

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Διπλωματική Εργασία

**«Υλοποίηση του αλγορίθμου SNOW-V, σε διάταξη FPGA, για κρυπτογράφηση δεδομένων υψηλής απόδοσης, 5G επικοινωνιών»**

Μαρίνος Γκίζας

Α.Μ. 1054348

Επιβλέπων

Νικόλαος Σκλάβος, Αναπληρωτής Καθηγητής

Μέλη Επιτροπής Αξιολόγησης

Νικολός Δημήτριος, Καθηγητής  
Βέργος Χαρίδημος, Καθηγητής  
Σκλάβος Νικόλαος, Αναπληρωτής Καθηγητής

Πάτρα, 2023

© Copyright συγγραφής Μαρίνος Γκίζας, 2023

© Copyright θέματος Νικόλαος Σκλάβος

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών & Πληροφορικής του Πανεπιστημίου Πατρών δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος.

**Περίληψη**

Τα τελευταία χρόνια, η ραγδαία εξέλιξη της τεχνολογίας, και ιδιαίτερα η ανάπτυξη του Internet of Things (IoT), έχει δημιουργήσει την ανάγκη για μετάβαση σε νέας γενιάς συστήματα ασύρματων δικτύων, την 5G. Αυτή η γενιά, έρχεται για να προσφέρει στους χρήστες μεγαλύτερες ταχύτητες, γρηγορότερες ανταποκρίσεις καθώς και χαμηλότερη κατανάλωση ενεργείας, κάτι το οποίο δεν μπορούσε να επιτευχθεί με την προηγούμενη 4η γενιά (4G). Ταυτόχρονα, όμως, κρίνεται αναγκαία η διασφάλιση των δεδομένων των χρηστών, με στόχο την ασφαλή και αξιόπιστη επικοινωνία μεταξύ τους. Αυτό επιτυγχάνεται μέσω της κρυπτογραφίας, η οποία έχει ως στόχο να διασφαλίσει την ιδιωτικότητα και την αυθεντικότητα της πληροφορίας.

Η παρούσα εργασία πραγματεύεται έναν νέο κρυπταλγόριθμο ροής, τον SNOW-V, που παρουσιάστηκε για πρώτη φορά το 2019 από τον Patrik Ekdahl και αποτελεί την βελτιωμένη έκδοση του SNOW 3G. Ένας από τους κυριότερους λόγους, για τον οποίο έπρεπε να υπάρξει νέα αρχιτεκτονική για τον αλγόριθμο SNOW, είναι ότι στην γενιά του 5G υπάρχει ανάγκη για μεγαλύτερη ασφάλεια, εξού και η αύξηση στην έξοδο του αλγόριθμου, από 32 σε 128 bits, που ήταν σε προηγούμενες εκδόσεις, καθώς και χρήση κλειδιού 256 bits από 128 που ήταν στον SNOW 3G, για την κωδικοποίηση της πληροφορίας. Με αυτόν τον τρόπο έγινε εφικτή και η αύξηση σε σημαντικό βαθμό της απόδοσης. Ο κρυπταλγόριθμος υλοποιήθηκε με την βοήθεια της γλώσσας προγραμματισμού VHDL (Very High-Speed Integrated Circuit Hardware Description Language) και χρησιμοποιήθηκε ο Επιτόπιος Πίνακας Προγραμματιζόμενων Πυλών (Field Programmable Gate Array) xcku035-ffva1156-2-i, της οικογένειας Kintex UltraScale της εταιρείας Xilinx, όπου επιτεύχθηκε η μέγιστη συχνότητα των 237 MHz.

**Λέξεις-Κλειδιά:** Κρυπτογραφία • Ιδιωτικότητα • Αυθεντικότητα • SNOW-V • 5G • Κρυπταλγόριθμος ροής • FPGA.

**Abstract**

In recent years, the rapid evolution of technology, and especially the growth of the Internet of Things (IoT), has created the need to move towards a new generation of wireless network systems, the 5G. This generation is offering users higher speeds, faster responses and low energy consumption, which is something that could not be achieved by the previous 4th Generation (4G). At the same time, though, the data of the users must be safeguarded, since the communication between them needs to be safer and more reliable. This can be succeeded through cryptography, which targets the guarantee of privacy and the authenticity of the information.

This thesis is analyzing a new stream cipher, the SNOW-V, which was introduced for the first time in 2019 by Patrik Ekdahl and constitutes an improved version of SNOW 3G. One of the main reasons a new architecture had to be created for the SNOW algorithm is that in the 5G we need greater security. For this reason, a 128 bits keystream is used, rather than 32 bits that we used in previous versions and 256 bits of key, rather than 128 bits that were used in SNOW 3G, in order to code the information. In this way, throughput got largely increased. This stream cipher was captured in the programming language VHDL (Very High-Speed Integrated Circuit Hardware Description Language) and implemented in the FPGA (Field Programmable Gate Array) part xcku035-ffva1156-2-I, of the Kintex UltraScale’s family by Xilinx, achieving Maximum Frequency 237MHz.

**Keywords:** Cryptography • Privacy • Authenticity • SNOW-V • 5G • Stream cipher • FPGA

**Ευχαριστίες**

Ξεκινώντας, θα ήθελα να ευχαριστήσω τον καθηγητή και επιβλέποντα στην διπλωματική μου εργασία, Νικόλαο Σκλάβο, για τις γνώσεις που μου παρείχε κατά την διάρκεια των σπουδών μου και την εμπιστοσύνη που μου έδειξε.

Επιπλέον, θα ήθελα να ευχαριστήσω ιδιαιτέρως τους γονείς μου και την αδερφή μου, που ήταν δίπλα μου με τις πολύτιμες συμβουλές τους και με την στήριξή τους σε όλους τους τομείς της, μέχρι τώρα, πορείας μου.

Τέλος, ένα μεγάλο ευχαριστώ στους φίλους μου, που έκαναν τα χρόνια των σπουδών μου λίγο πιο εύκολα και ευχάριστα, με όλες τις στιγμές που περάσαμε μαζί, είτε εύκολες είτε δύσκολες.

**Περιεχόμενα**

[1 1](#_Toc146479249)

[Εισαγωγή 1](#_Toc146479250)

[1.1 Προβλήματα Ασφάλειας και Ιδιωτικότητας στις Επικοινωνίες Νέας Γενιάς 5G 1](#_Toc146479251)

[1.2 Στόχοι της Διπλωματικής Εργασίας 2](#_Toc146479252)

[1.3 Διάρθρωση της Διπλωματικής Εργασίας 3](#_Toc146479253)

[1.4 Συσχέτιση αλγόριθμου SNOW-V με 5G δίκτυα 3](#_Toc146479254)

[2 5](#_Toc146479255)

[Κρυπτογραφία 5](#_Toc146479256)

[2.1 Εισαγωγή στην Κρυπτογραφία 5](#_Toc146479257)

[2.2 Βασικές Έννοιες Κρυπτοσυστήματος 6](#_Toc146479258)

[2.3 Είδη Κρυπτογραφίας 8](#_Toc146479259)

[2.3.1 Συμμετρική Κρυπτογραφία 8](#_Toc146479260)

[2.3.2 Ασύμμετρη Κρυπτογραφία 9](#_Toc146479261)

[2.4 Στόχοι Κρυπτογραφίας 10](#_Toc146479262)

[3 13](#_Toc146479263)

[Σχεδίαση Κρυπταλγόριθμων Ροής 13](#_Toc146479264)

[3.1 Εισαγωγή 13](#_Toc146479265)

[3.2 Κατηγορίες Κρυπταλγόριθμων Ροής 14](#_Toc146479266)

[3.2.1 LFSR (Linear Feedback Shift Register) 14](#_Toc146479267)

[3.2.2 NFSR (Non-Linear Feedback Shift Register) 15](#_Toc146479268)

[4 16](#_Toc146479269)

[Ολοκληρωμένα Κυκλώματα 16](#_Toc146479270)

[4.1 FPGA (Field Programmable Gate Array) 16](#_Toc146479271)

[4.2 ASIC (Application-Specific Integrated Circuit) 17](#_Toc146479272)

[4.2.1 Τύποι ASIC 17](#_Toc146479273)

[4.3 Διαφορές FPGA με ASIC 18](#_Toc146479274)

[4.4 Η Γλώσσα Περιγραφής Υλικού VHDL 19](#_Toc146479275)

[5 20](#_Toc146479276)

[Οικογένεια SNOW 20](#_Toc146479277)

[5.1 Εισαγωγή στην οικογένεια των SNOW αλγορίθμων 20](#_Toc146479278)

[5.2 SNOW 1.0 20](#_Toc146479279)

[5.2.1 Ευπάθειες SNOW 1.0 22](#_Toc146479280)

[5.3 SNOW 2.0 22](#_Toc146479281)

[5.3.1 Ευπάθειες SNOW 2.0 24](#_Toc146479282)

[5.4 SNOW 3G 24](#_Toc146479283)

[5.4.1 Ευπάθειες SNOW 3G 25](#_Toc146479284)

[5.5 Εισαγωγή στον SNOW-V 26](#_Toc146479285)

[5.6 Γενική Μορφή Σχεδιασμού 28](#_Toc146479286)

[5.6.1 LFSR 28](#_Toc146479287)

[5.6.2 FSM 29](#_Toc146479288)

[5.6.3 Initialization stage 32](#_Toc146479289)

[5.7 Επιθέσεις στον SNOW-V 33](#_Toc146479290)

[5.7.1 Επιθέσεις Αρχικοποίησης 33](#_Toc146479291)

[5.7.2 Επιθέσεις Ανταλλαγής Χρόνου/Μνήμης/Δεδομένων 34](#_Toc146479292)

[5.7.3 Επιθέσεις Μάντεψε-και-Καθόρισε 34](#_Toc146479293)

[5.7.4 Αλγεβρικές Επιθέσεις 35](#_Toc146479294)

[6 36](#_Toc146479295)

[Προτεινόμενος Σχεδιασμός SNOW-V 36](#_Toc146479296)

[6.1 Βελτιστοποίηση Σχεδιασμού του SNOW-V 36](#_Toc146479297)

[6.2 Λειτουργία Προτεινόμενου Σχεδιασμού 38](#_Toc146479298)

[7 42](#_Toc146479299)

[Προσομοίωση και Υλοποίηση Σχεδιασμού 42](#_Toc146479300)

[7.1 Εξομοίωση Λειτουργίας Προτεινόμενου Σχεδιασμού 42](#_Toc146479301)

[7.2 Υλοποίηση Προτεινόμενου Σχεδιασμού 46](#_Toc146479302)

[7.3 Σύγκριση Προτεινόμενου Σχεδιασμού με Αντίστοιχους Αλγόριθμους 47](#_Toc146479303)

[7.4 Σύγκριση Αποτελεσμάτων με άλλες Υλοποιήσεις του SNOW-V 48](#_Toc146479304)

[8 49](#_Toc146479305)

[Συμπεράσματα και Προοπτικές 49](#_Toc146479306)

[Βιβλιογραφία 51](#_Toc146479307)

[Παράρτημα Α: Πρόγραμμα Σύνθεσης και Εξομοίωσης Xilinx Vivado 2022.2 54](#_Toc146479308)

[Παράρτημα Β: Κώδικας SNOW-V 62](#_Toc146479309)

**Λίστα Εικόνων**

**Εικόνα 1**: Μοντέλο Τυπικού Κρυπτοσυστήματος 7

**Εικόνα 2**: Κρυπτογράφηση - Αποκρυπτογράφηση 8

**Εικόνα 3**: Παράδειγμα Ασύμμετρης Κρυπτογραφίας 9

**Εικόνα 4**: CIA [14] 10

**Εικόνα 5**: Ασφαλές Σύστημα 12

**Εικόνα 6**: Λειτουργία Κρυπταλγόριθμου Ροής 14

**Εικόνα 7**: Αρχιτεκτονική ενός γενικού γραμμικού καταχωρητή ολίσθησης με ανατροφοδότηση 15

**Εικόνα 8**: Δομή FPGA [21] 16

**Εικόνα 9**: Αρχιτεκτονική SNOW 1.0 21

**Εικόνα 10**: Η λειτουργία της συνάρτησης Mix\_Columns στον SNOW 2.0 23

**Εικόνα 11**: Αρχιτεκτονική SNOW 2.0 23

**Εικόνα 12**: Αρχιτεκτονική SNOW 3G 25

**Εικόνα 13**: Ψευδοκώδικας SNOW-V 26

**Εικόνα 14**: Initialization mode SNOW-V 27

**Εικόνα 15**: FSMupdate() SNOW-V 27

**Εικόνα 16**: LFSRupdate() SNOW-V 28

**Εικόνα 17**: LFSR (Linear Feedback Shift Register) 29

**Εικόνα 18**: FSM (Finite State Machine) 29

**Εικόνα 19**: S-Box [26] 30

**Εικόνα 20**: Shift Rows [27] 31

**Εικόνα 21**: Μητρώο Πολλαπλασιασμού 31

**Εικόνα 22**: Συνολικός Σχεδιασμός SNOW-V 33

**Εικόνα 23**: Προτεινόμενος Σχεδιασμός LFSR 37

**Εικόνα 24**: Αρχιτεκτονική α, β 38

**Εικόνα 25**: Αρχιτεκτονική α-1, β-1 39

**Εικόνα 26**: Multiply\_XOR BLOCK-A 39

**Εικόνα 27**: Multiply\_XOR BLOCK-B 40

**Εικόνα 28**: Αναμενόμενα αποτελέσματα test vector #1 [28] 41

**Εικόνα 29**: Αποτελέσματα test vector #1 - Μέρος Α 41

**Εικόνα 30**: Αποτελέσματα test vector #1 - Μέρος Β 42

**Εικόνα 31**: Αναμενόμενα αποτελέσματα test vector #2 [29] 42

**Εικόνα 32**: Αποτελέσματα test vector #2 - Μέρος A 43

**Εικόνα 33**: Αποτελέσματα test vector #2 - Μέρος Β 43

**Εικόνα 34**: Αναμενόμενα αποτελέσματα test vector #3 [29] 44

**Εικόνα 35**: Αποτελέσματα test vector #3 - Μέρος A 44

**Εικόνα 36**: Αποτελέσματα test vector #3 - Μέρος Β 45

**Λίστα Πινάκων**

**Πίνακας 1:** Αποτελέσματα Υλοποίησης SNOW-V 45

**Πίνακας 2**: Σύγκριση SNOW-V με άλλους κρυπταλγόριθμους ροής 46

**Πίνακας 3**: Σύγκριση Αποτελεσμάτων με άλλες Υλοποιήσεις του SNOW-V στη Βιβλιογραφία 47

**Συντομογραφίες**

|  |  |
| --- | --- |
| 3DES | Triple Data Encryption Standard |
| 3GPP | 3rd Generation Partnership Project |
| 5G | 5th Generation |
| AES | Advanced Encryption Standard |
| AR | Advanced Reality |
| ASIC | Application-Specific Integrated Circuit |
| CIA | Confidentiality Integrity Availability |
| CLB | Configurable Logic Block |
| DES | Data Encryption Standard |
| eMBB | Enhanced Mobile Broadband |
| FF | Flip-Flop |
| FPGA | Field Programmable Gate Array |
| FSM | Finite State Machine |
| GBPS | Giga Bits Per Second |
| GDPR | General Data Protection Regulation |
| IEEE | Institute of Electrical and Electronics Engineers |
| IMT | International Mobile Communication |
| ISO | International Organization for Standardization |
| IV | Initialization Vector |
| LFSR | Linear Feedback Shift Register |
| LTE | Long-Term Evolution |
| LUT | Loop-Up Table |
| MMS | Multimedia Messaging Service |
| mMTC | Massive Machine-Type Communications |
| NFSR | Non-linear Feedback Shift Register |
| NIST | National Institute of Standards and Technology |
| PCI DSS | Payment Card Industry Data Security Standard |
| QoS | Quality of Service |
| SMS | Short Message Service |
| uRLLC | Ultra-Reliable and Low Latency Communications |
| VHDL | VHSIC Hardware Description Language |
| VHSIC | Very High Speed Integrated Circuit |
| VR | Virtual Reality |

**Γλωσσάρι ή Απόδοση Όρων**

|  |  |
| --- | --- |
| 3DES | Πρότυπο Τριπλών Δεδομένων Κρυπτογράφησης |
| AES | Πρότυπο Προχωρημένης Κρυπτογράφησης |
| AR | Επαυξημένη Πραγματικότητα |
| Architecture | Αρχιτεκτονική |
| ASIC | Ολοκληρωμένα Κυκλώματα Ειδικά για την Εφαρμογή |
| Authenticatiοn | Αυθεντικοποίηση |
| Availability | Διαθεσιμότητα |
| Bandwidth | Εύρος ζώνης |
| Bits | Δυαδικά ψηφία |
| Bitstream | Ακολουθία δυαδικών ψηφίων |
| Block ciphers | Κρυπταλγόριθμοι μπλοκ |
| Cipher text | Κρυπτογραφημένο κείμενο |
| CLB | Λογικά Μπλοκ με δυνατότητα Διαμόρφωσης |
| Coefficients | Συντελεστές |
| Confidentiality | Εμπιστευτικότητα |
| Cryptosystem | Κρυπτοσύστημα |
| Cybersecurity | Κυβερνοασφάλεια |
| Decryption algorithm | Αλγόριθμος Αποκρυπτογράφησης |
| Decryption key | Κλειδί Αποκρυπτογράφησης |
| DES | Πρότυπο Δεδομένων Κρυπτογράφησης |
| eMBB | Ενισχυμένη Κινητή Ευρυζωνικότητα |
| Encoding | Κωδικοποίηση |
| Encryption | Κρυπτογράφηση |
| Encryption algorithm | Αλγόριθμος Κρυπτογράφησης |
| Encryption key | Κλειδί Κρυπτογράφησης |
| Entity | Οντότητα |
| FPGA | Επιτόπια Συστοιχία Προγραμματιζόμενων Πυλών |
| FSM | Μηχανή Πεπερασμένης Κατάστασης |
| GDPR | Γενικός Κανονισμός Προστασίας Δεδομένων της ΕΕ |
| Hardware | Υλικό |
| Implementation | Υλοποίηση |
| IMT | Διεθνείς Κινητές Τηλεπικοινωνίες |
| Initialization mode | Λειτουργία Αρχικοποίησης |
| Integrity | Ακεραιότητα |
| IoT | Διαδίκτυο των Πραγμάτων |
| IV | Διάνυσμα Αρχικοποίησης |
| Key | Κλειδί |
| Keystream | Ακολουθία κλειδιών |
| Keystream generator | Γεννήτρια ακολουθίας κλειδιών |
| Latency | Καθυστέρηση |
| Least significant bit | Λιγότερο σημαντικό δυαδικό ψηφίο |
| LFSR | Γραμμικός Καταχωρητής Ολίσθησης με Ανατροφοδότηση |
| Library | Βιβλιοθήκη |
| LTE | Μακροπρόθεσμη Εξέλιξη |
| mMTC | Επικοινωνίες τύπου μηχανής |
| Most significant bit | Πιο σημαντικό δυαδικό ψηφίο |
| NFSR | Καταχωρητής Ολίσθησης Χωρίς Ανατροφοδότηση |
| NIST | Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας |
| Non-repudiation | Μη-άρνησης |
| NSA | Διεθνής Υπηρεσία Ασφαλείας |
| Permutation Byte-Oriented | Μετάθεση Ανά Byte |
| Plain text | Απλό κείμενο |
| QoS | Ποιότητα των Υπηρεσιών |
| Registers | Καταχωρητές |
| Shift registers | Καταχωρητές Ολίσθησης |
| Stream ciphers | Κρυπταλγόριθμοι Ροής |
| Test vectors | Διανύσματα Ελέγχου |
| uRLLC | Επικοινωνίες Χαμηλής Καθυστέρησης και Μέγιστης Αξιοπιστίας |
| VHDL | Γλώσσα Περιγραφής Υλικού |
| VHSIC | Ολοκληρωμένα Πολύ Υψηλής Ταχύτητας |
| VR | Εικονική Πραγματικότητα |

# 

# Εισαγωγή

## Προβλήματα Ασφάλειας και Ιδιωτικότητας στις Επικοινωνίες Νέας Γενιάς 5G

Οι ασύρματες επικοινωνίες αποτελούν ένα μικρό “θαύμα” των τελευταίων δεκαετιών, μιας και κατέστη δυνατή η μεταφορά πληροφοριών, χωρίς την σύνδεση των επικοινωνούντων πλευρών, με καλώδιο ή ηλεκτρικούς αγωγούς, κάτι που μέχρι πρότινος θεωρούταν αδιανόητο [1]. Η εξέλιξη τους είναι ραγδαία και αυτό αποτυπώνεται στο γεγονός ότι η πρώτη γενιά ασύρματης τηλεπικοινωνίας (1G), όπου ήταν τα αναλογικά κινητά τηλέφωνα, αναπτύχθηκε το 1980 ενώ η δεύτερη (2G) μόλις 11 χρόνια αργότερα, το 1991, στην Φινλανδία. Σε αυτή τη γενιά, γίνεται η μετάβαση στην ψηφιακή κινητή τηλεπικοινωνία αυξάνοντας την ταχύτητα στη μεταφορά φωνής καθώς και το bandwidth, ενώ ταυτόχρονα εμφανίζονται για πρώτη φορά η Υπηρεσία Σύντομου Μηνύματος (SMS) και η Υπηρεσία Μηνύματος Πολυμέσων (MMS) [2]. Το 1998 παρουσιάστηκε η τρίτη γενιά (3G) και τέθηκε σε εφαρμογή το 2001. Οι αναβαθμίσεις που έφερε, αφορούσαν μεγαλύτερες ταχύτητες internet, κάτι το οποίο βοήθησε στην ανάπτυξη επικοινωνιών με φωνή και εικόνα μέσω διαδικτύου [3]. Την ίδια εποχή, ιδρύεται και το 3GPP (3rd Generation Partnership Project), ένα project που ενώνει 7 οργανισμούς, με σκοπό την ανάπτυξη προτύπων, τα οποία θα είναι παγκοσμίως αποδεκτά για τα κινητά συστήματα της 3G. Η αμέσως επόμενη γενιά , 4G ή αλλιώς LTE (Long-Term Evolution) , άρχισε επισήμως να λειτουργεί από το 2008 αυξάνοντας τις ταχύτητες σύνδεσης του διαδικτύου, φτάνοντας ακόμα και σε επίπεδα Gbps (Giga bits per second).

Όπως γίνεται εύκολα αντιληπτό, οι γενιές που διαδέχονται η μία την άλλη, έχουν μια διαφορά μεταξύ τους περίπου 10 ετών. Το ίδιο γίνεται και με την πιο πρόσφατη γενιά των ασύρματων επικοινωνιών που τέθηκε για πρώτη φορά σε λειτουργία το 2018 και ονομάζεται 5G. Πλέον ήταν απαραίτητη η μετάβαση σε αυτή, καθώς ανά τον κόσμο υπάρχουν δισεκατομμύρια, ταυτόχρονα, συνδεδεμένες συσκευές, με αποτέλεσμα να υπάρχει μειωμένο bandwidth και αυξημένο latency, κάτι το οποίο δεν βοηθούσε την εξέλιξη της τεχνολογίας. Το 5G υπόσχεται να καλύψει την πρόταση του IMT (International Mobile Communication), όπου αφορούσε 3 βασικές απαιτήσεις. Η πρώτη αφορά την ενισχυμένη κινητή ευρυζωνικότητα (eMBB) και έχει ως στόχο να ικανοποιήσει τις ανάγκες των χρηστών για ευρεία κάλυψη του δικτύου αλλά και μεγαλύτερο throughput. Έτσι, μπορούν να απολαύσουν υπηρεσίες όπως υψηλής απόδοσης ζωντανής ροής προγραμμάτων, ενισχυμένης πραγματικότητας (AR) και εικονικής πραγματικότητας (VR) [4], παρέχοντάς τους την πολυαναμενόμενη QoS (Quality of Service). Η δεύτερη προσφέρει επικοινωνίες με χαμηλή καθυστέρηση (latency), υψηλή διαθεσιμότητα και μεγάλη αξιοπιστία (URLLC) [4]. Η ύπαρξη τους κρίνεται αναγκαία καθώς πλέον με τα αυτόνομα μέσα μεταφοράς, τις απομακρυσμένες υπηρεσίες -όπως για παράδειγμα μια χειρουργική επέμβαση- αλλά και του IoT, χρειαζόμαστε αστραπιαίες αποκρίσεις για την επίτευξη της αποτελεσματικότητας που αναζητούμε. Η τελευταία, έχει να κάνει με επικοινωνίες μεταξύ μηχανών (mMTC), όπου θα υπάρχει πολύ μεγάλη γεωγραφική κάλυψη και τεράστιος αριθμός συνδεδεμένων συσκευών χαμηλού κόστους, με αποτέλεσμα να έχουμε πολλά δεδομένα την ίδια χρονική στιγμή που τα χρειαζόμαστε [4] [5].

Ταυτόχρονα, όμως, με την αύξηση των υπηρεσιών που θα προσφέρονται από τα δίκτυα 5G, υπάρχει και ανάλογη ανάγκη για αύξηση της ασφάλειας, καθώς όλο και περισσότεροι θα προσπαθούν να υποκλέψουν διαφόρων ειδών πληροφορίες. Ήδη τα δίκτυα 4G είναι επιρρεπή σε επιθέσεις, οι οποίες στοχεύουν στους κόμβους που απαιτούν χαμηλή ενέργεια για την πρόσβαση σε αυτούς. Αυτή η ευπάθεια θα αποτελεί μέρος και των 5G δικτύων, αφού οι κόμβοι πλέον θα συνενωθούν, κάνοντας όλο και περισσότερες συσκευές να επικοινωνούν μεταξύ τους, είτε αφορούν IoT είτε για επικοινωνία, άρα θα αυξηθούν περισσότερο και οι απειλές που προκύπτουν [6]. Επιπροσθέτως, όλες οι επικοινωνίες μεταξύ των συσκευών απαιτούν αυθεντικοποίηση, έτσι ώστε να είμαστε σίγουροι με ποιόν επικοινωνούμε. Συνέπεια αυτού, μαζί με το γεγονός της αύξησης των bits που χρησιμοποιούνται στα 5G, είναι η αναβάθμιση της κρυπτογραφίας, μιας και χρειάζονται νέοι τρόποι κρυπτογράφησης που θα ανταποκρίνονται στις νέες απαιτήσεις που έχουν προκύψει [7].

## Στόχοι της Διπλωματικής Εργασίας

Η παρούσα Διπλωματική Εργασία έχει ως στόχο τον σχεδιασμό και υλοποίηση του SNOW-V, ενός αλγόριθμου για την κρυπτογράφηση δεδομένων. Ο συγκεκριμένος έχει ένα επιπλέον θετικό, ότι το κλειδί που θα κρυπτογραφηθεί είναι των 256 bits και όχι των 128 σε σχέση με προηγούμενες εκδόσεις. Επιτεύχθηκε και η παραλληλοποίηση των LFSR, ενημερώνοντάς τα πλήρως σε κάθε κύκλο ρολογιού, αυξάνοντας σημαντικά το throughput. Επιπλέον, με τον προτεινόμενο σχεδιασμό, εκτός από κρυπτογράφηση υψηλής απόδοσης, έχουμε και υψηλότερη συχνότητα λειτουργίας αλλά και χρήση του λιγότερου δυνατού υλικού.

Η υλοποίηση του κρυπταλγόριθμου έγινε σε UltraScale FPGA της Xilinx, και η γλώσσα περιγραφής υλικού που χρησιμοποιήθηκε είναι η VHDL. Η επαλήθευση του κυκλώματος και της λειτουργίας του έγινε με την βοήθεια του εργαλείου εξομοίωσης και σύνθεσης Vivado. Χρησιμοποιήθηκαν διανύσματα ελέγχου (test vectors) από τους δημιουργούς του αλγόριθμου για την επιβεβαίωση της ορθής λειτουργίας του. Τα αποτελέσματα συγκρίθηκαν με αντίστοιχους κρυπταλγόριθμους για να διαπιστωθεί όντως ότι εξυπηρετεί τον σκοπό για τον οποίο δημιουργήθηκε.

## Διάρθρωση της Διπλωματικής Εργασίας

Στο πρώτο κεφάλαιο, που ονομάζεται Εισαγωγή, κάνουμε μια σύντομη ιστορική αναδρομή στα δίκτυα τηλεπικοινωνιών, επικεντρώνοντας το ενδιαφέρον μας στα δίκτυα της 5ης γενιάς, τις υπηρεσίες που θα προσφέρει αλλά και στους κινδύνους που ελλοχεύουν.

Στο δεύτερο κεφάλαιο με τίτλο Κρυπτογραφία, δίνουμε τα χαρακτηριστικά του βασικού συστήματος κρυπτογραφίας, αναφερόμαστε στα δύο είδη της κρυπτογραφίας αλλά και στις προϋποθέσεις που πρέπει να πληροί ένα σύστημα κρυπτογράφησης για να είναι ασφαλές.

Στο τρίτο κεφάλαιο, Σχεδίαση Κρυπταλγόριθμων Ροής, αναφέρουμε τα κύρια μέρη και την λειτουργία των κρυπταλγόριθμων ροής και τους διακρίνουμε σε κατηγορίες ανάλογα με τον τρόπο που έχουν υλοποιηθεί, αναλύοντας κάθε μία από αυτές.

Στο τέταρτο κεφάλαιο, του οποίου ο τίτλος είναι Ολοκληρωμένα Κυκλώματα, αναλύουμε την δομή των FPGA και πως μέσω αυτών κατασκευάζουμε τα επιθυμητά κυκλώματα. Επίσης, παρέχονται πληροφορίες για την γλώσσα περιγραφής υλικού VHDL.

Στο πέμπτο κεφάλαιο που ονομάζεται Σχεδιασμός SNOW-V, παρουσιάζουμε τον γενικό σχεδιασμό που έχει δοθεί για τον stream cipher SNOW-V και δίνεται ο προτεινόμενος σχεδιασμός αυτής της διπλωματικής εργασίας.

Το έκτο κεφάλαιο, με τίτλο Προσομοίωση και Υλοποίηση σχεδιασμού, πραγματεύεται την προσομοίωση του προτεινόμενου σχεδιασμού για την επαλήθευσή του, γίνεται η σύνθεση και η υλοποίηση του και συγκρίνονται τα αποτελέσματα με άλλους αλγόριθμους.

Στο έβδομο και τελευταίο κεφάλαιο με τίτλο Συμπεράσματα και Προοπτικές παρουσιάζεται μια ανακεφαλαίωση, εξάγοντας τα αντίστοιχα συμπεράσματα για την παρούσα Διπλωματική Εργασία και γίνεται αναφορά σε διαθέσιμες προοπτικές για εξέλιξη του προτεινόμενου σχεδιασμού.

## Συσχέτιση αλγόριθμου SNOW-V με 5G δίκτυα

Όπως έχει ήδη αναφερθεί, η ασφάλεια στα 5G δίκτυα, αποτελεί έναν σημαντικό λόγο για την μεταπήδηση σε επόμενης γενιάς δίκτυα και ο SNOW-V αποτελεί έναν από τους πυλώνες για την βελτίωσή της. Πρακτικά, ο SNOW-V μπορεί να χρησιμοποιηθεί ως hardware σε οποιαδήποτε συσκευή επικοινωνίας και σε διάφορους τομείς. Για παράδειγμα, θα μπορούσε να αποτελεί μέρος των στρατιωτικών συσκευών επικοινωνίας, μιας και σε αυτές τις περιπτώσεις, η ανάγκη για κρυπτογράφηση των δεδομένων για να μην γίνονται υποκλοπές από τρίτους και αποκτηθεί πρόσβαση σε σημαντικές πληροφορίες είναι ύψιστη. Ταυτόχρονα, βρίσκει εφαρμογή και σε IoT συσκευές, όπου η επικοινωνία τους πρέπει να είναι αδιάκοπη και με σωστές πληροφορίες για να είναι η λειτουργία τους ασφαλής και αποδοτική. Τέλος, αξίζει να αναφερθεί ότι μπορεί να χρησιμοποιηθεί από απλούς καταναλωτές στα κινητά τους τηλέφωνα για την μη υποκλοπή των δεδομένων τους. Συμπεραίνοντας, η κοινωνία, με τις ταχύτητες 5G και των νέων τεχνολογικών επιτευγμάτων, προχωράει συνεχώς, κάνοντας τις πληροφορίες που αποστέλλονται πολύ σημαντικές. Ο SNOW-V κρυπταλγόριθμος ροής είναι ένας τρόπος μέσω του υλικού των συσκευών προκειμένου να επιτευχθούν τα απαραίτητα επίπεδα ασφαλείας για επικοινωνίες χωρίς ανεπιθύμητα αποτελέσματα.

# 

# Κρυπτογραφία

## Εισαγωγή στην Κρυπτογραφία

Αν προσπαθούσαμε να ορίσουμε την κρυπτογραφία, θα λέγαμε ότι είναι οι πρακτικές εκείνες, οι οποίες χρησιμοποιούνται για την απόκρυψη και προστασία των πληροφοριών από τρίτους. Παρόλο που αποτελεί ένα μεγάλο κομμάτι των συζητήσεων για την ασφάλεια στις μέρες μας, δεν είναι κάτι που ανακαλύφθηκε πρόσφατα, καθώς η ιστορία της κρυπτογραφίας ξεκινάει το 3.900 π.Χ., όχι όμως με την μορφή που την γνωρίζουμε σήμερα. Σκοπός της, ήταν η αντικατάσταση των γραμμάτων για να γίνει πιο ελκυστικό γραμματικά το κείμενο στο τάφο ενός Αιγύπτιου αρχηγού, του Khnumhotep II. Η πρώτη καταγραφή κρυπτογραφίας για την απόκρυψη πληροφοριών έγινε τον 35ο αιώνα π.Χ, στην Μεσοποταμία, όπου ένας γραφέας ήθελε να αποκρύψει τον τρόπο γυαλίσματος κεραμικών. Η πιο αξιοσημείωτη, και προηγμένη για την εποχή, βέβαια χρήση της κρυπτογραφίας συναντάται στην Ρώμη και ονομάζεται “κώδικας του Καίσαρα”. Αποτελεί την βάση της σημερινής κρυπτογραφίας και ο τρόπος λειτουργίας του ήταν η αλλαγή κάθε γράμματος ενός κειμένου, με τον τρίτο χαρακτήρα έπειτα από αυτόν στο αλφάβητο. Για παράδειγμα, το γράμμα Α γινόταν Δ [8].

Σήμερα, την κρυπτογραφία θα μπορούσαμε να την περιγράψουμε ως μια αέναη μάχη μεταξύ δύο πλευρών. Στην μία, υπάρχουν οι νόμιμες επικοινωνίες και ανταλλαγές μηνυμάτων και θα χαρακτηρίζαμε αυτόν τον κόσμο ως “ανοιχτό”. Στην δεύτερη, υπάρχουν αυτοί που θέλουν να διαβάλουν αυτές τις πληροφορίες και να τις επεξεργαστούν οι ίδιοι, με σκοπό να επωφεληθούν με όποιον τρόπο είναι δυνατόν. Οι μεν, επιθυμούν τα μηνύματα που στέλνονται μεταξύ τους να είναι ακατανόητα και δύσκολα προσβάσιμα σε τρίτους, ενώ οι δε θα ήθελαν να καταλαβαίνουν με μεγάλη ευκολία τα μηνύματα. Η σχέση μεταξύ αυτών των δύο κόσμων είναι αλληλένδετη, καθώς όσο υπάρχουν άνθρωποι που θέλουν να παρεμβαίνουν στις απεσταλμένες πληροφορίες, τόσο μεγαλύτερη επιθυμία υπάρχει για μεγαλύτερη απόκρυψη και συνεπώς βελτίωση των μεθόδων της κρυπτογραφίας [9].

Υπάρχουν, βέβαια, νόμοι και πρότυπα που έχουν δημιουργηθεί έτσι ώστε διάφοροι οργανισμοί και εταιρείες να μπορούν να διασφαλίσουν τα δεδομένα τους καθώς και των πελατών τους. Ο πιο ισχυρός νόμος ονομάζεται GDPR (General Data Protection Regulation), τέθηκε σε ισχύ το 2018 από την Ευρωπαϊκή Ένωση και προβλέπει πολύ αυστηρές ποινές σε περίπτωση παραβίασης του [10]. Όσον αφορά τα πρότυπα, υπάρχουν σε πληθώρα, αλλά τα πιο σημαντικά είναι τα ISO ([International Organization for Standardization](https://www.iso.org/home.html)), NIST ( [National Institute of Standards and Technology](https://www.nist.gov/)) και το PCI DSS (Payment Card Industry Data Security Standard [11].

## Βασικές Έννοιες Κρυπτοσυστήματος

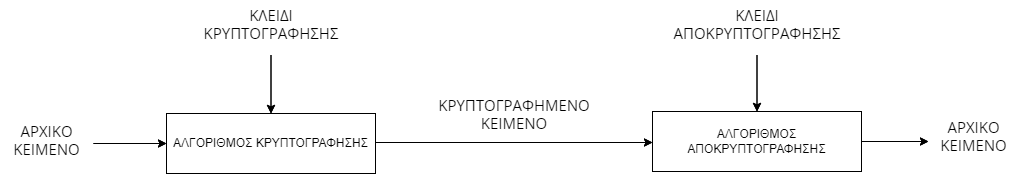
Η εξήγηση των βασικών εννοιών ενός κρυπτοσυστήματος είναι απαραίτητη έτσι ώστε να γίνει κατανοητός ο τρόπος λειτουργίας της κρυπτογραφίας και κατ’ επέκταση των αλγορίθμων κρυπτογραφίας. Αρχικά, θα χρησιμοποιήσουμε μία εικόνα (Εικόνα 1), που αναφέρεται στον τρόπο με τον οποίο λειτουργεί ένα τυπικό κρυπτοσύστημα (cryptosystem). Ο αποστολέας έχει κάποιου είδους πληροφορία που θέλει να μεταφέρει στον παραλήπτη. Αφού κρυπτογραφηθεί, με τρόπο που θα εξηγήσουμε αργότερα, με την βοήθεια κλειδιού (key) που ο ίδιος επιλέγει, αποστέλλεται μέσω ενός μη ασφαλούς καναλιού στον παραλήπτη. Ο παραλήπτης θα λάβει αρχικά το κλειδί που θα του σταλεί από τον αποστολέα, μέσω ενός ασφαλούς καναλιού και θα το χρησιμοποιήσει για την αποκρυπτογράφηση της πληροφορίας που έλαβε. Έτσι θα αποκτήσει την αρχική πληροφορία που ήθελε να του μεταφέρει ο αποστολέας.

Εικόνα που περιέχει κείμενο, διάγραμμα, Σχέδιο, παράλληλα

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 1**: Μοντέλο Τυπικού Κρυπτοσυστήματος

Η μεταφορά, ωστόσο, της πληροφορίας θα γίνει μέσω ενός μη ασφαλούς καναλιού και αυτό θα δώσει την ευκαιρία σε κάποιον επιτιθέμενο να πάρει αυτή την πληροφορία, να την αναλύσει και να την χρησιμοποιήσει προς όφελός του. Σκοπός αυτού του συστήματος είναι να καταστήσει όσο πιο ασφαλή γίνεται την πληροφορία κατά την μεταφορά μέσω της κρυπτογράφησης (encryption) με την βοήθεια ενός αλγορίθμου κρυπτογράφησης. Έστω ότι θέλουμε να μεταδώσουμε ένα κείμενο, τότε, όπως φαίνεται και στην παρακάτω εικόνα (Εικόνα 2), το αρχικό κείμενο (plaintext) μέσω ενός αλγορίθμου κρυπτογράφησης (encryption algorithm) και ενός κλειδιού (encryption key) μετατρέπεται σε κρυπτογραφημένο κείμενο (cipher text). Στον παραλήπτη γίνεται η αντίστροφη διαδικασία, όπου το κείμενο που παρέλαβε μαζί με το κλειδί αποκρυπτογράφησης (decryption key) χρησιμοποιούνται ως είσοδοι στον αλγόριθμο αποκρυπτογράφησης (decryption algorithm) και μας δίνουν ως αποτέλεσμα το αρχικό μας κείμενο που είχε αποσταλεί. Προφανώς στην θέση του κειμένου μπορεί να είναι το οτιδήποτε μπορεί να σταλεί όπως μια εικόνα ή ένα φωνητικό μήνυμα, αφού αυτά θα κωδικοποιηθούν (encoding) σε δυαδικά ψηφία (bits).



**Εικόνα 2**: Κρυπτογράφηση - Αποκρυπτογράφηση

## Είδη Κρυπτογραφίας

Τα είδη της κρυπτογραφίας διακρίνονται σε δυο κατηγορίες, τη συμμετρική (ή κρυπτογραφία συμμετρικού κλειδιού) και την ασύμμετρη (ή κρυπτογραφία δημοσίου κλειδιού). Η διαφορά τους έγκειται στο γεγονός ότι στην πρώτη το κλειδί είναι κοινό και μυστικό, ενώ στην δεύτερη είναι δημόσιο.

### Συμμετρική Κρυπτογραφία

Στη συμμετρική κρυπτογραφία, το κλειδί μεταφέρεται μέσω ενός ασφαλούς καναλιού μεταξύ του αποστολέα και του παραλήπτη ώστε να μην γίνει γνωστό στον έξω κόσμο. Αυτό το κλειδί το χρησιμοποιεί ο αποστολέας για να κρυπτογραφήσει την πληροφορία που θέλει να στείλει και ο παραλήπτης την αποκρυπτογραφεί με την χρήση του ίδιου κλειδιού. Οι πιο βασικοί αλγόριθμοι που χρησιμοποιούνται σε αυτού του είδους την κρυπτογραφία είναι οι DES, AES, 3DES.

* DES  
  Ο DES (Data Encryption Standard) δημιουργήθηκε το 1972 από την IBM, η οποία βασίστηκε σε έναν προηγούμενο σχεδιασμό του Horst Feistel χρησιμοποιώντας κλειδί των 64 bits. Έπειτα όμως από διαβούλευση με το Εθνικό Συμβούλιο Ασφαλείας (NSA) της Αμερικής, διαφοροποιήθηκε για γίνει πιο αποδοτικός απέναντι σε διαφορικές κρυπταναλύσεις, μειώνοντας έτσι τον αριθμό του κλειδιού σε 56 bits. Το 1999, όμως, έσπασε δημόσια κάτι το οποίο τον κατέστησε μη ασφαλή και έχει να χρησιμοποιηθεί από τότε [12].
* 3DES

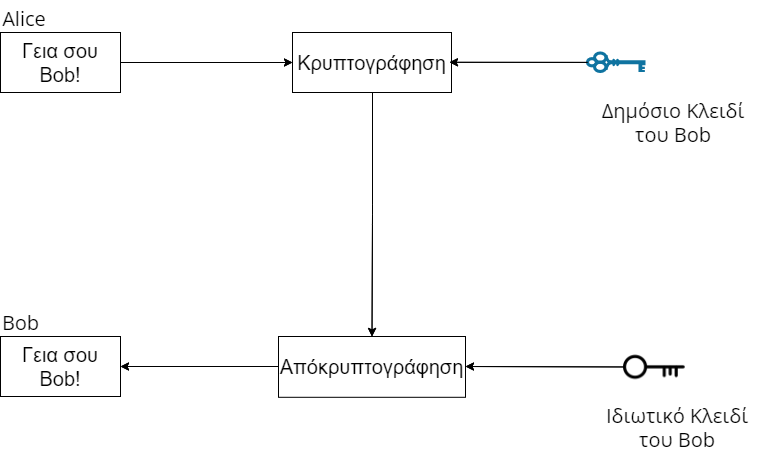
O 3DES αποτελεί αναβάθμιση του DES καθώς δέχεται κλειδί των 192 bits. Έπειτα χωρίζεται σε 3 κλειδιά των 64 bits, όπου το πρώτο κρυπτογραφεί τα δεδομένα εισόδου, το δεύτερο τα αποκρυπτογραφεί και το τρίτο πάλι τα κρυπτογραφεί.

* AES

Ο AES (Advanced Encryption Standard) παρουσιάστηκε για πρώτη φορά το 1997 από το NIST. Τα κλειδιά τα οποία μπορεί να δεχτεί ποικίλουν, αφού μπορούν να είναι 128,192, ακόμα και 256 bits. Με τον AES θα ασχοληθούμε πιο διεξοδικά στο 4ο κεφάλαιο.

### Ασύμμετρη Κρυπτογραφία

Στην ασύμμετρη κρυπτογραφία, υπάρχουν δύο κλειδιά. Ένα είναι ιδιωτικό και μοναδικό για κάθε χρήστη, που χρησιμοποιείται για την αποκρυπτογράφηση της πληροφορίας και κανένας άλλος δεν έχει πρόσβαση σε αυτό, και ένα δημόσιο όπου είναι διαθέσιμο σε όλους. Η σχέση αυτών των δύο κλειδιών είναι συμπληρωματική. Θα δώσουμε ένα παράδειγμα για να γίνει πιο κατανοητό. Έστω, ότι έχουμε δύο άτομα, τον Bob και την Alice. Και οι δύο έχουν από ένα μυστικό κλειδί και ένα δημόσιο. Αν η Alice θέλει να στείλει ένα μήνυμα στον Bob, θα το κρυπτογραφήσει με το δημόσιο κλειδί του Bob που έχει πρόσβαση και θα του το στείλει. Από την πλευρά του ο Bob, μόνο με την χρήση του ιδιωτικού του κλειδιού, που το γνωρίζει μόνο αυτός μπορεί να το αποκρυπτογραφήσει και να διαβάσει το μήνυμα. Το ίδιο θα συμβεί και στην αντίστροφη περίπτωση. Παρακάτω παρουσιάζεται και μια απεικόνιση του παραδείγματος για καλύτερη κατανόηση.



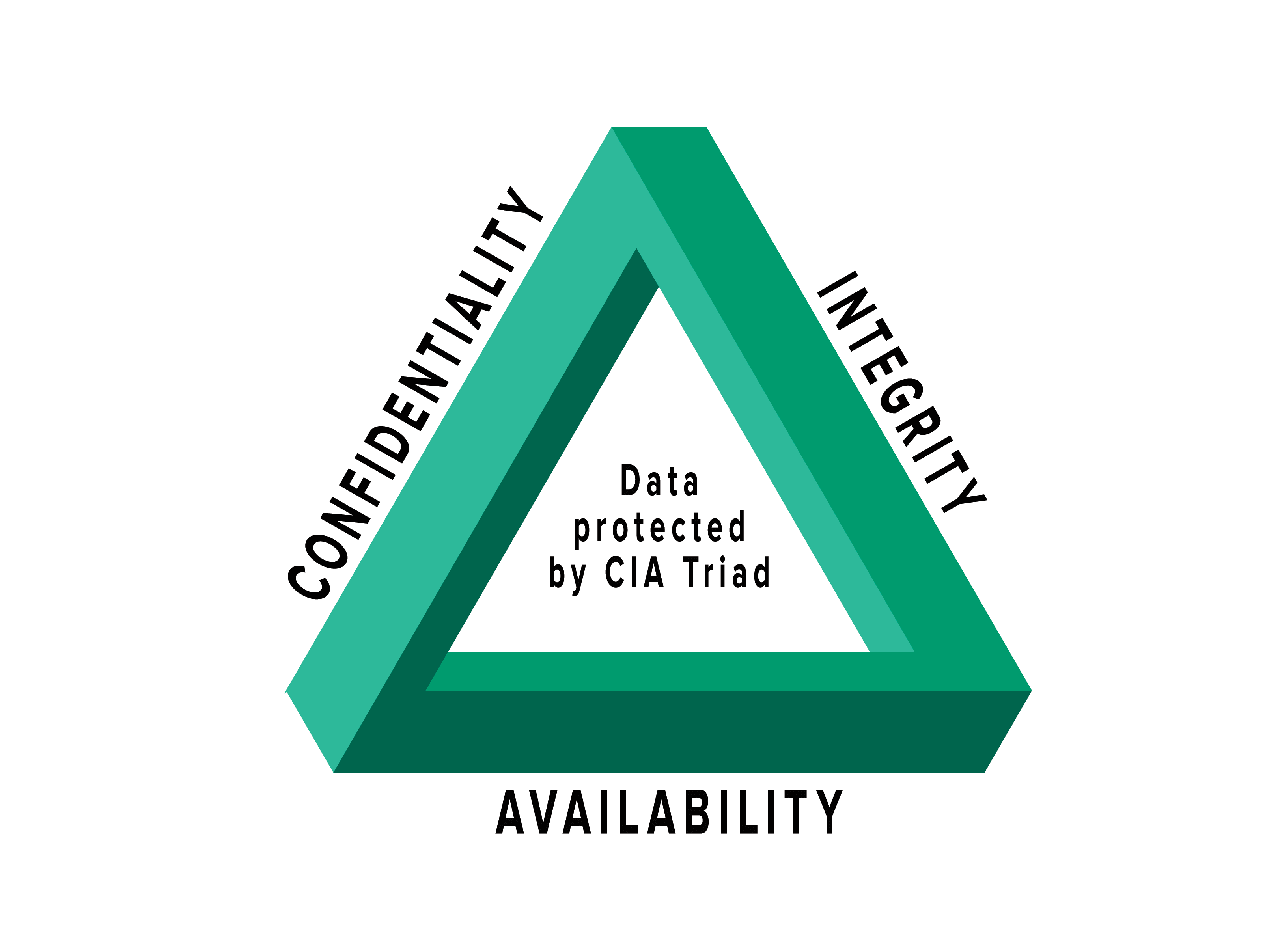
**Εικόνα 3**: Παράδειγμα Ασύμμετρης Κρυπτογραφίας

Εκτός από διαφορές στον τρόπο με τον οποίο λειτουργούν τα δύο είδη κρυπτογραφίας, υπάρχουν διαφορές και στο επίπεδο της ασφάλειας που προσφέρουν. Στην ασύμμετρη κρυπτογραφία, έχουμε περισσότερη ασφάλεια εξαιτίας του αυξημένου μεγέθους του κλειδιού σε σχέση με την συμμετρική. Η συμμετρική, ωστόσο, είναι πιο γρήγορη για τον ίδιο λόγο, αλλά για αποθήκευση ευαίσθητων πληροφοριών χρησιμοποιείται η ασύμμετρη, κάτι το οποίο την έχει κάνει πιο δημοφιλή στον κόσμο [12].

## Στόχοι Κρυπτογραφίας

Η μοντέρνα κρυπτογραφία πρέπει να βασίζεται σε κάποιες αρχές και τεχνικές, προκειμένου να ανταπεξέλθει στις αυξημένες απαιτήσεις της νέας εποχής επικοινωνιών. Ιδιαίτερα με την εμφάνιση του 5G και της προσπάθειας του ανθρώπου να επιτύχει επικοινωνία μεταξύ μηχανών και ανθρώπων, αλλά και μηχανών μεταξύ τους, γίνεται κατανοητό ότι δεν πρέπει να υπάρξει καμία παρέμβαση τρίτων διότι μπορεί να έχει καταστρεπτικές συνέπειες.

Γενικότερα, στην Κυβερνοασφάλεια (cyber security) οι βασικές αρχές της είναι οι εξής τρεις: Εμπιστευτικότητα (Confidentiality), Ακεραιότητα (Integrity), Διαθεσιμότητα (Availability). Όλες αυτές μαζί συνθέτουν την τριάδα CIA, όπως φαίνεται και στην Εικόνα 4.



**Εικόνα 4**: CIA [13]

Ειδικότερα στην σύγχρονη κρυπτογραφία, κρίθηκε αναγκαία η επέκταση αυτού του μοντέλου έτσι ώστε όλο και περισσότεροι στόχοι να επιτυγχάνονται για καλύτερη προστασία των δεδομένων. Συνεπώς, η κρυπτογραφία υπηρετεί πλέον πέντε στόχους που όλοι μαζί συνθέτουν ένα ασφαλές σύστημα, το οποίο αναπαρίσταται σχηματικά στην Εικόνα 5.

1. **Εμπιστευτικότητα:** Με τον όρο εμπιστευτικότητα αναφερόμαστε στην προστασία της πληροφορίας, δηλαδή ότι μόνο άτομα που έχουν εξουσιοδότηση μπορούν να έχουν πρόσβαση σε αυτήν, καθώς και στην διασφάλιση ότι η πληροφορίας μας είναι κωδικοποιημένη για να αποκρυφθεί από τρίτους [14].
2. **Ακεραιότητα:** Η ακεραιότητα βεβαιώνει ότι το μήνυμα που ελήφθη είναι το ίδιο με αυτό που στάλθηκε και δεν έχει παρέμβει κάποιος χωρίς άδεια. Η κρυπτογράφηση ενός μηνύματος, δεν το προστατεύει και δεν του προσθέτει ακεραιότητα, αφού κάποιος μπορεί να επέμβει και να μεταδώσει ένα διαφορετικό μήνυμα από το αρχικό, με αποτέλεσμα ο παραλήπτης να λάβει κάποιο άλλο μήνυμα [15].
3. **Διαθεσιμότητα:** Αναφερόμαστε στην ικανότητα μιας πληροφορίας να είναι διαθέσιμη οποιαδήποτε στιγμή απαιτηθεί από διαπιστευμένα άτομα.
4. **Αυθεντικοποίηση (Authentication):** Η αυθεντικοποίηση καλείται να επικυρώσει ότι η ταυτότητα του αποστολέα ενός μηνύματος είναι αυτή που ισχυρίζεται ο αποστολέας πως είναι. Δηλαδή, αποφεύγεται η παραπλάνηση του παραλήπτη για την πηγή από την οποία έχει προέλθει το μήνυμα [16].
5. **Μη-άρνηση (Non-repudiation):** Σκοπός της μη-άρνησης είναι να προσφέρει ακράδαντα στοιχεία ότι ένα άτομο έκανε κάποιες συγκεκριμένες πράξεις παρόλο που το ίδιο το άτομο μπορεί να τις απαρνείται. Ταυτόχρονα προστατεύει και διάφορα φαινόμενα λογοκλοπής, αφού και με την συμβολή την ψηφιακής υπογραφής, είναι δυνατόν να γνωρίζουμε ποιος πραγματικά είναι αυτός που αρχικά υπέγραψε κάποια συγκεκριμένη πληροφορία[17]**.**

**Εικόνα 5**: Ασφαλές Σύστημα

# 

# Σχεδίαση Κρυπταλγόριθμων Ροής

## Εισαγωγή

Οι κρυπταλγόριθμοι, όπως περιγράφεται και από το όνομά τους, είναι αλγόριθμοι που χρησιμοποιούνται για την κρυπτογράφηση πληροφορίας. Αρχικά, διακρίνονται σε δύο κατηγορίες: τους κρυπταλγόριθμους τμήματος (block ciphers) και τους κρυπταλγόριθμους ροής (stream ciphers). Οι πρώτοι χωρίζουν το plaintext, που έχει ακριβές μήκος και δεν γίνεται να μειωθεί ή να αυξηθεί σε μικρότερα κομμάτια και κάθε ένα από αυτά εισάγεται στον αλγόριθμο κρυπτογράφησης. Οι κρυπταλγόριθμοι ροής, με τους οποίους θα ασχοληθούμε ενδελεχώς, έχουν ως τρόπο λειτουργίας τους μία χρονικά μεταβαλλόμενη μετατροπή των ψηφίων του αρχικού κειμένου. Συγκεκριμένα, για κάθε ψηφίο του plaintext γίνεται η πράξη XOR με μία ψευδοτυχαία ακολουθία bits (keystream), η οποία έχει παραχθεί από μία γεννήτρια κλειδοροής (keystream generator) [18]. Στους σύγχρονους κρυπταλγόριθμους ροής έχουμε δύο στάδια. Το πρώτο ονομάζεται ως “στάδιο αρχικοποίησης” και το δεύτερο “στάδιο παραγωγής keystream”. Κατά την διάρκεια του σταδίου αρχικοποίησης, φορτώνεται στις εισόδους του αλγόριθμου ένα διάνυσμα αρχικοποίησης (IV) και το μυστικό κλειδί (key), τα οποία αναμιγνύονται και ενημερώνουν τις εσωτερικές καταστάσεις του αλγορίθμου μέσω διαφόρων συναρτήσεων. Στο δεύτερο στάδιο, που συνήθως περιέχει μια FSM (Finite-State Machine), παράγουμε το keystream ενώ ταυτόχρονα ανανεώνουμε πάλι τις εσωτερικές καταστάσεις του αλγορίθμου [19].

Υπάρχουν αρκετές διαφορές μεταξύ των δύο κρυπταλγόριθμων. Μερικές από αυτές είναι οι παρακάτω:

1. Οι κρυπταλγόριθμοι ροής είναι πιο περίπλοκοι, κάνοντας έτσι την αποκρυπτογράφηση της πληροφορίας δυσκολότερη.
2. Οι κρυπταλγόριθμοι τμήματος είναι πιο αργοί σε σχέση με τους ροής.
3. Οι stream ciphers έχουν σε κάθε κύκλο ρολογιού διαφορετική συνάρτηση για την κρυπτογράφηση της πληροφορίας, ενώ οι block ciphers έχουν την ίδια.
4. Στους block ciphers μπορούμε να χρησιμοποιήσουμε ξανά ένα κλειδί, αλλά αυτό στους stream ciphers μπορεί να προκαλέσει ευπάθεια στο keystream.

Η Εικόνα 6 που ακολουθεί, φανερώνει τον τρόπο που λειτουργεί ένας κρυπταλγόριθμος ροής.

ΓΕΝΝΗΤΡΙΑ ΚΛΕΙΔΟΡΟΗΣ

XOR

Plaintext

Ciphertext

Keystream

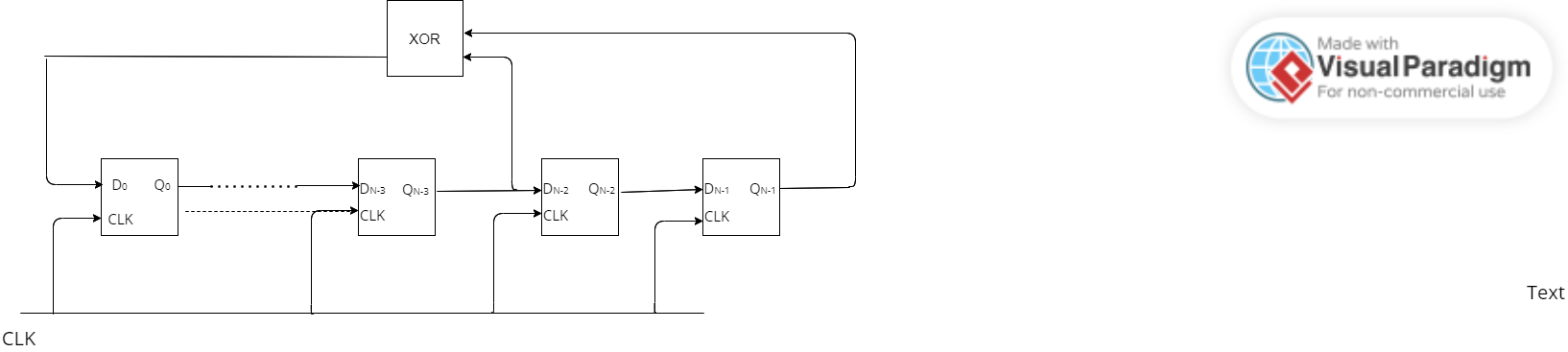
**Εικόνα 6**: Λειτουργία Κρυπταλγόριθμου Ροής

## Κατηγορίες Κρυπταλγόριθμων Ροής

Οι τρόποι κατασκευής των αλγορίθμων χωρίζονται σε διάφορες κατηγορίες ανάλογα στο που βασίζονται. Μερικές από αυτές είναι οι εξής: κατασκευή με γραμμικό καταχωρητή ολίσθησης με ανατροφοδότηση (LFSR - Linear Feedback Shift Register), και κατασκευή με μη- γραμμικό καταχωρητή ολίσθησης με ανατροφοδότηση (NFSR - Nonlinear Feedback Shift Register). Ο κρυπταλγόριθμος ροής SNOW-V που αναλύεται σε αυτή την εργασία, ανήκει στην πρώτη κατηγορία, μιας και χρησιμοποιεί την τεχνική του γραμμικού καταχωρητή ολίσθησης με ανατροφοδότηση.

### LFSR (Linear Feedback Shift Register)

Ένας γραμμικός καταχωρητής ολίσθησης με ανατροφοδότηση είναι μια σειρά από Ν συνδεδεμένα flip-flops, γνωστή και ως καταχωρητές ολίσθησης, όπου η έξοδος του ενός, είναι η είσοδος του επόμενου, και η ανατροφοδότηση του πρώτου γίνεται μέσω μιας συνάρτησης συντιθέμενης από προηγούμενες καταστάσεις. Συνολικά υπάρχουν 2Ν καταστάσεις (ή αλλιώς βαθμίδες), αλλά η κατάσταση στην οποία όλα τα FF είναι 0, συμβαίνει μόνο αν ξεκινήσουμε με αυτή, συνεπώς έχουμε 2Ν-1. Oι καταχωρητές μοιράζονται το ίδιο ρολόι και σε κάθε κύκλο ρολογιού περνάνε την είσοδό τους στην έξοδο. Στην Εικόνα 7 που ακολουθεί, παρουσιάζεται η γενική αρχιτεκτονική ενός γραμμικού καταχωρητή ολίσθησης με ανατροφοδότηση.



**Εικόνα 7**: Αρχιτεκτονική ενός γενικού γραμμικού καταχωρητή ολίσθησης με ανατροφοδότηση

### NFSR (Non-Linear Feedback Shift Register)

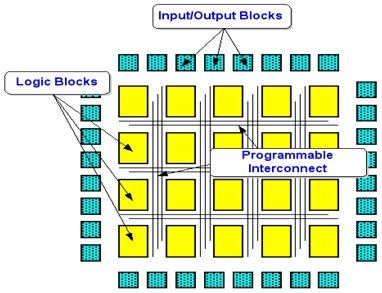
Ένας μη γραμμικός καταχωρητής ολίσθησης με ανατροφοδότηση δεν διαφέρει πάρα πολύ από τον γραμμικό καταχωρητή ολίσθησης με ανατροφοδότηση αφού είναι ένας καταχωρητής ολίσθησης, με την διαφορά ότι η συνάρτηση ανατροφοδότησης δεν είναι πλέον γραμμική όπως προηγουμένως. Αν έχουμε ως f την μη-γραμμική συνάρτηση ανατροφοδότησης και r είναι η επόμενη κατάσταση, τότε μπορούμε να ορίσουμε έναν NFSR ως:

# 

# Ολοκληρωμένα Κυκλώματα

## FPGA (Field Programmable Gate Array)

Η επιτόπια συστοιχία προγραμματιζόμενων πυλών (FPGA) είναι ένα προγραμματιζόμενο ολοκληρωμένο κύκλωμα που αποτελείται από προγραμματιζόμενες διασυνδέσεις και ένα σύνολο προγραμματιζόμενων μπλοκ λογικής (CLB - Configurable Logic Blocks), τα οποία περιέχουν λογικές πύλες, flip-flops και Πίνακες Αναζήτησης (LUTs - Lookup Tables) που χρησιμοποιούνται ως μνήμη αλλά και κυκλώματα επιτέλεσης πράξεων όπως οι αθροιστές. Επιπλέον, περιέχουν μπλοκ Εισόδων/Εξόδων για την εισαγωγή και εξαγωγή της πληροφορίας. Η Εικόνα 8 αποτελεί μια απεικόνιση ενός FPGA καθώς και των μερών του. Για την δημιουργία των επιθυμητών κυκλωμάτων, απλά συνδέονται κατάλληλα οι διασυνδέσεις μεταξύ των προγραμματιζόμενων μπλοκ λογικής. Ο προγραμματισμός των FPGA γίνεται μέσω γλωσσών περιγραφής υλικού, όπως είναι η Verilog και η VHDL, με την οποία θα ασχοληθούμε στο επόμενο μέρος αυτού του κεφαλαίου και αυτό το πρόγραμμα το μετατρέπουμε σε μια ροή δυαδικής πληροφορίας (bitstream), η οποία φορτώνεται στο κύκλωμα.



**Εικόνα 8**: Δομή FPGA [20]

Υπάρχουν αρκετά θετικά από την χρήση των FPGAs. Ένα από τα σημαντικότερα αποτελεί το γεγονός ότι είναι επαναπρογραμματιζόμενα χωρίς να υπάρχει κάποιος περιορισμός για το πόσες φορές μπορεί να χρησιμοποιηθεί εκ νέου. Έτσι, δεν χρειάζεται κάθε φορά να χρησιμοποιούμε καινούργιο hardware για ένα διαφορετικό κύκλωμα. Τέλος, προσφέρουν μεγάλη επεξεργαστική ταχύτητα και είναι πιο οικονομικά σε σχέση με άλλα προγραμματιζόμενα κυκλώματα όπως είναι τα ASIC (Application Specific Integrated Circuit), κάνοντας τα πιο ελκυστικά.

Πλέον τα FPGA αποτελούν ένα από τα πιο σύγχρονα μέσα κατασκευής ολοκληρωμένων κυκλωμάτων. Βρίσκουν χρηστικότητα σε πολλούς και διαφορετικούς τομείς και μερικοί από αυτούς είναι οι εξής:

* Αυτοκινητοβιομηχανία,
* Ιατρική,
* Επεξεργασία εικόνας και ήχου,
* Στρατιωτικές τεχνολογίες,
* Εφαρμογές διαστήματος.

## ASIC (Application-Specific Integrated Circuit)

Τα ολοκληρωμένα κυκλώματα εξειδικευμένης εφαρμογής (ASIC) αναφέρονται σε ολοκληρωμένα κυκλώματα που ο σχεδιασμός του έχει ως στόχο την εκτέλεση συγκεκριμένων λειτουργιών ή εφαρμογών. Είναι υψηλής απόδοσης ολοκληρωμένα κυκλώματα και έχουν χαμηλή κατανάλωση ενέργειας κάνοντάς τα ιδανικά για την χρήση τους σε ψηφιακή επεξεργασία σήματος, αισθητήρες και άλλες προηγμένες εφαρμογές. Για την σχεδίαση και την κατασκευή ενός ASIC χρειάζονται εξειδικευμένες γνώσεις μιας και είναι αρκετά πολύπλοκη διαδικασία. Επιπλέον, παρόλο που είναι αρκετά ακριβά για την κατασκευή τους και συνήθως συνίσταται να χρησιμοποιούνται σε προϊόντα που έχουν μεγάλη κατανάλωση, έχουν αρκετά μικρό μέγεθος συνδυάζοντας λογικές πύλες, μνήμες, μικροεπεξεργαστή και άλλα σε ένα μόνο ολοκληρωμένο. Αυτά τα ASIC είναι γνωστά και ως SoC (System-On-Chip), δηλαδή ολόκληρα συστήματα σε ένα μόνο ολοκληρωμένο και είναι δυνατόν να βρεθούν σε ηλεκτρονικές συσκευές όπως τα κινητά τηλέφωνα και οι τηλεοράσεις, που έχουν μόνιμη και ίδια λειτουργία σε όλη την διάρκεια της ζωής τους.

### Τύποι ASIC

Ανάλογα με την ποσότητα προσαρμογής που μπορεί να κάνει ο προγραμματιστής, τα ASIC χωρίζονται σε τρεις κατηγορίες. Τα πλήρως προσαρμοσμένα (full custom), τα ημί-προσαρμοσμένα (semi-custom) και τα ASIC δομημένου σχεδιασμού (structured).

* Πλήρως προσαρμοσμένα: Τα ολοκληρωμένα αυτά κυκλώματα σχεδιάζονται από την αρχή και από την βάση τους για μία προκαθορισμένη εφαρμογή έχοντας ως στόχο την υψηλότερη δυνατή ταχύτητα, την χαμηλότερη κατανάλωση αλλά και το μικρότερο μέγεθος που μπορεί να επιτευχθεί. Στα αρνητικά αυτών των ολοκληρωμένων βρίσκεται ο αυξημένος χρόνος για σχεδιασμό και παραγωγής τους και η χρήση πολύπλοκων σχεδιασμών που απαιτούν υψηλή σχεδιαστική ικανότητα.
* Ημι-προσαρμοσμένα: Είναι ολοκληρωμένα μερικώς προσαρμοσμένα για να εκτελούν διάφορες λειτουργίες και να είναι επιτρεπτές μερικές τροποποιήσεις.
* Δομημένα: Χαρακτηριστικά αυτών των ολοκληρωμένων είναι η χρήση προδιαγεγραμμένων μεταλλικών στρωμάτων για την κατασκευή τους, μειώνοντας έτσι την διάρκεια παραγωγής, και πρόβλεψης για το τι θα σχεδιαστεί, μειώνοντας τον χρόνο σχεδιασμού. Αποτελούν νέα μόδα στην βιομηχανία των ημιαγωγών.

## Διαφορές FPGA με ASIC

Υπάρχουν διαφόρων ειδών διαφορές, είτε αυτές είναι ποιοτικές είτε οικονομικές ακόμα και λειτουργικές. Παρακάτω παρουσιάζονται οι σημαντικότερες εξ αυτών:

1. Τα ASIC έχουν πιο πολύπλοκη ροή σχεδιασμού κάνοντας την διαδικασία της σχεδίασης πιο έντονη και επίπονη.
2. Τα FPGA είναι πιο ευέλικτα μιας και υπάρχει η δυνατότητα επαναπρογραμματισμού του ίδιου ολοκληρωμένου, αλλάζοντάς του έτσι την λειτουργία του χωρίς καμία επίπτωση στην απόδοσή του.
3. Όσον αφορά την απόδοση, η υπεροχή των ASIC είναι δεδομένη μιας και αυτός είναι ένας από τους λόγους για τους οποίους σχεδιάζονται εξ αρχής.
4. Τα ASIC έχουν μεγαλύτερο κόστος έρευνας, σχεδιασμού, ανάπτυξης και ελέγχου τους απ’ ότι τα FPGA και ταυτόχρονα απαιτούν περισσότερο χρόνο για την διάθεση τους στην αγορά.
5. Τα FPGA έχουν μεγαλύτερη κατανάλωση ρεύματος απ’ ότι τα ASIC, κάτι το οποίο αποτελεί σημαντικό σημείο, ειδικότερα όταν επιλέγονται για σχεδιασμό μιας συσκευής που στηρίζεται στην δύναμη της μπαταρίας.

Με βάση τα παραπάνω, γίνεται αντιληπτό ότι τα ASIC χρησιμοποιούνται σε πιο εξειδικευμένες περιπτώσεις όπου αναζητείται η απόδοση, ενώ τα FPGA είναι πιο φθηνά και επαναχρησιμοποιούμενα.

## Η Γλώσσα Περιγραφής Υλικού VHDL

Η σχεδίαση και δομή των ολοκληρωμένων κυκλωμάτων, γίνεται με γλώσσες περιγραφής υλικού. Μία από αυτές είναι και η VHDL, η οποία είναι συντόμευση των λέξεων VHSIC Hardware Description Language. Το ακρωνύμιο VHSIC (Very High Speed Integrated Circuit) μας υποδεικνύει ότι πρόκειται για μεγάλης ταχύτητας ολοκληρωμένα κυκλώματα. Είναι γλώσσα σχετικά χαμηλού επιπέδου, με δομές που χρησιμοποιούνται για τον χειρισμό της παραλληλίας που προσφέρει για την σχεδίαση του υλικού. Ταυτόχρονα, δεν κάνει διαχωρισμό μεταξύ κεφαλαίων και πεζών γραμμάτων (non case-sensitive), κάνοντας την VHDL μία από τις πιο εύχρηστες γλώσσες για την δημιουργία κώδικα. Οι πιο βασικές ενότητες για την δημιουργία κώδικα σε VHDL είναι οι εξής τρεις:

1. ΒΙΒΛΙΟΘΉΚΗ: Χρησιμοποιείται για τον ορισμό της βιβλιοθήκης που θα χρησιμοποιηθεί στο εκάστοτε αρχείο.
2. ΟΝΤΟΤΗΤΑ: Εδώ καθορίζονται οι είσοδοι και οι έξοδοι του κυκλώματος. Αν μπορούσαμε να το περιγράψουμε, θα το ονομάζαμε ως “μαύρο κουτί” όπου εσωτερικά του δεν γνωρίζουμε τι υπάρχει αλλά βλέπουμε μόνο τι εισέρχεται και τι εξέρχεται από αυτό.
3. ΑΡΧΙΤΕΚΤΟΝΙΚΗ: Περιέχει τον κώδικα που περιγράφει την συμπεριφορά και λειτουργία του “μαύρου κουτιού”.

Αναπτύχθηκε για πρώτη φορά στις αρχές του 1980 από το Υπουργείο Άμυνας των Ηνωμένων Πολιτειών, έχοντας ως στόχο την τεκμηρίωση κυκλωμάτων ASIC που χρησιμοποιούσαν. Η αρχική έκδοσή της αποτέλεσε πρότυπο του IEEE (Institute of Electrical and Electronics Engineers) το 1987, χωρίς, ωστόσο, να υποστηρίζει μέχρι τότε πολλαπλές τιμές σε μία μεταβλητή. Η τελευταία έκδοση της έρχεται το 2019 και έχει ονομαστεί ως IEEE 1076-2019.

# 

# Οικογένεια SNOW

## Εισαγωγή στην οικογένεια των SNOW αλγορίθμων

Ο αλγόριθμος SNOW-V είναι το τελευταίο μέλος μιας μεγάλης οικογένειας αλγόριθμων. Είναι μια λογική και προβλεπόμενη εξέλιξη των προηγούμενων μελών, που είχαν τον ίδιο σκοπό, αλλά η εφαρμογή τους ήταν σε δίκτυα προηγούμενης γενιάς. Συνεπώς, όσο οι γενιές των δικτύων προχωρούν, τόσο θα υπάρχουν και νέες εκδόσεις, μεγαλώνοντας ακόμα περισσότερο την οικογένεια των SNOW αλγορίθμων.

## SNOW 1.0

Το πρώτο μέλος της οικογένειας των SNOW αλγορίθμων είναι ο SNOW 1.0 (Εικόνα 9). Υποβλήθηκε στο NESSIE (New European Schemes for Signatures, Integrity and Encryption), ένα ευρωπαϊκό ερευνητικό έργο, το οποίο είχε ως στόχο τον εντοπισμό ασφαλών, χαμηλού επιπέδου αλγορίθμων κρυπτογραφίας. Για τον αλγόριθμο μπορεί να χρησιμοποιηθεί κλειδί είτε των 128 είτε των 256 bits, διάνυσμα αρχικοποίησης των 128 bits και παράγεται keystream των 32 bits. Αποτελείται από δύο μέρη, το LFSR και το FSM. Στο LFSR μέρος, όπου είναι η γεννήτρια ψευδοτυχαίων διανυσμάτων, υπάρχει ένας LFSR αποτελούμενος από 16 κελία των 32 δυαδικών ψηφίων ο καθένας. Σε κάθε κύκλο ρολογιού, γίνεται δεξιά ολίσθηση κατά μία θέση. Στο FSM υπάρχουν δύο καταχωρητές των 32 bits ο καθένας, 7 βήματα κυκλικής ολίσθησης προς τα αριστερά, 2 προσθέσεις mod32 και ένα S-Box. Αν τα κελιά του LFSR ορίζονται ως s0, s1,…..,s15 τότε το s0 αποτελεί την είσοδο του FSM, ενώ τα κελία s6, s12 ,s15 συνεισφέρουν στην ανατροφοδότηση του LFSR μαζί με το α το οποίο αποτελεί την ρίζα του πολυωνύμου

π(x) = x32 + x29 + x20 + x15 + x10 + x + 1.

Εικόνα που περιέχει διάγραμμα, τεχνικό σχέδιο, γραμμή, Σχέδιο

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 9**: Αρχιτεκτονική SNOW 1.0

Συνεπώς, το πολυώνυμο το οποίο δίνει την ανατροφοδότηση της γεννήτριας είναι το εξής:

p(x) = x15 + x12 + x6 + α-1.

Εκτός από τις λογικές πράξεις, στο FSM γίνεται και κυκλική ολίσθηση αριστερά 7 φορές. Αυτό σημαίνει ότι το διάνυσμα

x = ( x31, x30, ……., x0),

με 7 ολισθήσεις αριστερά γίνεται:

x = ( x24, x23, ……., x0, x31, ……, x1,x0).

Τέλος, υπάρχει και ένα S-Box, στο οποίο γίνεται αλλαγή των τιμών των bytes, ανάλογα με την τιμή της εισόδου. Το S-Box χωρίζεται σε 4 μικρότερα του ενός byte το καθένα, δηλαδή δυο δεκαεξαδικών ψηφίων. Η είσοδος χωρίζεται σε 4 bytes, από το πιο σημαντικό, στο λιγότερο σημαντικό byte, και το κάθε ένα αποτελεί την είσοδο στα μικρότερα S-Boxes. Το πρώτο δεκαεξαδικό ψηφίο ορίζει την σειρά και το δεύτερο δεκαεξαδικό την στήλη στο S-Box και η τιμή με την οποία θα γίνει η αλλαγή είναι το στοιχείο που βρίσκεται στον συνδυασμό της γραμμής με την στήλη. Για παράδειγμα, αν το byte που θα εισέλθει θα έχει την τιμή 4a στο δεκαεξαδικό σύστημα, τότε το στοιχείο που βρίσκεται στην γραμμή 4 και την στήλη 10, θα είναι αυτό που θα αντικαταστήσει το 4a. Μετά από την έξοδο από τα S-Boxes γίνεται και μετάθεση των bits. Η μετάθεση περιγράφεται από:

x = [3,10,20,24,0,14,17,29,7,13,18,25,5,12,23,27,1,8,21,26,4,9,19,31,2,11,16,28,6,15,22,30],

σύμφωνα με το οποίο το bit 31 γίνεται μετάθεση στην θέση 3, το 30 στην θέση 10 κ.ο.κ.

### Ευπάθειες SNOW 1.0

Παρόλο που στην αρχή ο SNOW 1.0 θεωρήθηκε ως ένας ασφαλής αλγόριθμος, έπειτα διαπιστώθηκε μέσω της χρήσης επιθέσεων εναντίων του ότι έκρυβε κάποιες αδυναμίες, οι οποίες τον καθιστούσαν εύκολα προσπελάσιμο. Η πρώτη αδυναμία έγκειται στο γεγονός ότι υπάρχει μόνο μια είσοδος στο FSM μέρος. Δίνεται η δυνατότητα, έτσι, σε έναν επιτιθέμενο να αντιστρέψει τις λειτουργίες και τις πράξεις που λαμβάνουν μέρος στο FSM και να αποκρυπτογραφήσει την πληροφορία μετά από μερικές προβλέψεις. Η δεύτερη ευπάθεια προκύπτει έπειτα από την χρήση συγκεκριμένων πράξεων που αφορά το πολυώνυμο ανατροφοδότησης p(x), καταλήγοντας σε μία γραμμική επαναληπτική εξίσωση που ισχύει για κάθε θέση των δυαδικών ψηφίων. Συνεπώς, οποιαδήποτε συσχέτιση βρέθηκε προηγουμένως στο FSM μπορεί να αποτελέσει και μία επίθεση διάκρισης ή ακόμα και επιθέσεις που αφορούν bits (bit-wise attacks). Βρέθηκαν, επιπλέον, και άλλες συσχετίσεις οι οποίες οφείλονταν από την αλληλεπίδραση μεταξύ της μετάθεσης των bits μετά τα S-Boxes και της κυκλικής ολίσθησης των 7 θέσεων [21].

## SNOW 2.0

Οι παραπάνω λόγοι οδήγησαν στην δημιουργία ενός ελαφρά διαφοροποιημένου αλγόριθμου για να καλύψει τις προηγούμενες αδυναμίες. Αρχικά στο LFSR η ανατροφοδότηση γίνεται με διαφορετικό πολυώνυμο που είναι το εξής:

π(x) = αx16 + x14 + α−1x5 + 1.

Όπως γίνεται αντιληπτό χρησιμοποιούνται πλέον δύο σταθερές (α, α-1), έχοντας ως στόχο την βελτίωση της ανθεκτικότητας σε bit-wise επιθέσεις. Για το FSM οι αλλαγές είναι περισσότερες καθώς εκεί υπήρξαν και οι περισσότερες ευπάθειες. Αρχικά, αυξήθηκαν οι είσοδοι από μία σε δύο, λύνοντας έτσι το πρόβλημα που μπορούσε να δημιουργηθεί από κάποιον επιτιθέμενο στην προηγούμενη έκδοση. Ταυτόχρονα, αφαιρέθηκε εντελώς η αριστερή κυκλική ολίσθηση κατά 7 θέσεις. Τέλος, αλλαγές έγιναν και στο S-Box, αφού μετά την αλλαγή τιμών από το S-Box του Rijndael, προστέθηκε η λειτουργία Mix\_Columns. Σε αυτή, οι έξοδοι των τιμών από το S-Box που συμβολίζονται ως SR[w0], SR[w1], SR[w2], SR[w3], με SR[w0] να είναι η έξοδος των λιγότερο σημαντικών δυαδικών ψηφίων και SR[w3] η έξοδος των περισσότερο σημαντικών ψηφίων, πολλαπλασιάζονται με ένα μητρώο όπως φαίνεται και στην Εικόνα 10:

Εικόνα που περιέχει κείμενο, γραμματοσειρά, αριθμός, λευκό

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 10**: Η λειτουργία της συνάρτησης Mix\_Columns στον SNOW 2.0

Τα r0,r1,r2,r3 αποτελούν τις εξόδους που προκύπτουν συνολικά από το σύστημα του S-Box. Στην εικόνα 11 παρουσιάζεται η αρχιτεκτονική του αλγόριθμου SNOW 2.0

Εικόνα που περιέχει διάγραμμα, τεχνικό σχέδιο, Σχέδιο, γραμμή

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 11**: Αρχιτεκτονική SNOW 2.0

Αρκετά πράγματα έχουν παραμείνει τα ίδια και δύο από αυτά είναι οι είσοδοι και οι έξοδοι του συστήματος, όπου το κλειδί είναι των 128 ή 256 bits, το διάνυσμα αρχικοποίησης των 128 bits και το keystream που παράγεται σε κάθε κύκλο ρολογιού είναι των 32 bits.

### Ευπάθειες SNOW 2.0

Στον SNOW 2.0, μιας και λύθηκαν οι ευπάθειες του προηγούμενου SNOW, υπήρχε η εντύπωση ότι πλέον ήταν ένας ασφαλής αλγόριθμος. Αυτή η εντύπωση σταμάτησε να επικρατεί όταν παρατηρήθηκε η ευπάθεια σε γραμμικές επιθέσεις με μάσκα. Ο κύριος λόγος που υφίσταται αυτή η αδυναμία είναι η χρήση των στοιχείων α και α-1 ως πολλαπλασιαστές των κύριων στοιχείων του LFSR, εφόσον θεωρούνται ως γραμμικοί μετασχηματισμοί [22]. Τέλος, η χρήση ενός κλειδιού των 256 bits αποτελεί έναν ακόμα λόγο όπου ο SNOW 2.0 είναι αδύναμος σε γραμμικές επιθέσεις με μάσκα.

## SNOW 3G

Ο κρυπταλγόριθμος ροής SNOW 3G (Εικόνα 12) είναι η αναβάθμιση του SNOW 2.0 με εφαρμογή σε δίκτυα 3G. Δημιουργήθηκε το 2006 και ακολουθεί τις απαιτήσεις του 3GPP για τον χρόνο και τους πόρους της μνήμης. Έχει ως εισόδους, κλειδί των 128 bits και διάνυσμα αρχικοποίησης των 128 bits, ενώ ως έξοδο παράγει μια ακολουθία των 32 δυαδικών ψηφίων. Το LFSR μέρος αποτελείται από 16 κελία των 32 δυαδικών ψηφίων το καθένα, ενώ το FSM από 3 καταχωρητές των 32 bits, δύο γύρους αλλαγών τιμών μέσω S-Boxes καθώς και δύο προσθέσεις modulo 32. Το πολυώνυμο με βάση το οποίο γίνεται η ανατροφοδότηση του LFSR είναι το εξής:

π(x) = αx16 + x14 + α−1x5 + 1.

Σε κάθε κύκλο ρολογιού (t>0) η έξοδος παράγεται από μια πρόσθεση modulo 32 του st15 με το R1t και το αποτέλεσμα αυτής της πράξης γίνεται XOR με το R2t. Τα δύο S-Boxes χωρίζονται σε 4 μικρότερα το καθένα, έτσι ώστε τα 4 bytes που θα χωριστεί η είσοδος, να αποτελούν την είσοδο στα μικρότερα S-Boxes.

Εικόνα που περιέχει διάγραμμα, τεχνικό σχέδιο, Σχέδιο, σχηματικό

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 12**: Αρχιτεκτονική SNOW 3G

Όπως γίνεται αντιληπτό, ο SNOW 3G δεν διαφέρει σε πολλά σημεία από τον SNOW 2.0. Η πρώτη διαφορά είναι η μείωση του κλειδιού από 256 σε 128 bits. Ο λόγος πίσω από αυτή την αλλαγή είναι η διαπίστωση ότι τα 256 δυαδικά ψηφία του κλειδιού δημιουργούσαν την ευπάθεια που έχει ο SNOW 2.0 στις γραμμικές επιθέσεις με μάσκα.

### Ευπάθειες SNOW 3G

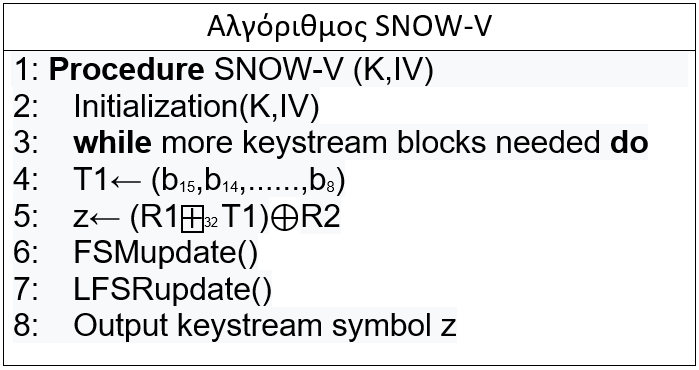
Οι αδυναμίες του SNOW 3G βρίσκονται σε επιθέσεις που έχουν να κάνουν με την ανάκτηση των αρχικών τιμών του LFSR, γνωρίζοντας κάποια δυαδικά ψηφία του keystream (correlation attacks), καθώς και επιθέσεις που έχουν ως στόχο την ανίχνευση κάποιας μη τυχαίας συμπεριφοράς για την αναγνώριση του μοτίβου λειτουργίας του αλγόριθμου (distinguishing attack) [23].

Οι προηγούμενοι αλγόριθμοι, όπως και ο SNOW 3G, έχουν και ευπάθειες σε επιθέσεις που χρειάζεται η φυσική παρουσία του επιτιθέμενου. Μία από αυτές είναι και η εισαγωγή σφαλμάτων (fault injection) στο σύστημα με σκοπό την ανίχνευση αυτών στην έξοδο και μέσω αυτής να γίνει κατανοητή η λειτουργία του συστήματος [24].

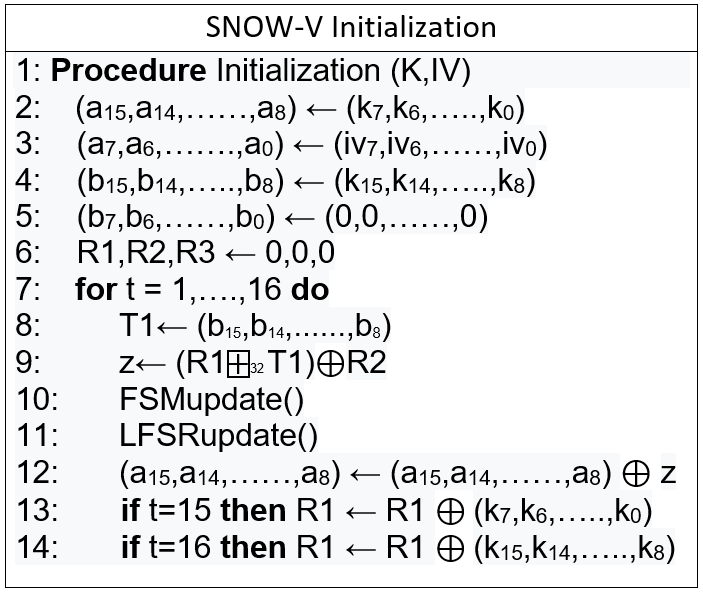
Η επίλυση όλων αυτών των αδυναμιών του SNOW 3G αλλά και των υπόλοιπων αλγορίθμων, γίνεται στην νέα έκδοση, με νέα αρχιτεκτονική του αλγόριθμου SNOW-V, ο οποίος για την προστασία των παραπάνω επιθέσεων, αυξάνεται το κλειδί από 128 σε 256 bits, η έξοδος από 32 σε 128 δυαδικά ψηφία, και δημιουργώντας δύο LFSRs που αλληλοεπιδρούν, αντί για ένα, όπως στις προηγούμενες εκδόσεις, μεγαλώνοντας έτσι τον βαθμό δυσκολίας για τον επιτιθέμενο να ανακτήσει οποιαδήποτε πληροφορία για την λειτουργία του συστήματος και κατ’ επέκταση για την πληροφορία που κρυπτογραφείται.

## Εισαγωγή στον SNOW-V

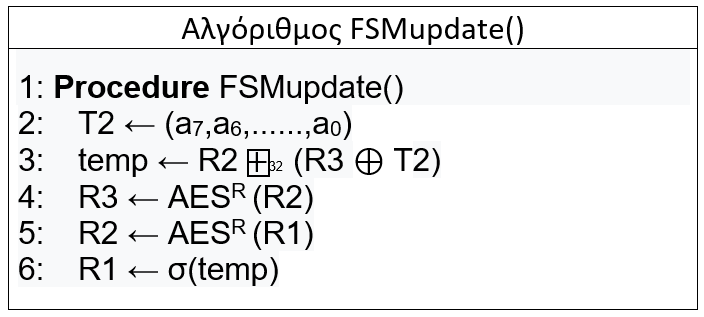
Στην παρούσα ενότητα θα παρουσιάσουμε τον stream cipher τον οποίο πραγματεύεται αυτή η διπλωματική, τον SNOW-V. Σχεδιάστηκε και υλοποιήθηκε με τέτοιο τρόπο ώστε να έχουμε όσο το δυνατόν λιγότερη χρήση υλικού ενώ παράλληλα να επιτυγχάνουμε ικανοποιητική απόδοση. Δέχεται 256 bits κλειδί και 128 bits διάνυσμα αρχικοποίησης, παράγοντας σε κάθε κύκλο ρολογιού κρυπτογραφημένο μήνυμα 128 bits. Αρχιτεκτονικά αποτελείται από δύο κύρια μέρη, το LFSR μέρος και το FSM μέρος. Συνεχίζοντας, θα παρουσιάσουμε τους απαραίτητους ψευδοκώδικες του αλγόριθμου για την υλοποίησή του. Ο κύριος ψευδοκώδικας του SNOW-V παρουσιάζεται στην Εικόνα 13. Η εκτέλεσή του αποτελείται από δύο φάσεις: το initialization mode (Εικόνα 14), όπου γίνεται η αρχικοποίηση του και το running-key mode, όπου συνεχίζουμε να εκτελούμε επαναλήψεις μέχρι να έχουμε παράξει την ακολουθία κλειδιών του μεγέθους που επιθυμούμε. Παρατηρούμε ότι και οι δύο φάσεις του αλγορίθμου καλούν τους αλγόριθμους LFSRupdate() (Εικόνα 15) και FSMupdate() (Εικόνα 16) για την ενημέρωση των μερών LFSR και FSM αντίστοιχα.



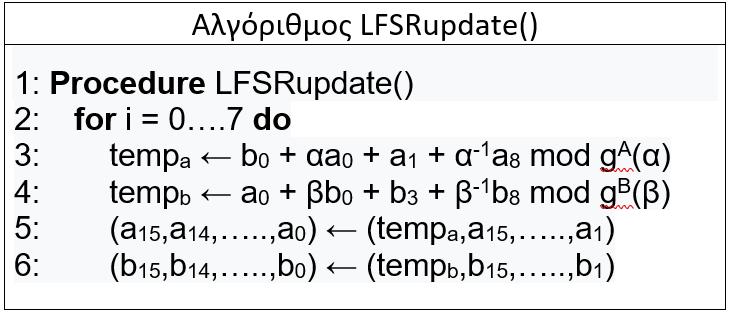
**Εικόνα 13**: Ψευδοκώδικας SNOW-V



**Εικόνα 14**: Initialization mode SNOW-V



**Εικόνα 15**: FSMupdate() SNOW-V

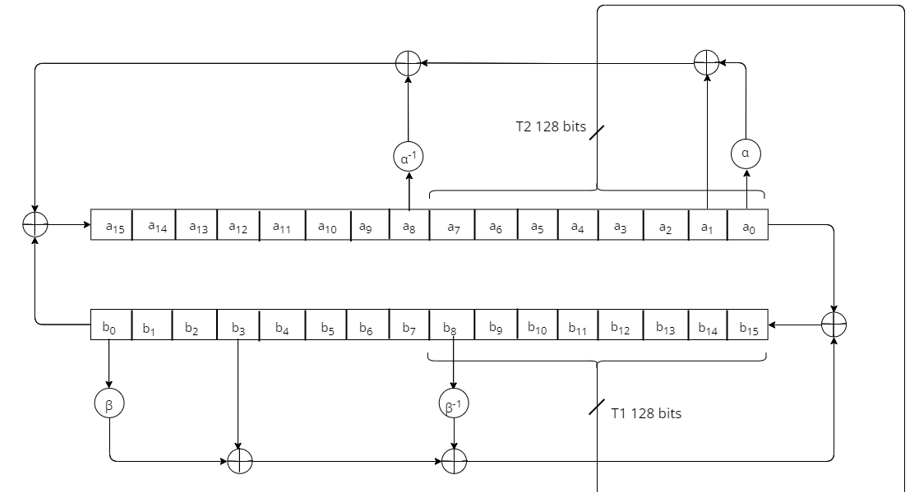


**Εικόνα 16**: LFSRupdate() SNOW-V

## Γενική Μορφή Σχεδιασμού

### LFSR

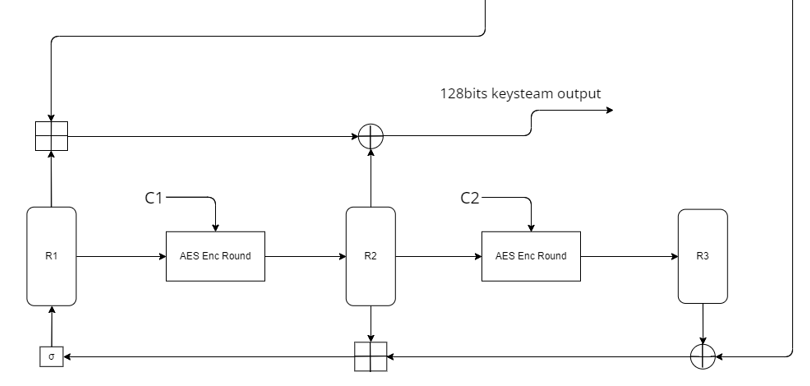
Με βάση την αρχιτεκτονική που έχει προτείνει ο ίδιος ο Patrik Ekdahl, το μέρος που αποκαλείται LFSR αποτελείται από δύο LFSRs: τον LFSR-A και τον LFSR-B. Ο καθένας έχει 16 κελιά των 16 bits το καθένα και συμβολίζονται ως a15 έως a0 για τον LFSR-A, και b15 έως b0 για τον LFSR-B αντίστοιχα. Έχει ως εισόδους το αρχικό διάνυσμα (IV) μήκους 128 bits και το κλειδί μήκους 256 bits που εισάγονται στους δύο LFSR. Σε κάθε κύκλο ρολογιού έχουμε ανανέωση των στοιχείων a15 και b15 με βάση τις εκφράσεις στις γραμμές 3,4 του ψευδοκώδικα της Εικόνας 9. Σε κάθε LFSRupdate() χρονίζουμε τους δύο LFSRs 8 φορές, ανανεώνοντας 256 bits από τα συνολικά 512. Ταυτόχρονα, τα 8 λιγότερο σημαντικά κελιά του LFSR-A και τα 8 περισσότερο σημαντικά κελιά του LFSR-B αποτελούν τα διανύσματα T2 και T1 αντίστοιχα, τα οποία ενημερώνονται πλήρως σε κάθε ενημέρωση των LFSR και θα αποτελούν τις εισόδους στο δεύτερο μέρος, το FSM. Συνεπώς, T1 = [b15, b14, b13, b12, b11, b10, b9, b8] και T2 = [a7, a6, a5, a4, a3, a2, a1, a0]. Ο σχεδιασμός του LFSR μέρους απεικονίζεται στην Εικόνα 17.



**Εικόνα 17**: LFSR (Linear Feedback Shift Register)

### FSM

Το μέρος FSM δέχεται ως είσοδο τα Τ1, Τ2 από το LFSR part και έχει ως έξοδο 128 bits που είναι η έξοδος ακολουθίας κλειδιών (keystream output). Αποτελείται από τρεις καταχωρητές (R1, R2, R3) των 128 bits ο καθένας, δυο λογικές πύλες XOR με σύμβολο το , ένα γύρο μετάθεσης των bytes (Permutation Byte-Oriented), δύο γύρους κωδικοποίησης AES καθώς και δύο προσθέσεις modulo 32, όπως φαίνεται στην Εικόνα 18.



**Εικόνα 18**: FSM (Finite State Machine)

Στη συνέχεια θα περιγράψουμε τη λειτουργία των συστατικών μερών του FSM.

AES: Δέχεται ως εισόδους, την έξοδο είτε του καταχωρητή R1, είτε του R2, που είναι 128 bits η κάθε μία, και ένα κλειδί των 128 bits, όπου εδώ τα συμβολίζουμε με C1 και C2 αντίστοιχα για τους δύο γύρους AES. Τα C1, C2 στην συγκεκριμένη περίπτωση έχουν την τιμή 0 σε όλη την διάρκεια εκτέλεσης του αλγόριθμου. Τα βήματα σε κάθε AES είναι τα εξής:

1. SubBytes (Αλλαγή Bytes): Σε αυτή την φάση του AES χωρίζουμε τα 128 bits της εισόδου σε bytes, δηλαδή σε ομάδες των 8 bits. Έπειτα, ανάλογα με την τιμή που έχουν τα bytes, τα μετατρέπουμε στα αντίστοιχα που ορίζει το παρακάτω κουτί, το λεγόμενο S-Box (Εικόνα 19). Με αυτόν τον τρόπο έχουμε καταφέρει να μην έχουμε πλέον την αρχική πληροφορία που είναι ευανάγνωστη, αλλά πλέον την έχουμε κωδικοποιήσει. Τα δεδομένα εισέρχονται ως είσοδος στην δεύτερη φάση που είναι η ολίσθηση σειρών.

Εικόνα που περιέχει ορθογώνιο παραλληλόγραμμο, τετράγωνο, συμμετρία, στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 19**: S-Box [25]

1. ShiftRows (Ολίσθηση σειρών): Το επόμενο στάδιο του AES αφορά την κυκλική ολίσθηση των σειρών έτσι ώστε να υπάρξει ανακάτεμα των τιμών για να μην είναι εύκολη η ανάκτηση δεδομένων από αγνώστους.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, διάγραμμα, ορθογώνιο παραλληλόγραμμο

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 20**: Shift Rows [26]

Όπως παρατηρούμε και από την Εικόνα 20, η πρώτη σειρά bytes δεν ολισθαίνει καθόλου, η δεύτερη ολισθαίνει κυκλικά κατά μία θέση, η τρίτη κατά δύο θέσεις και η τελευταία κατά τρεις θέσεις. Το Κ0,0 αποτελεί το λιγότερο σημαντικό byte, το Κ1,0 είναι το δεύτερο λιγότερο σημαντικό byte κ.ο.κ, μέχρι το Κ3,3 που είναι το πιο σημαντικό.

1. MixColumns (Ανάμιξη Στηλών): Στο στάδιο αυτό κάνουμε πολλαπλασιασμό, στοιχείο προς στοιχείο, του μητρώου που δημιουργήθηκε μετά από την ολίσθηση γραμμών, με το μητρώο της Εικόνας 21.

Εικόνα που περιέχει λευκό, γραμματοσειρά, σχεδίαση

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 21**: Μητρώο Πολλαπλασιασμού

Το τελικό αποτέλεσμα, το κάνουμε XOR με το αντίστοιχο κλειδί.

Permutation (Μετάθεση): Κατά την μετάθεση των bytes, αυτό που συμβαίνει είναι ότι αλλάζει η σειρά που είναι ταξινομημένα τα bytes στα 128 bits του data bus. Η νέα σειρά που θα έχουν καθορίζεται από τη συνάρτηση μετάθεσης:

σ = [ 0 , 4 , 8 , 12 , 1 , 5 , 9 , 13 , 2 , 6 , 10 , 14 , 3 , 7 , 11 , 15 ].

Έτσι, μετακινούμε το byte 0 στην θέση 0, το byte 4 στην θέση 0, το byte 8 στην θέση 2 και το ίδιο συνεχίζεται και για τα υπόλοιπα στοιχεία.

Modulo 32 addition: Για την πρόσθεση modulo 32 χωρίζουμε τις δύο εισόδους των 128 bits, σε υπολέξεις, δηλαδή σε ακολουθίες των 32 bits. Κάνουμε παράλληλη πρόσθεση με κρατούμενο τις αντίστοιχες υπολέξεις από τις δύο εισόδους χωρίς όμως να διαδίδουμε το κρατούμενο από τις προσθέσεις των λιγότερο σημαντικών υπολέξεων σε αυτές των περισσότερο σημαντικών.

### Initialization stage

Κατά το στάδιο της αρχικοποίησης, το κλειδί K των 256 bits χωρίζεται σε ομάδες των 16 bits ως εξής:

K = (k15, k14,……, k1, k0),

όπου στο k15 βρίσκονται τα πιο σημαντικά bits και στο k0 τα λιγότερο σημαντικά. Αντίστοιχα γίνεται το ίδιο και για το αρχικό διάνυσμα IV των 128 bits:

IV = (iv7, iv6,……, iv1, iv0).

Στην συνέχεια, αυτές οι τιμές πρέπει να φορτωθούν στα κελιά των LFSR όπως αναλύθηκαν στο 5.2.1. Για το LFSR-A έχουμε:

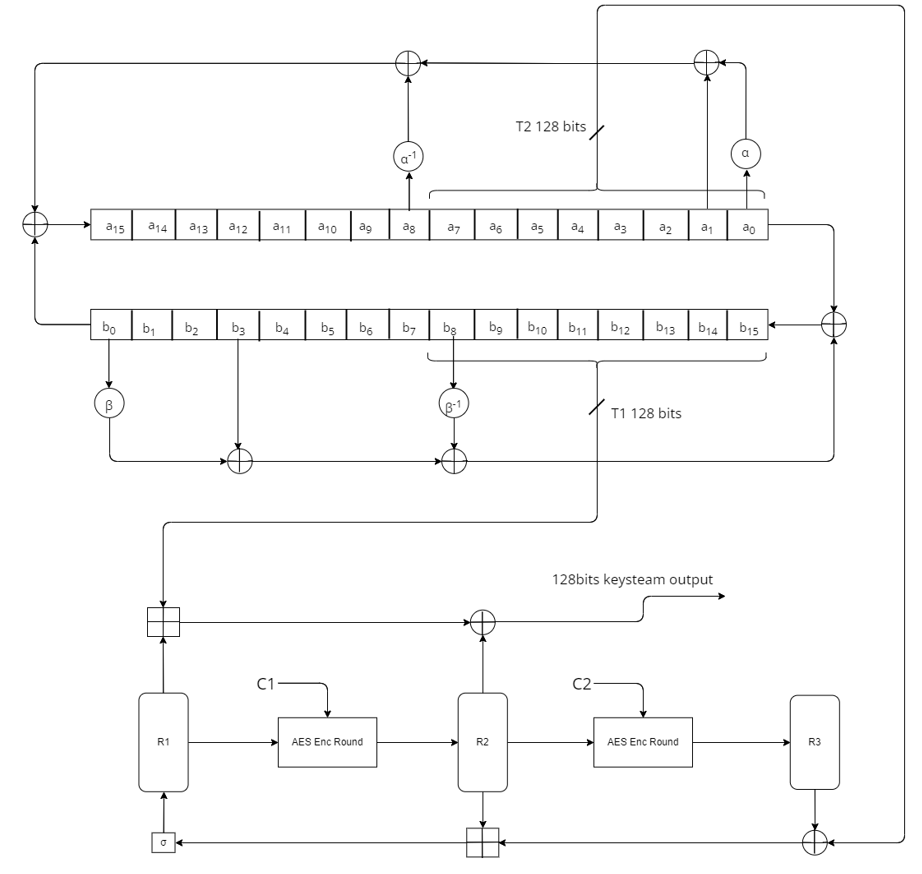
(a15, a14,……, a1, a0) = (k7, k6,……., k0, iv7, iv6,……, iv0)

και για το LFSR-B αντίστοιχα:

(b15, b14,……, b1, b0) = (k15, k14,……., k8, 0, 0,……, 0).

Έπειτα, η αρχικοποίηση αποτελείται από 16 επαναλήψεις, όπου ενημερώνεται ο stream cipher όπως και στο running-key mode, δηλαδή στο στάδιο της παραγωγής των keystreams, με την διαφορά ότι σε κάθε επανάληψη, κάνουμε XOR της εξόδου που παράγεται από το FSM με τα 8 πιο σημαντικά κελιά του LFSR-A. Επιπροσθέτως, στις τελευταίες δύο επαναλήψεις κάνουμε XOR το κλειδί με το διάνυσμα που προκύπτει μετά το permutation στον R1.

Ο συνολικός σχεδιασμός του αλγόριθμου SNOW-V απεικονίζεται στην Εικόνα 22.



**Εικόνα 22**: Συνολικός Σχεδιασμός SNOW-V

## Επιθέσεις στον SNOW-V

### Επιθέσεις Αρχικοποίησης

Έχοντας αναλύσει τους προηγούμενους αλγόριθμους καθώς και τις απαιτήσεις της νέας γενιάς των δικτύων, είναι φανερό ότι ο αλγόριθμος SNOW-V πρέπει να είναι ανθεκτικός σε αρκετά είδη επιθέσεων προκειμένου να μπορέσει να ανταπεξέλθει στις νέες προκλήσεις, κρατώντας ασφαλή τα συστήματα τηλεπικοινωνιών. Το πρώτο είδος επιθέσεων στο οποίο ο αλγόριθμος είναι ανεκτικός, είναι αυτό της αρχικοποίησης. Αυτές εφαρμόζονται είτε στα δυαδικά ψηφία του διανύσματος αρχικοποίησης είτε και σε αυτά του κλειδιού. Δεν θα μπορούσε ο επιτιθέμενος με ευκολία να εξάγει αποτελέσματα από το στάδιο της αρχικοποίησης μιας και αποτελείται από 16 γύρους και σε κάθε έναν εκτελούνται δύο

επιπλέον γύροι AES κρυπτογράφησης. Άρα, θεωρείται ανθεκτικός σε αυτού του είδους τις επιθέσεις.

### Επιθέσεις Ανταλλαγής Χρόνου/Μνήμης/Δεδομένων

Ο επόμενος τύπος επιθέσεων ονομάζεται Αντάλλαγμα Χρόνου/Μνήμης/Δεδομένων. Είναι μια γενική μέθοδος κατά την οποία γίνεται προσπάθεια αντιστροφής των κρυπταλγόριθμων, μέσω εξισορρόπησης του χρόνου που ξοδεύτηκε, της μνήμης που απαιτήθηκε και των δεδομένων που αποκτήθηκαν. Η λογική πίσω από αυτού του είδους τις επιθέσεις, είναι ότι σε πολλούς κρυπταλγόριθμους ροής, καθίσταται δυνατή η αντιστροφή της εσωτερικής ενημέρωσης καταστάσεων. Αν καταφέρει ο επιτιθέμενος να ανακτήσει μια κατάσταση σε μια συγκεκριμένη χρονική στιγμή, τότε θα μπορεί να ανακτήσει και όλες τις προηγούμενες καθώς και τις επόμενες. Συγκεκριμένα στον SNOW-V, δεν υπάρχουν προφανείς τρόποι για ανακατασκευή της εσωτερικής κατάστασης, μιας και αποτελείται από συνολικά 896 δυαδικά ψηφία, - εκ των οποίων τα 512 είναι οι δύο LFSRs και τα υπόλοιπα 348 αφορούν τους τρεις registers – αριθμός ο οποίος αντιστοιχεί σε μέγεθος 3.5 φορές του μυστικού κλειδιού. Εξαιτίας αυτού του μεγέθους, ικανοποιείται και ο κανόνας που έχει προκύψει από αυτού του είδους τις επιθέσεις, ο οποίος αναφέρει ότι για να θεωρηθεί ασφαλής ένας κρυπταλγόριθμος ροής από επιθέσεις Ανταλλάγματος Χρόνου/Μνήμης/Δεδομένων, οφείλει να έχει μέγεθος εσωτερικής κατάστασης τουλάχιστον δύο φορές το μέγεθος του μυστικού κλειδιού. Ακόμα και αν με κάποιο τρόπο κατάφερναν να ανακατασκευάσουν μια εσωτερική κατάσταση, δεν θα μπορούσαν να εξαγάγουν πολλές πληροφορίες για το μυστικό κλειδί ή τα keystreams και αυτό οφείλεται στο «καμουφλάρισμα» του κλειδιού στον καταχωρητή R1 κατά τους δύο τελευταίους γύρους της αρχικοποίησης. Γίνεται, λοιπόν, αντιληπτό ότι σε αυτού του είδους τις επιθέσεις ο αλγόριθμος SNOW-V είναι ανθεκτικός.

### Επιθέσεις Μάντεψε-και-Καθόρισε

Οι Μάντεψε-και-Καθόρισε επιθέσεις είναι αρκετά διαδομένες και αποτελούν μία, ακόμη, απειλή για τους κρυπταλγόριθμους ροής. Σε αυτές, ο επιτιθέμενος μαντεύει μέρος της κατάστασης στην οποία βρίσκεται το σύστημα και καθορίζει την τιμή των άλλων μερών του συστήματος, μέσω της εξίσωσης που ορίζει το keystream. Ο στόχος των επιθέσεων αυτών, είναι να καθοριστούν όσο το δυνατόν περισσότερα bits και να προβλεφθούν όσο γίνεται λιγότερα bits μέσω της εξίσωσης του keystream. Πιο συγκεκριμένα, στον SNOW-V, στην εξίσωση που ορίζεται το keystream, z(t) = (R1(t) 32 T1(t)) R2(t), υπάρχουν 3 τιμές των 128 bits η κάθε μία, οι οποίες είναι άγνωστες. Για κάθε keystream που θα προσπαθήσει ο επιτιθέμενος να βρει την τιμή του, θα πρέπει να μαντέψει τις 2 από τις 3 τιμές, δηλαδή 256 bits, κάτι το οποίο είναι πολύ δύσκολο να συμβεί, καθιστώντας τον SNOW-V ασφαλή από τέτοιου είδους επιθέσεις.

### Αλγεβρικές Επιθέσεις

Σε μία αλγεβρική επίθεση, ο επιτιθέμενος στοχεύει είτε το κλειδί, είτε την εσωτερική κατάσταση. Εάν στοχεύσει στο κλειδί, τότε απαιτείται να έρθει αντιμέτωπος με τα 16 στάδια της αρχικοποίησης, μαζί με δύο εξισώσεις που ορίζουν το keystream. Αν στοχεύσει στην εσωτερική κατάσταση, τότε προκειμένου να αντλήσει δεδομένα θα πρέπει να έρθει αντιμέτωπος με 7 keystream blocks, δηλαδή 7 \* 128 bits. Αυτά τα δυαδικά ψηφία θα έχουν υποστεί και επεξεργασία από modulo 32 προσθέσεις, καθώς και αρκετούς γύρους AES κρυπτογράφησης. Και στις δύο περιπτώσεις, προκύπτουν μη γραμμικές εξισώσεις, καθιστώντας πολύ δύσκολη την εξαγωγή δεδομένων από τον SNOW-V. Συνεπώς, είναι αρκετά ασφαλής απέναντι σε αλγεβρικές επιθέσεις.

# 

# Προτεινόμενος Σχεδιασμός SNOW-V

## Βελτιστοποίηση Σχεδιασμού του SNOW-V

Όπως γίνεται εύκολα αντιληπτό από την περιγραφή λειτουργίας του SNOW-V, εάν σε κάθε ενημέρωση των LFSR χρονίζεται ο stream cipher 8 φορές, τότε μόνο για το στάδιο της αρχικοποίησης χρειαζόμαστε 8 x 16 = 128 κύκλους ρολογιού. Αν και το running-key mode εκτελεστεί άλλες 8 φορές για την παραγωγή των keystreams, τότε προστίθενται στους 128 κύκλους ρολογιού άλλοι 8 x 8 = 64, ανεβάζοντας τον αριθμό των απαιτούμενων κύκλων ρολογιού σε 192.

Με τον σχεδιασμό της παρούσας διπλωματικής, έχει γίνει εφικτή η μείωση αυτού του αριθμού σε 16 για το initialization και σε 8 σε running-key mode για την παραγωγή 8 keystreams, άρα συνολικά 24 κύκλοι ρολογιού. Η μεγάλη αυτή μείωση των 168 κύκλων εξοικονομεί σημαντικό χρόνο από την εκτέλεση του αλγορίθμου κάνοντας τον stream cipher ταχύτερο. Ταυτόχρονα, καταφέρνει να μειώσει σε τεράστιες ποσότητες την κατανάλωση ρεύματος που απαιτείται μιας και θα λειτουργεί για λιγότερο χρονικό διάστημα.

Η βελτίωση αυτή των επιδόσεων επιτεύχθηκε μέσω της παραλληλοποίησης του LFSR μέρους, όπου χωρίζουμε τα LFSR-A και LFSR-B σε δύο καταχωρητές το καθένα. Οι δύο θα έχουν τα 128 περισσότερο σημαντικά δυαδικά ψηφία των LFSR-A και LFSR-B, ενώ οι άλλοι δύο τα 128 λιγότερο σημαντικά. Η έξοδος των καταχωρητών με τα περισσότερο σημαντικά ψηφία, θα αποτελεί την είσοδο των καταχωρητών των λιγότερο σημαντικών ψηφίων (Εικόνα 17).

Εικόνα που περιέχει διάγραμμα, Σχέδιο, τεχνικό σχέδιο, κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 23**: Προτεινόμενος Σχεδιασμός LFSR

## Λειτουργία Προτεινόμενου Σχεδιασμού

Σε αυτό το σημείο θα επεξηγήσουμε τον τρόπο λειτουργίας του προτεινόμενου σχεδιασμού, καθώς και τις διαφορές του με τον συμβατικό σχεδιασμό που περιεγράφηκε στο υποκεφάλαιο 5.5. Όσον αφορά την ανανέωση των στοιχείων, αυτή γίνεται μέσω του υποσυστήματος Multiply\_XOR όπως φαίνεται και στην Εικόνα 17. Πλέον, δε γίνεται ολίσθηση όλων των τιμών κατά μία θέση σε κάθε κύκλο ρολογιού και ενημέρωση μόνο των a15 και b15 που αποτελούνται από 16 δυαδικά ψηφία το καθένα, όπως στον προηγούμενο σχεδιασμό, αλλά σε κάθε κύκλο ρολογιού ενημερώνονται τα 128 πιο σημαντικά bits, τα οποία βρίσκονται στους καταχωρητές AH (LFSR-A High), BH (LFSR-B high), όπως αυτοί απεικονίζονται στην Εικόνα 17.

Το υποσύστημα Multiply\_XOR είναι υπεύθυνο να εκτελέσει τις απαραίτητες πράξεις για την ανατροφοδότηση του συστήματος. Με βάση τον προηγούμενο σχεδιασμό, σε κάθε κύκλο ρολογιού έπρεπε να εκτελεστεί το παρακάτω σύνολο πράξεων για το LFSR-A:

a15 = b0 + αa0 + a1 + α-1 a8 mod gA(α), (1)

και το παρακάτω για το LFSR-B:

b15 = a0 + βb0 + b3 + β-1 b8 mod gΒ(β). (2)

Στο υποσύστημα του προτεινόμενου σχεδιασμού, οι πράξεις αυτές γίνονται παράλληλα 8 φορές σε κάθε κύκλο ρολογιού, δηλαδή ανανεώνονται 128 bits. Για την δημιουργία των α, β

χρησιμοποιούνται οι παρακάτω υπομονάδες της Εικόνας 24. Αρχικά, η είσοδος ολισθαίνει αριστερά κατά μία θέση, και γίνεται η λογική πράξη XOR με τους συντελεστές (coefficients) 0x990f και 0xc963 αντίστοιχα για τα α, β. Αν το πιο σημαντικό ψηφίο της εισόδου είναι ‘1’, τότε στην έξοδο βγαίνει το αποτέλεσμα της πράξης XOR, αλλιώς αν είναι ‘0’ βγαίνει η ολισθημένη τιμή της εισόδου.

Εικόνα που περιέχει κείμενο, διάγραμμα, Σχέδιο, τεχνικό σχέδιο

Περιγραφή που δημιουργήθηκε αυτόματαp

**Εικόνα 24**: Αρχιτεκτονική α, β

Αντίστοιχα για την δημιουργία των α-1, β-1 χρησιμοποιούνται οι παρακάτω υπομονάδες (Εικόνα 25). Η είσοδος ολισθαίνει μία φορά δεξιά και έπειτα γίνεται XOR με τους συντελεστές 0xcc87 και 0xe4b1 αντίστοιχα για τα α-1 και β-1. Αν το λιγότερο σημαντικό ψηφίο της εισόδου είναι ‘1’ τότε στην έξοδο περνάει το αποτέλεσμα της πράξης XOR, ενώ αν είναι ΄0΄ περνάει η ολισθημένη τιμή.

Εικόνα που περιέχει κείμενο, διάγραμμα, Σχέδιο, τεχνικό σχέδιο

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 25**: Αρχιτεκτονική α-1, β-1

Για την επίτευξη της ανανέωσης των τιμών των a15,b15 χρειάζεται να εκτελεστούν όλες οι απαραίτητες πράξεις των σχέσεων (1) και (2). Αυτό θα γίνει με την βοήθεια των λογικών πυλών XOR. Ο παρακάτω σχεδιασμός χρησιμοποιείται 8 φορές για την παραγωγή των απαραίτητων τιμών για τον AH καταχωρητή.

Εικόνα που περιέχει διάγραμμα, γραμμή, σκίτσο/σχέδιο, Σχέδιο

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 26**: Multiply\_XOR BLOCK-A

Οι απαραίτητες μεταβλητές που θα χρησιμοποιηθούν στο κάθε ένα από τα 8 BLOCK-A για την παραγωγή των τιμών δίνονται από τον τύπο:

aA,i+8 = (αai ai+1 α-1ai+8 bi), με i = {0,1….,7}.

Αντίστοιχα, για την παραγωγή των τιμών του καταχωρητή BH εκτελείται ο παρακάτω σχεδιασμός 8 φορές.

Εικόνα που περιέχει διάγραμμα, γραμμή, σκίτσο/σχέδιο, λευκό

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 27**: Multiply\_XOR BLOCK-B

Οι απαραίτητες μεταβλητές που θα χρησιμοποιηθούν στο κάθε ένα από τα 8 BLOCK-B για την παραγωγή των τιμών δίνονται από τον τύπο:

bB,i+8 = (βbi βi+3 β-1bi+8 ai), με i = {0,1….,7}.

Ο σχεδιασμός του FSM part παραμένει ο ίδιος με τον σχεδιασμό που προτείνεται από τον Patrik Ekdahl.

# 

# Προσομοίωση και Υλοποίηση Σχεδιασμού

## Εξομοίωση Λειτουργίας Προτεινόμενου Σχεδιασμού

Για την εξομοίωση της λειτουργίας του SNOW-V έχουν επιλεχθεί test vectors τα οποία προκύπτουν από τον ίδιο τον σχεδιαστή του αλγορίθμου, καθώς και από σχετική βιβλιογραφία. Η εξομοίωση ξεκινάει με την αρχικοποίηση των καταχωρητών στην τιμή 0 για μισό κύκλο ρολογιού. Με την πρώτη θετική ακμοπυροδότηση του ρολογιού, οι registers περνάνε στην έξοδο την τιμή της εισόδου τους εκείνη την χρονική στιγμή και ξεκινάει η λειτουργία του αλγορίθμου.

Οι τιμές στα test vectors έχουν ταξινομηθεί από το least significant bit στο most significant bit, ενώ στην εξομοίωση η ταξινόμηση είναι ανάποδη. Επιπλέον, για την επαλήθευση των αποτελεσμάτων θα χρησιμοποιηθούν τα αποτελέσματα των πρώτων κύκλων ρολογιού και αυτών που παράγουν το keystream στις τελευταίες 8 επαναλήψεις.

**Test Vector #1**

KEY = 0x 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

IV = 0x 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Τα αναμενόμενα και προκύπτοντα αποτελέσματα του πρώτου test vector φαίνονται στην Εικόνα 28, και στις εικόνες 29 και 30 αντίστοιχα. Σύγκριση αναμενόμενων και προκυπτόντων αποτελεσμάτων παρουσιάζεται και για τα test vectors 2 και 3, στις Εικόνες 31-36.

Αναμενόμενα αποτελέσματα test vector #1:

**Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, ασπρόμαυρο

Περιγραφή που δημιουργήθηκε αυτόματα**

**Εικόνα 28**: Αναμενόμενα αποτελέσματα test vector #1 [27]

Αποτελέσματα προτεινόμενου σχεδιασμού:

Εικόνα που περιέχει στιγμιότυπο οθόνης, γραμμή

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 29**: Αποτελέσματα test vector #1 - Μέρος Α

**Εικόνα που περιέχει στιγμιότυπο οθόνης, λογισμικό πολυμέσων

Περιγραφή που δημιουργήθηκε αυτόματα**

**Εικόνα 30**: Αποτελέσματα test vector #1 - Μέρος Β

**Test Vector #2**

KEY = 0x ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

IV = 0x ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

Αναμενόμενα αποτελέσματα test vector #2:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, ασπρόμαυρο, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 31**: Αναμενόμενα αποτελέσματα test vector #2 [28]

Αποτελέσματα προτεινόμενου σχεδιασμού:

Εικόνα που περιέχει στιγμιότυπο οθόνης, λογισμικό πολυμέσων, λογισμικό, λογισμικό γραφικών

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 32**: Αποτελέσματα test vector #2 - Μέρος A

**Εικόνα που περιέχει στιγμιότυπο οθόνης, λογισμικό πολυμέσων

Περιγραφή που δημιουργήθηκε αυτόματα**

**Εικόνα 33**: Αποτελέσματα test vector #2 - Μέρος Β

**Test Vector #3**

KEY = 0x 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f 0a 1a 2a 3a 4a 5a 6a 7a 8a 9a aa ba ca da ea fa

IV = 0x 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10

Αναμενόμενα αποτελέσματα test vector #3:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, έγγραφο, ασπρόμαυρο

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 34**: Αναμενόμενα αποτελέσματα test vector #3 [28]

Αποτελέσματα προτεινόμενου σχεδιασμού:

Εικόνα που περιέχει στιγμιότυπο οθόνης, γραμμή

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 35**: Αποτελέσματα test vector #3 - Μέρος A

Εικόνα που περιέχει στιγμιότυπο οθόνης, γραμμή

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 36**: Αποτελέσματα test vector #3 - Μέρος Β

Τα test benches για την παραγωγή των αποτελεσμάτων βρίσκονται στο Παράρτημα A.

## Υλοποίηση Προτεινόμενου Σχεδιασμού

Τα αποτελέσματα του Πίνακα 1 προέκυψαν από την εκτέλεση της διαδικασίας Υλοποίησης (Implementation) του λογισμικού Vivado. Για τον καθορισμό της μέγιστης συχνότητας χρησιμοποιήθηκε το εργαλείο Constraints Wizard του Vivado όπου τέθηκε περιορισμός ρολογιού και η μέγιστη συχνότητα υπολογίστηκε με το worst negative slack. Ο υπολογισμός του throughput έγινε πολλαπλασιάζοντας τον αριθμό των bits που εξάγονται σε κάθε κύκλο ρολογιού, με την μέγιστη συχνότητα. Όλες οι παρακάτω πληροφορίες είναι διαθέσιμες στο “Utilization Report” και “Timing Report” του εργαλείου Vivado.

**Πίνακας 1:** Αποτελέσματα Υλοποίησης SNOW-V

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Αλγόριθμος** | **LUTs** | **FFs** | **Max Frequency (MHz)** | **Throughput (Gbps)** | **Device part** |
| SNOW-V | 2740/203128 | 896/406256 | 237 | 30,336 | xcku035-ffva1156-2-I |

## Σύγκριση Προτεινόμενου Σχεδιασμού με Αντίστοιχους Αλγόριθμους

Για την συγκριτική μέτρηση της απόδοσης του SNOW-V, επιλέχθηκαν αλγόριθμοι με το ίδιο security level, δηλαδή κλειδί των 256 bits. Μερικοί από αυτούς είναι ο ZUC-256, AES-256, Salsa20, Helix. Στο σημείο αυτό πρέπει να τονιστεί ότι η σύγκριση διαφορετικών αποτελεσμάτων υλοποίησης που βασίζονται σε διαφορετικές δομές FPGA και επιλογές υλοποίησης είναι σχεδόν αδύνατη και μερικές φορές άδικη (π.χ. σε ορισμένες περιπτώσεις οι προτεινόμενες αρχιτεκτονικές δεν είναι απαραίτητα καλύτερες ή χρησιμοποιούνται διαφορετικές τεχνικές βελτιστοποίησης). Ωστόσο, αυτές οι συγκρίσεις εξακολουθούν να έχουν αξία καθώς παρέχουν μια επισκόπηση της βιβλιογραφίας όσον αφορά τις υλοποιήσεις σε υλικό αλγορίθμων συγκρίσιμων με τον SNOW-V. Η σύγκριση τους παρουσιάζεται στον Πίνακα 2.

**Πίνακας 2**: Σύγκριση SNOW-V με άλλους κρυπταλγόριθμους ροής

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Αλγόριθμος** | **LUTs** | **Area (Slices)** | **Max Frequency (MHz)** | **Throughput (Gbps)** | **Device** |
| SNOW-V | 2740 | 283 | 237 | 30,336 | xcku035-ffva1156-2-I |
| ZUC-256[29] | - | 249 | 272 | 9 | V6  XC6vIx75t-3 |
| AES-256[30] | 15376 | - | 332,941 | 0,91 | Virtex-7 XC7VX690T |
| Salsa20 [31] | 958 | 266 | 175,22 | 1,12 | Spartan 6  XC6SLX75 |
| Helix [32] | - | 418 | 32 | 1,024 | Virtex-IITM 2V250FG256 |

Με βάση τα παραπάνω αποτελέσματα είναι εμφανές ότι ο προτεινόμενος αλγόριθμος του SNOW-V σε σχέση με τους υπόλοιπους stream ciphers συνδυάζει μειωμένη χρήση υλικού αλλά και μεγάλο throughput.

## Σύγκριση Αποτελεσμάτων με άλλες Υλοποιήσεις του SNOW-V

Στον Πίνακα 4 γίνεται σύγκριση μεταξύ της προτεινόμενης υλοποίησης του αλγόριθμου SNOW-V και άλλων υλοποιήσεων από την Βιβλιογραφία.

**Πίνακας 3**: Σύγκριση Αποτελεσμάτων με άλλες Υλοποιήσεις του SNOW-V στη Βιβλιογραφία

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Αλγόριθμος** | **LUTs** | **Area (Slices)** | **Max Frequency (MHz)** | **Throughput (Gbps)** | **Device** |
| Προτεινόμενος SNOW-V | 2740 | 283 | 237 | 30,336 | xcku035-ffva1156-2-I |
| SNOW-V [33] | 2109 | 530 | 224 | 2,6 | XC7A35T (BASYS3) |

Ο σχεδιασμός με τον οποίο συγκρίνεται ο σχεδιασμός της παρούσας διπλωματικής, ακολουθεί την λογική που πρότεινε ο Patrik Ekdahl. Για τους δύο LFSR, του LFSR μέρους του SNOW-V, χρησιμοποιούνται 16 κελιά των 16 bits, όπου σε κάθε κύκλο ρολογιού γίνεται ολίσθηση κατά μία θέση. Έτσι, μόνο για το στάδιο της αρχικοποίησης χρησιμοποιούνται 128 κύκλοι ρολογιού. Ταυτόχρονα, για να ενημερωθούν τα διανύσματα Τ1, Τ2 που αποτελούν τις εισόδους στο FSM μέρος, χρησιμοποιούνται 8 κύκλοι ρολογιού.

Αντιθέτως, στον προτεινόμενο σχεδιασμό του SNOW-V, η ενημέρωση των τιμών των LFSR γίνεται παράλληλα και σε κάθε κύκλο ρολογιού υπολογίζονται 8 διαφορετικές τιμές. Με αυτόν τον τρόπο, η ενημέρωση των τιμών των Τ1, Τ2 και κατ’ επέκταση του FSM μέρους, γίνεται 8 φορές πιο γρήγορα. Ως συμπέρασμα από τα παραπάνω εξάγεται ότι αυτή η μεγάλη διαφορά στην ταχύτητα, με την οποία λειτουργεί ο προτεινόμενος σχεδιασμός, έχει αντίκτυπο και στην καθυστέρηση του συστήματος, μειώνοντας την αρκετά. Λογικό επακόλουθο της μείωσης αυτής, είναι και η τεράστια διαφορά που αποτυπώνεται στο throughput, με τον προτεινόμενο σχεδιασμό να είναι εμφανώς πολύ ανώτερος του άλλου σχεδιασμού [33]. Επιπλέον, παρόλο που ο προτεινόμενος σχεδιασμός χρειάζεται περισσότερα LUTs, αυτό δεν αποτυπώνεται και στον αριθμό των slices που χρησιμοποιούνται σε κάθε ένα από τα FPGA, μειώνοντας έτσι το απαιτούμενο υλικό για τον σχεδιασμό. Για την μέγιστη συχνότητα λειτουργίας, η διαφορά είναι πολύ μικρή και δεν είναι ασφαλής η διεξαγωγή σύγκρισης μιας και δεν χρησιμοποιείται η ίδια συσκευή για τους δύο σχεδιασμούς.

Ένα άλλο συμπέρασμα που αντλούμε, είναι η έλλειψη των υλοποιήσεων του αλγορίθμου SNOW-V σε FPGA στη βιβλιογραφία. Σε συνδυασμό με τα αποτελέσματα του υποκεφαλαίου 7.3, που καθιστούν τον SNOW-V πιο αποδοτικό από αλγόριθμους παρόμοιούς του, επισημαίνεται η ανάγκη προτάσεων βελτιστοποίησής του.

# 

# Συμπεράσματα και Προοπτικές

Τα δίκτυα 5G έχουν ανοίξει νέους ορίζοντες στην επικοινωνία και την τεχνολογία και το μέλλον τους φαίνεται εξαιρετικά ενθαρρυντικό, καθώς θα συμβάλουν στην ανάπτυξη πληθώρας καινοτόμων εφαρμογών και υπηρεσιών. Η ανάπτυξη τεχνολογιών μεταξύ μηχανών και ανθρώπου αλλά και μηχανών μεταξύ τους θα γίνει πολύ πιο γρήγορη και φιλική στον χρήστη μέσω της αποστολής μεγάλων δεδομένων και την μείωση του latency. Παράλληλα, θα δημιουργηθούν νέες ευκαιρίες για βελτίωση της ποιότητας ζωής και οικονομική ανάπτυξη. Ταυτόχρονα, όμως, για γίνει δυνατή η πλήρης εκμετάλλευση των προνομίων των 5G δικτύων, θα πρέπει να ληφθούν υπόψιν οι διάφορες προκλήσεις ασφάλειας και ιδιωτικότητας που προκύπτουν.

Αρχικά, πολλές από τις επιθέσεις που στοχεύουν τα δίκτυα 4G θα είναι εξίσου επικρατείς και στα 5G δίκτυα, καθώς οι κόμβοι των δικτύων θα συνδυάζονται πλέον, προκαλώντας αύξηση του αριθμού των συσκευών που επικοινωνούν μεταξύ τους στο IoT. Συνεπώς, η αυθεντικοποίηση καθίσταται απαραίτητη στη μεταφορά μηνυμάτων των συσκευών, ώστε να επαληθεύεται η ταυτότητα του αποστολέα και του παραλήπτη. Καταλήγουμε πως η εξέλιξη και αναβάθμιση της κρυπτογραφίας γίνεται απαραίτητη, καθώς απαιτούνται νέες μέθοδοι κρυπτογράφησης για την ικανοποίηση των κριτηρίων που καλείται να καλύψει το 5G.

Καθώς στα ασύρματα δίκτυα 5G πολλοί από τους κόμβους θα είναι εικονικοποιημένοι, μειώνεται το διαθέσιμο εξειδικευμένο υλικό που μπορεί να χρησιμοποιηθεί για την υλοποίηση κρυπτογραφικών αλγορίθμων. Μια άλλη πρόκληση που πρέπει να ληφθεί υπόψη, είναι πως η οργάνωση 3GPP κινείται προς αλγορίθμους με κλειδιά μεγέθους 256 bit, με αποτέλεσμα αλγόριθμοι που χρησιμοποιούνται τωρινά να απορριφθούν για χρήση σε 5G δίκτυα. Ένας από τους πολλούς τρόπους για την αύξηση της ασφάλειας σε συστήματα με περιορισμένους πόρους είναι η χρήση κρυπτογραφικών λύσεων σε υλικό σε ηλεκτρονικές συσκευές. Με αυτόν τον τρόπο μειώνεται κατά πολύ η δυνατότητα κλοπής των δεδομένων όπου θα οδηγήσει σε αλλαγή της επιθυμητής λειτουργίας. Οι κρυπταλγόριθμοι ροής, ειδικά, αποτελούσαν πάντα μια ελκυστική λύση για προβλήματα ασφαλείας σε ασύρματα δίκτυα προηγούμενων γενεών.

Η παρούσα Διπλωματική Εργασία παρέχει μια αρκετά καλή πρόταση για την επίτευξη των επιθυμητών στόχων. Αφορά τον σχεδιασμό και υλοποίηση του κρυπταλγόριθμου ροής SNOW-V, που επί του παρόντος εξετάζεται από την 3GPP για τη χρήση του στο 5G. Ως κρυπταλγόριθμος με μήκος κλειδιού 256 bit πιθανώς να χρησιμοποιηθεί σε δίκτυα κινητών επικοινωνιών για να προσφέρουν προστασία από πιθανούς μελλοντικούς αντιπάλους που μπορούν να αξιοποιήσουν κβαντικούς υπολογιστές.

Ο προτεινόμενος σχεδιασμός, μέσω της παραλληλοποίησης του LFSR μέρους, επιτρέπει την αύξηση του πλάτους της ακολουθίας κλειδιών που υπολογίζεται κατά τη διάρκεια ενός κύκλου ρολογιού. Έκανε δυνατή τη μείωση του απαραίτητου υλικού, τη βελτίωση της συχνότητας λειτουργίας αλλά και την αύξηση του throughput.

Επιπλέον, τα αποτελέσματά μας επιδεικνύουν ότι ο SNOW-V έχει σημαντικό πλεονέκτημα έναντι αντίστοιχων αλγορίθμων. Αρχικά ίσως παρατηρείται μια ισοβαθμία μεταξύ τους, τόσο σε υλικό που χρησιμοποιείται όσο και σε συχνότητα λειτουργίας. Η μεγάλη, όμως, διαφορά βρίσκεται στο throughput καθώς η αύξηση που επιτυγχάνεται με τον αλγόριθμο SNOW-V αγγίζει μέχρι και το 3000%. Σε σχέση με άλλες υλοποιήσεις του αλγόριθμου SNOW-V, η μείωση του απαιτούμενου υλικού αγγίζει το 50%, η συχνότητα λειτουργίας είναι περίπου η ίδια, ενώ και πάλι η διαφορά στο throughput είναι τεράστια, καθώς παρατηρείται αύξηση κατά 1500%.

Μία από τις πολλές προοπτικές για εξέλιξη της παρούσας Διπλωματικής Εργασίας είναι να παραλληλοποιηθεί και το υποσύστημα του FSM ώστε σε κάθε κύκλο ρολογιού να διπλασιάζεται ο αριθμός των keystreams που παράγονται και να αυξηθεί έτσι στο διπλάσιο και το throughput. Μια άλλη ιδέα θα αποτελούσε η ανάπτυξη μιας υλοποίησης ανθεκτική σε side-channel επιθέσεις.

# Βιβλιογραφία

[1] G. P. Kaur, J. Birla και J. Ahlawat, «Generations of Wireless Technology,» International Journal of Computer Science and Management Studies, pp. 435-441, 2011.

[2] A. U. Gawas, «An overview on evolution of mobile wireless communication networks: 1G-6G,» International Journal on Recent and Innovation Trends in Computing and Communication, pp. 3130-3133, 2015.

[3] J. Pisarov και G. Mester, «The impact of 5G technology on life in 21st century,» IPSI BgD Transactions on Advanced Research (TAR), 2020.

[4] P. Popovski, K. F. Trillingsgaard, O. Simeone και G. Durisi, «5G Wireless Network Slicing for eMBB, URLLC, and mMTC: A Communication-Theoretic View,» IEEE Access, pp. 55765-55779, 2018.

[5] J. A. del Peral-Rosado, R. Raulefs, J. A. López-Salcedo και G. Seco-Granados, «Survey of Cellular Mobile Radio Localization Methods: From 1G to 5G,» IEEE Communications Surveys & Tutorials, pp. 1124-1148, 2018.

[6] I. Ahmad, S. Shahabuddin, T. Kumar, J. Okwuibe, A. Gurtov και M. Ylianttila, «Security for 5G and Beyond,» IEEE Communications Surveys & Tutorials, pp. 3682-3722, 2019.

[7] I. Ahmad, T. Kumar, M. Liyanage, J. Okwuibe, M. Ylianttila και A. Gurtov, «Overview of 5G Security Challenges and Solutions,» IEEE Communications Standards Magazine, pp. 36-43, 2018.

[8] Y. Inan, «Analyzing The Classic Caesar Method Cryptography,» σε 4th International Conference on Computational Mathematics and Engineering Sciences (CMES-2019), 2019, pp. 213-220.

[9] W.Stallings, Cryptography and Network Security: Principles and Practice, Pearson Higher Education, 2020.

[10] GDPR.EU, «GDPR.EU,» Protagon AG, [Ηλεκτρονικό]. Available: https://gdpr.eu/tag/gdpr/. [Πρόσβαση 2 Ιουνίου 2023].

[11] S. E. Donaldson, S. G. Siegel, C. K. Williams, A. Aslam, S. E. Donaldson, S. G. Siegel, C. K. Williams και A. Aslam, «Cybersecurity frameworks,» Enterprise Cybersecurity: How to Build a Successful Cyberdefense Program Against Advanced Threats, pp. 297-309, 2015.

[12] F. Maqsood, M. Ahmed, M. M. Ali και M. A. Shah, «Cryptography: a comparative analysis for modern techniques,» International Journal of Advanced Computer Science and Applications, 2017.

[13] D. Walkowski, «F5 Labs,» 8 Ιούνιος 2019. [Ηλεκτρονικό]. Available: https://www.f5.com/labs/learning-center/what-is-the-cia-triad. [Πρόσβαση 6 Ιουνίου 2023].

[14] K. S. Mohamed, Cryptography Concepts: Confidentiality, Springer International Publishing, 2020.

[15] R. Cramer, I. Damgård και Y. Ishai, «Theory of Cryptography,» σε Proceedings of the 9th Theory of Cryptography Conference, TCC, Springer, 2012.

[16] J. Cederlof και J.-Å. Larsson, «Security Aspects of the Authentication Used in Quantum Cryptography,» IEEE Transactions on Information Theory, pp. 1735-1741, April 2008.

[17] J. T. Force, «Security and privacy controls for information systems and organizations,» National Institute of Standards and Technology, 2017.

[18] R. A. Rueppel, Analysis and design of stream ciphers, Springer Science & Business Media, 2012.

[19] L. Jiao, Y. Hao και D. Feng, «Stream cipher designs: a review,» Science China Information Sciences, pp. 1-25, 2020.

[20] Microcontrollerslab, «INTRODUCTION TO FIELD PROGRAMMABLE GATE ARRAYS (FPGA),» [Ηλεκτρονικό]. Available: https://microcontrollerslab.com/fpga-introduction-block-diagram/. [Πρόσβαση 15 Ιουνίου 2023].

[21] P. Ekdahl και T. Johansson, «A new version of the stream cipher SNOW,» σε Selected Areas in Cryptography: 9th Annual International Workshop, SAC 2002 St. John’s, Newfoundland, Canada, August 15--16, 2002 Revised Papers 9, Springer, 2003, pp. 47-61.

[22] D. Watanabe, A. Biryukov και C. De Canniere, «A distinguishing attack of SNOW 2.0 with linear masking method,» σε Selected Areas in Cryptography: 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14-15, 2003. Revised Papers 10, Ottawa, Springer, 2004, pp. 222-233.

[23] J. Yang, T. Johansson και A. Maximov, «Vectorized linear approximations for attacks on SNOW 3G,» IACR Transactions on Symmetric Cryptology, pp. 249-271, 2019.

[24] B. Debraize και I. M. Corbella, «Fault Analysis of the Stream Cipher Snow 3G,» σε 2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), Lussane, 2009, pp. 103-110.

[25] M. T. Tran, D. K. Bui και A. D. Duong, «Gray S-Box for Advanced Encryption Standard,» σε 2008 International Conference on Computational Intelligence and Security, China, IEEE, 2008, pp. 253-258.

[26] R. Riyaldhi, Rojali και A. Kurniawan, «Improvement of Advanced Encryption Standard Algorithm With Shift Row and S.Box Modification Mapping in Mix Column,» Procedia Computer Science, pp. 401-407, 2017.

[27] P. Ekdahl, T. Johansson, A. Maximov και J. Yang, «A new SNOW stream cipher called SNOW-V,» Cryptology ePrint Archive, 2018.

[28] J. Yang, «Contributions to Confidentiality and Integrity Algorithms for 5G,» Lund University, 2021.

[29] A. Xu, Y. Wu, J. Yang, M. Zhu, Q. Zhao και L. Liu, «A High-throughput Hardware Implementation of ZUC-256 Stream Cipher,» σε 2022 4th International Conference on Communications, Information System and Computer Engineering (CISCE), 2022, pp. 24-27.

[30] N. S. Srinivas και M. Akramuddin, «FPGA based hardware implementation of AES Rijndael algorithm for Encryption and Decryption,» σε 2016 international conference on electrical, electronics, and optimization techniques (ICEEOT), 2016, pp. 1769-1776.

[31] B. Mazumdar, S. S. Ali και O. Sinanoglu, «A compact implementation of Salsa20 and its power analysis vulnerabilities,» ACM Transactions on Design Automation of Electronic Systems (TODAES), pp. 1-26, 2016.

[32] M. D. Galanis, P. Kitsos, G. Kostopoulos, N. Sklavos και C. E. Goutis, «Comparison of the Hardware Implementation of Stream Ciphers.,» Int. Arab J. Inf. Technol., pp. 267-274, 2005.

[33] L. Pyrgas και P. Kitsos, «5G Security: FPGA Implementation of SNOW-V Stream Cipher,» σε 2021 24th Euromicro Conference on Digital System Design (DSD), Palermo, IEEE Xplore, 2021, pp. 381-384.

**Παράρτημα Α: Πρόγραμμα Σύνθεσης και Εξομοίωσης Xilinx Vivado 2022.2**

Το Vivado 2022.2 είναι ένα λογισμικό, το οποίο δημιουργήθηκε από την Xilinx (σημερινή AMD) για τον σχεδιασμό, την ανάλυση, την εξομοίωση και την σύνθεση κυκλωμάτων, που έχουν αποτυπωθεί από γλώσσες περιγραφής υλικού. Επιπλέον, μπορεί να δημιουργεί την ακολουθία από bits, που αντιστοιχεί στον σχεδιασμό που δημιουργήθηκε, με σκοπό να φορτωθεί σε κάποιο FGPA.

Αυτό το λογισμικό χρησιμοποιήθηκε στην παρούσα εργασία και παρακάτω παρουσιάζονται κάποια βασικά βήματα χρήσης του.

1. **Δημιουργία νέου έργου**

Για την δημιουργία ενός νέου έργου, επιλέγετε το *Create Project* κάτω από την ετικέτα *Quick Start* όπως φαίνεται στην παρακάτω εικόνα.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, οθόνη

Περιγραφή που δημιουργήθηκε αυτόματα

Στο παράθυρο που εμφανίζεται, επιλέγετε *Next*.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, εικονίδιο υπολογιστή

Περιγραφή που δημιουργήθηκε αυτόματα

Αφού επιλεχθεί σε ποια θέση θα αποθηκευτεί το αρχείο και με ποιο όνομα, το επόμενο βήμα είναι να καθοριστεί ο τύπος του έργου που θα δημιουργηθεί. Στο παρόν έργο, είναι ένα *RTL Project* και επιλέγετε *Next* για να προχωρήσετε παρακάτω.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, εικονίδιο υπολογιστή

Περιγραφή που δημιουργήθηκε αυτόματα

Στην συνέχεια, επιλέγετε ένα κομμάτι που ανήκει στην οικογένεια της Xilinx, το οποίο να είναι κατάλληλο για το έργο και επιλέγετε *Next*.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, εικονίδιο υπολογιστή

Περιγραφή που δημιουργήθηκε αυτόματα

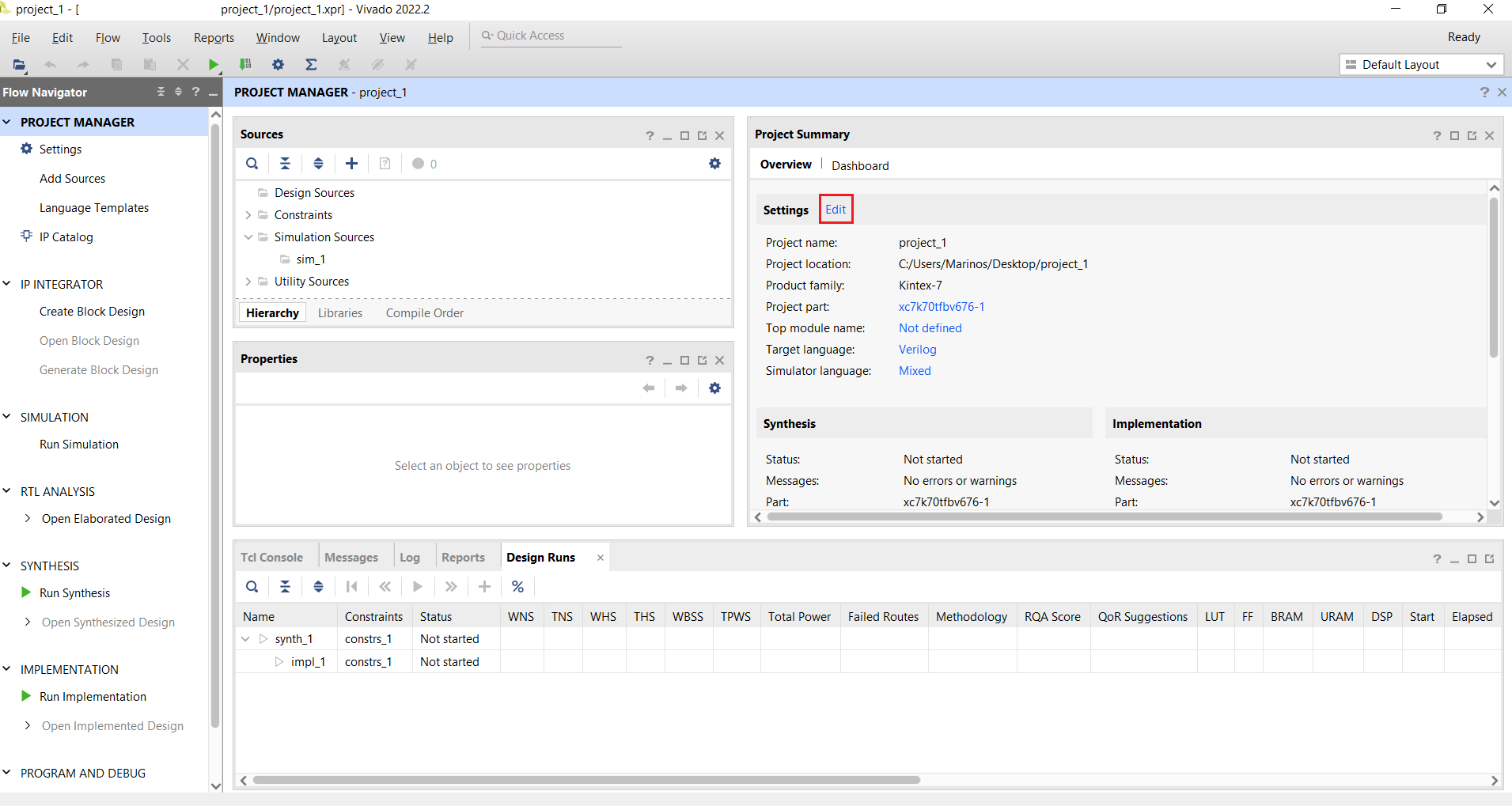
Το τελευταίο παράθυρο που εμφανίζεται, δίνει μια σύνοψη των επιλογών που έχουν γίνει. Για οριστικοποίηση και δημιουργία του έργου επιλέγετε το *Finish*.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, εικονίδιο υπολογιστή

Περιγραφή που δημιουργήθηκε αυτόματα

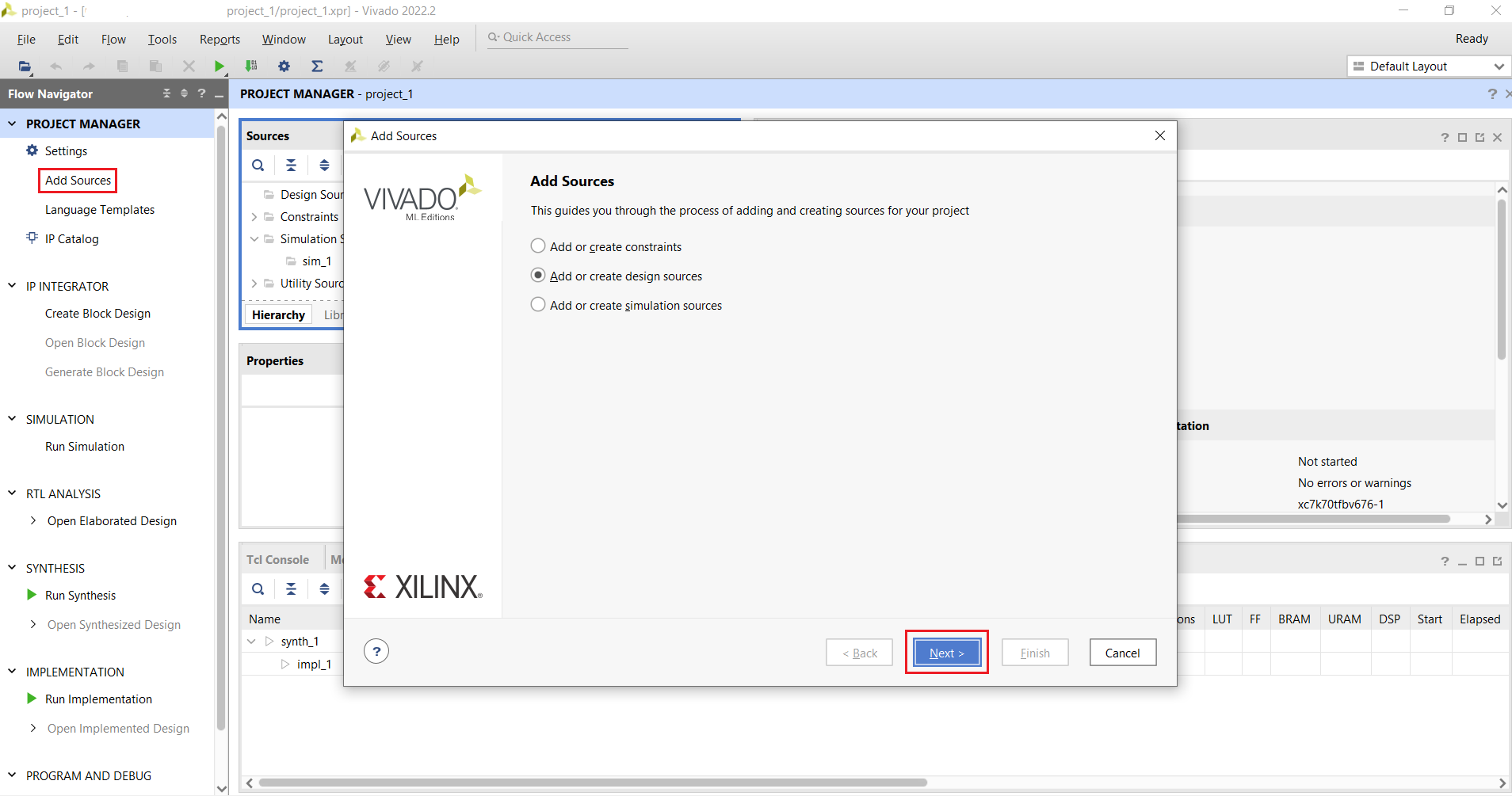
1. **Ρυθμίσεις διεπαφής**

Επιλέγοντας *Edit* είναι δυνατή η αλλαγή των ρυθμίσεων της διεπαφής και του έργου, όπως η γλώσσα που θα χρησιμοποιηθεί, ποια οικογένεια της Xilinx κ.α.

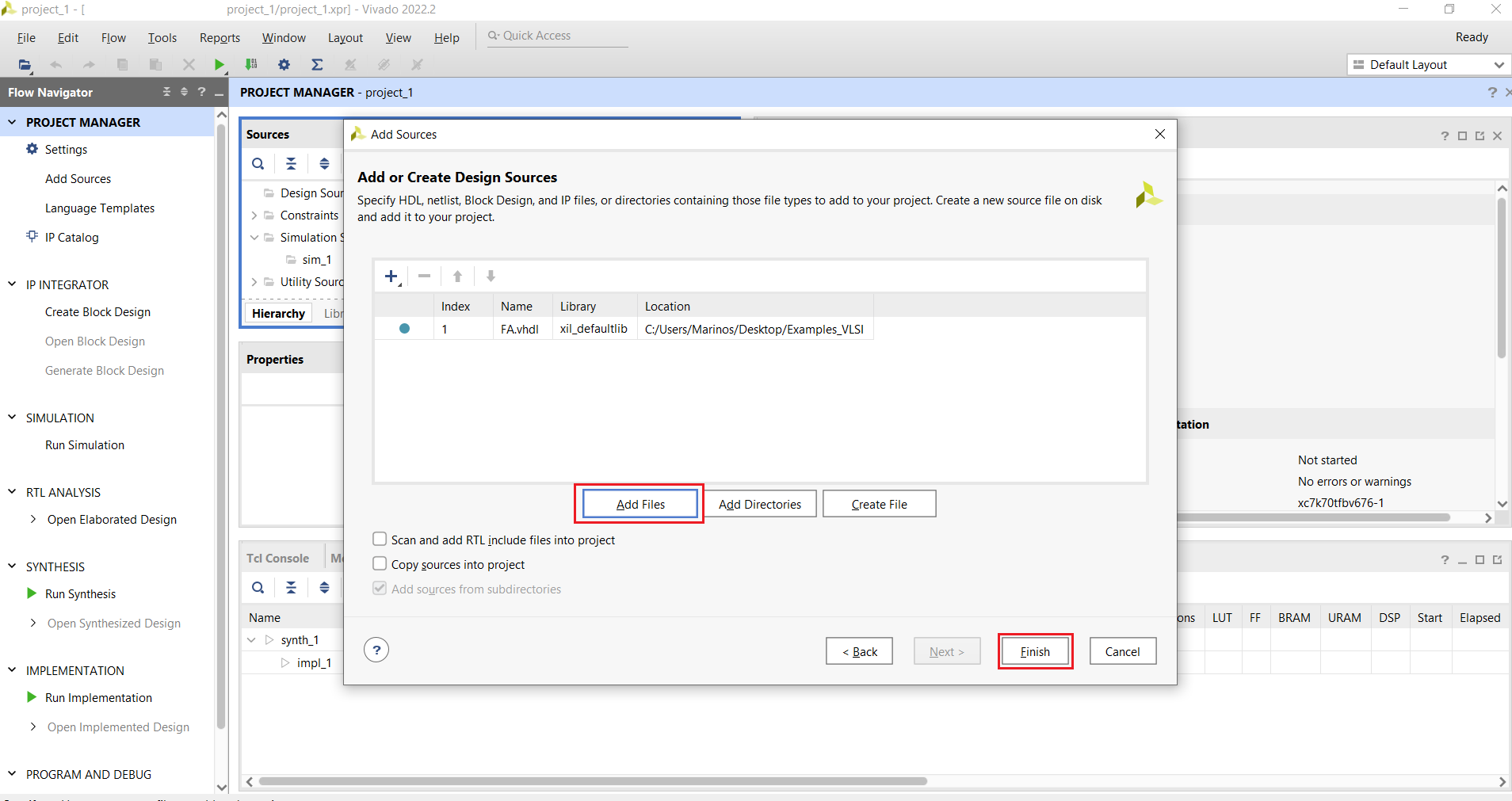


1. **Προσθήκη αρχείων στο έργο**

Επιλέγοντας *Add Sources* από τον Project Manager, εμφανίζεται ένα παράθυρο διαλόγου στο οποίο καθορίζεται τι είδους αρχείο θα προστεθεί. Για αρχεία σχεδιασμού επιλέγετε το δεύτερο και προχωράτε στο επόμενο βήμα πατώντας *Next*.

****

Στην συνέχεια , μέσω του *Add Files* επιλέγονται τα αρχεία που θα προστεθούν στον έργο και επιλέγετε *Finish*.



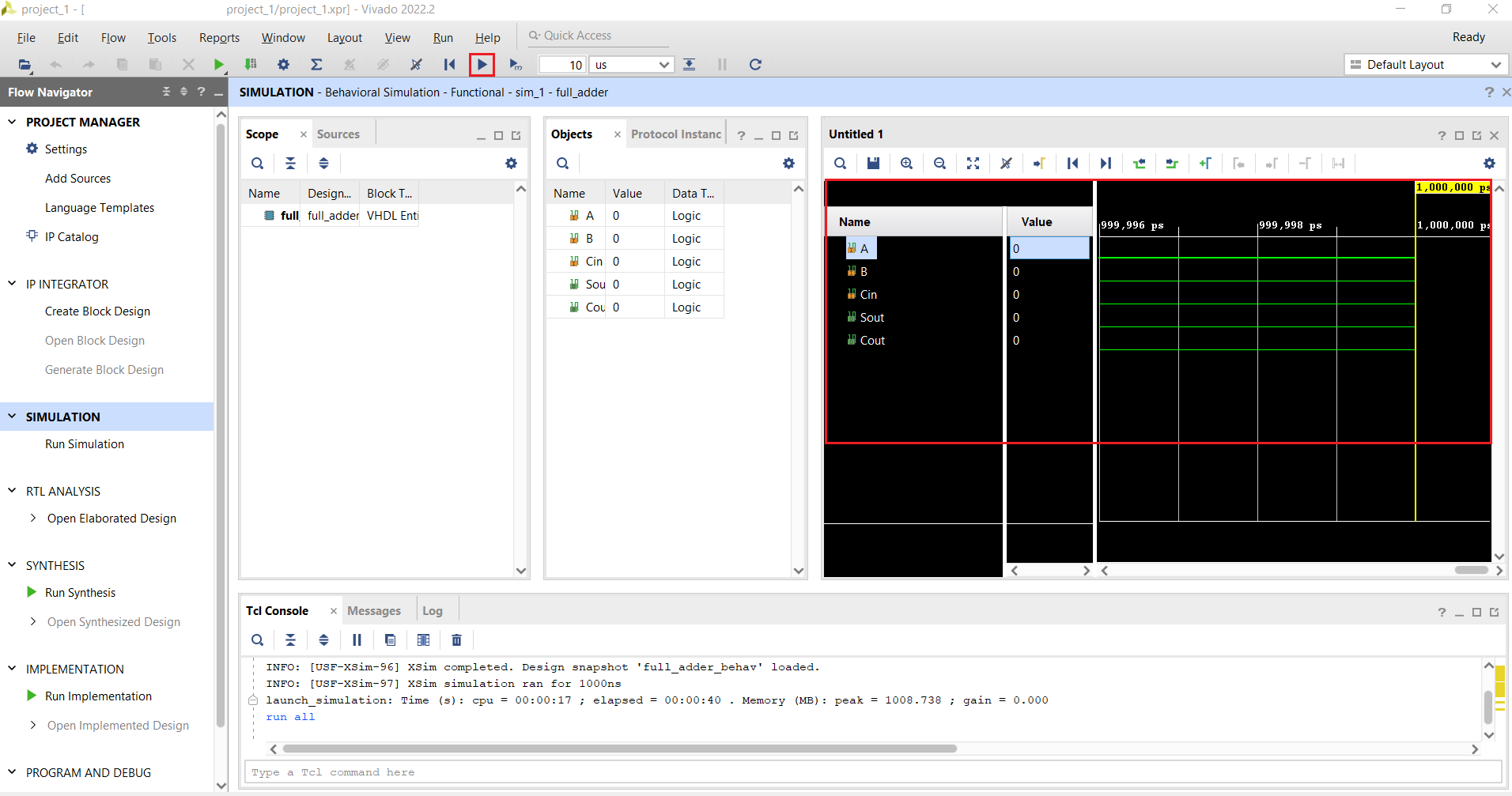
1. **Εξομοίωση σχεδιασμού**

Για να γίνει η εξομοίωση του σχεδιασμού, επιλέγεται το *Run Simulation* -> *Run Behavioral Simulation*.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, αριθμός

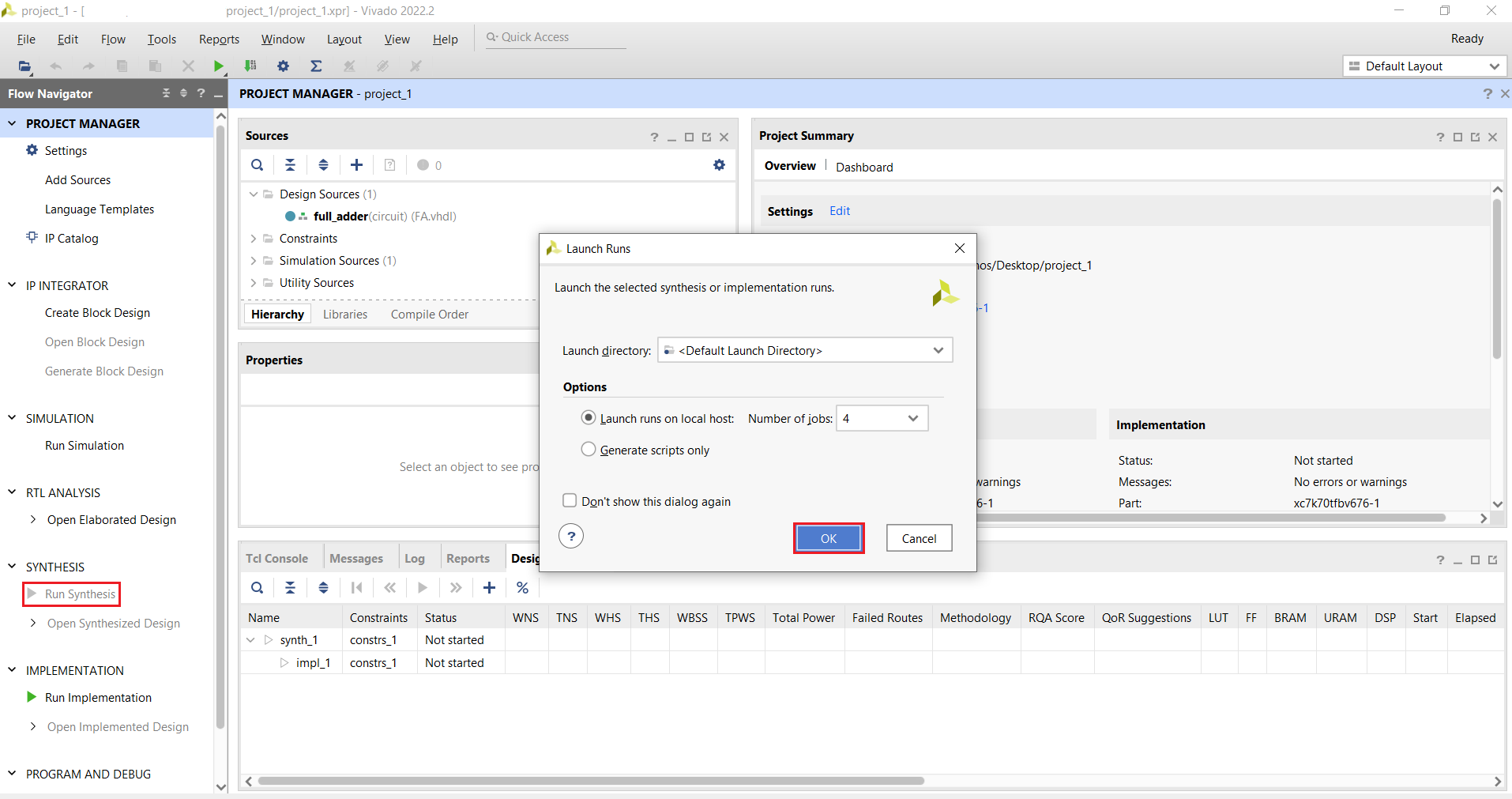
Περιγραφή που δημιουργήθηκε αυτόματα

Στο παράθυρο που εμφανίζεται, μπορούν να προστεθούν τιμές στις εισόδους και πατώντας το *Run All* να εμφανιστούν οι τιμές των εισόδων και των εξόδων σε κάθε στιγμή ως κυματομορφές.



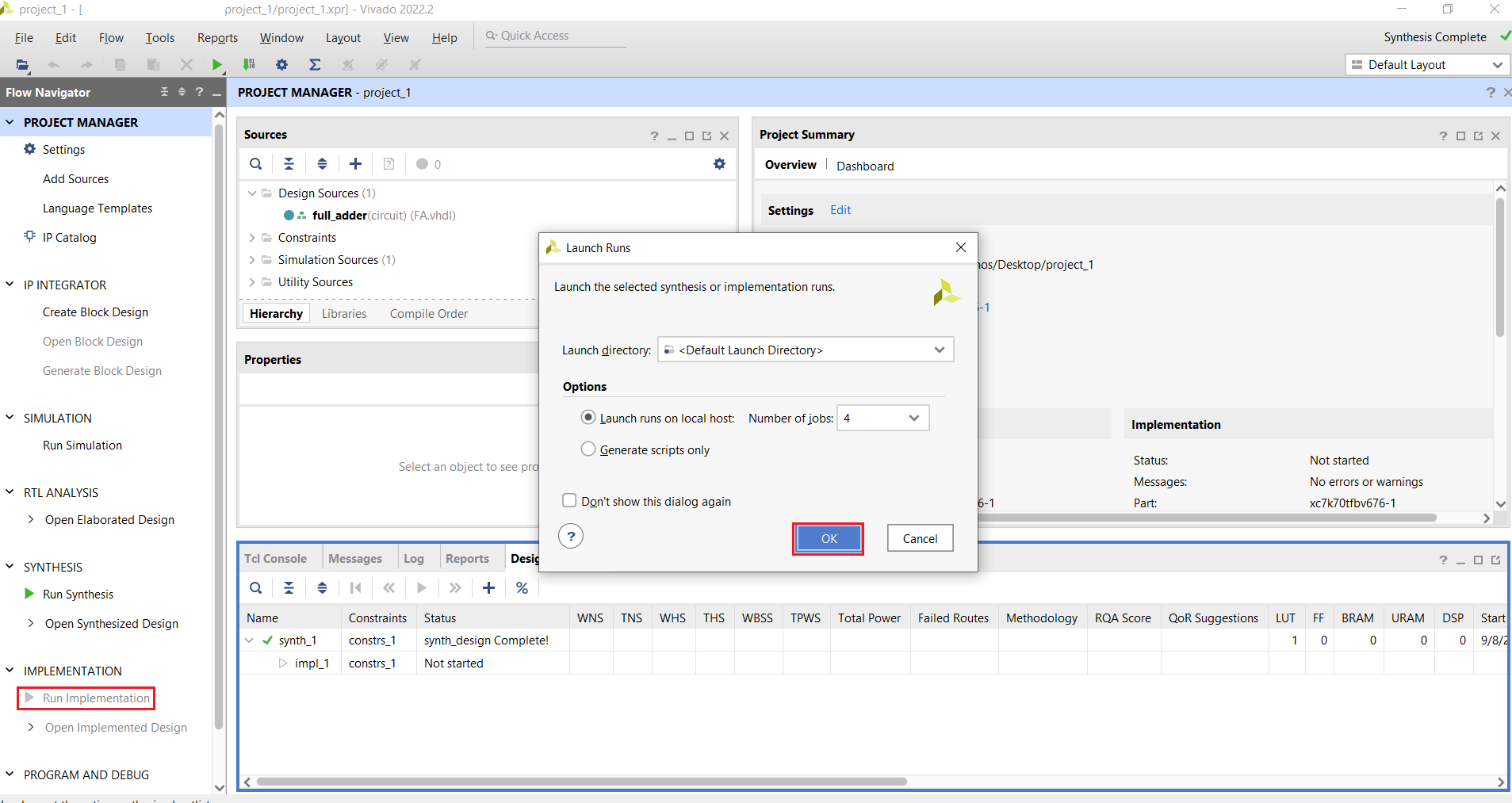
1. **Σύνθεση σχεδιασμού**

Η σύνθεση του σχεδιασμού γίνεται επιλέγοντας το *Run Synthesis*. Στο παράθυρο που ανοίγεται, επιλέγονται οι επιθυμητές ρυθμίσεις για την εκτέλεση της σύνθεσης και για να αρχίσει η διαδικασία επιλέγεται το *OK*. Αφού ολοκληρωθεί χωρίς λάθη η σύνθεση, το επόμενο βήμα είναι η υλοποίηση.

****

1. **Υλοποίηση σχεδιασμού**

Για την υλοποίηση του σχεδιασμού, επιλέγεται το *Run Implementation* και στο παράθυρο διαλόγου που εμφανίζεται, αφού επιλεχθούν οι επιθυμητές ρυθμίσεις υλοποίησης, επιλέγεται το *OK*. Με αυτόν τον τρόπο, ο σχεδιασμός σας, φορτώνεται εικονικά στο device που έχει επιλεχθεί.



Μετά το πέρας της υλοποίησης, εμφανίζονται διάφορα Reports από τα οποία εξάγονται χρήσιμες πληροφορίες για τον σχεδιασμό, όπως η ισχύς, οι χρόνοι και το utilization. Επιπλέον, μπορείτε να δείτε και το σχηματικό του σχεδιασμού.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, λογισμικό πολυμέσων

Περιγραφή που δημιουργήθηκε αυτόματα

1. **Δημιουργία Bitstream**

Αν υπάρχει το device και επιθυμείτε να το φορτώσετε πραγματικά σε αυτό, τότε επιλέγετε το *Generate Bitstream* και δημιουργείται το αντίστοιχο αρχείο που θα φορτωθεί στο device.

Εικόνα που περιέχει στιγμιότυπο οθόνης, κείμενο, λογισμικό, λογισμικό πολυμέσων

Περιγραφή που δημιουργήθηκε αυτόματα

**Παράρτημα Β: Κώδικας SNOW-V**

**D\_FF**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY D\_FF IS

PORT(

d\_in: IN std\_logic;

q\_out: OUT std\_logic;

clk,rst: IN std\_logic

);

END ENTITY;

ARCHITECTURE Circuit OF D\_FF IS

BEGIN

PROCESS(clk,rst)

BEGIN

IF (rst='1') THEN

q\_out <= '0';

ELSIF (clk'EVENT and clk='1') THEN

q\_out <= d\_in;

END IF;

END PROCESS;

END ARCHITECTURE;

**Mux**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY Mux IS

GENERIC(N:integer);

PORT(

a,b: IN std\_logic\_vector (N-1 downto 0);

c: OUT std\_logic\_vector(N-1 downto 0);

sel: IN std\_logic

);

END ENTITY;

ARCHITECTURE Circuit OF Mux IS

BEGIN

PROCESS(a,b,sel)

BEGIN

CASE sel IS

WHEN '0' => c<=a;

WHEN '1' => c<=b;

WHEN OTHERS => c<= (others=>'X');

END CASE;

END PROCESS;

END ARCHITECTURE;

**Register128**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY Register128 IS

GENERIC(N:integer:=128);

PORT(

reg\_in: IN std\_logic\_vector (N-1 downto 0);

clk,rst: IN std\_logic;

reg\_out:OUT std\_logic\_vector (N-1 downto 0)

);

END ENTITY;

ARCHITECTURE Circuit OF Register128 IS

-- COMPONENT D-FLIP FLOP

COMPONENT D\_FF IS

PORT(

d\_in: IN std\_logic;

q\_out: OUT std\_logic;

clk,rst: IN std\_logic

);

END COMPONENT;

BEGIN

ff\_0: D\_FF PORT MAP( d\_in => reg\_in(0), clk => clk, rst => rst, q\_out => reg\_out(0));

ff\_1: D\_FF PORT MAP( d\_in => reg\_in(1), clk => clk, rst => rst, q\_out => reg\_out(1));

ff\_2: D\_FF PORT MAP( d\_in => reg\_in(2), clk => clk, rst => rst, q\_out => reg\_out(2));

ff\_3: D\_FF PORT MAP( d\_in => reg\_in(3), clk => clk, rst => rst, q\_out => reg\_out(3));

ff\_4: D\_FF PORT MAP( d\_in => reg\_in(4), clk => clk, rst => rst, q\_out => reg\_out(4));

ff\_5: D\_FF PORT MAP( d\_in => reg\_in(5), clk => clk, rst => rst, q\_out => reg\_out(5));

ff\_6: D\_FF PORT MAP( d\_in => reg\_in(6), clk => clk, rst => rst, q\_out => reg\_out(6));

ff\_7: D\_FF PORT MAP( d\_in => reg\_in(7), clk => clk, rst => rst, q\_out => reg\_out(7));

ff\_8: D\_FF PORT MAP( d\_in => reg\_in(8), clk => clk, rst => rst, q\_out => reg\_out(8));

ff\_9: D\_FF PORT MAP( d\_in => reg\_in(9), clk => clk, rst => rst, q\_out => reg\_out(9));

ff\_10: D\_FF PORT MAP( d\_in => reg\_in(10), clk => clk, rst => rst, q\_out => reg\_out(10));

ff\_11: D\_FF PORT MAP( d\_in => reg\_in(11), clk => clk, rst => rst, q\_out => reg\_out(11));

ff\_12: D\_FF PORT MAP( d\_in => reg\_in(12), clk => clk, rst => rst, q\_out => reg\_out(12));

ff\_13: D\_FF PORT MAP( d\_in => reg\_in(13), clk => clk, rst => rst, q\_out => reg\_out(13));

ff\_14: D\_FF PORT MAP( d\_in => reg\_in(14), clk => clk, rst => rst, q\_out => reg\_out(14));

ff\_15: D\_FF PORT MAP( d\_in => reg\_in(15), clk => clk, rst => rst, q\_out => reg\_out(15));

ff\_16: D\_FF PORT MAP( d\_in => reg\_in(16), clk => clk, rst => rst, q\_out => reg\_out(16));

ff\_17: D\_FF PORT MAP( d\_in => reg\_in(17), clk => clk, rst => rst, q\_out => reg\_out(17));

ff\_18: D\_FF PORT MAP( d\_in => reg\_in(18), clk => clk, rst => rst, q\_out => reg\_out(18));

ff\_19: D\_FF PORT MAP( d\_in => reg\_in(19), clk => clk, rst => rst, q\_out => reg\_out(19));

ff\_20: D\_FF PORT MAP( d\_in => reg\_in(20), clk => clk, rst => rst, q\_out => reg\_out(20));

ff\_21: D\_FF PORT MAP( d\_in => reg\_in(21), clk => clk, rst => rst, q\_out => reg\_out(21));

ff\_22: D\_FF PORT MAP( d\_in => reg\_in(22), clk => clk, rst => rst, q\_out => reg\_out(22));

ff\_23: D\_FF PORT MAP( d\_in => reg\_in(23), clk => clk, rst => rst, q\_out => reg\_out(23));

ff\_24: D\_FF PORT MAP( d\_in => reg\_in(24), clk => clk, rst => rst, q\_out => reg\_out(24));

ff\_25: D\_FF PORT MAP( d\_in => reg\_in(25), clk => clk, rst => rst, q\_out => reg\_out(25));

ff\_26: D\_FF PORT MAP( d\_in => reg\_in(26), clk => clk, rst => rst, q\_out => reg\_out(26));

ff\_27: D\_FF PORT MAP( d\_in => reg\_in(27), clk => clk, rst => rst, q\_out => reg\_out(27));

ff\_28: D\_FF PORT MAP( d\_in => reg\_in(28), clk => clk, rst => rst, q\_out => reg\_out(28));

ff\_29: D\_FF PORT MAP( d\_in => reg\_in(29), clk => clk, rst => rst, q\_out => reg\_out(29));

ff\_30: D\_FF PORT MAP( d\_in => reg\_in(30), clk => clk, rst => rst, q\_out => reg\_out(30));

ff\_31: D\_FF PORT MAP( d\_in => reg\_in(31), clk => clk, rst => rst, q\_out => reg\_out(31));

ff\_32: D\_FF PORT MAP( d\_in => reg\_in(32), clk => clk, rst => rst, q\_out => reg\_out(32));

ff\_33: D\_FF PORT MAP( d\_in => reg\_in(33), clk => clk, rst => rst, q\_out => reg\_out(33));

ff\_34: D\_FF PORT MAP( d\_in => reg\_in(34), clk => clk, rst => rst, q\_out => reg\_out(34));

ff\_35: D\_FF PORT MAP( d\_in => reg\_in(35), clk => clk, rst => rst, q\_out => reg\_out(35));

ff\_36: D\_FF PORT MAP( d\_in => reg\_in(36), clk => clk, rst => rst, q\_out => reg\_out(36));

ff\_37: D\_FF PORT MAP( d\_in => reg\_in(37), clk => clk, rst => rst, q\_out => reg\_out(37));

ff\_38: D\_FF PORT MAP( d\_in => reg\_in(38), clk => clk, rst => rst, q\_out => reg\_out(38));

ff\_39: D\_FF PORT MAP( d\_in => reg\_in(39), clk => clk, rst => rst, q\_out => reg\_out(39));

ff\_40: D\_FF PORT MAP( d\_in => reg\_in(40), clk => clk, rst => rst, q\_out => reg\_out(40));

ff\_41: D\_FF PORT MAP( d\_in => reg\_in(41), clk => clk, rst => rst, q\_out => reg\_out(41));

ff\_42: D\_FF PORT MAP( d\_in => reg\_in(42), clk => clk, rst => rst, q\_out => reg\_out(42));

ff\_43: D\_FF PORT MAP( d\_in => reg\_in(43), clk => clk, rst => rst, q\_out => reg\_out(43));

ff\_44: D\_FF PORT MAP( d\_in => reg\_in(44), clk => clk, rst => rst, q\_out => reg\_out(44));

ff\_45: D\_FF PORT MAP( d\_in => reg\_in(45), clk => clk, rst => rst, q\_out => reg\_out(45));

ff\_46: D\_FF PORT MAP( d\_in => reg\_in(46), clk => clk, rst => rst, q\_out => reg\_out(46));

ff\_47: D\_FF PORT MAP( d\_in => reg\_in(47), clk => clk, rst => rst, q\_out => reg\_out(47));

ff\_48: D\_FF PORT MAP( d\_in => reg\_in(48), clk => clk, rst => rst, q\_out => reg\_out(48));

ff\_49: D\_FF PORT MAP( d\_in => reg\_in(49), clk => clk, rst => rst, q\_out => reg\_out(49));

ff\_50: D\_FF PORT MAP( d\_in => reg\_in(50), clk => clk, rst => rst, q\_out => reg\_out(50));

ff\_51: D\_FF PORT MAP( d\_in => reg\_in(51), clk => clk, rst => rst, q\_out => reg\_out(51));

ff\_52: D\_FF PORT MAP( d\_in => reg\_in(52), clk => clk, rst => rst, q\_out => reg\_out(52));

ff\_53: D\_FF PORT MAP( d\_in => reg\_in(53), clk => clk, rst => rst, q\_out => reg\_out(53));

ff\_54: D\_FF PORT MAP( d\_in => reg\_in(54), clk => clk, rst => rst, q\_out => reg\_out(54));

ff\_55: D\_FF PORT MAP( d\_in => reg\_in(55), clk => clk, rst => rst, q\_out => reg\_out(55));

ff\_56: D\_FF PORT MAP( d\_in => reg\_in(56), clk => clk, rst => rst, q\_out => reg\_out(56));

ff\_57: D\_FF PORT MAP( d\_in => reg\_in(57), clk => clk, rst => rst, q\_out => reg\_out(57));

ff\_58: D\_FF PORT MAP( d\_in => reg\_in(58), clk => clk, rst => rst, q\_out => reg\_out(58));

ff\_59: D\_FF PORT MAP( d\_in => reg\_in(59), clk => clk, rst => rst, q\_out => reg\_out(59));

ff\_60: D\_FF PORT MAP( d\_in => reg\_in(60), clk => clk, rst => rst, q\_out => reg\_out(60));

ff\_61: D\_FF PORT MAP( d\_in => reg\_in(61), clk => clk, rst => rst, q\_out => reg\_out(61));

ff\_62: D\_FF PORT MAP( d\_in => reg\_in(62), clk => clk, rst => rst, q\_out => reg\_out(62));

ff\_63: D\_FF PORT MAP( d\_in => reg\_in(63), clk => clk, rst => rst, q\_out => reg\_out(63));

ff\_64: D\_FF PORT MAP( d\_in => reg\_in(64), clk => clk, rst => rst, q\_out => reg\_out(64));

ff\_65: D\_FF PORT MAP( d\_in => reg\_in(65), clk => clk, rst => rst, q\_out => reg\_out(65));

ff\_66: D\_FF PORT MAP( d\_in => reg\_in(66), clk => clk, rst => rst, q\_out => reg\_out(66));

ff\_67: D\_FF PORT MAP( d\_in => reg\_in(67), clk => clk, rst => rst, q\_out => reg\_out(67));

ff\_68: D\_FF PORT MAP( d\_in => reg\_in(68), clk => clk, rst => rst, q\_out => reg\_out(68));

ff\_69: D\_FF PORT MAP( d\_in => reg\_in(69), clk => clk, rst => rst, q\_out => reg\_out(69));

ff\_70: D\_FF PORT MAP( d\_in => reg\_in(70), clk => clk, rst => rst, q\_out => reg\_out(70));

ff\_71: D\_FF PORT MAP( d\_in => reg\_in(71), clk => clk, rst => rst, q\_out => reg\_out(71));

ff\_72: D\_FF PORT MAP( d\_in => reg\_in(72), clk => clk, rst => rst, q\_out => reg\_out(72));

ff\_73: D\_FF PORT MAP( d\_in => reg\_in(73), clk => clk, rst => rst, q\_out => reg\_out(73));

ff\_74: D\_FF PORT MAP( d\_in => reg\_in(74), clk => clk, rst => rst, q\_out => reg\_out(74));

ff\_75: D\_FF PORT MAP( d\_in => reg\_in(75), clk => clk, rst => rst, q\_out => reg\_out(75));

ff\_76: D\_FF PORT MAP( d\_in => reg\_in(76), clk => clk, rst => rst, q\_out => reg\_out(76));

ff\_77: D\_FF PORT MAP( d\_in => reg\_in(77), clk => clk, rst => rst, q\_out => reg\_out(77));

ff\_78: D\_FF PORT MAP( d\_in => reg\_in(78), clk => clk, rst => rst, q\_out => reg\_out(78));

ff\_79: D\_FF PORT MAP( d\_in => reg\_in(79), clk => clk, rst => rst, q\_out => reg\_out(79));

ff\_80: D\_FF PORT MAP( d\_in => reg\_in(80), clk => clk, rst => rst, q\_out => reg\_out(80));

ff\_81: D\_FF PORT MAP( d\_in => reg\_in(81), clk => clk, rst => rst, q\_out => reg\_out(81));

ff\_82: D\_FF PORT MAP( d\_in => reg\_in(82), clk => clk, rst => rst, q\_out => reg\_out(82));

ff\_83: D\_FF PORT MAP( d\_in => reg\_in(83), clk => clk, rst => rst, q\_out => reg\_out(83));

ff\_84: D\_FF PORT MAP( d\_in => reg\_in(84), clk => clk, rst => rst, q\_out => reg\_out(84));

ff\_85: D\_FF PORT MAP( d\_in => reg\_in(85), clk => clk, rst => rst, q\_out => reg\_out(85));

ff\_86: D\_FF PORT MAP( d\_in => reg\_in(86), clk => clk, rst => rst, q\_out => reg\_out(86));

ff\_87: D\_FF PORT MAP( d\_in => reg\_in(87), clk => clk, rst => rst, q\_out => reg\_out(87));

ff\_88: D\_FF PORT MAP( d\_in => reg\_in(88), clk => clk, rst => rst, q\_out => reg\_out(88));

ff\_89: D\_FF PORT MAP( d\_in => reg\_in(89), clk => clk, rst => rst, q\_out => reg\_out(89));

ff\_90: D\_FF PORT MAP( d\_in => reg\_in(90), clk => clk, rst => rst, q\_out => reg\_out(90));

ff\_91: D\_FF PORT MAP( d\_in => reg\_in(91), clk => clk, rst => rst, q\_out => reg\_out(91));

ff\_92: D\_FF PORT MAP( d\_in => reg\_in(92), clk => clk, rst => rst, q\_out => reg\_out(92));

ff\_93: D\_FF PORT MAP( d\_in => reg\_in(93), clk => clk, rst => rst, q\_out => reg\_out(93));

ff\_94: D\_FF PORT MAP( d\_in => reg\_in(94), clk => clk, rst => rst, q\_out => reg\_out(94));

ff\_95: D\_FF PORT MAP( d\_in => reg\_in(95), clk => clk, rst => rst, q\_out => reg\_out(95));

ff\_96: D\_FF PORT MAP( d\_in => reg\_in(96), clk => clk, rst => rst, q\_out => reg\_out(96));

ff\_97: D\_FF PORT MAP( d\_in => reg\_in(97), clk => clk, rst => rst, q\_out => reg\_out(97));

ff\_98: D\_FF PORT MAP( d\_in => reg\_in(98), clk => clk, rst => rst, q\_out => reg\_out(98));

ff\_99: D\_FF PORT MAP( d\_in => reg\_in(99), clk => clk, rst => rst, q\_out => reg\_out(99));

ff\_100: D\_FF PORT MAP( d\_in => reg\_in(100), clk => clk, rst => rst, q\_out => reg\_out(100));

ff\_101: D\_FF PORT MAP( d\_in => reg\_in(101), clk => clk, rst => rst, q\_out => reg\_out(101));

ff\_102: D\_FF PORT MAP( d\_in => reg\_in(102), clk => clk, rst => rst, q\_out => reg\_out(102));

ff\_103: D\_FF PORT MAP( d\_in => reg\_in(103), clk => clk, rst => rst, q\_out => reg\_out(103));

ff\_104: D\_FF PORT MAP( d\_in => reg\_in(104), clk => clk, rst => rst, q\_out => reg\_out(104));

ff\_105: D\_FF PORT MAP( d\_in => reg\_in(105), clk => clk, rst => rst, q\_out => reg\_out(105));

ff\_106: D\_FF PORT MAP( d\_in => reg\_in(106), clk => clk, rst => rst, q\_out => reg\_out(106));

ff\_107: D\_FF PORT MAP( d\_in => reg\_in(107), clk => clk, rst => rst, q\_out => reg\_out(107));

ff\_108: D\_FF PORT MAP( d\_in => reg\_in(108), clk => clk, rst => rst, q\_out => reg\_out(108));

ff\_109: D\_FF PORT MAP( d\_in => reg\_in(109), clk => clk, rst => rst, q\_out => reg\_out(109));

ff\_110: D\_FF PORT MAP( d\_in => reg\_in(110), clk => clk, rst => rst, q\_out => reg\_out(110));

ff\_111: D\_FF PORT MAP( d\_in => reg\_in(111), clk => clk, rst => rst, q\_out => reg\_out(111));

ff\_112: D\_FF PORT MAP( d\_in => reg\_in(112), clk => clk, rst => rst, q\_out => reg\_out(112));

ff\_113: D\_FF PORT MAP( d\_in => reg\_in(113), clk => clk, rst => rst, q\_out => reg\_out(113));

ff\_114: D\_FF PORT MAP( d\_in => reg\_in(114), clk => clk, rst => rst, q\_out => reg\_out(114));

ff\_115: D\_FF PORT MAP( d\_in => reg\_in(115), clk => clk, rst => rst, q\_out => reg\_out(115));

ff\_116: D\_FF PORT MAP( d\_in => reg\_in(116), clk => clk, rst => rst, q\_out => reg\_out(116));

ff\_117: D\_FF PORT MAP( d\_in => reg\_in(117), clk => clk, rst => rst, q\_out => reg\_out(117));

ff\_118: D\_FF PORT MAP( d\_in => reg\_in(118), clk => clk, rst => rst, q\_out => reg\_out(118));

ff\_119: D\_FF PORT MAP( d\_in => reg\_in(119), clk => clk, rst => rst, q\_out => reg\_out(119));

ff\_120: D\_FF PORT MAP( d\_in => reg\_in(120), clk => clk, rst => rst, q\_out => reg\_out(120));

ff\_121: D\_FF PORT MAP( d\_in => reg\_in(121), clk => clk, rst => rst, q\_out => reg\_out(121));

ff\_122: D\_FF PORT MAP( d\_in => reg\_in(122), clk => clk, rst => rst, q\_out => reg\_out(122));

ff\_123: D\_FF PORT MAP( d\_in => reg\_in(123), clk => clk, rst => rst, q\_out => reg\_out(123));

ff\_124: D\_FF PORT MAP( d\_in => reg\_in(124), clk => clk, rst => rst, q\_out => reg\_out(124));

ff\_125: D\_FF PORT MAP( d\_in => reg\_in(125), clk => clk, rst => rst, q\_out => reg\_out(125));

ff\_126: D\_FF PORT MAP( d\_in => reg\_in(126), clk => clk, rst => rst, q\_out => reg\_out(126));

ff\_127: D\_FF PORT MAP( d\_in => reg\_in(127), clk => clk, rst => rst, q\_out => reg\_out(127));

END ARCHITECTURE;

**Subunits**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY Subunits IS

PORT(

subunits\_in,coeff\_in: IN std\_logic\_vector(15 downto 0);

subunits\_out: OUT std\_logic\_vector(15 downto 0)

);

END ENTITY;

--Coeff\_a = 0x990f

--Coeff b = 0xc963

ARCHITECTURE Circuit OF Subunits IS

COMPONENT Mux IS

GENERIC(N:integer);

PORT(

a,b: IN std\_logic\_vector (N-1 downto 0);

c: OUT std\_logic\_vector(N-1 downto 0);

sel: IN std\_logic

);

END COMPONENT;

SIGNAL Shifted,Xored: std\_logic\_vector (15 downto 0);

BEGIN

Shifted <= subunits\_in(14 downto 0) & '0';

Xored <= coeff\_in XOR Shifted;

mux0: Mux GENERIC MAP ( N => 16) PORT MAP( a => Shifted, b => Xored, sel => subunits\_in(15), c => subunits\_out );

END ARCHITECTURE;

**Inv\_Subunits**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY Inv\_Subunits IS

PORT(

inv\_subunits\_in,coeff\_in: IN std\_logic\_vector(15 downto 0);

inv\_subunits\_out: OUT std\_logic\_vector(15 downto 0)

);

END ENTITY;

-- Coeff a = 0xcc87

-- Coeff b = 0xe4b1

ARCHITECTURE Circuit OF Inv\_Subunits IS

COMPONENT Mux IS

GENERIC(N:integer);

PORT(

a,b: IN std\_logic\_vector (N-1 downto 0);

c: OUT std\_logic\_vector(N-1 downto 0);

sel: IN std\_logic

);

END COMPONENT;

SIGNAL Shifted,Xored: std\_logic\_vector (15 downto 0);

BEGIN

Shifted <= '0' & inv\_subunits\_in(15 downto 1) ;

Xored <= Coeff\_in XOR Shifted;

mux0: Mux GENERIC MAP ( N => 16) PORT MAP( a => Shifted, b => Xored, sel => inv\_subunits\_in(0), c => inv\_subunits\_out );

END ARCHITECTURE;

**Multiply\_XOR**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY Multiply\_XOR IS

PORT(

a\_high,a\_low,b\_high,b\_low: IN std\_logic\_vector(127 downto 0);

a\_high\_new\_out,b\_high\_new\_out: OUT std\_logic\_vector(127 downto 0)

);

END ENTITY;

ARCHITECTURE Circuit OF Multiply\_XOR IS

COMPONENT Subunits IS

PORT(

subunits\_in,coeff\_in: IN std\_logic\_vector(15 downto 0);

subunits\_out: OUT std\_logic\_vector(15 downto 0)

);

END COMPONENT;

COMPONENT Inv\_Subunits IS

PORT(

inv\_subunits\_in,coeff\_in: IN std\_logic\_vector(15 downto 0);

inv\_subunits\_out: OUT std\_logic\_vector(15 downto 0)

);

END COMPONENT;

--Signals for A PART

SIGNAL sub\_a0\_out,sub\_a1\_out,sub\_a2\_out,sub\_a3\_out,sub\_a4\_out,sub\_a5\_out,sub\_a6\_out,sub\_a7\_out: std\_logic\_vector(15 downto 0);

SIGNAL mul\_a0\_out,mul\_a1\_out,mul\_a2\_out,mul\_a3\_out,mul\_a4\_out,mul\_a5\_out,mul\_a6\_out,mul\_a7\_out: std\_logic\_vector(15 downto 0);

SIGNAL inv\_sub\_a0\_out,inv\_sub\_a1\_out,inv\_sub\_a2\_out,inv\_sub\_a3\_out,inv\_sub\_a4\_out,inv\_sub\_a5\_out,inv\_sub\_a6\_out,inv\_sub\_a7\_out: std\_logic\_vector(15 downto 0);

SIGNAL mul\_a8\_out,mul\_a9\_out,mul\_a10\_out,mul\_a11\_out,mul\_a12\_out,mul\_a13\_out,mul\_a14\_out,mul\_a15\_out: std\_logic\_vector(15 downto 0);

--Signals for B PART

SIGNAL sub\_b0\_out,sub\_b1\_out,sub\_b2\_out,sub\_b3\_out,sub\_b4\_out,sub\_b5\_out,sub\_b6\_out,sub\_b7\_out: std\_logic\_vector(15 downto 0);

SIGNAL mul\_b0\_out,mul\_b1\_out,mul\_b2\_out,mul\_b3\_out,mul\_b4\_out,mul\_b5\_out,mul\_b6\_out,mul\_b7\_out: std\_logic\_vector(15 downto 0);

SIGNAL inv\_sub\_b0\_out,inv\_sub\_b1\_out,inv\_sub\_b2\_out,inv\_sub\_b3\_out,inv\_sub\_b4\_out,inv\_sub\_b5\_out,inv\_sub\_b6\_out,inv\_sub\_b7\_out: std\_logic\_vector(15 downto 0);

SIGNAL mul\_b8\_out,mul\_b9\_out,mul\_b10\_out,mul\_b11\_out,mul\_b12\_out,mul\_b13\_out,mul\_b14\_out,mul\_b15\_out: std\_logic\_vector(15 downto 0);

--Signals for XOR A PART

SIGNAL xor\_a0,xor\_a1,xor\_a2,xor\_a3,xor\_a4,xor\_a5,xor\_a6,xor\_a7:std\_logic\_vector(15 downto 0);

SIGNAL xor\_a8,xor\_a9,xor\_a10,xor\_a11,xor\_a12,xor\_a13,xor\_a14,xor\_a15:std\_logic\_vector(15 downto 0);

--Signals for XOR B PART

SIGNAL xor\_b0,xor\_b1,xor\_b2,xor\_b3,xor\_b4,xor\_b5,xor\_b6,xor\_b7:std\_logic\_vector(15 downto 0);

SIGNAL xor\_b8,xor\_b9,xor\_b10,xor\_b11,xor\_b12,xor\_b13,xor\_b14,xor\_b15:std\_logic\_vector(15 downto 0);

BEGIN

-- alpha is produced

sub\_a0: Subunits PORT MAP (subunits\_in => a\_low(15 downto 0), coeff\_in => x"990f",subunits\_out => sub\_a0\_out);

sub\_a1: Subunits PORT MAP (subunits\_in => a\_low(31 downto 16), coeff\_in => x"990f",subunits\_out => sub\_a1\_out);

sub\_a2: Subunits PORT MAP (subunits\_in => a\_low(47 downto 32), coeff\_in => x"990f",subunits\_out => sub\_a2\_out);

sub\_a3: Subunits PORT MAP (subunits\_in => a\_low(63 downto 48), coeff\_in => x"990f",subunits\_out => sub\_a3\_out);

sub\_a4: Subunits PORT MAP (subunits\_in => a\_low(79 downto 64), coeff\_in => x"990f",subunits\_out => sub\_a4\_out);

sub\_a5: Subunits PORT MAP (subunits\_in => a\_low(95 downto 80), coeff\_in => x"990f",subunits\_out => sub\_a5\_out);

sub\_a6: Subunits PORT MAP (subunits\_in => a\_low(111 downto 96), coeff\_in => x"990f",subunits\_out => sub\_a6\_out);

sub\_a7: Subunits PORT MAP (subunits\_in => a\_low(127 downto 112), coeff\_in => x"990f",subunits\_out => sub\_a7\_out);

--alpha^-1 is produced

inv\_sub\_a0: inv\_Subunits PORT MAP (inv\_subunits\_in => a\_high(15 downto 0), coeff\_in => x"cc87",inv\_subunits\_out => inv\_sub\_a0\_out);

inv\_sub\_a1: inv\_Subunits PORT MAP (inv\_subunits\_in => a\_high(31 downto 16), coeff\_in => x"cc87",inv\_subunits\_out => inv\_sub\_a1\_out);

inv\_sub\_a2: inv\_Subunits PORT MAP (inv\_subunits\_in => a\_high(47 downto 32), coeff\_in => x"cc87",inv\_subunits\_out => inv\_sub\_a2\_out);

inv\_sub\_a3: inv\_Subunits PORT MAP (inv\_subunits\_in => a\_high(63 downto 48), coeff\_in => x"cc87",inv\_subunits\_out => inv\_sub\_a3\_out);

inv\_sub\_a4: inv\_Subunits PORT MAP (inv\_subunits\_in => a\_high(79 downto 64), coeff\_in => x"cc87",inv\_subunits\_out => inv\_sub\_a4\_out);

inv\_sub\_a5: inv\_Subunits PORT MAP (inv\_subunits\_in => a\_high(95 downto 80), coeff\_in => x"cc87",inv\_subunits\_out => inv\_sub\_a5\_out);

inv\_sub\_a6: inv\_Subunits PORT MAP (inv\_subunits\_in => a\_high(111 downto 96), coeff\_in => x"cc87",inv\_subunits\_out => inv\_sub\_a6\_out);

inv\_sub\_a7: inv\_Subunits PORT MAP (inv\_subunits\_in => a\_high(127 downto 112), coeff\_in => x"cc87",inv\_subunits\_out => inv\_sub\_a7\_out);

-- beta is produced

sub\_b0: Subunits PORT MAP (subunits\_in => b\_low(15 downto 0), coeff\_in => x"c963",subunits\_out => sub\_b0\_out);

sub\_b1: Subunits PORT MAP (subunits\_in => b\_low(31 downto 16), coeff\_in => x"c963",subunits\_out => sub\_b1\_out);

sub\_b2: Subunits PORT MAP (subunits\_in => b\_low(47 downto 32), coeff\_in => x"c963",subunits\_out => sub\_b2\_out);

sub\_b3: Subunits PORT MAP (subunits\_in => b\_low(63 downto 48), coeff\_in => x"c963",subunits\_out => sub\_b3\_out);

sub\_b4: Subunits PORT MAP (subunits\_in => b\_low(79 downto 64), coeff\_in => x"c963",subunits\_out => sub\_b4\_out);

sub\_b5: Subunits PORT MAP (subunits\_in => b\_low(95 downto 80), coeff\_in => x"c963",subunits\_out => sub\_b5\_out);

sub\_b6: Subunits PORT MAP (subunits\_in => b\_low(111 downto 96), coeff\_in => x"c963",subunits\_out => sub\_b6\_out);

sub\_b7: Subunits PORT MAP (subunits\_in => b\_low(127 downto 112), coeff\_in => x"c963",subunits\_out => sub\_b7\_out);

--beta^-1 is produced

inv\_sub\_b0: inv\_Subunits PORT MAP (inv\_subunits\_in => b\_high(15 downto 0), coeff\_in => x"e4b1",inv\_subunits\_out => inv\_sub\_b0\_out);

inv\_sub\_b1: inv\_Subunits PORT MAP (inv\_subunits\_in => b\_high(31 downto 16), coeff\_in => x"e4b1",inv\_subunits\_out => inv\_sub\_b1\_out);

inv\_sub\_b2: inv\_Subunits PORT MAP (inv\_subunits\_in => b\_high(47 downto 32), coeff\_in => x"e4b1",inv\_subunits\_out => inv\_sub\_b2\_out);

inv\_sub\_b3: inv\_Subunits PORT MAP (inv\_subunits\_in => b\_high(63 downto 48), coeff\_in => x"e4b1",inv\_subunits\_out => inv\_sub\_b3\_out);

inv\_sub\_b4: inv\_Subunits PORT MAP (inv\_subunits\_in => b\_high(79 downto 64), coeff\_in => x"e4b1",inv\_subunits\_out => inv\_sub\_b4\_out);

inv\_sub\_b5: inv\_Subunits PORT MAP (inv\_subunits\_in => b\_high(95 downto 80), coeff\_in => x"e4b1",inv\_subunits\_out => inv\_sub\_b5\_out);

inv\_sub\_b6: inv\_Subunits PORT MAP (inv\_subunits\_in => b\_high(111 downto 96), coeff\_in => x"e4b1",inv\_subunits\_out => inv\_sub\_b6\_out);

inv\_sub\_b7: inv\_Subunits PORT MAP (inv\_subunits\_in => b\_high(127 downto 112), coeff\_in => x"e4b1",inv\_subunits\_out => inv\_sub\_b7\_out);

--BLOCK A

xor\_a0 <= sub\_a0\_out XOR a\_low(31 downto 16);

xor\_a1 <= sub\_a1\_out XOR a\_low(47 downto 32);

xor\_a2 <= sub\_a2\_out XOR a\_low(63 downto 48);

xor\_a3 <= sub\_a3\_out XOR a\_low(79 downto 64);

xor\_a4 <= sub\_a4\_out XOR a\_low(95 downto 80);

xor\_a5 <= sub\_a5\_out XOR a\_low(111 downto 96);

xor\_a6 <= sub\_a6\_out XOR a\_low(127 downto 112);

xor\_a7 <= sub\_a7\_out XOR a\_high(15 downto 0);

xor\_a8 <=inv\_sub\_a0\_out XOR b\_low(15 downto 0);

xor\_a9 <= inv\_sub\_a1\_out XOR b\_low(31 downto 16);

xor\_a10 <= inv\_sub\_a2\_out XOR b\_low(47 downto 32);

xor\_a11 <= inv\_sub\_a3\_out XOR b\_low(63 downto 48);

xor\_a12 <= inv\_sub\_a4\_out XOR b\_low(79 downto 64);

xor\_a13 <=inv\_sub\_a5\_out XOR b\_low(95 downto 80);

xor\_a14 <= inv\_sub\_a6\_out XOR b\_low(111 downto 96);

xor\_a15 <= inv\_sub\_a7\_out XOR b\_low(127 downto 112);

a\_high\_new\_out(15 downto 0) <= xor\_a0 XOR xor\_a8;

a\_high\_new\_out(31 downto 16) <= xor\_a1 XOR xor\_a9;

a\_high\_new\_out(47 downto 32) <= xor\_a2 XOR xor\_a10;

a\_high\_new\_out(63 downto 48) <= xor\_a3 XOR xor\_a11;

a\_high\_new\_out(79 downto 64) <= xor\_a4 XOR xor\_a12;

a\_high\_new\_out(95 downto 80) <= xor\_a5 XOR xor\_a13;

a\_high\_new\_out(111 downto 96) <= xor\_a6 XOR xor\_a14;

a\_high\_new\_out(127 downto 112) <= xor\_a7 XOR xor\_a15;

--BLOCK B

xor\_b0 <= sub\_b0\_out XOR b\_low(63 downto 48);

xor\_b1 <= sub\_b1\_out XOR b\_low(79 downto 64);

xor\_b2 <= sub\_b2\_out XOR b\_low(95 downto 80);

xor\_b3 <= sub\_b3\_out XOR b\_low(111 downto 96);

xor\_b4 <= sub\_b4\_out XOR b\_low(127 downto 112);

xor\_b5 <= sub\_b5\_out XOR b\_high(15 downto 0);

xor\_b6 <= sub\_b6\_out XOR b\_high(31 downto 16);

xor\_b7 <= sub\_b7\_out XOR b\_high(47 downto 32);

xor\_b8 <= inv\_sub\_b0\_out XOR a\_low(15 downto 0);

xor\_b9 <= inv\_sub\_b1\_out XOR a\_low(31 downto 16);

xor\_b10 <= inv\_sub\_b2\_out XOR a\_low(47 downto 32);

xor\_b11 <= inv\_sub\_b3\_out XOR a\_low(63 downto 48);

xor\_b12 <= inv\_sub\_b4\_out XOR a\_low(79 downto 64);

xor\_b13 <= inv\_sub\_b5\_out XOR a\_low(95 downto 80);

xor\_b14 <= inv\_sub\_b6\_out XOR a\_low(111 downto 96);

xor\_b15 <= inv\_sub\_b7\_out XOR a\_low(127 downto 112);

b\_high\_new\_out(15 downto 0) <= xor\_b0 XOR xor\_b8;

b\_high\_new\_out(31 downto 16) <= xor\_b1 XOR xor\_b9;

b\_high\_new\_out(47 downto 32) <= xor\_b2 XOR xor\_b10;

b\_high\_new\_out(63 downto 48) <= xor\_b3 XOR xor\_b11;

b\_high\_new\_out(79 downto 64) <= xor\_b4 XOR xor\_b12;

b\_high\_new\_out(95 downto 80) <= xor\_b5 XOR xor\_b13;

b\_high\_new\_out(111 downto 96) <= xor\_b6 XOR xor\_b14;

b\_high\_new\_out(127 downto 112) <= xor\_b7 XOR xor\_b15;

END ARCHITECTURE;

**LFSR\_Parallel**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY LFSR\_Parallel IS

PORT(

k\_in : IN std\_logic\_vector(255 downto 0);

iv\_in,keystream\_in: IN std\_logic\_vector(127 downto 0);

t1\_out,t2\_out: OUT std\_logic\_vector(127 downto 0);

clk,rst,sel\_mux\_in,sel\_mux\_a\_high2\_in: IN std\_logic

);

END ENTITY;

ARCHITECTURE Circuit OF LFSR\_Parallel IS

COMPONENT Multiply\_XOR IS

PORT(

a\_high,a\_low,b\_high,b\_low: IN std\_logic\_vector(127 downto 0);

a\_high\_new\_out,b\_high\_new\_out: OUT std\_logic\_vector(127 downto 0)

);

END COMPONENT;

COMPONENT Register128 IS

GENERIC(N:integer:=128);

PORT(

reg\_in: IN std\_logic\_vector (N-1 downto 0);

clk,rst: IN std\_logic;

reg\_out:OUT std\_logic\_vector (N-1 downto 0)

);

END COMPONENT;

COMPONENT Mux IS

GENERIC(N:integer);

PORT(

a,b: IN std\_logic\_vector (N-1 downto 0);

c: OUT std\_logic\_vector(N-1 downto 0);

sel: IN std\_logic

);

END COMPONENT;

SIGNAL a\_high\_out,a\_low\_out,b\_high\_out,b\_low\_out,a\_high\_new\_out,b\_high\_new\_out: std\_logic\_vector(127 downto 0);

SIGNAL mux\_b\_low\_out,mux\_a\_low\_out,mux\_b\_high\_out,mux\_a\_high\_out,mux\_a\_high2\_out:std\_logic\_vector(127 downto 0);

SIGNAL a\_high\_xor\_keystream:std\_logic\_vector(127 downto 0);

BEGIN

mux\_BL: Mux GENERIC MAP (N=>128) PORT MAP(a => b\_high\_out, b => (0 => '0', OTHERS => '0'), c => mux\_b\_low\_out, sel => sel\_mux\_in);

mux\_BH: Mux GENERIC MAP (N=>128) PORT MAP(a => b\_high\_new\_out, b => k\_in(255 downto 128), c => mux\_b\_high\_out, sel => sel\_mux\_in);

mux\_AL: Mux GENERIC MAP (N=>128) PORT MAP(a => a\_high\_out, b => iv\_in(127 downto 0), c => mux\_a\_low\_out, sel => sel\_mux\_in);

mux\_AH: Mux GENERIC MAP (N=>128) PORT MAP(a => a\_high\_new\_out, b => k\_in(127 downto 0), c => mux\_a\_high\_out, sel => sel\_mux\_in);

a\_high\_xor\_keystream <= a\_high\_new\_out XOR keystream\_in;

mux\_AH2: mux GENERIC MAP (N=>128) PORT MAP(a => mux\_a\_high\_out, b => a\_high\_xor\_keystream, c => mux\_a\_high2\_out, sel => sel\_mux\_a\_high2\_in);

BL: Register128 PORT MAP(reg\_in => mux\_b\_low\_out, clk => clk, rst => rst, reg\_out => b\_low\_out);

BH: Register128 PORT MAP(reg\_in => mux\_b\_high\_out, clk => clk, rst => rst, reg\_out => b\_high\_out);

AL: Register128 PORT MAP(reg\_in => mux\_a\_low\_out, clk => clk, rst => rst, reg\_out => a\_low\_out);

AH: Register128 PORT MAP(reg\_in => mux\_a\_high2\_out, clk => clk, rst => rst, reg\_out => a\_high\_out);

MX: Multiply\_XOR PORT MAP(a\_high => a\_high\_out, a\_low => a\_low\_out, b\_high => b\_high\_out, b\_low => b\_low\_out, a\_high\_new\_out => a\_high\_new\_out, b\_high\_new\_out =>b\_high\_new\_out);

t1\_out <= b\_high\_out;

t2\_out <= a\_low\_out;

END ARCHITECTURE;

**FA**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY FA IS

PORT(

fa\_in0,fa\_in1,fa\_c\_in: IN std\_logic;

fa\_sum\_out,fa\_c\_out: OUT std\_logic

);

END ENTITY;

ARCHITECTURE Circuit OF FA IS

BEGIN

fa\_sum\_out<= (fa\_in0 XOR fa\_in1 ) XOR fa\_c\_in;

fa\_c\_out<= (fa\_in0 AND fa\_in1 ) OR ((fa\_in0 XOR fa\_in1 ) AND fa\_c\_in);

END ARCHITECTURE;

**RippleCarryAdder**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY RippleCarryAdder IS

PORT(

rcadd\_in0,rcadd\_in1: IN std\_logic\_vector(31 downto 0);

rcadd\_c\_in: IN std\_logic:='0';

rcadd\_sum\_out: OUT std\_logic\_vector(31 downto 0);

rcadd\_c\_out: OUT std\_logic

);

END ENTITY;

ARCHITECTURE Circuit OF RippleCarryAdder IS

--Component FullAdder

COMPONENT FA IS

PORT(

fa\_in0,fa\_in1,fa\_c\_in: IN std\_logic;

fa\_sum\_out,fa\_c\_out: OUT std\_logic

);

END COMPONENT;

SIGNAL c\_out0,c\_out1,c\_out2,c\_out3,c\_out4,c\_out5,c\_out6,c\_out7,c\_out8,c\_out9,c\_out10,c\_out11,c\_out12,c\_out13,c\_out14,c\_out15,c\_out16,c\_out17,c\_out18,c\_out19,c\_out20,c\_out21,c\_out22,c\_out23,c\_out24,c\_out25,c\_out26,c\_out27,c\_out28,c\_out29,c\_out30,c\_out31: std\_logic;

BEGIN

FA\_0: FA PORT MAP( fa\_in0 => rcadd\_in0(0), fa\_in1 => rcadd\_in1(0), fa\_c\_in => rcadd\_c\_in, fa\_sum\_out => rcadd\_sum\_out(0), fa\_c\_out => c\_out0 );

FA\_1: FA PORT MAP( fa\_in0 => rcadd\_in0(1), fa\_in1 => rcadd\_in1(1), fa\_c\_in => c\_out0, fa\_sum\_out => rcadd\_sum\_out(1), fa\_c\_out => c\_out1 );

FA\_2: FA PORT MAP( fa\_in0 => rcadd\_in0(2), fa\_in1 => rcadd\_in1(2), fa\_c\_in => c\_out1, fa\_sum\_out => rcadd\_sum\_out(2), fa\_c\_out => c\_out2 );

FA\_3: FA PORT MAP( fa\_in0 => rcadd\_in0(3), fa\_in1 => rcadd\_in1(3), fa\_c\_in => c\_out2, fa\_sum\_out => rcadd\_sum\_out(3), fa\_c\_out => c\_out3 );

FA\_4: FA PORT MAP( fa\_in0 => rcadd\_in0(4), fa\_in1 => rcadd\_in1(4), fa\_c\_in => c\_out3, fa\_sum\_out => rcadd\_sum\_out(4), fa\_c\_out => c\_out4 );

FA\_5: FA PORT MAP( fa\_in0 => rcadd\_in0(5), fa\_in1 => rcadd\_in1(5), fa\_c\_in => c\_out4, fa\_sum\_out => rcadd\_sum\_out(5), fa\_c\_out => c\_out5 );

FA\_6: FA PORT MAP( fa\_in0 => rcadd\_in0(6), fa\_in1 => rcadd\_in1(6), fa\_c\_in => c\_out5, fa\_sum\_out => rcadd\_sum\_out(6), fa\_c\_out => c\_out6 );

FA\_7: FA PORT MAP( fa\_in0 => rcadd\_in0(7), fa\_in1 => rcadd\_in1(7), fa\_c\_in => c\_out6, fa\_sum\_out => rcadd\_sum\_out(7), fa\_c\_out => c\_out7 );

FA\_8: FA PORT MAP( fa\_in0 => rcadd\_in0(8), fa\_in1 => rcadd\_in1(8), fa\_c\_in => c\_out7, fa\_sum\_out => rcadd\_sum\_out(8), fa\_c\_out => c\_out8 );

FA\_9: FA PORT MAP( fa\_in0 => rcadd\_in0(9), fa\_in1 => rcadd\_in1(9), fa\_c\_in => c\_out8, fa\_sum\_out => rcadd\_sum\_out(9), fa\_c\_out => c\_out9 );

FA\_10: FA PORT MAP( fa\_in0 => rcadd\_in0(10), fa\_in1 => rcadd\_in1(10), fa\_c\_in => c\_out9, fa\_sum\_out => rcadd\_sum\_out(10), fa\_c\_out => c\_out10 );

FA\_11: FA PORT MAP( fa\_in0 => rcadd\_in0(11), fa\_in1 => rcadd\_in1(11), fa\_c\_in => c\_out10, fa\_sum\_out => rcadd\_sum\_out(11), fa\_c\_out => c\_out11 );

FA\_12: FA PORT MAP( fa\_in0 => rcadd\_in0(12), fa\_in1 => rcadd\_in1(12), fa\_c\_in => c\_out11, fa\_sum\_out => rcadd\_sum\_out(12), fa\_c\_out => c\_out12 );

FA\_13: FA PORT MAP( fa\_in0 => rcadd\_in0(13), fa\_in1 => rcadd\_in1(13), fa\_c\_in => c\_out12, fa\_sum\_out => rcadd\_sum\_out(13), fa\_c\_out => c\_out13 );

FA\_14: FA PORT MAP( fa\_in0 => rcadd\_in0(14), fa\_in1 => rcadd\_in1(14), fa\_c\_in => c\_out13, fa\_sum\_out => rcadd\_sum\_out(14), fa\_c\_out => c\_out14 );

FA\_15: FA PORT MAP( fa\_in0 => rcadd\_in0(15), fa\_in1 => rcadd\_in1(15), fa\_c\_in => c\_out14, fa\_sum\_out => rcadd\_sum\_out(15), fa\_c\_out => c\_out15 );

FA\_16: FA PORT MAP( fa\_in0 => rcadd\_in0(16), fa\_in1 => rcadd\_in1(16), fa\_c\_in => c\_out15, fa\_sum\_out => rcadd\_sum\_out(16), fa\_c\_out => c\_out16 );

FA\_17: FA PORT MAP( fa\_in0 => rcadd\_in0(17), fa\_in1 => rcadd\_in1(17), fa\_c\_in => c\_out16, fa\_sum\_out => rcadd\_sum\_out(17), fa\_c\_out => c\_out17 );

FA\_18: FA PORT MAP( fa\_in0 => rcadd\_in0(18), fa\_in1 => rcadd\_in1(18), fa\_c\_in => c\_out17, fa\_sum\_out => rcadd\_sum\_out(18), fa\_c\_out => c\_out18 );

FA\_19: FA PORT MAP( fa\_in0 => rcadd\_in0(19), fa\_in1 => rcadd\_in1(19), fa\_c\_in => c\_out18, fa\_sum\_out => rcadd\_sum\_out(19), fa\_c\_out => c\_out19 );

FA\_20: FA PORT MAP( fa\_in0 => rcadd\_in0(20), fa\_in1 => rcadd\_in1(20), fa\_c\_in => c\_out19, fa\_sum\_out => rcadd\_sum\_out(20), fa\_c\_out => c\_out20 );

FA\_21: FA PORT MAP( fa\_in0 => rcadd\_in0(21), fa\_in1 => rcadd\_in1(21), fa\_c\_in => c\_out20, fa\_sum\_out => rcadd\_sum\_out(21), fa\_c\_out => c\_out21 );

FA\_22: FA PORT MAP( fa\_in0 => rcadd\_in0(22), fa\_in1 => rcadd\_in1(22), fa\_c\_in => c\_out21, fa\_sum\_out => rcadd\_sum\_out(22), fa\_c\_out => c\_out22 );

FA\_23: FA PORT MAP( fa\_in0 => rcadd\_in0(23), fa\_in1 => rcadd\_in1(23), fa\_c\_in => c\_out22, fa\_sum\_out => rcadd\_sum\_out(23), fa\_c\_out => c\_out23 );

FA\_24: FA PORT MAP( fa\_in0 => rcadd\_in0(24), fa\_in1 => rcadd\_in1(24), fa\_c\_in => c\_out23, fa\_sum\_out => rcadd\_sum\_out(24), fa\_c\_out => c\_out24 );

FA\_25: FA PORT MAP( fa\_in0 => rcadd\_in0(25), fa\_in1 => rcadd\_in1(25), fa\_c\_in => c\_out24, fa\_sum\_out => rcadd\_sum\_out(25), fa\_c\_out => c\_out25 );

FA\_26: FA PORT MAP( fa\_in0 => rcadd\_in0(26), fa\_in1 => rcadd\_in1(26), fa\_c\_in => c\_out25, fa\_sum\_out => rcadd\_sum\_out(26), fa\_c\_out => c\_out26 );

FA\_27: FA PORT MAP( fa\_in0 => rcadd\_in0(27), fa\_in1 => rcadd\_in1(27), fa\_c\_in => c\_out26, fa\_sum\_out => rcadd\_sum\_out(27), fa\_c\_out => c\_out27 );

FA\_28: FA PORT MAP( fa\_in0 => rcadd\_in0(28), fa\_in1 => rcadd\_in1(28), fa\_c\_in => c\_out27, fa\_sum\_out => rcadd\_sum\_out(28), fa\_c\_out => c\_out28 );

FA\_29: FA PORT MAP( fa\_in0 => rcadd\_in0(29), fa\_in1 => rcadd\_in1(29), fa\_c\_in => c\_out28, fa\_sum\_out => rcadd\_sum\_out(29), fa\_c\_out => c\_out29 );

FA\_30: FA PORT MAP( fa\_in0 => rcadd\_in0(30), fa\_in1 => rcadd\_in1(30), fa\_c\_in => c\_out29, fa\_sum\_out => rcadd\_sum\_out(30), fa\_c\_out => c\_out30 );

FA\_31: FA PORT MAP( fa\_in0 => rcadd\_in0(31), fa\_in1 => rcadd\_in1(31), fa\_c\_in => c\_out30, fa\_sum\_out => rcadd\_sum\_out(31), fa\_c\_out => c\_out31 );

END ARCHITECTURE;

**Modulo32**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY Modulo32 IS

GENERIC(N: integer:=128);

PORT(

mod32\_in0,mod32\_in1: IN std\_logic\_vector (N-1 downto 0);

mod32\_out: OUT std\_logic\_vector (N-1 downto 0)

);

END ENTITY;

ARCHITECTURE Circuit OF Modulo32 IS

--Component of Ripple Carry Adder(without the use of the last carry out bit)

COMPONENT RippleCarryAdder IS

PORT(

rcadd\_in0,rcadd\_in1: IN std\_logic\_vector(31 downto 0);

rcadd\_c\_in: IN std\_logic:='0';

rcadd\_sum\_out: OUT std\_logic\_vector(31 downto 0);

rcadd\_c\_out: OUT std\_logic

);

END COMPONENT;

SIGNAL cout0,cout1,cout2,cout3: std\_logic;

BEGIN

RCA\_0: RippleCarryAdder PORT MAP( rcadd\_in0 => mod32\_in0 (31 downto 0), rcadd\_in1 => mod32\_in1 (31 downto 0), rcadd\_c\_in => '0', rcadd\_sum\_out => mod32\_out (31 downto 0), rcadd\_c\_out => cout0);

RCA\_1: RippleCarryAdder PORT MAP( rcadd\_in0 => mod32\_in0 (63 downto 32), rcadd\_in1 => mod32\_in1 (63 downto 32), rcadd\_c\_in => '0', rcadd\_sum\_out => mod32\_out (63 downto 32), rcadd\_c\_out => cout1);

RCA\_2: RippleCarryAdder PORT MAP( rcadd\_in0 => mod32\_in0 (95 downto 64), rcadd\_in1 => mod32\_in1 (95 downto 64), rcadd\_c\_in => '0', rcadd\_sum\_out => mod32\_out (95 downto 64), rcadd\_c\_out => cout2);

RCA\_3: RippleCarryAdder PORT MAP( rcadd\_in0 => mod32\_in0 (127 downto 96), rcadd\_in1 => mod32\_in1 (127 downto 96), rcadd\_c\_in => '0', rcadd\_sum\_out => mod32\_out (127 downto 96), rcadd\_c\_out => cout3);

END ARCHITECTURE;

**S\_Box**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY S\_Box IS

PORT(

s\_box\_in: IN std\_logic\_vector (7 downto 0);

s\_box\_out: OUT std\_logic\_vector (7 downto 0)

);

END ENTITY;

ARCHITECTURE Circuit OF S\_Box IS

BEGIN

PROCESS(s\_box\_in)

BEGIN

CASE s\_box\_in IS

--1st line. 0x

WHEN x"00" => s\_box\_out <= x"63"; -- x is used to indicate that we work with hexadecimal numbers

when x"01" => s\_box\_out <= x"7c";

when x"02" => s\_box\_out <= x"77";

when x"03" => s\_box\_out <= x"7b";

when x"04" => s\_box\_out <= x"f2";

when x"05" => s\_box\_out <= x"6b";

when x"06" => s\_box\_out <= x"6f";

when x"07" => s\_box\_out <= x"c5";

when x"08" => s\_box\_out <= x"30";

when x"09" => s\_box\_out <= x"01";

when x"0a" => s\_box\_out <= x"67";

when x"0b" => s\_box\_out <= x"2b";

when x"0c" => s\_box\_out <= x"fe";

when x"0d" => s\_box\_out <= x"d7";

when x"0e" => s\_box\_out <= x"ab";

when x"0f" => s\_box\_out <= x"76";

--2nd line. 1x

when x"10" => s\_box\_out <= x"ca";

when x"11" => s\_box\_out <= x"82";

when x"12" => s\_box\_out <= x"c9";

when x"13" => s\_box\_out <= x"7d";

when x"14" => s\_box\_out <= x"fa";

when x"15" => s\_box\_out <= x"59";

when x"16" => s\_box\_out <= x"47";

when x"17" => s\_box\_out <= x"f0";

when x"18" => s\_box\_out <= x"ad";

when x"19" => s\_box\_out <= x"d4";

when x"1a" => s\_box\_out <= x"a2";

when x"1b" => s\_box\_out <= x"af";

when x"1c" => s\_box\_out <= x"9c";

when x"1d" => s\_box\_out <= x"a4";

when x"1e" => s\_box\_out <= x"72";

when x"1f" => s\_box\_out <= x"c0";

--3rd line. 2x

when x"20" => s\_box\_out <= x"b7";

when x"21" => s\_box\_out <= x"fd";

when x"22" => s\_box\_out <= x"93";

when x"23" => s\_box\_out <= x"26";

when x"24" => s\_box\_out <= x"36";

when x"25" => s\_box\_out <= x"3f";

when x"26" => s\_box\_out <= x"f7";

when x"27" => s\_box\_out <= x"cc";

when x"28" => s\_box\_out <= x"34";

when x"29" => s\_box\_out <= x"a5";

when x"2a" => s\_box\_out <= x"e5";

when x"2b" => s\_box\_out <= x"f1";

when x"2c" => s\_box\_out <= x"71";

when x"2d" => s\_box\_out <= x"d8";

when x"2e" => s\_box\_out <= x"31";

when x"2f" => s\_box\_out <= x"15";

--4th line. 3x

when x"30" => s\_box\_out <= x"04";

when x"31" => s\_box\_out <= x"c7";

when x"32" => s\_box\_out <= x"23";

when x"33" => s\_box\_out <= x"c3";

when x"34" => s\_box\_out <= x"18";

when x"35" => s\_box\_out <= x"96";

when x"36" => s\_box\_out <= x"05";

when x"37" => s\_box\_out <= x"9a";

when x"38" => s\_box\_out <= x"07";

when x"39" => s\_box\_out <= x"12";

when x"3a" => s\_box\_out <= x"80";

when x"3b" => s\_box\_out <= x"e2";

when x"3c" => s\_box\_out <= x"eb";

when x"3d" => s\_box\_out <= x"27";

when x"3e" => s\_box\_out <= x"b2";

when x"3f" => s\_box\_out <= x"75";

--5th line. 4x

when x"40" => s\_box\_out <= x"09";

when x"41" => s\_box\_out <= x"83";

when x"42" => s\_box\_out <= x"2c";

when x"43" => s\_box\_out <= x"1a";

when x"44" => s\_box\_out <= x"1b";

when x"45" => s\_box\_out <= x"6e";

when x"46" => s\_box\_out <= x"5a";

when x"47" => s\_box\_out <= x"a0";

when x"48" => s\_box\_out <= x"52";

when x"49" => s\_box\_out <= x"3b";

when x"4a" => s\_box\_out <= x"d6";

when x"4b" => s\_box\_out <= x"b3";

when x"4c" => s\_box\_out <= x"29";

when x"4d" => s\_box\_out <= x"e3";

when x"4e" => s\_box\_out <= x"2f";

when x"4f" => s\_box\_out <= x"84";

--6th line. 5x

when x"50" => s\_box\_out <= x"53";

when x"51" => s\_box\_out <= x"d1";

when x"52" => s\_box\_out <= x"00";

when x"53" => s\_box\_out <= x"ed";

when x"54" => s\_box\_out <= x"20";

when x"55" => s\_box\_out <= x"fc";

when x"56" => s\_box\_out <= x"b1";

when x"57" => s\_box\_out <= x"5b";

when x"58" => s\_box\_out <= x"6a";

when x"59" => s\_box\_out <= x"cb";

when x"5a" => s\_box\_out <= x"be";

when x"5b" => s\_box\_out <= x"39";

when x"5c" => s\_box\_out <= x"4a";

when x"5d" => s\_box\_out <= x"4c";

when x"5e" => s\_box\_out <= x"58";

when x"5f" => s\_box\_out <= x"cf";

--7th line. 6x

when x"60" => s\_box\_out <= x"d0";

when x"61" => s\_box\_out <= x"ef";

when x"62" => s\_box\_out <= x"aa";

when x"63" => s\_box\_out <= x"fb";

when x"64" => s\_box\_out <= x"43";

when x"65" => s\_box\_out <= x"4d";

when x"66" => s\_box\_out <= x"33";

when x"67" => s\_box\_out <= x"85";

when x"68" => s\_box\_out <= x"45";

when x"69" => s\_box\_out <= x"f9";

when x"6a" => s\_box\_out <= x"02";

when x"6b" => s\_box\_out <= x"7f";

when x"6c" => s\_box\_out <= x"50";

when x"6d" => s\_box\_out <= x"3c";

when x"6e" => s\_box\_out <= x"9f";

when x"6f" => s\_box\_out <= x"a8";

--8th line. 7x

when x"70" => s\_box\_out <= x"51";

when x"71" => s\_box\_out <= x"a3";

when x"72" => s\_box\_out <= x"40";

when x"73" => s\_box\_out <= x"8f";

when x"74" => s\_box\_out <= x"92";

when x"75" => s\_box\_out <= x"9d";

when x"76" => s\_box\_out <= x"38";

when x"77" => s\_box\_out <= x"f5";

when x"78" => s\_box\_out <= x"bc";

when x"79" => s\_box\_out <= x"b6";

when x"7a" => s\_box\_out <= x"da";

when x"7b" => s\_box\_out <= x"21";

when x"7c" => s\_box\_out <= x"10";

when x"7d" => s\_box\_out <= x"ff";

when x"7e" => s\_box\_out <= x"f3";

when x"7f" => s\_box\_out <= x"d2";

--9th line. 8x

when x"80" => s\_box\_out <= x"cd";

when x"81" => s\_box\_out <= x"0c";

when x"82" => s\_box\_out <= x"13";

when x"83" => s\_box\_out <= x"ec";

when x"84" => s\_box\_out <= x"5f";

when x"85" => s\_box\_out <= x"97";

when x"86" => s\_box\_out <= x"44";

when x"87" => s\_box\_out <= x"17";

when x"88" => s\_box\_out <= x"c4";

when x"89" => s\_box\_out <= x"a7";

when x"8a" => s\_box\_out <= x"7e";

when x"8b" => s\_box\_out <= x"3d";

when x"8c" => s\_box\_out <= x"64";

when x"8d" => s\_box\_out <= x"5d";

when x"8e" => s\_box\_out <= x"19";

when x"8f" => s\_box\_out <= x"73";

--10th line. 9x

when x"90" => s\_box\_out <= x"60";

when x"91" => s\_box\_out <= x"81";

when x"92" => s\_box\_out <= x"4f";

when x"93" => s\_box\_out <= x"dc";

when x"94" => s\_box\_out <= x"22";

when x"95" => s\_box\_out <= x"2a";

when x"96" => s\_box\_out <= x"90";

when x"97" => s\_box\_out <= x"88";

when x"98" => s\_box\_out <= x"46";

when x"99" => s\_box\_out <= x"ee";

when x"9a" => s\_box\_out <= x"b8";

when x"9b" => s\_box\_out <= x"14";

when x"9c" => s\_box\_out <= x"de";

when x"9d" => s\_box\_out <= x"5e";

when x"9e" => s\_box\_out <= x"0b";

when x"9f" => s\_box\_out <= x"db";

--11th line. ax

when x"a0" => s\_box\_out <= x"e0";

when x"a1" => s\_box\_out <= x"32";

when x"a2" => s\_box\_out <= x"3a";

when x"a3" => s\_box\_out <= x"0a";

when x"a4" => s\_box\_out <= x"49";

when x"a5" => s\_box\_out <= x"06";

when x"a6" => s\_box\_out <= x"24";

when x"a7" => s\_box\_out <= x"5c";

when x"a8" => s\_box\_out <= x"c2";

when x"a9" => s\_box\_out <= x"d3";

when x"aa" => s\_box\_out <= x"ac";

when x"ab" => s\_box\_out <= x"62";

when x"ac" => s\_box\_out <= x"91";

when x"ad" => s\_box\_out <= x"95";

when x"ae" => s\_box\_out <= x"e4";

when x"af" => s\_box\_out <= x"79";

--12th line. bx

when x"b0" => s\_box\_out <= x"e7";

when x"b1" => s\_box\_out <= x"c8";

when x"b2" => s\_box\_out <= x"37";

when x"b3" => s\_box\_out <= x"6d";

when x"b4" => s\_box\_out <= x"8d";

when x"b5" => s\_box\_out <= x"d5";

when x"b6" => s\_box\_out <= x"4e";

when x"b7" => s\_box\_out <= x"a9";

when x"b8" => s\_box\_out <= x"6c";

when x"b9" => s\_box\_out <= x"56";

when x"ba" => s\_box\_out <= x"f4";

when x"bb" => s\_box\_out <= x"ea";

when x"bc" => s\_box\_out <= x"65";

when x"bd" => s\_box\_out <= x"7a";

when x"be" => s\_box\_out <= x"ae";

when x"bf" => s\_box\_out <= x"08";

--13th line. cx

when x"c0" => s\_box\_out <= x"ba";

when x"c1" => s\_box\_out <= x"78";

when x"c2" => s\_box\_out <= x"25";

when x"c3" => s\_box\_out <= x"2e";

when x"c4" => s\_box\_out <= x"1c";

when x"c5" => s\_box\_out <= x"a6";

when x"c6" => s\_box\_out <= x"b4";

when x"c7" => s\_box\_out <= x"c6";

when x"c8" => s\_box\_out <= x"e8";

when x"c9" => s\_box\_out <= x"dd";

when x"ca" => s\_box\_out <= x"74";

when x"cb" => s\_box\_out <= x"1f";

when x"cc" => s\_box\_out <= x"4b";

when x"cd" => s\_box\_out <= x"bd";

when x"ce" => s\_box\_out <= x"8b";

when x"cf" => s\_box\_out <= x"8a";

--14th line. dx

when x"d0" => s\_box\_out <= x"70";

when x"d1" => s\_box\_out <= x"3e";

when x"d2" => s\_box\_out <= x"b5";

when x"d3" => s\_box\_out <= x"66";

when x"d4" => s\_box\_out <= x"48";

when x"d5" => s\_box\_out <= x"03";

when x"d6" => s\_box\_out <= x"f6";

when x"d7" => s\_box\_out <= x"0e";

when x"d8" => s\_box\_out <= x"61";

when x"d9" => s\_box\_out <= x"35";

when x"da" => s\_box\_out <= x"57";

when x"db" => s\_box\_out <= x"b9";

when x"dc" => s\_box\_out <= x"86";

when x"dd" => s\_box\_out <= x"c1";

when x"de" => s\_box\_out <= x"1d";

when x"df" => s\_box\_out <= x"9e";

--15th line. ex

when x"e0" => s\_box\_out <= x"e1";

when x"e1" => s\_box\_out <= x"f8";

when x"e2" => s\_box\_out <= x"98";

when x"e3" => s\_box\_out <= x"11";

when x"e4" => s\_box\_out <= x"69";

when x"e5" => s\_box\_out <= x"d9";

when x"e6" => s\_box\_out <= x"8e";

when x"e7" => s\_box\_out <= x"94";

when x"e8" => s\_box\_out <= x"9b";

when x"e9" => s\_box\_out <= x"1e";

when x"ea" => s\_box\_out <= x"87";

when x"eb" => s\_box\_out <= x"e9";

when x"ec" => s\_box\_out <= x"ce";

when x"ed" => s\_box\_out <= x"55";

when x"ee" => s\_box\_out <= x"28";

when x"ef" => s\_box\_out <= x"df";

--16th line. fx

when x"f0" => s\_box\_out <= x"8c";

when x"f1" => s\_box\_out <= x"a1";

when x"f2" => s\_box\_out <= x"89";

when x"f3" => s\_box\_out <= x"0d";

when x"f4" => s\_box\_out <= x"bf";

when x"f5" => s\_box\_out <= x"e6";

when x"f6" => s\_box\_out <= x"42";

when x"f7" => s\_box\_out <= x"68";

when x"f8" => s\_box\_out <= x"41";

when x"f9" => s\_box\_out <= x"99";

when x"fa" => s\_box\_out <= x"2d";

when x"fb" => s\_box\_out <= x"0f";

when x"fc" => s\_box\_out <= x"b0";

when x"fd" => s\_box\_out <= x"54";

when x"fe" => s\_box\_out <= x"bb";

when x"ff" => s\_box\_out <= x"16";

when OTHERS => s\_box\_out <= "XXXXXXXX";

END CASE;

END PROCESS;

END ARCHITECTURE;

**SubBytes**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY SubBytes IS

Port(

sub\_bytes\_in: IN std\_logic\_vector (127 downto 0);

sub\_bytes\_out: OUT std\_logic\_vector (127 downto 0)

);

END ENTITY;

ARCHITECTURE Circuit OF SubBytes IS

COMPONENT S\_Box IS

PORT(

s\_box\_in: IN std\_logic\_vector (7 downto 0);

s\_box\_out: OUT std\_logic\_vector (7 downto 0)

);

END COMPONENT;

--StateArray of Bytes

--{0} {4} {8} {12}

--{1} {5} {9} {13}

--{2} {6} {10} {14}

--{3} {7} {11} {15}

BEGIN

S\_Box0: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(7 downto 0), s\_box\_out => sub\_bytes\_out(7 downto 0));

S\_Box1: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(15 downto 8), s\_box\_out => sub\_bytes\_out(15 downto 8));

S\_Box2: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(23 downto 16), s\_box\_out => sub\_bytes\_out(23 downto 16));

S\_Box3: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(31 downto 24), s\_box\_out => sub\_bytes\_out(31 downto 24));

S\_Box4: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(39 downto 32), s\_box\_out => sub\_bytes\_out(39 downto 32));

S\_Box5: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(47 downto 40), s\_box\_out => sub\_bytes\_out(47 downto 40));

S\_Box6: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(55 downto 48), s\_box\_out => sub\_bytes\_out(55 downto 48) );

S\_Box7: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(63 downto 56), s\_box\_out => sub\_bytes\_out(63 downto 56));

S\_Box8: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(71 downto 64), s\_box\_out => sub\_bytes\_out(71 downto 64) );

S\_Box9: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(79 downto 72), s\_box\_out => sub\_bytes\_out(79 downto 72));

S\_Box10: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(87 downto 80), s\_box\_out => sub\_bytes\_out(87 downto 80));

S\_Box11: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(95 downto 88), s\_box\_out => sub\_bytes\_out(95 downto 88));

S\_Box12: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(103 downto 96), s\_box\_out => sub\_bytes\_out(103 downto 96));

S\_Box13: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(111 downto 104), s\_box\_out => sub\_bytes\_out(111 downto 104));

S\_Box14: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(119 downto 112), s\_box\_out => sub\_bytes\_out(119 downto 112));

S\_Box15: S\_Box PORT MAP(s\_box\_in => sub\_bytes\_in(127 downto 120), s\_box\_out => sub\_bytes\_out(127 downto 120));

END ARCHITECTURE;

**ShiftRows**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY ShiftRows IS

PORT(

shift\_rows\_in: IN std\_logic\_vector (127 downto 0);

shift\_rows\_out: OUT std\_logic\_vector (127 downto 0)

);

END ENTITY;

ARCHITECTURE Circuit OF ShiftRows IS

BEGIN

-- 0 4 8 12 1st line, 0 shifts

-- 5 9 13 1 2nd line, 1 shift

-- 10 14 2 6 3rd line, 2 shifts

-- 15 3 7 11 4th line, 3 shifts

shift\_rows\_out(7 downto 0) <= shift\_rows\_in(7 downto 0) ; --Byte 0

shift\_rows\_out(15 downto 8) <= shift\_rows\_in (47 downto 40); --Byte 1

shift\_rows\_out(23 downto 16) <= shift\_rows\_in (87 downto 80); --Byte 2

shift\_rows\_out(31 downto 24) <= shift\_rows\_in (127 downto 120); --Byte 3

shift\_rows\_out(39 downto 32) <= shift\_rows\_in (39 downto 32); --Byte 4

shift\_rows\_out(47 downto 40) <= shift\_rows\_in (79 downto 72); --Byte 5

shift\_rows\_out(55 downto 48) <= shift\_rows\_in (119 downto 112); --Byte 6

shift\_rows\_out(63 downto 56) <= shift\_rows\_in (31 downto 24); --Byte 7

shift\_rows\_out(71 downto 64) <= shift\_rows\_in (71 downto 64); --Byte 8

shift\_rows\_out(79 downto 72) <= shift\_rows\_in (111 downto 104); --Byte 9

shift\_rows\_out(87 downto 80) <= shift\_rows\_in (23 downto 16); --Byte 10

shift\_rows\_out(95 downto 88) <= shift\_rows\_in (63 downto 56); --Byte 11

shift\_rows\_out(103 downto 96) <= shift\_rows\_in (103 downto 96); --Byte 12

shift\_rows\_out(111 downto 104) <= shift\_rows\_in (15 downto 8); --Byte 13

shift\_rows\_out(119 downto 112) <= shift\_rows\_in (55 downto 48); --Byte 14

shift\_rows\_out(127 downto 120) <= shift\_rows\_in (95 downto 88); --Byte 15

END ARCHITECTURE;

**MixColumns**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY MixColumns IS

PORT(

mix\_columns\_in: IN std\_logic\_vector (127 downto 0);

mix\_columns\_out: OUT std\_logic\_vector (127 downto 0)

);

END ENTITY;

ARCHITECTURE Circuit OF MixColumns IS

SIGNAL r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,r14,r15,r16,r17,r18,r19,r20,r21,r22,r23,r24,r25,r26,r27,r28,r29,r30,r31: std\_logic\_vector (7 downto 0);

-- 02 03 01 01

-- 01 02 03 01

-- 01 01 02 03

-- 03 01 01 02

BEGIN

--1st row,1st column

PROCESS(mix\_columns\_in(7 downto 0))

BEGIN

CASE mix\_columns\_in(7) IS

when '0' => r0 <= mix\_columns\_in(6 downto 0) & '0';

when '1' => r0 <= (mix\_columns\_in(6 downto 0) & '0') XOR "00011011";

when OTHERS => r0<= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(15 downto 8))

BEGIN

CASE mix\_columns\_in(15) IS

when '0' => r1 <= mix\_columns\_in (14 downto 8) & '0' XOR mix\_columns\_in (15 downto 8);

when '1' => r1 <= mix\_columns\_in (14 downto 8) & '0' XOR mix\_columns\_in (15 downto 8) XOR "00011011";

when OTHERS => r1 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--2nd row,1st column

PROCESS(mix\_columns\_in(15 downto 8))

BEGIN

CASE mix\_columns\_in(15) IS

when '0' => r2 <= mix\_columns\_in(14 downto 8) & '0';

when '1' => r2 <= (mix\_columns\_in(14 downto 8) & '0') XOR "00011011";

when OTHERS => r2 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(23 downto 16))

BEGIN

CASE mix\_columns\_in(23) IS

when '0' => r3 <= (mix\_columns\_in(22 downto 16) & '0') XOR mix\_columns\_in (23 downto 16);

when '1' => r3 <= (mix\_columns\_in (22 downto 16) & '0') XOR mix\_columns\_in (23 downto 16) XOR "00011011";

when OTHERS => r3 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--3rd row,1st column

PROCESS(mix\_columns\_in(23 downto 16))

BEGIN

CASE mix\_columns\_in(23) IS

when '0' => r4 <= mix\_columns\_in(22 downto 16) & '0';

when '1' => r4 <= (mix\_columns\_in(22 downto 16) & '0') XOR "00011011";

when OTHERS => r4 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(31 downto 24))

BEGIN

CASE mix\_columns\_in(31) IS

when '0' => r5 <= (mix\_columns\_in(30 downto 24) & '0') XOR mix\_columns\_in(31 downto 24);

when '1' => r5 <= (mix\_columns\_in (30 downto 24) & '0') XOR mix\_columns\_in (31 downto 24) XOR "00011011";

when OTHERS => r5 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--4th row,1st column

PROCESS(mix\_columns\_in(31 downto 24))

BEGIN

CASE mix\_columns\_in(31) IS

when '0' => r6 <= mix\_columns\_in(30 downto 24) & '0';

when '1' => r6 <= (mix\_columns\_in(30 downto 24) & '0') XOR "00011011";

when OTHERS => r6 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(7 downto 0))

BEGIN

CASE mix\_columns\_in(7) IS

when '0' => r7 <= (mix\_columns\_in(6 downto 0) & '0') XOR mix\_columns\_in(7 downto 0);

when '1' => r7 <= (mix\_columns\_in(6 downto 0) & '0')XOR mix\_columns\_in(7 downto 0) XOR "00011011";

when OTHERS => r7 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--1st row,2nd column

PROCESS(mix\_columns\_in(39 downto 32))

BEGIN

CASE mix\_columns\_in(39) IS

when '0' => r8 <= mix\_columns\_in(38 downto 32) & '0';

when '1' => r8 <= (mix\_columns\_in(38 downto 32) & '0') XOR "00011011";

when OTHERS => r8 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(47 downto 40))

BEGIN

CASE mix\_columns\_in(47) IS

when '0' => r9 <= (mix\_columns\_in(46 downto 40) & '0') XOR mix\_columns\_in(47 downto 40);

when '1' => r9 <= (mix\_columns\_in(46 downto 40) & '0')XOR mix\_columns\_in(47 downto 40) XOR "00011011";

when OTHERS => r9 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--2nd rown,2nd column

PROCESS(mix\_columns\_in(47 downto 40))

BEGIN

CASE mix\_columns\_in(47) IS

when '0' => r10 <= mix\_columns\_in(46 downto 40) & '0';

when '1' => r10 <= (mix\_columns\_in(46 downto 40) & '0') XOR "00011011";

when OTHERS => r10 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(55 downto 48))

BEGIN

CASE mix\_columns\_in(55) IS

when '0' => r11 <= (mix\_columns\_in(54 downto 48) & '0') XOR mix\_columns\_in(55 downto 48);

when '1' => r11 <= (mix\_columns\_in(54 downto 48) & '0')XOR mix\_columns\_in(55 downto 48) XOR "00011011";

when OTHERS => r11 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--3rd row, 2nd column

PROCESS(mix\_columns\_in(55 downto 48))

BEGIN

CASE mix\_columns\_in(55) IS

when '0' => r12 <= mix\_columns\_in(54 downto 48) & '0';

when '1' => r12 <= (mix\_columns\_in(54 downto 48) & '0') XOR "00011011";

when OTHERS => r12 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(63 downto 56))

BEGIN

CASE mix\_columns\_in(63) IS

when '0' => r13 <= (mix\_columns\_in(62 downto 56) & '0') XOR mix\_columns\_in(63 downto 56);

when '1' => r13 <= (mix\_columns\_in(62 downto 56) & '0')XOR mix\_columns\_in(63 downto 56) XOR "00011011";

when OTHERS => r13 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--4th row,2nd column

PROCESS(mix\_columns\_in(63 downto 56))

BEGIN

CASE mix\_columns\_in(63) IS

when '0' => r14 <= mix\_columns\_in(62 downto 56) & '0';

when '1' => r14 <= (mix\_columns\_in(62 downto 56) & '0') XOR "00011011";

when OTHERS => r14 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(39 downto 32))

BEGIN

CASE mix\_columns\_in(39) IS

when '0' => r15 <= (mix\_columns\_in(38 downto 32) & '0') XOR mix\_columns\_in(39 downto 32);

when '1' => r15 <= (mix\_columns\_in(38 downto 32) & '0')XOR mix\_columns\_in(39 downto 32) XOR "00011011";

when OTHERS => r15 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--1st row,3rd column

PROCESS(mix\_columns\_in(71 downto 64))

BEGIN

CASE mix\_columns\_in(71) IS

when '0' => r16 <= mix\_columns\_in(70 downto 64) & '0';

when '1' => r16 <= (mix\_columns\_in(70 downto 64) & '0') XOR "00011011";

when OTHERS => r16 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(79 downto 72))

BEGIN

CASE mix\_columns\_in(79) IS

when '0' => r17 <= (mix\_columns\_in(78 downto 72) & '0') XOR mix\_columns\_in(79 downto 72);

when '1' => r17 <= (mix\_columns\_in(78 downto 72) & '0')XOR mix\_columns\_in(79 downto 72) XOR "00011011";

when OTHERS => r17 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--2nd row,3rd column

PROCESS(mix\_columns\_in(79 downto 72))

BEGIN

CASE mix\_columns\_in(79) IS

when '0' => r18 <= mix\_columns\_in(78 downto 72) & '0';

when '1' => r18 <= (mix\_columns\_in(78 downto 72) & '0') XOR "00011011";

when OTHERS => r18 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(87 downto 80))

BEGIN

CASE mix\_columns\_in(87) IS

when '0' => r19 <= (mix\_columns\_in(86 downto 80) & '0') XOR mix\_columns\_in(87 downto 80);

when '1' => r19 <= (mix\_columns\_in(86 downto 80) & '0')XOR mix\_columns\_in(87 downto 80) XOR "00011011";

when OTHERS => r19 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--3rd row,3rd column

PROCESS(mix\_columns\_in(87 downto 80))

BEGIN

CASE mix\_columns\_in(87) IS

when '0' => r20 <= mix\_columns\_in(86 downto 80) & '0';

when '1' => r20 <= (mix\_columns\_in(86 downto 80) & '0') XOR "00011011";

when OTHERS => r20 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(95 downto 88))

BEGIN

CASE mix\_columns\_in(95) IS

when '0' => r21 <= (mix\_columns\_in(94 downto 88) & '0') XOR mix\_columns\_in(95 downto 88);

when '1' => r21 <= (mix\_columns\_in(94 downto 88) & '0')XOR mix\_columns\_in(95 downto 88) XOR "00011011";

when OTHERS => r21 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--4th row,3rd column

PROCESS(mix\_columns\_in(95 downto 88))

BEGIN

CASE mix\_columns\_in(95) IS

when '0' => r22 <= mix\_columns\_in(94 downto 88) & '0';

when '1' => r22 <= (mix\_columns\_in(94 downto 88) & '0') XOR "00011011";

when OTHERS => r22 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(71 downto 64))

BEGIN

CASE mix\_columns\_in(71) IS

when '0' => r23 <= (mix\_columns\_in(70 downto 64) & '0') XOR mix\_columns\_in(71 downto 64);

when '1' => r23 <= (mix\_columns\_in(70 downto 64) & '0')XOR mix\_columns\_in(71 downto 64) XOR "00011011";

when OTHERS => r23 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--1st row,4th column

PROCESS(mix\_columns\_in(103 downto 96))

BEGIN

CASE mix\_columns\_in(103) IS

when '0' => r24 <= mix\_columns\_in(102 downto 96) & '0';

when '1' => r24 <= (mix\_columns\_in(102 downto 96) & '0') XOR "00011011";

when OTHERS => r24 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(111 downto 104))

BEGIN

CASE mix\_columns\_in(111) IS

when '0' => r25 <= (mix\_columns\_in(110 downto 104) & '0') XOR mix\_columns\_in(111 downto 104);

when '1' => r25 <= (mix\_columns\_in(110 downto 104) & '0')XOR mix\_columns\_in(111 downto 104) XOR "00011011";

when OTHERS => r25 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--2nd row,4th column

PROCESS(mix\_columns\_in(111 downto 104))

BEGIN

CASE mix\_columns\_in(111) IS

when '0' => r26 <= mix\_columns\_in(110 downto 104) & '0';

when '1' => r26 <= (mix\_columns\_in(110 downto 104) & '0') XOR "00011011";

when OTHERS => r26 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(119 downto 112))

BEGIN

CASE mix\_columns\_in(119) IS

when '0' => r27 <= (mix\_columns\_in(118 downto 112) & '0') XOR mix\_columns\_in(119 downto 112);

when '1' => r27 <= (mix\_columns\_in(118 downto 112) & '0')XOR mix\_columns\_in(119 downto 112) XOR "00011011";

when OTHERS => r27 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--3rd row,4th column

PROCESS(mix\_columns\_in(119 downto 112))

BEGIN

CASE mix\_columns\_in(119) IS

when '0' => r28 <= mix\_columns\_in(118 downto 112) & '0';

when '1' => r28 <= (mix\_columns\_in(118 downto 112) & '0') XOR "00011011";

when OTHERS => r28 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(127 downto 120))

BEGIN

CASE mix\_columns\_in(127) IS

when '0' => r29 <= (mix\_columns\_in(126 downto 120) & '0') XOR mix\_columns\_in(127 downto 120);

when '1' => r29 <= (mix\_columns\_in(126 downto 120) & '0')XOR mix\_columns\_in(127 downto 120) XOR "00011011";

when OTHERS => r29 <= (OTHERS => 'U');

END CASE;

END PROCESS;

--4th row,4th column

PROCESS(mix\_columns\_in(127 downto 120))

BEGIN

CASE mix\_columns\_in(127) IS

when '0' => r30 <= mix\_columns\_in(126 downto 120) & '0';

when '1' => r30 <= (mix\_columns\_in(126 downto 120) & '0') XOR "00011011";

when OTHERS => r30 <= (OTHERS => 'U');

END CASE;

END PROCESS;

PROCESS(mix\_columns\_in(103 downto 96))

BEGIN

CASE mix\_columns\_in(103) IS

when '0' => r31 <= (mix\_columns\_in(102 downto 96) & '0') XOR mix\_columns\_in(103 downto 96);

when '1' => r31 <= (mix\_columns\_in(102 downto 96) & '0') XOR mix\_columns\_in(103 downto 96) XOR "00011011";

when OTHERS => r31 <= (OTHERS => 'U');

END CASE;

END PROCESS;

mix\_columns\_out(7 downto 0) <= r0 XOR r1 XOR mix\_columns\_in(23 downto 16) XOR mix\_columns\_in(31 downto 24);--Position 0,0

mix\_columns\_out(15 downto 8) <= mix\_columns\_in(7 downto 0) XOR r2 XOR r3 XOR mix\_columns\_in( 31 downto 24); --Position 1,0

mix\_columns\_out(23 downto 16) <= mix\_columns\_in(7 downto 0) XOR mix\_columns\_in(15 downto 8) XOR r4 XOR r5;-- Position 2,0

mix\_columns\_out(31 downto 24) <= r6 XOR mix\_columns\_in(15 downto 8) XOR mix\_columns\_in(23 downto 16) XOR r7;--Position 3,0

mix\_columns\_out(39 downto 32) <= r8 XOR r9 XOR mix\_columns\_in(55 downto 48) XOR mix\_columns\_in(63 downto 56);--Position 0,1

mix\_columns\_out(47 downto 40) <= mix\_columns\_in(39 downto 32) XOR r10 XOR r11 XOR mix\_columns\_in( 63 downto 56);--Position 1,1

mix\_columns\_out(55 downto 48) <= mix\_columns\_in(39 downto 32) XOR mix\_columns\_in(47 downto 40) XOR r12 XOR r13;--Position 2,1

mix\_columns\_out(63 downto 56) <= r14 XOR mix\_columns\_in( 47 downto 40) XOR mix\_columns\_in(55 downto 48) XOR r15;--Position 3,1

mix\_columns\_out(71 downto 64) <= r16 XOR r17 XOR mix\_columns\_in(87 downto 80) XOR mix\_columns\_in(95 downto 88);--Position 0,2

mix\_columns\_out(79 downto 72) <= mix\_columns\_in(71 downto 64) XOR r18 XOR r19 XOR mix\_columns\_in(95 downto 88);--Position 1,2

mix\_columns\_out(87 downto 80) <= mix\_columns\_in(71 downto 64) XOR mix\_columns\_in(79 downto 72) XOR r20 XOR r21;--Position 2,2

mix\_columns\_out(95 downto 88) <= r22 XOR mix\_columns\_in(79 downto 72) XOR mix\_columns\_in(87 downto 80) XOR r23;--Position 3,2

mix\_columns\_out(103 downto 96) <= r24 XOR r25 XOR mix\_columns\_in(119 downto 112) XOR mix\_columns\_in(127 downto 120);--Position 0,3

mix\_columns\_out(111 downto 104) <= mix\_columns\_in(103 downto 96) XOR r26 XOR r27 XOR mix\_columns\_in(127 downto 120);--Position 1,3

mix\_columns\_out(119 downto 112) <= mix\_columns\_in(103 downto 96) XOR mix\_columns\_in(111 downto 104) XOR r28 XOR r29;--Position 2,3

mix\_columns\_out(127 downto 120) <= r30 XOR mix\_columns\_in(111 downto 104) XOR mix\_columns\_in(119 downto 112) XOR r31;--Position 3,3

END ARCHITECTURE;

**PermutationBO**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY PermutationBO IS

PORT(

pbo\_in: IN std\_logic\_vector(127 downto 0);

pbo\_out: OUT std\_logic\_vector(127 downto 0)

);

END ENTITY;

ARCHITECTURE Circuit OF PermutationBO IS

BEGIN

--Permutation block sigma = [ 0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15 ]

pbo\_out(7 downto 0) <= pbo\_in (7 downto 0);

pbo\_out(15 downto 8) <= pbo\_in (39 downto 32);

pbo\_out(23 downto 16) <= pbo\_in (71 downto 64);

pbo\_out(31 downto 24) <= pbo\_in (103 downto 96);

pbo\_out(39 downto 32) <= pbo\_in (15 downto 8);

pbo\_out(47 downto 40) <= pbo\_in (47 downto 40);

pbo\_out(55 downto 48) <= pbo\_in (79 downto 72);

pbo\_out(63 downto 56) <= pbo\_in (111 downto 104);

pbo\_out(71 downto 64) <= pbo\_in (23 downto 16);

pbo\_out(79 downto 72) <= pbo\_in (55 downto 48);

pbo\_out(87 downto 80) <= pbo\_in (87 downto 80);

pbo\_out(95 downto 88) <= pbo\_in (119 downto 112);

pbo\_out(103 downto 96) <= pbo\_in (31 downto 24);

pbo\_out(111 downto 104) <= pbo\_in (63 downto 56);

pbo\_out(119 downto 112) <= pbo\_in (95 downto 88);

pbo\_out(127 downto 120) <= pbo\_in (127 downto 120);

END ARCHITECTURE;

**AES\_C1**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY AES\_C1 IS

PORT(

aes\_c1\_in: IN std\_logic\_vector (127 downto 0);

c1: IN std\_logic\_vector (127 downto 0);

aes\_c1\_out: OUT std\_logic\_vector (127 downto 0)

);

END ENTITY;

ARCHITECTURE Circuit OF AES\_C1 IS

COMPONENT SubBytes IS

PORT(

sub\_bytes\_in: IN std\_logic\_vector (127 downto 0);

sub\_bytes\_out: OUT std\_logic\_vector (127 downto 0)

);

END COMPONENT;

COMPONENT ShiftRows IS

PORT(

shift\_rows\_in: IN std\_logic\_vector (127 downto 0);

shift\_rows\_out: OUT std\_logic\_vector (127 downto 0)

);

END COMPONENT;

COMPONENT MixColumns IS

PORT(

mix\_columns\_in: IN std\_logic\_vector (127 downto 0);

mix\_columns\_out: OUT std\_logic\_vector (127 downto 0)

);

END COMPONENT;

SIGNAL SubBytes\_ShiftRows,ShiftRows\_MixColumns,MixColumns\_Out: std\_logic\_vector(127 downto 0);

BEGIN

SubBytes0: SubBytes PORT MAP (sub\_bytes\_in => aes\_c1\_in, sub\_bytes\_out=> SubBytes\_ShiftRows);

ShiftRows0: ShiftRows PORT MAP(shift\_rows\_in => SubBytes\_ShiftRows,shift\_rows\_out => ShiftRows\_MixColumns);

MixColumns0: MixColumns PORT MAP (mix\_columns\_in => ShiftRows\_MixColumns,mix\_columns\_out => MixColumns\_Out);

aes\_c1\_out <= c1 XOR MixColumns\_Out;

END ARCHITECTURE;

**AES\_C2**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY AES\_C2 IS

PORT(

aes\_c2\_in: IN std\_logic\_vector (127 downto 0);

c2: IN std\_logic\_vector (127 downto 0);

aes\_c2\_out: OUT std\_logic\_vector (127 downto 0)

);

END ENTITY;

ARCHITECTURE Circuit OF AES\_C2 IS

COMPONENT SubBytes IS

PORT(

sub\_bytes\_in: IN std\_logic\_vector (127 downto 0);

sub\_bytes\_out: OUT std\_logic\_vector (127 downto 0)

);

END COMPONENT;

COMPONENT ShiftRows IS

PORT(

shift\_rows\_in: IN std\_logic\_vector (127 downto 0);

shift\_rows\_out: OUT std\_logic\_vector (127 downto 0)

);

END COMPONENT;

COMPONENT MixColumns IS

PORT(

mix\_columns\_in: IN std\_logic\_vector (127 downto 0);

mix\_columns\_out: OUT std\_logic\_vector (127 downto 0)

);

END COMPONENT;

SIGNAL SubBytes\_ShiftRows,ShiftRows\_MixColumns,MixColumns\_out: std\_logic\_vector(127 downto 0);

BEGIN

SubBytes\_inst: SubBytes PORT MAP (sub\_bytes\_in => aes\_c2\_in, sub\_bytes\_out=> SubBytes\_ShiftRows);

ShiftRows\_inst: ShiftRows PORT MAP(shift\_rows\_in => SubBytes\_ShiftRows, shift\_rows\_out => ShiftRows\_MixColumns);

MixColumns\_inst: MixColumns PORT MAP (mix\_columns\_in => ShiftRows\_MixColumns, mix\_columns\_out => MixColumns\_out);

aes\_c2\_out <= c2 XOR MixColumns\_out;

END ARCHITECTURE;

**FSM**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY FSM IS

PORT(

lfsr\_t1\_in,lfsr\_t2\_in: IN std\_logic\_vector(127 downto 0);

k\_in: IN std\_logic\_vector(255 downto 0);

clk,rst,sel\_mux\_r\_0\_in,sel\_mux\_r1\_in,sel\_mux\_key\_in: IN std\_logic;

keystream\_out: OUT std\_logic\_vector(127 downto 0)

);

END ENTITY;

ARCHITECTURE Circuit OF FSM IS

COMPONENT Modulo32 IS

GENERIC(N: integer:=128);

PORT(

mod32\_in0,mod32\_in1: IN std\_logic\_vector (N-1 downto 0);

mod32\_out: OUT std\_logic\_vector (N-1 downto 0)

);

END COMPONENT;

COMPONENT Mux IS

GENERIC(N:integer);

PORT(

a,b: IN std\_logic\_vector (N-1 downto 0);

c: OUT std\_logic\_vector(N-1 downto 0);

sel: IN std\_logic

);

END COMPONENT;

COMPONENT PermutationBO IS

PORT(

pbo\_in: IN std\_logic\_vector(127 downto 0);

pbo\_out: OUT std\_logic\_vector(127 downto 0)

);

END COMPONENT;

COMPONENT Register128 IS

GENERIC(N:integer:=128);

PORT(

reg\_in: IN std\_logic\_vector (N-1 downto 0);

clk,rst: IN std\_logic;

reg\_out:OUT std\_logic\_vector (N-1 downto 0)

);

END COMPONENT;

COMPONENT AES\_C1 IS

PORT(

aes\_c1\_in: IN std\_logic\_vector (127 downto 0);

c1: IN std\_logic\_vector (127 downto 0);

aes\_c1\_out: OUT std\_logic\_vector (127 downto 0)

);

END COMPONENT;

COMPONENT AES\_C2 IS

PORT(

aes\_c2\_in: IN std\_logic\_vector (127 downto 0);

c2: IN std\_logic\_vector (127 downto 0);

aes\_c2\_out: OUT std\_logic\_vector (127 downto 0)

);

END COMPONENT;

SIGNAL r3\_out\_xor\_t2\_in,permuation\_out,r1\_out,r3\_out,r2\_out,key\_xor\_permuation: std\_logic\_vector(127 downto 0);

SIGNAL mux\_key\_out,mux\_r1\_out,mux\_r2\_out,mux\_r3\_out,mux\_r1\_0\_out,mux\_r2\_0\_out,mux\_r3\_0\_out: std\_logic\_vector(127 downto 0);

SIGNAL aes2\_out,aes1\_out,mod32b\_out,mod32a\_out: std\_logic\_vector(127 downto 0);

BEGIN

r3\_out\_xor\_t2\_in <= lfsr\_t2\_in XOR r3\_out;

mod32a: Modulo32 PORT MAP(mod32\_in0 => r3\_out\_xor\_t2\_in, mod32\_in1 => r2\_out, mod32\_out => mod32a\_out);

AES2: AES\_C2 PORT MAP(aes\_c2\_in => r2\_out, c2 => (0 => '0', OTHERS => '0'), aes\_c2\_out => aes2\_out);

mux\_r3\_0: Mux GENERIC MAP(N => 128) PORT MAP(a => aes2\_out, b => (0 => '0', OTHERS => '0'), c => mux\_r3\_0\_out, sel => sel\_mux\_r\_0\_in);

r3: Register128 PORT MAP(reg\_in => mux\_r3\_0\_out, clk => clk, rst => rst, reg\_out => r3\_out);

AES1: AES\_C1 PORT MAP(aes\_c1\_in => r1\_out, c1 => (0 => '0', OTHERS => '0'), aes\_c1\_out => aes1\_out);

mux\_r2\_0: Mux GENERIC MAP(N => 128) PORT MAP(a => aes1\_out, b => (0 => '0', OTHERS => '0'), c => mux\_r2\_0\_out, sel => sel\_mux\_r\_0\_in);

r2: Register128 PORT MAP(reg\_in => mux\_r2\_0\_out, clk => clk, rst => rst, reg\_out => r2\_out);

mux\_K: Mux GENERIC MAP(N => 128) PORT MAP(a => k\_in(127 downto 0), b => k\_in(255 downto 128), c => mux\_key\_out, sel => sel\_mux\_key\_in);

pbo: PermutationBO PORT MAP(pbo\_in => mod32a\_out , pbo\_out => permuation\_out);

key\_xor\_permuation <= mux\_key\_out XOR permuation\_out;

mux\_r1: Mux GENERIC MAP(N => 128) PORT MAP(a => key\_xor\_permuation, b => permuation\_out, c => mux\_r1\_out, sel => sel\_mux\_r1\_in);

mux\_r1\_0: Mux GENERIC MAP(N => 128) PORT MAP(a => mux\_r1\_out, b => (0 => '0', OTHERS => '0'), c => mux\_r1\_0\_out, sel => sel\_mux\_r\_0\_in);

r1: Register128 PORT MAP(reg\_in => mux\_r1\_0\_out, clk => clk, rst => rst, reg\_out => r1\_out);

mod32b: Modulo32 PORT MAP(mod32\_in0 => r1\_out, mod32\_in1 => lfsr\_t1\_in, mod32\_out => mod32b\_out);

keystream\_out <= r2\_out XOR mod32b\_out;

END ARCHITECTURE;

**SnowV\_Core**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY SnowV\_Core IS

PORT(

key\_in: IN std\_logic\_vector(255 downto 0);

iv\_in: IN std\_logic\_vector(127 downto 0);

keystream\_out: INOUT std\_logic\_vector(127 downto 0);

clk,rst,sel\_mux\_in,sel\_mux\_a\_high2\_in,sel\_mux\_r\_0\_in,sel\_mux\_r1\_in,sel\_mux\_key\_in: std\_logic

);

END ENTITY;

ARCHITECTURE Circuit OF SnowV\_Core IS

COMPONENT LFSR\_Parallel IS

PORT(

k\_in : IN std\_logic\_vector(255 downto 0);

iv\_in,keystream\_in: IN std\_logic\_vector(127 downto 0);

t1\_out,t2\_out: OUT std\_logic\_vector(127 downto 0);

clk,rst,sel\_mux\_in,sel\_mux\_a\_high2\_in: IN std\_logic

);

END COMPONENT;

COMPONENT FSM IS

PORT(

lfsr\_t1\_in,lfsr\_t2\_in: IN std\_logic\_vector(127 downto 0);

k\_in: IN std\_logic\_vector(255 downto 0);

clk,rst,sel\_mux\_r\_0\_in,sel\_mux\_r1\_in,sel\_mux\_key\_in: IN std\_logic;

keystream\_out: OUT std\_logic\_vector(127 downto 0)

);

END COMPONENT;

SIGNAL t1,t2 : std\_logic\_vector(127 downto 0);

BEGIN

LFSR\_inst: LFSR\_Parallel PORT MAP(k\_in => key\_in, iv\_in => iv\_in,keystream\_in => keystream\_out,

t1\_out => t1, t2\_out => t2,

clk => clk, rst => rst,

sel\_mux\_in => sel\_mux\_in ,sel\_mux\_a\_high2\_in => sel\_mux\_a\_high2\_in);

FSM\_inst: FSM PORT MAP(lfsr\_t1\_in => t1, lfsr\_t2\_in => t2, k\_in => key\_in,

clk => clk, rst => rst,

keystream\_out => keystream\_out, sel\_mux\_r\_0\_in => sel\_mux\_r\_0\_in ,

sel\_mux\_r1\_in=> sel\_mux\_r1\_in, sel\_mux\_key\_in => sel\_mux\_key\_in);

END ARCHITECTURE;

**Testbench για test vector #1**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY SnowV\_Core\_tb IS

--PORT();

END ENTITY;

ARCHITECTURE Circuit OF SnowV\_Core\_tb IS

COMPONENT SnowV\_Core IS

PORT(

key\_in: IN std\_logic\_vector(255 downto 0);

iv\_in: IN std\_logic\_vector(127 downto 0);

keystream\_out: INOUT std\_logic\_vector(127 downto 0);

clk,rst,sel\_mux\_in,sel\_mux\_a\_high2\_in,sel\_mux\_r\_0\_in,sel\_mux\_r1\_in,sel\_mux\_key\_in: std\_logic

);

END COMPONENT;

SIGNAL key: std\_logic\_vector(255 downto 0):= X"0000000000000000000000000000000000000000000000000000000000000000";

SIGNAL iv: std\_logic\_vector(127 downto 0):= X"00000000000000000000000000000000";

SIGNAL sel\_mux\_in,sel\_mux\_r\_0\_in ,sel\_mux\_r1\_in: std\_logic := '1';

SIGNAL rst,clk,sel\_mux\_a\_high2\_in,sel\_mux\_key\_in: std\_logic:= '0';

SIGNAL keystream\_out: std\_logic\_vector(127 downto 0);

BEGIN

DUT: SnowV\_Core PORT MAP(key\_in => key, iv\_in => iv, keystream\_out => keystream\_out, clk => clk, rst => rst,

sel\_mux\_in => sel\_mux\_in, sel\_mux\_a\_high2\_in => sel\_mux\_a\_high2\_in,

sel\_mux\_r\_0\_in => sel\_mux\_r\_0\_in ,

sel\_mux\_r1\_in=> sel\_mux\_r1\_in, sel\_mux\_key\_in => sel\_mux\_key\_in);

clk <= NOT clk after 5ns;

rst<= '1','0' after 5ns;

PROCESS

BEGIN

wait for 5ns;

sel\_mux\_r\_0\_in <= '0';

sel\_mux\_in <='0';

FOR i IN 1 TO 15 LOOP

sel\_mux\_a\_high2\_in <= '1';

wait for 10ns;

IF (i=14) THEN

sel\_mux\_r1\_in <='0';

sel\_mux\_key\_in <='0';

ELSIF (i=15) THEN

sel\_mux\_key\_in <='1';

end if;

END LOOP;

wait for 10ns;

sel\_mux\_r1\_in <= '1';

sel\_mux\_a\_high2\_in <= '0';

wait for 70ns;

END PROCESS;

END ARCHITECTURE;

**Testbench για test vector #2**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY SnowV\_Core\_tb IS

--PORT();

END ENTITY;

ARCHITECTURE Circuit OF SnowV\_Core\_tb IS

COMPONENT SnowV\_Core IS

PORT(

key\_in: IN std\_logic\_vector(255 downto 0);

iv\_in: IN std\_logic\_vector(127 downto 0);

keystream\_out: INOUT std\_logic\_vector(127 downto 0);

clk,rst,sel\_mux\_in,sel\_mux\_a\_high2\_in,sel\_mux\_r\_0\_in,sel\_mux\_r1\_in,sel\_mux\_key\_in: std\_logic

);

END COMPONENT;

SIGNAL key: std\_logic\_vector(255 downto 0):= X"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff";

SIGNAL iv: std\_logic\_vector(127 downto 0):= X"ffffffffffffffffffffffffffffffff";

SIGNAL sel\_mux\_in,sel\_mux\_r\_0\_in ,sel\_mux\_r1\_in: std\_logic := '1';

SIGNAL rst,clk,sel\_mux\_a\_high2\_in,sel\_mux\_key\_in: std\_logic:= '0';

SIGNAL keystream\_out: std\_logic\_vector(127 downto 0);

BEGIN

DUT: SnowV\_Core PORT MAP(key\_in => key, iv\_in => iv, keystream\_out => keystream\_out, clk => clk, rst => rst,

sel\_mux\_in => sel\_mux\_in, sel\_mux\_a\_high2\_in => sel\_mux\_a\_high2\_in,

sel\_mux\_r\_0\_in => sel\_mux\_r\_0\_in ,

sel\_mux\_r1\_in=> sel\_mux\_r1\_in, sel\_mux\_key\_in => sel\_mux\_key\_in);

clk <= NOT clk after 5ns;

rst<= '1','0' after 5ns;

PROCESS

BEGIN

wait for 5ns;

sel\_mux\_r\_0\_in <= '0';

sel\_mux\_in <='0';

FOR i IN 1 TO 15 LOOP

sel\_mux\_a\_high2\_in <= '1';

wait for 10ns;

IF (i=14) THEN

sel\_mux\_r1\_in <='0';

sel\_mux\_key\_in <='0';

ELSIF (i=15) THEN

sel\_mux\_key\_in <='1';

end if;

END LOOP;

wait for 10ns;

sel\_mux\_r1\_in <= '1';

sel\_mux\_a\_high2\_in <= '0';

wait for 70ns;

END PROCESS;

END ARCHITECTURE;

**Testbench για test vector #3**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.all;

ENTITY SnowV\_Core\_tb IS

--PORT();

END ENTITY;

ARCHITECTURE Circuit OF SnowV\_Core\_tb IS

COMPONENT SnowV\_Core IS

PORT(

key\_in: IN std\_logic\_vector(255 downto 0);

iv\_in: IN std\_logic\_vector(127 downto 0);

keystream\_out: INOUT std\_logic\_vector(127 downto 0);

clk,rst,sel\_mux\_in,sel\_mux\_a\_high2\_in,sel\_mux\_r\_0\_in,sel\_mux\_r1\_in,sel\_mux\_key\_in: std\_logic

);

END COMPONENT;

SIGNAL key: std\_logic\_vector(255 downto 0):= X"faeadacabaaa9a8a7a6a5a4a3a2a1a0a5f5e5d5c5b5a59585756555453525150";

SIGNAL iv: std\_logic\_vector(127 downto 0):= X"1032547698badcfeefcdab8967452301";

SIGNAL sel\_mux\_in,sel\_mux\_r\_0\_in ,sel\_mux\_r1\_in: std\_logic := '1';

SIGNAL rst,clk,sel\_mux\_a\_high2\_in,sel\_mux\_key\_in: std\_logic:= '0';

SIGNAL keystream\_out: std\_logic\_vector(127 downto 0);

BEGIN

DUT: SnowV\_Core PORT MAP(key\_in => key, iv\_in => iv, keystream\_out => keystream\_out, clk => clk, rst => rst,

sel\_mux\_in => sel\_mux\_in, sel\_mux\_a\_high2\_in => sel\_mux\_a\_high2\_in,

sel\_mux\_r\_0\_in => sel\_mux\_r\_0\_in ,

sel\_mux\_r1\_in=> sel\_mux\_r1\_in, sel\_mux\_key\_in => sel\_mux\_key\_in);

clk <= NOT clk after 5ns;

rst<= '1','0' after 5ns;

PROCESS

BEGIN

wait for 5ns;

sel\_mux\_r\_0\_in <= '0';

sel\_mux\_in <='0';

FOR i IN 1 TO 15 LOOP

sel\_mux\_a\_high2\_in <= '1';

wait for 10ns;

IF (i=14) THEN

sel\_mux\_r1\_in <='0';

sel\_mux\_key\_in <='0';

ELSIF (i=15) THEN

sel\_mux\_key\_in <='1';

end if;

END LOOP;

wait for 10ns;

sel\_mux\_r1\_in <= '1';

sel\_mux\_a\_high2\_in <= '0';

wait for 70ns;

END PROCESS;

END ARCHITECTURE;