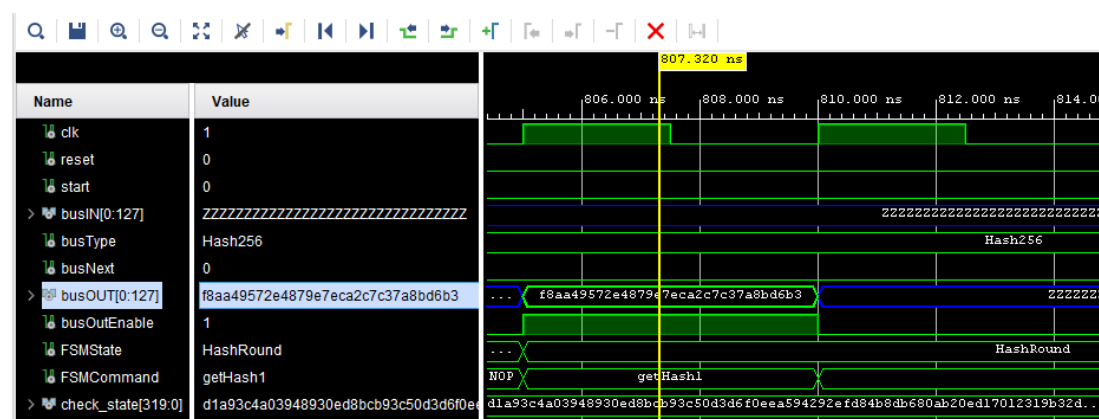


Ακολουθώς παράγεται το Tag (Εικόνα 32), ακολουθούν τα πρώτα 128 MSB της Hash256 (Εικόνα 33), και τα υπόλοιπα 128 LSB στην συνέχεια (Εικόνα 34).

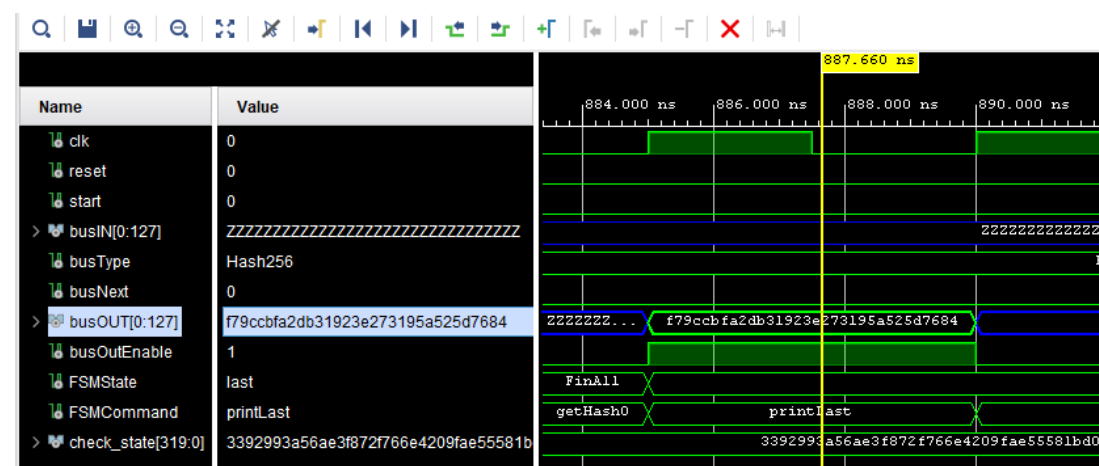
Η αναγνώριση του τύπου της εξόδου (Message, Tag, Hash), γίνεται από τον καταχωρητή busType. Και τα δεδομένα αντλούνται από το busOUT όταν το busOutEnable είναι High.



Εικόνα 32: System Tag Output



Εικόνα 33: Hash256 MSB Output (255-128)



Εικόνα 34: Hash256 LSB Output (127-0)

Επιπροσθέτως οι εξοδοι εκτυπώνονται μέσω εντολών report στο log (Εικόνα 35), όπου δίπλα τους υπάρχει και η χρονική στιγμή παραγωγής τους (time stamp).



```
Tcl Console Messages Log x
Time resolution is 1 ps
Note: Ciphertext = BB8311B402AD7127895B94111236BFDC
Time: 230 ns Iteration: 0 Process: /testbench/line__90 File: E:/temp/Ascon/VHDL/DE - Copy/sources/testbench.vhd
Note: Tag = 5B0E104F3341062956D3243F10CFA21F
Time: 360 ns Iteration: 0 Process: /testbench/line__90 File: E:/temp/Ascon/VHDL/DE - Copy/sources/testbench.vhd
Note: Hash 256 = F8AA49572E4879E7ECA2C7C37A8BD6B3F79CCBFA2DB31923E273195A525D7684
Time: 890 ns Iteration: 0 Process: /testbench/ASCONTestBench/line__61 File: E:/temp/Ascon/VHDL/DE - Copy/sources/Ascon.vhd
```

Εικόνα 35: Vivado Simulation Log

5.1.2 Δημιουργία δεδομένων ελέγχου

Η σύνοψη των δεδομένων εξόδου της Εικόνας 35, διασταυρώνεται με την έξοδο του **pyAscon.py**. Στην συνέχεια επαληθεύονται από τα KATs (Known Answer Tests) [25] τα οποία παρέχονται από την ASCON team.

Το **pyAscon.py** [10] είναι ένα πρόγραμμα γραμμένο σε python, το οποίο μπορεί να κάνει κωδικοποίηση/αποκωδικοποίηση μηνυμάτων χρησιμοποιώντας τον αλγόριθμο ASCON. Σημαντική λεπτομέρεια είναι, ότι πρέπει να γίνει χρήση της τελευταίας έκδοσης (AEAD128), παλαιότερες εκδόσεις (v1.1, v1.2) δεν επεξεργάζονται τα δεδομένα σε little endian και έχουν διαφορετικά διανύσματα αρχικοποίησης (IV), με αποτέλεσμα να παράγουν διαφορετική έξοδο.

Συνεπώς παραμετροποιώντας κατάλληλα το `if __name__ == "__main__":`, είναι δυνατή η εκτέλεση κώδικα κατευθείαν από το script. Ως δεδομένα εισόδου για τον AEAD128 χρησιμοποιούνται τα Key, Nonce, AD, Message και για την Hash256 το Message, όπως παρουσιάζεται στο **pyAscon Data Input.py**. Με την εκτέλεση του προγράμματος παράγονται τα αποτελέσματα της Εικόνας 36 (pyAscon Data Out). Η σύγκριση τους με αυτά της Εικόνας 35 (Vivado Simulation Log), καταδεικνύει την πλήρη ταύτισή τους (Εικόνα 37, pyAscon/Vivado Results Compare).

```
PS E:\temp\Ascon> C:\Users\30694\AppData\Local\Microsoft\WindowsApps\python3.13.exe e:/temp/Ascon/Python/testings/test_1.py
message: 0x05319cf7ffec692672cf2722d8df7f27 (32 bytes)
customization: 0x (0 bytes)
key: 0xf8aa49572e4879e7eca2c7c37a8bd6b3f79ccbf2db31923e273195a525d7684 (64 bytes)
nonce: 0xea9432a05bf1e53c853086621be55680 (32 bytes)
plaintext: 0x05319cf7ffec692672cf2722d8df7f27 (32 bytes)
ass.data: 0x54686973207465787420697320617373 (32 bytes)
ciphertext: 0xbb8311b402ad7127895b94111236bfdc (32 bytes)
tag: 0x5b0e104f3341062956d3243f10cfa21f (32 bytes)
received: 0x05319cf7ffec692672cf2722d8df7f27 (32 bytes)
```

Εικόνα 36: pyAscon Data Out



```
message: 0x05319cf7ffec692672cf2722d8df7f27 (32 bytes)
customization: 0x (0 bytes)
tag: 0xf8aa49572e4879e7eca2c7c37a8bd6b3f79ccbfa2db31923e273195a525d7684 (64 bytes)
key: 0x2e45359eff1340ab04b32a1d09bd7f3b (32 bytes)
nonce: 0xea9432a05bf1e53c853086621be55680 (32 bytes)
plaintext: 0x05319cf7ffec692672cf2722d8df7f27 (32 bytes)
ass.data: 0x54686973207465787420697320617373 (32 bytes)
ciphertext: 0xbb8311b402ad7127895b94111236bfdc (32 bytes)
tag: 0x5b0e104f3341062956d3243f10cfa21f (32 bytes)
received: 0x05319cf7ffec692672cf2722d8df7f27 (32 bytes)
PS E:\temp\Ascon>
```

```
Tcl Console Messages Log x
Time resolution is 1 ps
Note: Ciphertext = BB8311B402AD7127895B94111236BFDC
Time: 230 ns Iteration: 0 Process: /testbench/line__90 File: E:/temp/Ascon/VHDL/DE
Note: Tag = 5B0E104F3341062956D3243F10CFA21F
Time: 360 ns Iteration: 0 Process: /testbench/line__90 File: E:/temp/Ascon/VHDL/DE
Note: Hash 256 = F8AA49572E4879E7ECA2C7C37A8BD6B3F79CCBFA2DB31923E273195A525D7684
Time: 890 ns Iteration: 0 Process: /testbench/ASCONTestBench/line__61 File: E:/temp
```

Εικόνα 37: pyAscon/Vivado Results Compare

Προκειμένου να επαληθευθεί ότι τα παραγόμενα δεδομένα εξόδου του **pyAscon.py** συμμορφώνονται πλήρως με τις προδιαγραφές του προτύπου NIST SP 800-232, χρησιμοποιήθηκαν επιλεγμένα KATs [25]. Τα συγκεκριμένα διανύσματα ελέγχου εισήχθησαν στο **pyAscon Data Input.py** και τα αποτελέσματα που προέκυψαν συγκρίθηκαν με τις αντίστοιχες τιμές. Η πλήρης ταύτιση των αποτελεσμάτων επιβεβαίωσε την ορθότητα των δεδομένων αναφοράς.

5.1.3 Έλεγχος ενδιάμεσων τιμών

Για τη διευκόλυνση της ανάπτυξης και επαλήθευσης της υλοποίησης, κρίθηκε απαραίτητη η δυνατότητα παρακολούθησης της εσωτερικής κατάστασης (state) των 320 bits. Η δυνατότητα αυτή παρέχεται από το περιβάλλον **pyAscon.py** μέσω της συνάρτησης **pyAscon print_debug.py**, η οποία επιτρέπει την εκτύπωση των πέντε λέξεων του state σε οποιοδήποτε σημείο της διαδικασίας. Η προσέγγιση αυτή παρέχει μεγαλύτερη ευελιξία σε σχέση με τους ενσωματωμένους μηχανισμούς αποσφαλμάτωσης του **pyAscon.py**, καθώς δίνει τη δυνατότητα στοχευμένης παρακολούθησης συγκεκριμένων καταστάσεων. Για παράδειγμα, κατά τη διαδικασία αρχικοποίησης είναι δυνατή η τοποθέτηση της **print_debug**, πριν και μετά την κλήση της συνάρτησης **permutation** εντός της **ascon_initialize** (παρουσιάζεται στο **pyAscon State Check.py**), επιτρέποντας την άμεση παρατήρηση των μεταβολών του state. Με τον τρόπο αυτό καθίσταται δυνατή η λεπτομερής σύγκριση των ενδιάμεσων τιμών της υλοποίησης στο Vivado, διευκολύνοντας τον εντοπισμό και τη διόρθωση πιθανών αποκλίσεων κατά την ανάπτυξη του συστήματος.



Το χαρακτηριστικό παράδειγμα εξόδου του **pyAscon State Check.py** παρουσιάζεται στην Εικόνα 38. Στην συνέχεια αξιοποιώντας το περιβάλλον προσομοίωσης του Vivado, γίνεται εφικτή η άμεση σύγκριση των καταστάσεων και αρκετά ευκολότερη η αποσφαλμάτωση (αν αυτή χρειαστεί).

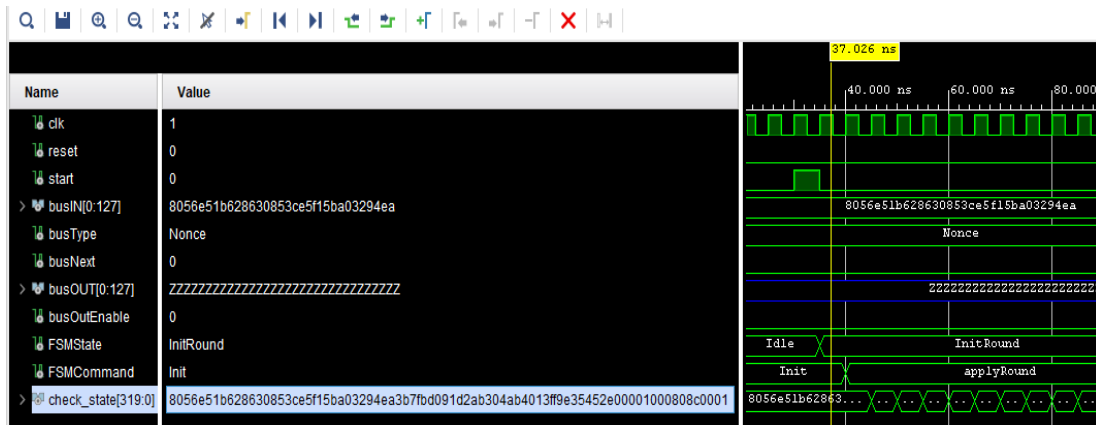
Η Εικόνα 39 παρουσιάζει την κατάσταση του state στο Vivado πριν τη φάση αρχικοποίησης του συστήματος. Ενώ η Εικόνα 40 παρουσιάζει την κατάσταση του state μετά τη φάση της αρχικοποίησης, και ο Πίνακας 18 κάνει μια σύνοψη των δεδομένων.

Επιπλέον, θα πρέπει να σημειωθεί ότι το **pyAscon.py** εμφανίζει τα δεδομένα εισόδου/εξόδου ως ακολουθίες μήκους 32 bytes (Εικόνα 38). Ωστόσο, η συγκεκριμένη ένδειξη αναφέρεται στο μέγεθος της αναπαράστασης των δεδομένων σε επίπεδο λογισμικού και όχι στο πραγματικό μήκος των επιμέρους πεδίων του αλγορίθμου. Για παράδειγμα, στην παρούσα υλοποίηση τα πεδία έχουν μήκος 128 bits (16 bytes).

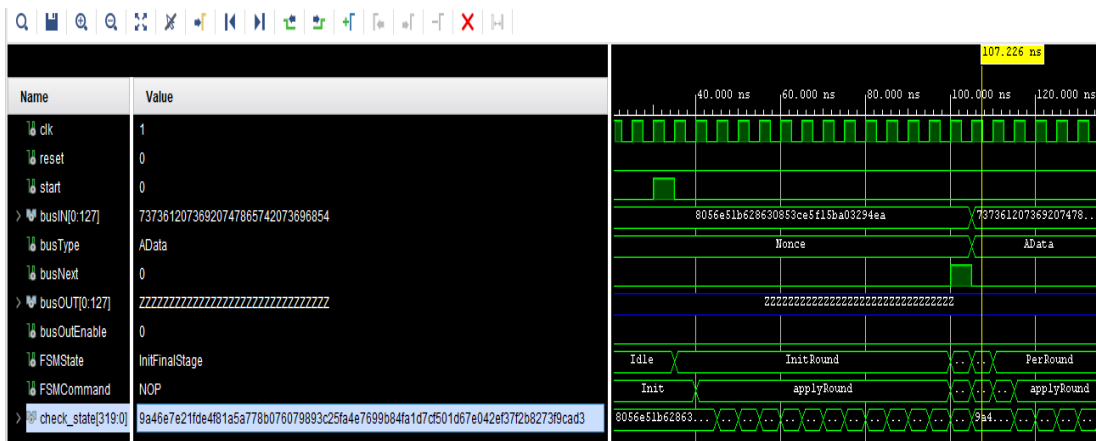
Σαν τελευταίο, ο καταχωρητής check_state των 320 bits στο Vivado είναι καθαρά για λόγους πληροφόρησης και αποσφαλμάτωσης, συνεπώς κατά την σύνθεση πρέπει να απενεργοποιηθεί. Αν αυτό δεν συμβεί τότε θα αποτύχει η σύνθεση λόγω των περιορισμένων πόρων σε I/O pins που διαθέτει η συσκευή της υλοποίησης.

```
-----  
Before Initialize  
Initialize  
S0=00001000808c0001  
S1=ab4013ff9e35452e  
S2=3b7fbd091d2ab304  
S3=3ce5f15ba03294ea  
S4=8056e51b62863085  
-----  
After Initialize  
Initialize  
S0=f37f2b8273f9cad3  
S1=1d7cf501d67e042e  
S2=c25fa4e7699b84fa  
S3=a5a778b076079893  
S4=9a46e7e21fde4f81  
-----  
key:          0x2e45359eff1340ab04b32a1d09bd7f3b (32 bytes)  
nonce:        0xea9432a05bf1e53c853086621be55680 (32 bytes)  
plaintext:    0x05319cf7ffec692672cf2722d8df7f27 (32 bytes)  
ass.data:     0x54686973207465787420697320617373 (32 bytes)  
ciphertext:   0xbb8311b402ad7127895b94111236bfdc (32 bytes)  
tag:          0x5b0e104f3341062956d3243f10cfa21f (32 bytes)  
received:     0x05319cf7ffec692672cf2722d8df7f27 (32 bytes)  
PS E:\temp\Ascon>
```

Εικόνα 38: pyAscon print debug example



Εικόνα 39: Vivado Waveform Before Init



Εικόνα 40: Vivado Waveform After Init

Πίνακας 18: Compare pyAscon/Vivado States

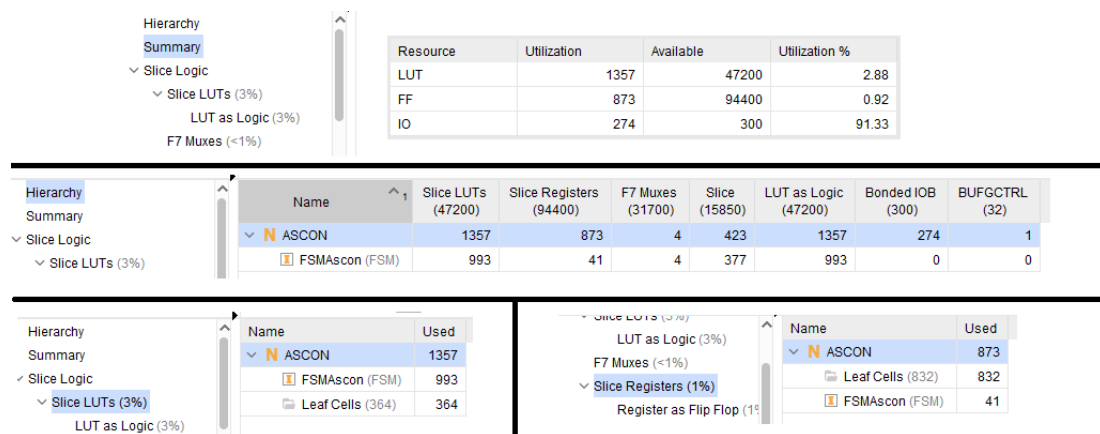
	pyAscon	Vivado
Before	S4=8056e51b62863085	8056e51b62863085
Before	S3=3ce5f15ba03294ea	3ce5f15ba03294ea
Before	S2=3b7fbd091d2ab304	3b7fbd091d2ab304
Before	S1=ab4013ff9e35452e	ab4013ff9e35452e
Before	S0=00001000808c0001	00001000808c0001
After	S4=9a46e7e21fde4f81	9a46e7e21fde4f81
After	S3=a5a778b076079893	a5a778b076079893
After	S2=c25fa4e7699b84fa	c25fa4e7699b84fa
After	S1=1d7cf501d67e042e	1d7cf501d67e042e
After	S0=f37f2b8273f9cad3	f37f2b8273f9cad3



5.2 Μετρήσεις

5.2.1 Επιφάνεια υλοποίησης συστήματος

Στην Εικόνα 41 παρουσιάζεται η συνολική επιφάνεια υλοποίησης των πόρων της συσκευής xc7a75tfgg676-1. Ειδικότερα, απεικονίζονται οι πόροι τύπου CLB (Configurable Logic Blocks), οι οποίοι αποτελούνται από ένα ή περισσότερα slices. Κάθε slice περιλαμβάνει επιμέρους δομικά στοιχεία, όπως LUTs, MUXs, FFs και άλλους λογικούς πόρους. Ως εκ τούτου, η χρήση ενός slice δεν συνεπάγεται απαραίτητα την αξιοποίηση όλων των διαθέσιμων στοιχείων που αυτό περιέχει, καθώς μέρος αυτών μπορεί να παραμείνει ανεκμετάλλετο. Στην Εικόνα 42 παρουσιάζονται οι βασικοί πόροι που χρησιμοποιήθηκαν για την υλοποίηση. Για την αξιολόγηση της απαιτούμενης επιφάνειας υλοποίησης, λαμβάνεται υπόψιν ο αριθμός των πόρων από την αναφορά της Εικόνας 41, και ανέρχεται σε 1357 LUTs.



Εικόνα 41: Σύνοψη implementation συστήματος

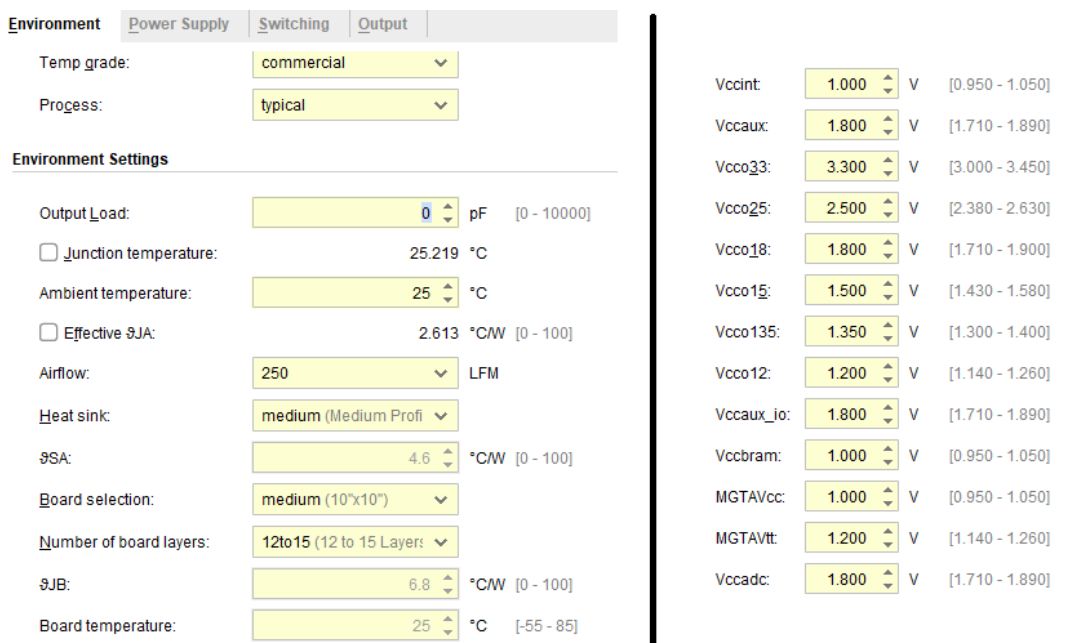
Ref Name	Used	Functional Category
FDRE	873	Flop & Latch
LUT5	548	LUT
LUT6	534	LUT
LUT4	237	LUT
LUT3	233	LUT
IBUF	134	IO
LUT2	132	LUT
OBUFT	128	IO
OBUF	12	IO
MUXF7	4	MuxFx
BUFG	1	Clock

Εικόνα 42: Σύνοψη πρωτευόντων δομικών στοιχείων υλοποίησης

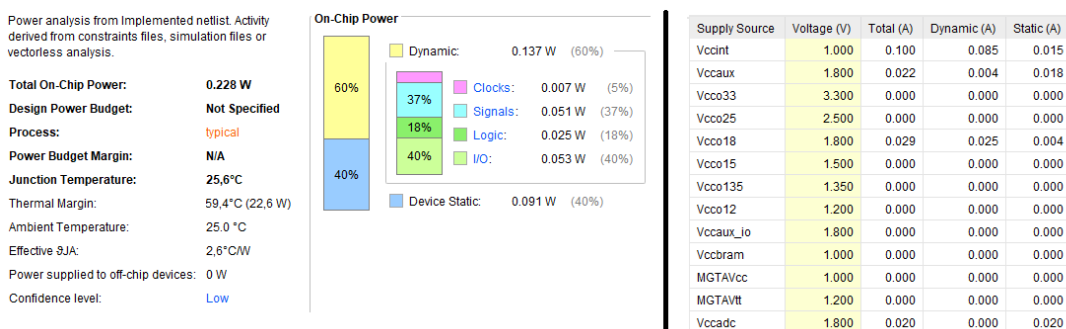
5.2.2 Κατανάλωση ενέργειας συστήματος

Θέτοντας τις συνθήκες λειτουργίας και τις τάσεις τροφοδοσίας που παρουσιάζονται στην Εικόνα 43 (Περιβάλλον εργασίας του Artix7), η εκτιμώμενη συνολική κατανάλωση ισχύος ανέρχεται στα 228 mW (Εικόνα 44, Αναφορά κατανάλωσης συστήματος στα 200MHz). Από αυτήν τα 91 mW είναι στατική, ενώ τα υπόλοιπα 137 mW είναι η δυναμική κατανάλωση του συστήματος.

Σημειώνεται επίσης ότι οι παραπάνω μετρήσεις αφορούν την λειτουργία του συστήματος με συχνότητα ρολογιού στα 200MHz. Επιπροσθέτως υπάρχει η αναφορά της Εικόνας 45, η οποία παρουσιάζει την κατανομή της κατανάλωσης ανά ακροδέκτη εισόδου/εξόδου (για πιο πλήρη ενημέρωση, γίνεται unfold το bus προς διερεύνηση στο περιβάλλον Vivado).



Εικόνα 43: Περιβάλλον εργασίας του Artix7



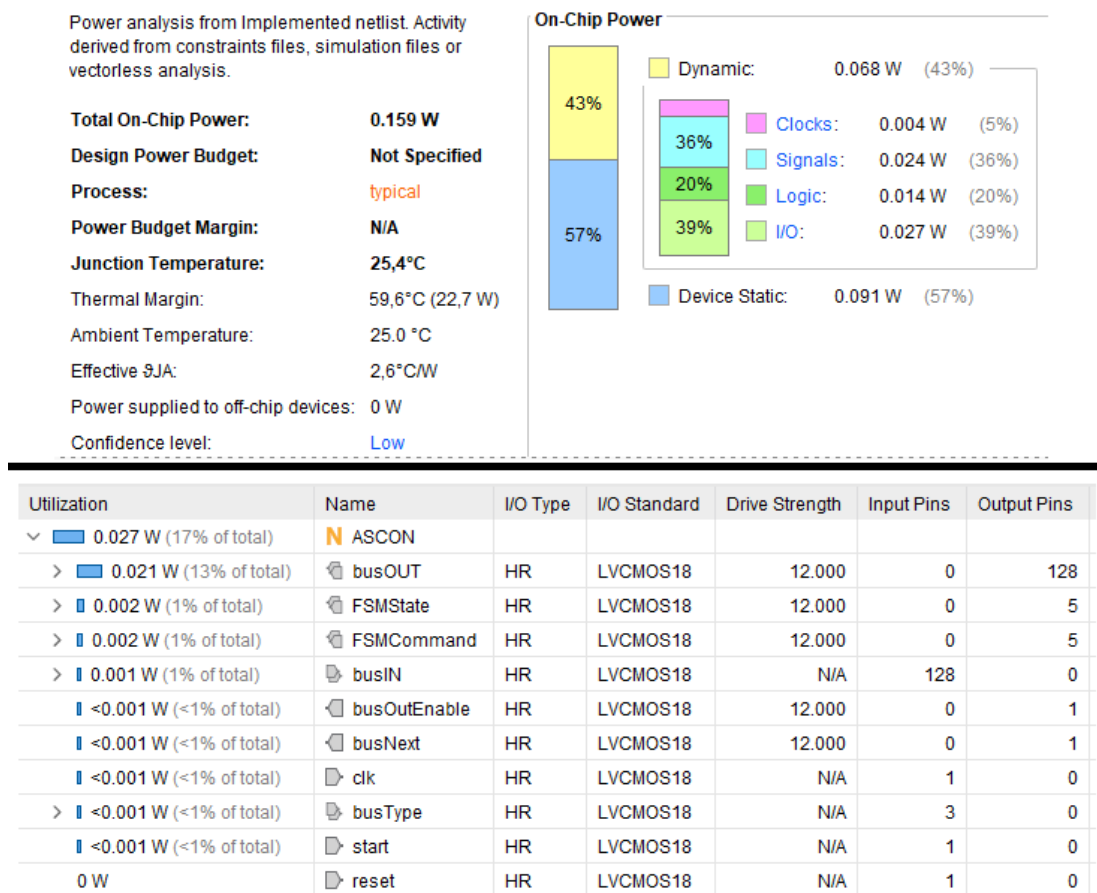
Εικόνα 44: Αναφορά κατανάλωσης συστήματος στα 200MHz



Utilization	Name	I/O Type	I/O Standard	Drive Strength	Input Pins	Output Pins
0.053 W (23% of total)	ASCON					
> 0.042 W (18% of total)	busOUT	HR	LVC MOS18	12.000	0	128
> 0.005 W (2% of total)	FSMState	HR	LVC MOS18	12.000	0	5
> 0.004 W (2% of total)	FSMCommand	HR	LVC MOS18	12.000	0	5
> 0.001 W (1% of total)	busIN	HR	LVC MOS18	N/A	128	0
0.001 W (<1% of total)	busOutEnable	HR	LVC MOS18	12.000	0	1
0.001 W (<1% of total)	busNext	HR	LVC MOS18	12.000	0	1
<0.001 W (<1% of total)	clk	HR	LVC MOS18	N/A	1	0
> <0.001 W (<1% of total)	busType	HR	LVC MOS18	N/A	3	0
<0.001 W (<1% of total)	start	HR	LVC MOS18	N/A	1	0
0 W	reset	HR	LVC MOS18	N/A	1	0

Εικόνα 45: Κατανάλωση I/O blocks στα 200MHz

Αν για κάποιο λόγο υπάρξουν απαιτήσεις ακόμη μικρότερης κατανάλωσης ισχύος, αυτό μπορεί να επιτευχθεί μειώνοντας την συχνότητα ρολογιού. Η ταχύτητα του συστήματος θα μειωθεί στο μισό, αλλά παράλληλα θα υπάρξει και μια πτώση της δυναμικής ισχύος σε παρόμοιο ποσοστό. Στην Εικόνα 46 (Συνολική αναφορά κατανάλωσης στα 100MHz) παρουσιάζεται και αυτή σαν δεδομένο για την απόκτηση μια ποιο εμπεριστατωμένης άποψης των διαφορετικών λειτουργιών. Εκεί παρατηρείται ότι η συνολική κατανάλωση πέφτει στα 159 mW, κάτι φυσιολογικό γιατί η στατική κατανάλωση παραμένει στα ίδια επίπεδα. Συνεπώς φαίνεται ότι η βελτιστοποίηση προέρχεται αποκλειστικά από την δυναμική (από 137 mW πέφτει στα 68 mW, που προσθέτοντας τα 91 mW της στατικής προκύπτουν 159 mW).



Εικόνα 46: Συνολική αναφορά κατανάλωσης στα 100MHz

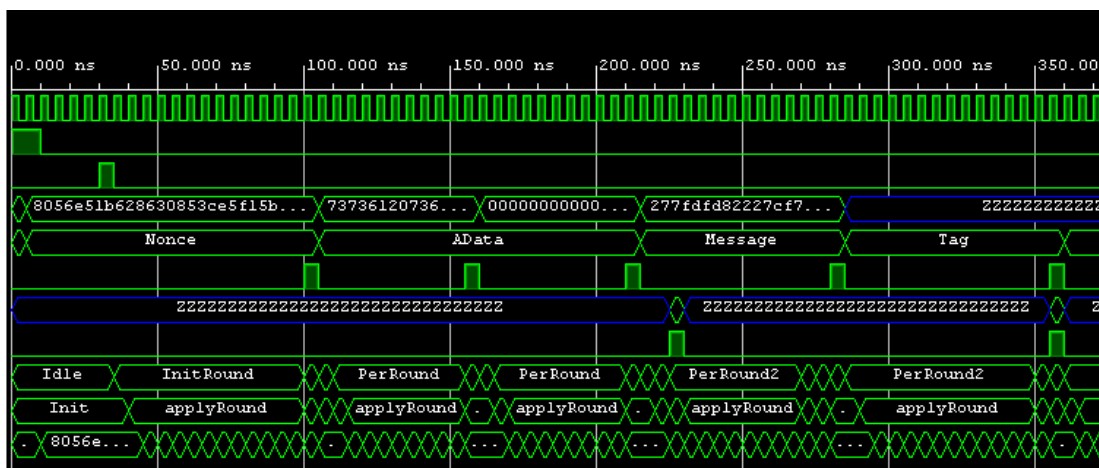
5.2.3 Throughput

Για τον υπολογισμό του throughput της υλοποίησης χρειάζονται κάποια δεδομένα:

1. **To rate του AEAD128:** Αναφέρεται ως 128 bits.
2. **Η συχνότητα χρονισμού του συστήματος:** Υλοποιείται στα 200 MHz.
3. **Πλήθος clock του συστήματος ανά block μηνύματος:** Όχι άμεσα προσδιορίσιμο.

Η τρίτη παράμετρος δεν μπορεί να προσδιοριστεί άμεσα, παρόλο που είναι γνωστό ότι υλοποιείται μια permutation ανά clock, και ότι ένα block μηνύματος χρειάζεται 8 permutations για να επεξεργαστεί. Επιπροσθέτως υπάρχουν clock ελέγχου και μετάβασης τιμών, οπότε για την ορθότητα του αποτελέσματος θα αναλυθεί το waveform του Vivado simulation.

Στην Εικόνα 47 παρουσιάζεται το συνολικό χρονικό διάγραμμα λειτουργίας του ASCON AEAD128. Για τον υπολογισμό της απόδοσης της υλοποίησης, η ανάλυση επικεντρώνεται στη φάση επεξεργασίας του μηνύματος (busType = Message). Η συγκεκριμένη επιλογή πραγματοποιείται επειδή τα blocks του μηνύματος εισέρχονται διαδοχικά στο σύστημα και αποτελούν το κύριο τμήμα της κρυπτογραφικής επεξεργασίας, ενώ η διαδικασία ολοκληρώνεται με τον υπολογισμό του authentication tag.



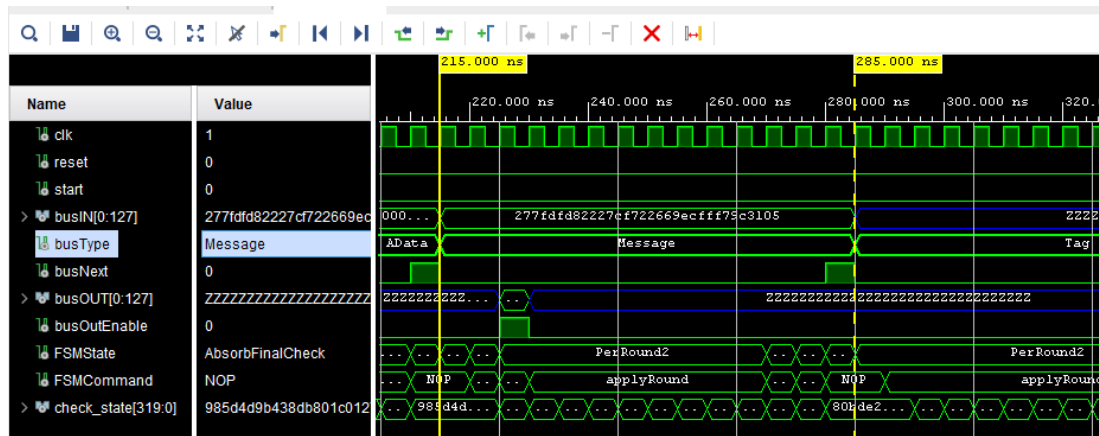
Εικόνα 47: Vivado Simulation Waveform

Στην Εικόνα 48 παρουσιάζεται ο χρόνος που χρειάζεται η επεξεργασία ενός block μηνύματος. Αυτός ανέρχεται στα $285-215 = 70$ nS ή 14 κύποι ρολογιού (clock). Στην συνέχεια υπολογίζεται το throughput χρησιμοποιώντας το clock ως:

$$throughput = \frac{rate}{number\ of\ clocks} \cdot clock\ freq = \frac{128\ bits}{14} \cdot 200 \cdot 10^6 c/s = 1828Mbps$$

ή διαφορετικά χρησιμοποιώντας τον χρόνο ως:

$$\begin{matrix} 128bits & \rightarrow & 70 \cdot 10^{-9}sec \\ x\ bits & \rightarrow & 1\ sec \end{matrix} \Rightarrow x = \frac{128bits}{70} \cdot 10^9 = 1,828Gbps$$



Εικόνα 48: Message Process Time

5.3 Σύγκριση αποτελεσμάτων

Για την αξιολόγηση της προτεινόμενης υλοποίησης πραγματοποιείται συγκριτική ανάλυση με αντίστοιχες υλοποιήσεις που αναφέρονται στη διεθνή βιβλιογραφία. Σκοπός της διαδικασίας είναι η αποτίμηση της αποδοτικότητας του συστήματος ως προς βασικές μετρικές απόδοσης και χρήσης πόρων. Ειδικότερα, εξετάζονται χαρακτηριστικά όπως η μέγιστη συχνότητα λειτουργίας, ο ρυθμός μετάδοσης δεδομένων, η απαιτούμενη επιφάνεια υλοποίησης και η κατανάλωση ισχύος. Μέσω της σύγκρισης αυτών των μεγεθών καθίσταται δυνατή η αξιολόγηση των πλεονεκτημάτων και των περιορισμών της προτεινόμενης αρχιτεκτονικής, σε σχέση με άλλες υλοποιήσεις του ASCON σε FPGA με παρόμοια χαρακτηριστικά.

Πίνακας 19: Πόροι συστήματος υλοποίησης (Current)

Implement- tation	Device	Clock freq (Mhz)	LUTs	Through- put (Mbps)	TP/area	Power (mW)	Slices	FFs
AEAD128	Artix-7	200	1357	1828	1,347	228	423	873
AEAD128	Artix-7	100	1357	914	0,674	159	423	873

Πίνακας 20: Πόροι υλοποίησης της βιβλιογραφικής αναφοράς [8]

Tech/gy	Freq. (MHz)	Thr/put (Mbps)	Area (LUTs)	Power (mWatt)	Comments
Zynq- 7000	10	106.67	1312	2.16	-
Artix 7	191	1528 / 873.1	1465	113 (at 75 MHz)	128/hash - iterative (Graz-v1)



Artix 7	213	2272 / 973.7	1541	121 (at 75 MHz)	128a/hash - iterative (Graz-v2)
Artix 7	201	2572.8 / 1608	2142	390 (at 75 MHz)	128a/hash - 2x unrolled (Graz-v3)
Artix 7	206	3296 / 1648	2249	484 (at 75 MHz)	128a/hash - 2x unrolled (Graz-v4)
Artix 7	150	2400 / 1600	2797	2,526 (75 MHz)	128a/hash - 3x unrolled (Graz-v5)
Artix 7	246	6297.6	2410	427 (at 75 MHz)	128a - 2x unrolled (GMU-v1)
Artix 7	307	4366.2	1790	128 (at 75 MHz)	128a - iterative (GMU-v2)
Artix 7	276	2533.4 / 1358.8	1375	123 (at 75 MHz)	128/hash - iterative (GMU2-v1)
Artix 7	234	3744 / 2139.4	2126	447 (at 75 MHz)	128/hash - 2x unrolled (GMU2-v2)
Artix 7	142	3029.3 / 1817.6	2493	2,605 (75 MHz)	128/hash - 3x unrolled (GMU2-v3)
Artix 7	233	1491.2	1913	100 (at 75 MHz)	128 - iterative (VT-v1)
Artix 7	219	1475.4 / 934.4	1928	104 (at 75 MHz)	128/hash - iterative (VT-v2)
Virtex 6	N/A	5085	1270	-	-
Artix 7	N/A	644.2	1790	265 (at 75 MHz)	128a - iterative - 16 Byte message
Kintex 7	303	N/A	1055	-	128
Kintex 7	312	N/A	1107	-	128a
Kintex 7	429	N/A	629	-	hash
Artix 7	107	457	1330	31	128
Virtex 7	20.9	1342.31	22636	-	128, Full unroll
Virtex 7	18.9	2419.88	25187	-	128a, Full unroll
Virtex 7	270.37	721.53	2708	-	128, 2-round recursive
Virtex 7	275.63	1357.08	2916	-	128a, 2-round recursive
Virtex 4	12.8	817.41	26943	-	128, Full unroll
Virtex 4	11.7	1496.25	30006	-	128a, Full unroll
Virtex 4	189.86	506.29	4021	-	128, 2-round recursive
Virtex 4	197.08	970.25	4215	-	128a, 2-round recursive
Artix-7	244	N/A	1548	-	RECO-HCON
Artix-7	271	1927	3144	-	Coprocessor, 128a
Artix-7	271	1875	3144	-	Coprocessor, hasha
Zynq-7000	200	N/A	7726	1373	-
Spartan 6	216	60.1	684	-	128
Spartan 6	216	119.16	684	-	128a



Spartan 6	208.5	1906.3	1402	-	128
Spartan 6	202.8	2884.3	1712	-	128a
Artix 7	260.75	668.4	770	-	hash - r / p = 1
Artix 7	64.07	792.56	3728	-	hash - r / p = 6
Kintex 7	268.24	646.46	702	-	hash - r / p = 1
Kintex 7	100.36	1354.49	3728	-	hash - r / p = 6
Virtex 7	272.03	697.54	773	-	hash - r / p = 1
Virtex 7	224.56	1108.22	1290	-	hash - r / p = 2
Virtex 7	162.68	1488.37	2550	-	hash - r / p = 4
Virtex 7	108.45	1391.3	3855	-	hash - r / p = 6
Spartan 6	146.1	114.0	1640	9.2	128 - Small - Unprotected
Spartan 6	148.4	237.4	1725	10.6	128 - Large - Unprotected
Spartan 6	117.0	91.4	3278	15.9	128 - Small - Protected
Spartan 6	119.8	191.7	3673	22.8	128 - Large - Protected

Πίνακας 21: Πόροι υλοποίησης της βιβλιογραφικής αναφοράς [26]

Implementation	Device	LUT	Maximum Frequency (MHz)	TP (Mbps)	TP/Area (Mbps/LUT)
This work	Virtex-7	6536	104	13312	2.036
[12]	Spartan-6	684	216	119.16	0.17
[6]	Spartan-6	1725	148.4	237.4	0.137
[14]	Spartan-6	1712	202.8	2884.3	1.684
[4]	Artix-7	1898	263	1683.2	0.887
[15]	Spartan-6	4072	483.35	6874.5	1.688
[17]	Virtex-7	3350	226.4	14512	4.33
[29]	Virtex-7	3678	132.1	8454	2.30
[30]	Kintex 7	n/a	104	6709	n/a
[31]	Artix-7	1548	244	n/a	n/a
[32]	Virtex-7	25187	n/a	2419	0.096



Πίνακας 22: Πόροι υλοποίησης της βιβλιογραφικής αναφοράς [27]

ASCON Architecture	FPGA Hardware Platform	Area (LUTs)	Power (mW)	Throughput (Mbps)	Efficiency (Mbps/LUT)
ASCON (original) [53]	Spartan 7	371 (slices)	99	6.646	17.914
ASCON (original) [53]	Kintex 7	376 (slices)	88	6.709	17.843
ASCON (unprotected) [58]	Spartan 6	2048	11.5	255.4	0.1247
ASCON [26]	Artix7	1808	26.8	39.0	-
ASCON-128 [27]	Artix7	1330	31	457	0.343
RECO-HCON (128) [29]	Artix7	1548	-	5926	-
RECO-HCON (128a) [29]	Artix7	1548	-	9077	-
RECO-HCON (hash) [29]	Artix7	1548	-	3160	-
RECO-HCON (hash-a) [29]	Artix7	1548	-	4534	-
ASCON (protected) [58]	Spartan 6	6364	37.5	134.6	0.0212
ASCON-128 (Logic One-bit) [53]	Spartan 7	373 (slices)	99	6705	17.783
ASCON-128 (Logic Interleaved-bit) [53]	Spartan 7	380 (slices)	99	6687	17.444
ASCON-128 (Logic CRC-3) [53]	Spartan 7	407 (slices)	99	6603	15.601
ASCON-128 (LUT One-bit) [53]	Spartan 7	372 (slices)	100	6448	17.333
ASCON-128 (LUT Interleaved-bit) [53]	Spartan 7	377 (slices)	100	6443	17.090
ASCON-128 (LUT CRC-3) [53]	Spartan 7	425 (slices)	100	6431	15.131
ASCON-128 (Logic One-bit) [53]	Kintex 7	381 (slices)	89	6705	17.598
ASCON-128 (Logic Interleaved-bit) [53]	Kintex 7	384 (slices)	89	6687	17.414
ASCON-128 (Logic CRC-3) [53]	Kintex 7	385 (slices)	89	6603	17.150
ASCON-128 (LUT One-bit) [53]	Kintex 7	363 (slices)	89	6776	18.667
ASCON-128 (LUT Interleaved-bit) [53]	Kintex 7	372 (slices)	89	6409	17.228
ASCON-128 (LUT CRC-3) [53]	Kintex 7	384 (slices)	89	6383	16.224
ASCON-128 (unrolled) [30]	Virtex4	26943	-	817.41	0.031
ASCON-128 (recursive) [30]	Virtex4	4021	-	506.29	0.125
ASCON-128 (unrolled) [30]	Virtex7	22636	-	1342.31	0.059
ASCON-128 (recursive) [30]	Virtex7	2708	-	721.53	0.266
ASCON-128 (unrolled) [30]	Spartan6	22636	-	688.83	0.031
ASCON-128 (recursive) [30]	Spartan6	2781	-	346.50	0.124
ASCON-128a (unrolled) [30]	Virtex4	30006	-	1496.25	0.049
ASCON-128a (recursive) [30]	Virtex4	4215	-	970.25	0.231
ASCON-128a (unrolled) [30]	Virtex7	25187	-	2419.88	0.096
ASCON-128a (recursive) [30]	Virtex7	2916	-	1357.08	0.465
ASCON-128a (unrolled) [30]	Spartan6	25187	-	1247.22	0.049
ASCON-128a (recursive) [30]	Spartan6	2918	-	638.44	0.218
Fault-injected ASCON [49]	SASEBO-GII	217	7.8	198	-
Key-bypass HT on ASCON [33]	SoC Cyclone V	827	22.4	-	-
Round-reduction HT on ASCON [33]	SoC Cyclone V	771	22.3	-	-

Πίνακας 23: Πόροι υλοποίησης της βιβλιογραφικής αναφοράς [28]

Device	Ascon Variant	Slices	LUTs	FFs	Clock Cycles	Frequency (F in MHz)	Time (T in μ s)	Throughput (TP in Mbps)	Power (mW)
Virtex-7 xc7vx690tffg1930-3	Ascon-128 Ascon-128a	348	1632	912	55 47	335	0.16 0.14	400 914	239
Kintex-7 xc7k480tffg1156-3	Ascon-128 Ascon-128a	265	1497	912	55 47	331	0.16 0.14	400 914	236
Spartan-7 xc7k480tffg1156-3	Ascon-128 Ascon-128a	330	1804	912	55 47	309	0.18 0.15	355 853	219
Artix-7 xc7a100tcs324-3	Ascon-128 Ascon-128a	303	1756	912	55 47	317	0.17 0.15	376 853	222

Πίνακας 24: Πόροι υλοποίησης της βιβλιογραφικής αναφοράς [29]

Ref	FPGA	LUTs	Freq. (MHz)	TP (Mbps)	TP/A (Mbps/LUT)
[14]	Spartan-6	0.68 k	216.0	60.1	0.260
[15]	Spartan-6	1.64 k	146.1	114.0	0.070
[16]	Spartan-6	1.91 k	174.4	1116.4	0.584
[18]	Spartan-6	2.06 k	206.2	315.2	0.153
[19]	Virtex-7	2.70 k	270.3	721.5	0.125

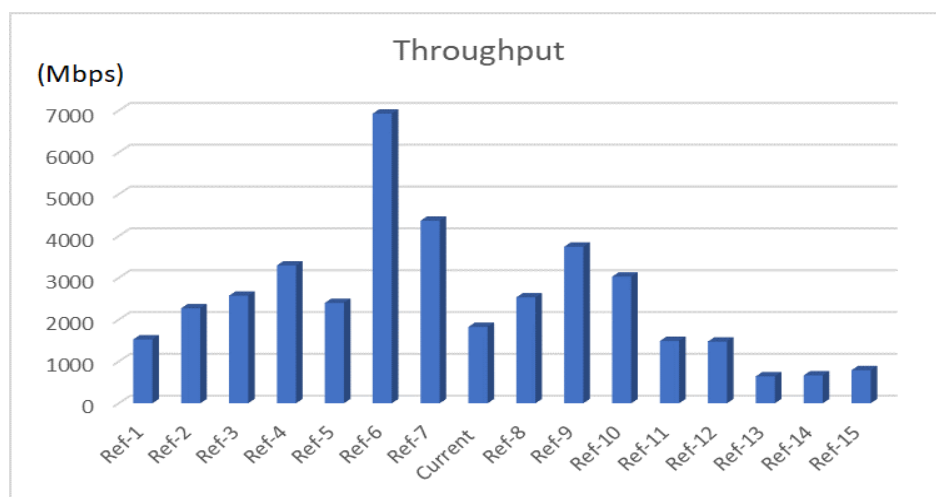
Στον Πίνακα 19 υπάρχει μια σύνοψη των μετρήσεων του παρόντος συστήματος, η οποία εξάχθηκε από την [Ενότητα 5.2](#) (Μετρήσεις). Επομένως τώρα γίνεται εφικτή και η αξιολόγηση μέσω στοιχείων της εργασίας που πραγματοποιήθηκε και παρουσιάζεται στην συνέχεια.

Στις Εικόνες 49, 50, 51, 52 αξιολογείται η παρούσα υλοποίηση (Current), με παρόμοια συστήματα (Artix-7) της βιβλιογραφικής αναφοράς [8] του Πίνακα 20.

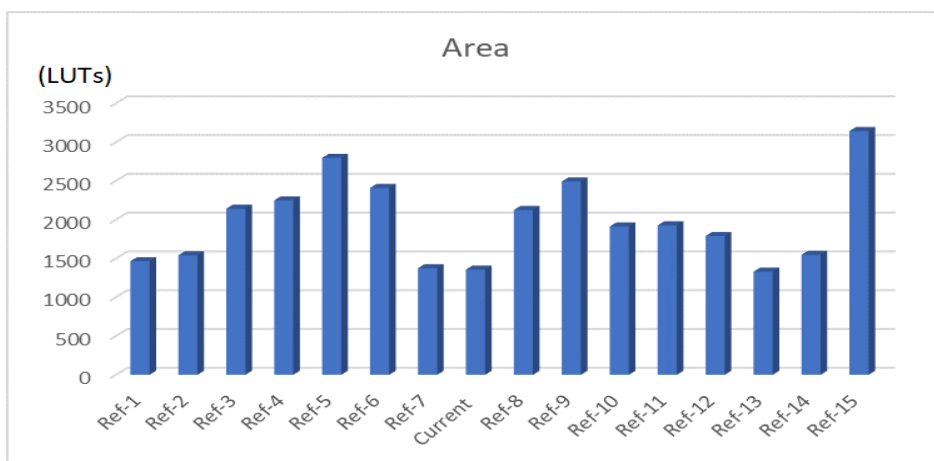
Στους Εικόνες 53, 54, 55 αξιολογείται η παρούσα υλοποίηση (Current), με παρόμοια συστήματα (Artix-7) των βιβλιογραφικών αναφορών [26] [27] [28], που παρουσιάζονται στους Πίνακες 21, 22, 23.

Σημείωση 1: Σε κάποιες βιβλιογραφικές αναφορές δεν δίνονται όλα τα στοιχεία αξιολογήσεων, στις Εικόνες 49 με 55 τα αντίστοιχα πεδία παραμένουν κενά. Επίσης στην Εικόνα 24 της βιβλιογραφικής αναφοράς [29] δεν υπάρχουν συστήματα Artix-7, παρόλα αυτά συμπεριλήφθηκε για την επίτευξη μιας πιο γενικευμένης αναφοράς (δεν αναφέρονται στοιχεία του στις εικόνες συγκρίσεων 49 με 55).

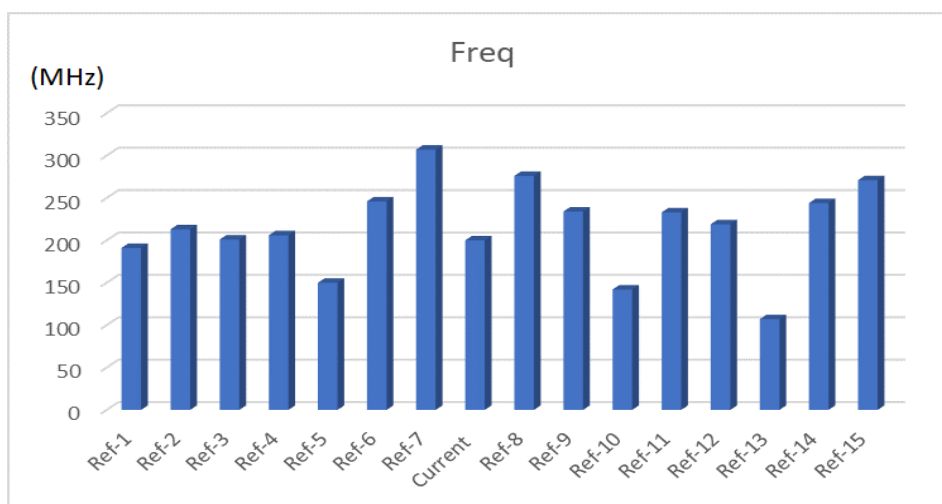
Σημείωση 2: Στην σύγκριση της κατανάλωσης ισχύος (Εικόνα 52) τα δεδομένα της βιβλιογραφίας (Πίνακας 20) είναι με συχνότητα χρονισμού στα 75 MHz, οπότε παρατέθηκε η κατανάλωση των 100 MHz του παρόντος συστήματος.



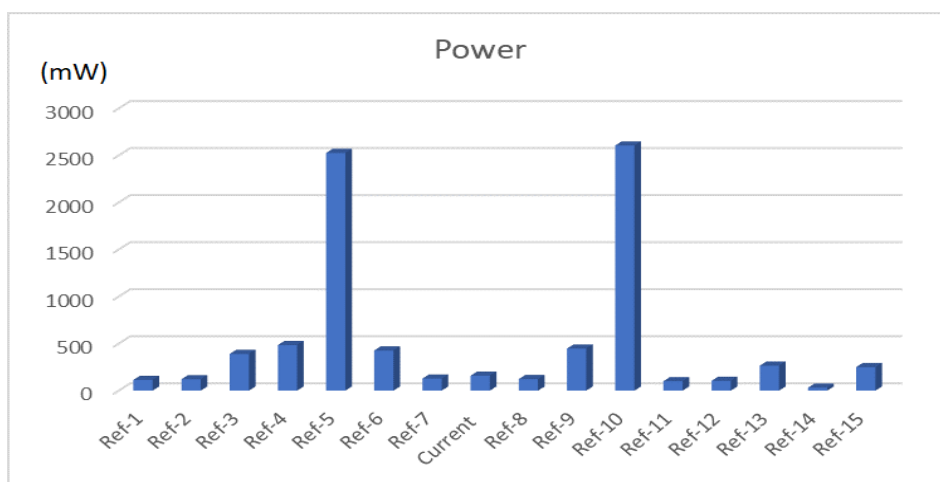
Εικόνα 49: Σύγκριση Throughput του σχεδιασμού με παρόμοιες εργασίες



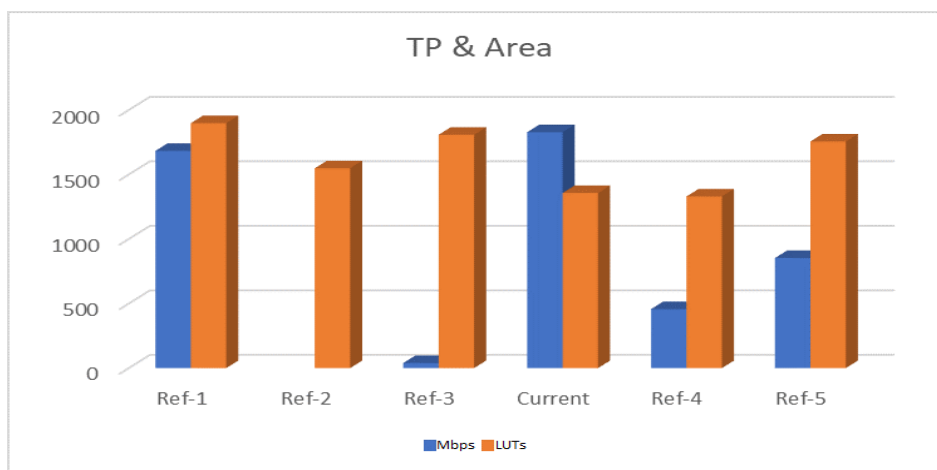
Εικόνα 50: Σύγκριση επιφάνειας υλοποίησης του σχεδιασμού με παρόμοιες εργασίες



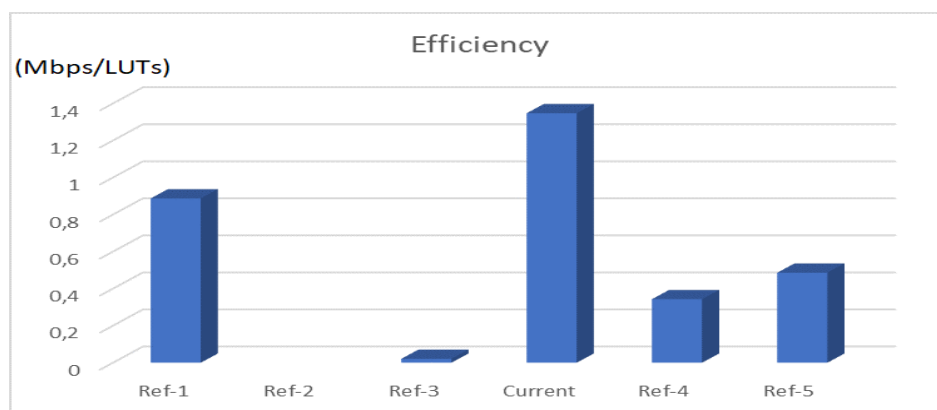
Εικόνα 51: Σύγκριση συχνότητας ρολογιού του σχεδιασμού με παρόμοιες εργασίες



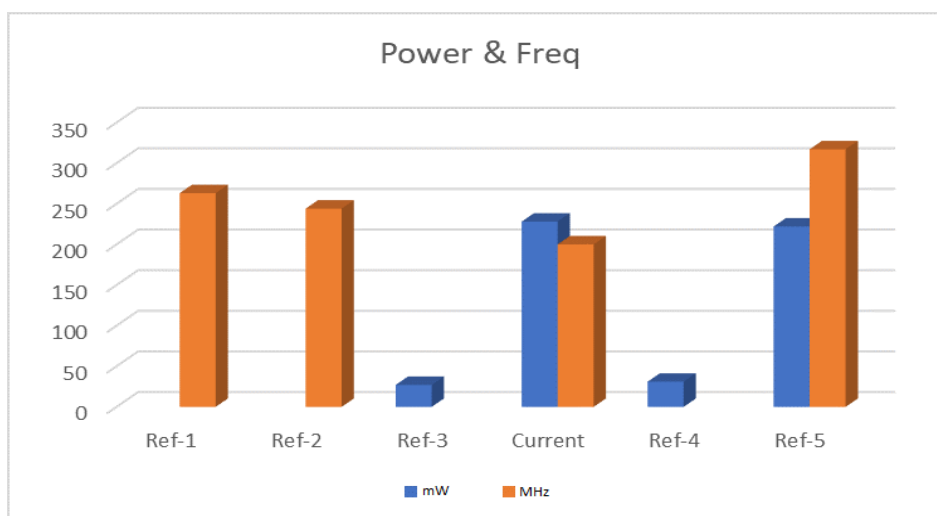
Εικόνα 52: Σύγκριση κατανάλωσης ισχύος του σχεδιασμού με παρόμοιες εργασίες



Εικόνα 53: Σύγκριση TP & Area του σχεδιασμού με παρόμοιες εργασίες



Εικόνα 54: Σύγκριση Mbps/LUTs του σχεδιασμού με παρόμοιες εργασίες



Εικόνα 55: Σύγκριση Κατανάλωσης & Συχνότητας του σχεδιασμού με παρόμοιες εργασίες

6. Συμπεράσματα

Στην παρούσα διπλωματική εργασία υλοποιήθηκε ένα σύστημα κρυπτογράφησης και αποκρυπτογράφησης βασισμένο στον αλγόριθμο ASCON AEAD128, αξιοποιώντας σύγχρονες τεχνικές σχεδίασης ψηφιακών συστημάτων. Για την ανάπτυξη και αξιολόγηση της υλοποίησης χρησιμοποιήθηκε το περιβάλλον Vivado, η γλώσσα περιγραφής υλικού VHDL και σαν ενσωματωμένο σύστημα ένα FPGA της σειράς Artix-7 [23]. Η VHDL επιλέχθηκε καθώς σαν HDL παρέχει υψηλό βαθμό ελέγχου και απόλυτη προσαρμοστικότητα σε επίπεδο υλικού επιτρέποντας την ανάπτυξη, υλοποίηση και παραμετροποίηση αποδοτικών υλοποιήσεων. Ως κρυπτογραφικός αλγόριθμος επιλέχθηκε ο ASCON AEAD128 ο οποίος προτυποποιείται μέσω του εγγράφου NIST SP 800-232 [7] και αποτελεί πλέον πρότυπο για την Lightweight Cryptography (LWC). Επιπροσθέτως υλοποιήθηκε και η συνάρτηση κατακερματισμού ASCON Hash256, που συνέβαλε στην καλύτερη κατανόηση της αρχιτεκτονικής του συνολικού αλγορίθμου, αναδεικνύοντας τον τρόπο που επιτυγχάνει υψηλή αποδοτικότητα μέσω της επαναχρησιμοποίησης κοινών δομικών μονάδων.

Η υλοποίηση βασίστηκε σε αρχιτεκτονική μιας μετάθεσης ανά κύκλο ρολογιού, που σαν αποτέλεσμα είχε την χαμηλή πολυπλοκότητα, μειωμένες απαιτήσεις σε επιφάνεια υλοποίησης και μικρή κατανάλωση ισχύος. Αν και αρχικά υπήρχε η πεποίθηση ότι η συγκεκριμένη επιλογή θα οδηγούσε σε μικρό throughput σε σχέση με άλλες δημοσιευμένες υλοποιήσεις, η συγκριτική αξιολόγηση των εργασιών [8] [26] [27] [28] και [29] έδειξε ότι είναι κοντά στον μέσο όρο. Παράλληλα όμως εμφάνισε και ικανοποιητικά αποτελέσματα τα οποία είναι πάνω από τον μέσο όρο. Αυτά είναι η σχέση throughput προς επιφάνειας υλοποίησης και η μικρή κατανάλωση ισχύος. Σημαντικό πλεονέκτημα επίσης είναι και η απλή αρχιτεκτονική που διαθέτει όπως και η modular σχεδίαση, καθώς διευκολύνει την κατανόηση, την συντήρηση, και κάνει ευκολότερη την ενσωμάτωση τυχόν μελλοντικών επεκτάσεων.

Ως μελλοντική εργασία προτείνεται η προσθήκη μηχανισμών padding και parsing, ώστε να γίνει δυνατή η αυτόνομη λειτουργία του. Επιπλέον θα μπορούσε να μειωθεί το μήκος των διαδρόμων εισόδου/εξόδου (busIN/busOUT), από 128 bits στα 32 bits, σε συνδυασμό με σειριακή εισαγωγή/εξαγωγή των δεδομένων. Η προσέγγιση αυτή θα διευκόλυνε την διασύνδεση του με άλλα ενσωματωμένα συστήματα περιορισμένων πόρων, χωρίς όμως την απώλεια απόδοσης, καθώς ο χρόνος φόρτωσης των δεδομένων παραμένει μικρότερος από τον χρόνο που απαιτείται για την επεξεργασία ενός block μηνύματος (όπως έχει προαναφερθεί, αυτός αντιστοιχεί σε 14 clock circles).



7. Αναφορές

- [1] P. Priyadarshini, P. Narayankar, D. Narayan και S. Meena, «A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish,» 2016.
- [2] V. A. Thakor, M. A. Razzaque και M. R. A. Khandaker, «Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities,» IEEE Access, 2021.
- [3] Amrita, C. P. Ekwueme, I. H. Adam και A. Dwivedi, «Lightweight Cryptography for Internet of Things: A Review,» 2024.
- [4] J. Xu, «Comparison and Application of FPGA and ASIC in Digital System Design,» 2024.
- [5] Z. Wu, K. Qiu και J. Zhang, «A Smart Microcontroller Architecture for the Internet of Things,» 2020.
- [6] I. Radhakrishnan, S. Jadon και P. B. Honnavalli, «Efficiency and Security Evaluation of Lightweight Cryptographic Algorithms for Resource-Constrained IoT Devices,» 2024.
- [7] M. S. Turan, K. A. McKay, J. Kang και J. Kelsey, «Ascon-Based Lightweight Cryptography Standards for Constrained Devices,» NIST, 2025.
- [8] E. Konstantopoulou, G. Athanasiou και N. Sklavos, «Review and Analysis of FPGA and ASIC Implementations of NIST Lightweight Cryptography Finalists,» 2025.
- [9] «ASCON github,» [Ηλεκτρονικό]. Available: <https://github.com/ascon>. [Πρόσβαση 4 2026].
- [10] «PyAscon,» [Ηλεκτρονικό]. Available: <https://github.com/meichlseder/pyascon>. [Πρόσβαση 4 2026].
- [11] S. S. Dhanda, B. Singh και P. Jindal, «Lightweight Cryptography: A Solution to Secure IoT,» 2020.
- [12] Z. A. Mohammed και K. A. Hussein, «Lightweight Cryptography Concepts and Algorithms: A Survey,» 2023.
- [13] «CAESAR competition 2014,» [Ηλεκτρονικό]. Available: <https://competitions.cr.yp.to/caesar-submissions.html>. [Πρόσβαση 4 2026].
- [14] «ASCON homepage,» [Ηλεκτρονικό]. Available: <https://ascon.isec.tugraz.at/index.html>. [Πρόσβαση 4 2026].
- [15] C. Dobraunig, F. Mendel, M. Eichlseder και M. Schl  ffer, «Ascon v1.2,» NIST, 2021.
- [16] L. Grassi, R. L  ftenegger, C. Rechberge και D. R. & M. Sch  fnegger, «On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy,» 2020.
- [17] Trimmerger και S. Stephen, «Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology,» Fellow IEEE.
- [18] L. Samal, R. Kori και K. Mahapatra, «A Secure FPGA-Based IoT Gateway for Smart Home Automation Using PUF-Based Authentication,» 2025.
- [19] «Xilinx,» [Ηλεκτρονικό]. Available: <https://en.wikipedia.org/wiki/Xilinx>. [Πρόσβαση 4 2026].
- [20] «AMD homepage,» [Ηλεκτρονικό]. Available: <https://www.amd.com/en.html>. [Πρόσβαση 4 2026].
- [21] «Xilinx series 7 datasheet,» [Ηλεκτρονικό]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.mouser.com/datasheet/2/903/ds180_7Series_Overview-1591537.pdf. [Πρόσβαση 4 2026].



- [22] «Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics,» [Ηλεκτρονικό]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.farnell.com/datasheets/2301213.pdf. [Πρόσβαση 4 2026].
- [23] «AMD product line,» [Ηλεκτρονικό]. Available: <https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/artix-7.html#product-table>. [Πρόσβαση 4 2026].
- [24] Σ. Παγιάτης, «Vivado Project on github,» [Ηλεκτρονικό]. Available: https://github.com/Pagiatis-Spiros/Ascon_AEAD128-Hash256_on_Vivado. [Πρόσβαση 05 2026].
- [25] «Ascon KATs,» 05 2025. [Ηλεκτρονικό]. Available: https://github.com/ascon/ascon-c/blob/main/crypto_aead/asconaead128/LWC_AEAD_KAT_128_128.txt.
- [26] T. Sy-Nam, H. Van-Thuc και B. Duy-Hieu, «A Hardware Architecture of NIST Lightweight Cryptography Applied in IPSec to Secure High-Throughput Low-Latency IoT Networks,» IEEE Access, 2023.
- [27] J. Kaur, C. Alvaro Cintas, K. Mehran Mozaffari και R. Azarderakhsh, «A Survey on the Implementations, Attacks, and Countermeasures of the Current NIST Lightweight Cryptography Standard,» 2023.
- [28] A. R. Alharbi, A. Aliaedi, A. Aliuhni, M. K. Alghuson, H. Aldawood και S. S. Jamal, «Evaluating Ascon Hardware on 7-series FPGA Devices,» IEEE Access, 2024.
- [29] S. Khan, K. Inayat, M. Fahad Bin, S. Yasir Ali, M. A. U. Rehman, A. Khalid, M. Imran και A. Abdusalomov, «Securing the IoT ecosystem: ASIC-based hardware realization of Ascon lightweight cipher,» 2024.

