ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αξιολόγηση σε υλισμικό (FPGAs) νέων αλγορίθμων κρυπτογραφίας τύπου Lightweight

Κυβέλος Γεώργιος Κωνσταντίνος

ΑΜ : 2465

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: Παρασκευάς Κίτσος

ΠΑΤΡΑ 2022

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή
Πάτρα, Ημερομηνία

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ
1.
2.
3.

Υπεύθυνη Δήλωση Φοιτητή

Βεβαιώνω ότι είμαι συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τη συγκεκριμένη εργασία.

Η έγκριση της πτυχιακής εργασίας από το Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Πελοποννήσου δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος.

Περίληψη

Το τελευταίο διάστημα υπάρχει έντονη η ανάγκη για εξεύρεση νέων ασφαλών αλγορίθμων κρυπτογράφησης κατάλληλων για εξειδικευμένες υπηρεσίες. Ως προς τη προσπάθεια αυτή τα τελευταία χρόνια υπάρχει σε ισχύ ένας νέος σημαντικός διαγωνισμός για τέτοιου τύπου αλγορίθμων στην οποία λαμβάνουν μέρος τόσο ερευνητικά ινστιτούτα και πανεπιστήμια όσο και γνωστές εταιρίες με σκοπό την εύρεση του καλύτερου αλγορίθμου καταρχάς από άποψη ασφάλειας για συστήματα. Οι αλγόριθμοι αυτοί είναι τύπου Lightweight και δημιουργήθηκαν για τον διαγωνισμό που διοργάνωσε η NIST (National institute of standards and technology). Σκοπός της πτυχιακής εργασίας αυτής είναι να υλοποιηθούν δύο από τους αλγόριθμους που έχουν προταθεί σε αυτόν το διαγωνισμό σε κώδικα VHDL και να γίνει η αντίστοιχη υλοποίηση τους σε FPGA. Οι αλγόριθμοι που επιλέχτηκαν να υλοποιηθούν είναι για αρχή ο αλγόριθμος Spix που υποβλήθηκε και σχεδιάστηκε από τους Riham AlTawy, Guang Gong, Morgan He, Kalikinkar Mandal, και Raghvendra Rohit, και ο δεύτερος αλγόριθμος που επιλέχτηκε είναι ο GIFT-COFB που υποβλήθηκε και σχεδιάστηκε από τους Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin Yu, Sasaki Siang, Meng Sim, Yosuke Todo. Όλα τα κομμάτια κώδικα που υλοποιήθηκαν, επιβεβαιώθηκαν σε επίπεδο προσομοίωσης με το πρόγραμμα Xilinx ISE Design Suite 14.7.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

1.1. Εισαγωγή

Σκοπός της παρούσας πτυχιακής εργασίας ήταν να επιλεχτούν δύο αλγόριθμοι κρυπτογράφησης από τους αλγορίθμους που υποβλήθηκαν στο διαγωνισμό που διοργάνωσε η NIST (National institute of standards and technology), και στη συνέχεια να υλοποιηθεί το υλικό που περιγράφει ο αλγόριθμος με τη χρήση της γλώσσας VHDL. Ο πρώτος αλγόριθμος που επιλέχτηκε είναι ο SPIX. Σκοπός της επιλογής αυτής, ήταν η εξοικείωση με τον τρόπο που υλοποιούνται οι αλγόριθμοι σε υλικό και η αρχιτεκτονική που χρησιμοποιείτε. Μεγάλη βαρύτητα στην επιλογή του αλγορίθμου SPIX έδωσε ο αλγόριθμος ACE που επίσης είχε υποβληθεί στο διαγωνισμό ο οποίος είχε υλοποιημένο υλικό σε κώδικα VHDL και βασίζεται στην ίδια αρχιτεκτονική με τον αλγόριθμο SPIX. Αυτό οδήγησε στην πιο εύκολη κατανόηση του υλικού και στον πιο εύκολο σχεδιασμό του από τη στιγμή που ως μέτρο αναφοράς χρησιμοποιήθηκε το υλικό του αλγορίθμου ACE. Μετά την κατανόηση των αρχιτεκτονικών που χρησιμοποιούνται στην υλοποίηση του υλικού και τη εξοικείωση με τους αλγορίθμους, επιλέχτηκε ένας δεύτερος αλγόριθμος ο οποίος θα έπρεπε να υλοποιηθεί εξ' ολοκλήρου από την αρχή σε υλικό χωρίς τη βοήθεια κάποιου παραπλήσιου κώδικα. Ο δεύτερος αλγόριθμος που επιλέχτηκε είναι ο GIFT-COFB. Για την καταγραφή του κώδικα VHDL χρησιμοποιήθηκε το πρόγραμμα Notepad ++ και για προσομοιώσεις και μικρό διορθώσεις του κώδικα χρησιμοποιήθηκε το πρόγραμμα ISE Project Navigator 14.7 της Xilinx. Για τη συγγραφή του κείμενου της παρούσας πτυχιακής εργασίας χρησιμοποιήθηκε το πρόγραμμα Microsoft Office 365.

Στο δεύτερο κεφάλαιο της παρούσας εργασίας, περιγράφεται η αρχιτεκτονική του αλγορίθμου SPIX. Αρχικά, το κεφάλαιο ξεκινάει αναλύοντας τις βασικές προδιαγραφές του αλγορίθμου και στη συνέχεια αναλύονται οι φάσεις λειτουργίας του. Ακολουθεί, η ανάλυση των βημάτων του πυρήνα του αλγορίθμου που είναι μετάθεση sLiSCP και αναλύονται όλοι οι μηχανισμοί και ένθετες μεταθέσεις που εκτελούνται μαζί με την κύρια μετάθεση. Στη συνέχεια, αναλύετε το κύκλωμα του αλγορίθμου δείχνοντας πως συνδέονται όλα τα στοιχεία μεταξύ τους. Πριν το τέλος, αναφέρονται τα αρχεία που περιέχουν τους κώδικες VHDL που γράφτηκαν και ποιες λειτουργίες η στοιχεία κάθε αρχείο κώδικα περιέχει. Στο τέλος, δίνεται η λογική που οδήγησε στις σχεδιαστικές αποφάσεις του αλγορίθμου SPIX.

Στο τρίτο κεφάλαιο της παρούσας εργασίας, περιγράφεται η αρχιτεκτονική του αλγορίθμου GIFT-COFB. Αρχικά, το κεφάλαιο ξεκινάει αναλύοντας τις βασικές προδιαγραφές του αλγορίθμου. Ακολουθεί, η λεπτομερής ανάλυση των βημάτων και μηχανισμών του μπλοκ κρυπτογράφησης GIFT. Στη συνέχεια, ακολουθεί η λεπτομερής ανάλυση των λειτουργιών που περιέχει το μπλοκ COFB. Στη συνέχεια, εξηγείτε ο χρονισμός του αλγορίθμου και αναλύετε το κύκλωμα του αλγορίθμου ενώ ταυτόχρονα εξηγείτε ο τρόπος που αλληλοεπιδρά το μπλοκ GIFT με το μπλοκ COFB. Πριν το τέλος, αναφέρονται τα αρχεία που περιέχουν τους κώδικες VHDL που γράφτηκαν και ποιες λειτουργίες η στοιχεία κάθε αρχείο κώδικα περιέχει. Στο τέλος, δίνεται η λογική που οδήγησε στις σχεδιαστικές αποφάσεις του αλγορίθμου GIFT-COFB.

Στο τέταρτο κεφάλαιο της παρούσας εργασίας, περιέχει τα αποτελέσματα σύνθεσης και αποτελέσματα των λειτουργικών προσομοιώσεων από το υλικό που γράφτηκε με τη χρήση της γλώσσας VHDL. Οι προσομοιώσεις έγιναν μετά από επιτυχή σύνθεση του υλικού και γίνεται σχολιασμός των αποτελεσμάτων για καλύτερη κατανόηση.

Στο πέμπτο κεφάλαιο, δίνονται τα συμπεράσματα που βγήκαν μετά την ολοκλήρωση της πτυχιακής εργασίας.

Στο τέλος, υπάρχει βιβλιογραφία με τα βιβλία και ιστότοπους που χρησιμοποιήθηκαν κατά την εκπόνηση της πτυχιακής εργασίας όσο ακόμα και τα αρχεία σε κώδικα VHDL που υλοποιήθηκαν.

Κεφάλαιο 2

Αλγόριθμος Spix

2.1. Γενική Περιγραφή

Ο Spix είναι ένας αλγόριθμος κρυπτογράφησης που σχεδιάστηκε από τους Riham AlTawy, Guang Gong, Morgan He, Kalikinkar Mandal, και Raghvendra Rohit και υποστηρίζει επεξεργασία μηνυμάτων και σχετικών δεδομένων (Associated Data). Χρησιμοποιείται για ελαφριές εφαρμογές που μπορούν να εγγυηθούν τη nonce μοναδικότητα για ασφάλεια. Ο αλγόριθμος αυτός προσφέρει 128-bit ασφάλεια και υποστηρίζει δύο λειτουργίες. Στο ανώτερο επίπεδο, ο αλγόριθμος Spix υιοθετεί την κατασκευή monkey duplex [6][7] η οποία υποστηρίζει δύο διαφορετικά είδη εκτέλεσης για την εφαρμογή της μετάθεσης που περιέχει. Τα δύο διαφορετικά είδη εκτέλεσης επηρεάζουν το αριθμό των γύρων που θα εκτελεστεί η μετάθεση. Στην ουσία ο Spix είναι ένας αλγόριθμος κρυπτογράφησης ο οποίος περιέχει δύο τρόπου εκτέλεσης της μετάθεσης sLiSCP-light [14]. Σε αυτό το κεφάλαιο, θα γίνει παρουσίαση των προδιαγραφών του αλγορίθμου Spix μαζί με τον τρόπο λειτουργίας του και τον τρόπο χρήσης των μεταθέσεων του. Στη συνέχεια θα ακολουθήσει λειτουργική προσομοίωση του αλγορίθμου και θα αναλυθεί η λογική στη σχεδίαση του.

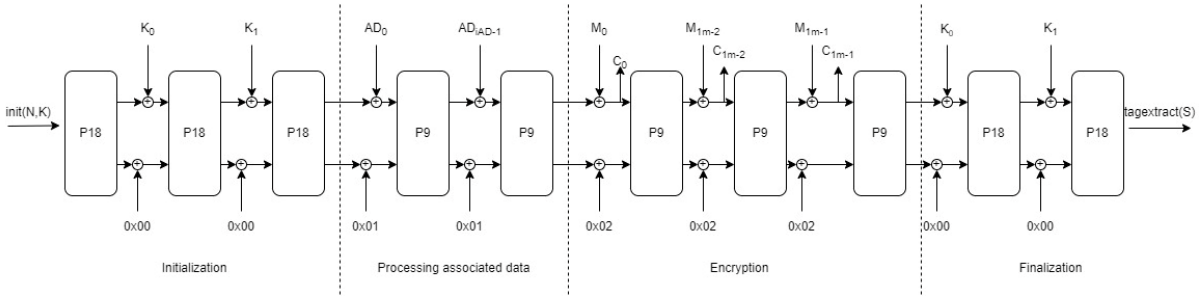
2.2. Επισκόπηση Spix

Ο αλγόριθμος Spix προσφέρει 128-bit ασφάλεια και υποστηρίζει δύο λειτουργίες. Η λειτουργία της κρυπτογράφησης δέχεται ένα κλειδί (K) μεγέθους k-bits, μία nonce μεγέθους k-bits, ένα μήνυμα (M) μεταβαλλόμενου μεγέθους και τα σχετικά δεδομένα (AD). Παράγει ένα κρυπτογραφημένο κείμενο (C) ίσου μεγέθους με το κείμενο και μία ετικέτα (T) μεγέθους k-bits η οποία χρησιμοποιείτε για την αυθεντικοποίηση της nonce, του μηνύματος και των σχετικών δεδομένων. Αντίστοιχα για τη λειτουργία της αποκρυπτογράφησης δέχεται ένα κλειδί, μία nonce, το κρυπτογραφημένο κείμενο, και σχετικά δεδομένα. Παράγει το αρχικό κείμενο αν η αυθεντικοποίηση με τη χρήση ετικέτας γίνει επιτυχώς αλλιώς παράγει μήνυμα σφάλματος.

2.3. Τρόπος λειτουργίας

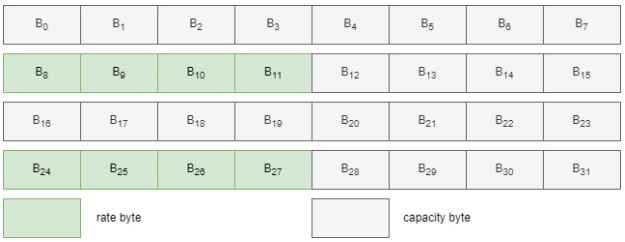
Η λειτουργία του αλγορίθμου Spix απεικονίζεται στο σχήμα 1. Επιπλέον ο αλγόριθμος χρησιμοποιεί φάσεις αρχικοποίησης και οριστικοποίησης που κάνουν την ανάκτηση κλειδιών δύσκολη και καθιστά την πλαστογράφηση μη εφικτή ακόμα και αν είναι γνωστή η εσωτερική κατάσταση του αλγορίθμου. Όπως απεικονίζεται στο παρακάτω σχήμα, η λειτουργία κρυπτογράφησης περιέχει τέσσερις φάσεις. Οι φάσεις είναι η αρχικοποίηση (Initialization), επεξεργασία σχετικών δεδομένων (Processing Associated Data), κρυπτογράφηση (Encryption) και οριστικοποίηση (Finalization). Η αποκρυπτογράφηση είναι αντίστοιχη με τη διαφορά ότι αντί για φάση κρυπτογράφησης έχει φάση αποκρυπτογράφησης. Οι φάσεις αρχικοποίησης και οριστικοποίησης χρησιμοποιούν μετάθεση sLiSCP δεκαοχτώ βημάτων για κάθε κομμάτι κλειδιού

και nonce που δέχεται ο αλγόριθμος ενώ οι υπόλοιπες φάσεις χρησιμοποιούν μετάθεση sLiSCP εννέα βημάτων για κάθε κομμάτι σχετικών δεδομένων και κειμένου προς κρυπτογράφηση που δέχεται ο αλγόριθμος. Οι φάσεις κρυπτογράφησης θα αναλυθούν σε επόμενα κεφάλαια.



Σχήμα 1 – Λειτουργία κρυπτογράφησης αλγορίθμου Spix

Ο πυρήνας του αλγορίθμου είναι η μετάθεση sLiSCP. Η μετάθεση αυτή επεξεργάζεται μία κατάσταση S μεγέθους 256 bits. Η κατάσταση αυτή που επεξεργάζεται, χωρίζεται σε rate bits (Sr) και capacity bits (Sc). Ο χωρισμός της κατάστασης απεικονίζεται στο σχήμα 2 όπου το κομμάτι κατάστασης rate απεικονίζεται με πράσινο χρώμα και το κομμάτι κατάστασης capacity απεικονίζεται με γκρι χρώμα. Όλο το κομμάτι της κατάστασης αρχικοποιείται με κλειδί και nonce ενώ το πράσινο μέρος της κατάστασης Sr χρησιμοποιείτε για να απορροφηθούν τα σχετικά δεδομένα και τα κείμενα προς κρυπτογράφηση στην κατάσταση του αλγορίθμου για επεξεργασία.



Σχήμα 2 – Απεικόνιση κατάστασης S

2.3.1 Φάση αρχικοποίησης

Σε αυτή τη φάση, η κατάσταση S που έχει ως είσοδο ο αλγόριθμος, αρχικοποιείται με τη nonce μεγέθους 128 bits και με το κλειδί επίσης μεγέθους 128 bits. Η nonce θα χωριστεί σε δύο κομμάτια (N0 και N1) των 64 bits και θα αποθηκευτούν στα πρώτα οχτώ bytes και στο τρίτο σετ των οχτώ bytes αντίστοιχα της κατάστασης. Στη συνέχεια το κλειδί επίσης θα χωριστεί σε δύο κομμάτια (K0 και K1) των 64 bits και θα αποθηκευτούν στο δεύτερο και τέταρτος σετ των οχτώ bits της κατάστασης. Αφού φορτωθεί η κατάσταση με τις τιμές, ακολουθεί η μετάθεση sLiSCP δεκαοχτώ βημάτων στην εσωτερική κατάσταση και η διαδικασία αρχικοποίησης ολοκληρώνεται και ο αλγόριθμος είναι έτοιμος να δεχτεί σχετικά δεδομένα και κείμενο προς κρυπτογράφηση. Επιπλέον σε αυτή τη φάση ο domain separator παράγει την τιμή 00 αλλά δεν θα απορροφηθεί στην εσωτερική κατάσταση. Η διαδικασία της αρχικοποίησης της κατάστασης με nonce και κλειδί απεικονίζεται και στο σχήμα 3.

$B_0, \ldots, B_7 \leftarrow N_0[0] \ldots, N_0[7]$

$B_{16}, \ldots, B_{23} \leftarrow N_1[0] \ldots, N_1[7]$

$B_8, \ldots, B_{15} \leftarrow K_0[0] \ldots, K_0[7]$

$B_{24}, \ldots, B_{31} \leftarrow K_1[0] \ldots, K_1[7]$

$S \leftarrow P^{18}(S)$

$S \leftarrow P^{18}(Sr \oplus K_i, S_c), i = 0, 1$

Σχήμα 3 – Αρχικοποίηση αλγορίθμου Spix

2.3.2 Επεξεργασία σχετικών δεδομένων

Σε αυτήν τη φάση τα σχετικά δεδομένα χωρίζονται σε μπλοκς των 64 bits. Κάθε μπλοκ σχετικών δεδομένων χωρίζεται σε δύο υπομπλοκ των 32 bits και απορροφάτε στο Sr κομμάτι της εσωτερικής κατάστασης S με τη χρήση της εντολής XOR. Το πρώτο υπομπλοκ αποθηκεύεται στο B8 έως B11 της Sr κατάστασης ενώ το δεύτερο υπομπλοκ αποθηκεύεται στο B24 έως B27 της Sr κατάστασης. Επιπλέον, η σταθερά τιμή 01 που παράγει ο domain separator απορροφάτε στο τέλος της εσωτερικής κατάστασης S με τη χρήση εντολής XOR ώστε να δείξει στον αλγόριθμο ότι είναι φάση επεξεργασίας σχετικών δεδομένων. Στη συνέχεια, εφαρμόζεται μετάθεση sLiSCP εννέα βημάτων στην εσωτερική κατάσταση. Αφού τελειώσει η επεξεργασία όλων των σχετικών δεδομένων τότε ο αλγόριθμος είναι έτοιμος να προχωρήσει σε φάση κρυπτογράφησης.

2.3.3 Φάση Κρυπτογράφησης

Παρόμοια με τη φάση επεξεργασίας σχετικών δεδομένων, έτσι και στη φάση κρυπτογράφησης κάθε μπλοκ μηνύματος απορροφάτε στο Sr κομμάτι της εσωτερικής κατάστασης S με την εντολή XOR. Το πρώτο υπομπλοκ μηνύματος αποθηκεύεται στο B8 έως B11 της Sr κατάστασης ενώ το δεύτερο υπομπλοκ αποθηκεύεται στο B24 έως B27 της Sr κατάστασης. Επιπλέον, η σταθερά τιμή 02 που παράγει ο domain separator απορροφάτε στο τέλος της εσωτερικής κατάστασης S με τη χρήση της εντολής XOR ώστε να δείξει στον αλγόριθμο ότι είναι φάση κρυπτογράφησης. Στη συνέχεια, εφαρμόζεται μετάθεση sLiSCP εννέα βημάτων στην εσωτερική κατάσταση. Αφού τελειώσει η επεξεργασία όλων των μηνυμάτων προς κρυπτογράφηση τότε ο αλγόριθμος είναι έτοιμος να προχωρήσει σε φάση οριστικοποίησης.

2.3.4 Φάση οριστικοποίησης

Σε αυτή τη φάση, ο domain separator παράγει τιμή 00 και εφαρμόζεται στο τέλος της εσωτερικής κατάστασης με τη χρήση της εντολής XOR ώστε να δείξει στον αλγόριθμο ότι είναι φάση οριστικοποίησης. Η τιμή αυτή υποδεικνύει την έναρξη της οριστικοποίησης και τα μπλοκ κλειδιών απορροφώνται στην κατάσταση. Στη συνέχεια εφαρμόζεται η μετάθεση sLiSCP δεκαοχτώ βημάτων. Τέλος, σε αυτή τη φάση παράγεται και ετικέτα μεγέθους 128 bits για λόγους επαλήθευσης σε μετέπειτα αποκρυπτογράφηση.

2.3.5 Αποκρυπτογράφηση

Η διαδικασία αποκρυπτογράφησης είναι παρόμοια με τη διαδικασία κρυπτογράφησης, με τη διαφορά ότι αντί φάση κρυπτογράφησης εκτελείτε φάση αποκρυπτογράφησης.

2.4. Μηχανισμοί αλγορίθμου Spix.

Στο προηγούμενο κεφάλαιο αναφέρθηκε η παραγωγή μία σταθερά τιμής που παράγεται από ένα μηχανισμό που ονομάζεται domain separator. Ο domain separator είναι ένας μηχανισμός που παράγει μία σταθερά τιμή μεγέθους των δύο bits η οποία χρησιμοποιείτε για να δείχνει στον αλγόριθμο σε πια φάση να αλλάζει. Η σταθερά τιμή αποθηκεύεται στο τέλος της εσωτερικής κατάστασης S του αλγορίθμου. Οι τιμές που αντιστοιχούν στην κάθε φάση που περιέχει ο αλγόριθμος, απεικονίζονται στο σχήμα 4.

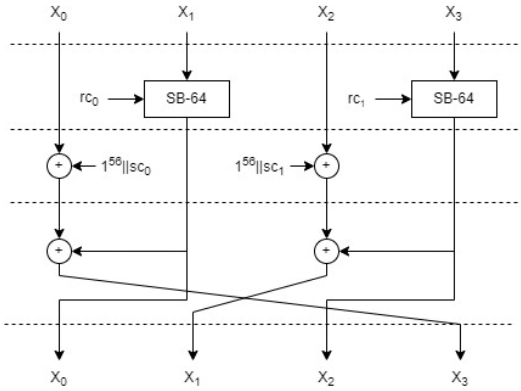
Initialization	Proc. AD	Enc. & Dec.	Finalization
0x00	0x01	0x02	0x00

Σχήμα 4 – Domain separator σταθερές

Επιπλέον πολύ σημαντικός μηχανισμός είναι το padding. Όπως αναφέρθηκε και πριν ο αλγόριθμός δέχεται μήνυμα μεταβαλλόμενου μεγέθους, αλλά μπορεί να επεξεργαστεί συγκεκριμένο μέγεθος μηνύματος. Έτσι λοιπόν, η λειτουργία padding προσθέτει μηδενικά στα σχετικά δεδομένα και στα μηνύματα προς κρυπτογράφηση έτσι ώστε όταν ο αλγόριθμος τα χωρίσει σε μπλοκ να είναι όλα ίσου μεγέθους. Αν το μήνυμα είναι ήδη σε σωστό μέγεθος, τότε θα το φορτώσει κατευθείαν στον αλγόριθμο. Ο έλεγχος του μηχανισμού αυτού γίνεται κάθε φορά που ένα νέο μπλοκ δεδομένων εισέρχεται στον αλγόριθμο για επεξεργασία.

2.5. Μετάθεση sLiSCP-light

Σε προηγούμενο κεφάλαιο κατά την περιγραφή των φάσεων της διαδικασίας κρυπτογράφησης αναφέρθηκε μία μετάθεση sLiSCP η οποία εκτελείτε για συγκεκριμένο αριθμό βημάτων αναλόγως την φάση. Το όνομα sLiSCP χρησιμοποιείτε για συντομία του sLiSCP-light και είναι βασικό κομμάτι του αλγορίθμου Spix. Η μετάθεση sLiSCP δέχεται είσοδο μεγέθους 256 bits και παράγει μία έξοδο ίσου μεγέθους. Η μετάθεση χωρίζει την είσοδο που δέχεται σε τέσσερα μπλοκ των 64 bits κάθε ένα. Στη συνέχεια ξεκινάει το μετασχηματισμό των τεσσάρων μπλοκ εκτελώντας τρία βήματα. Το πρώτο βήμα είναι η αντικατάσταση των μπλοκ (SSb), στη συνέχεια ακολουθεί η προσθήκη σταθερών βήματος (ASc) και τελευταίο βήμα είναι η αλλαγή των θέσεων των μπλοκ (MSb). Στο σχήμα 5 δίνεται η γραφική αναπαράσταση των βημάτων της μετάθεσης sLiSCP.



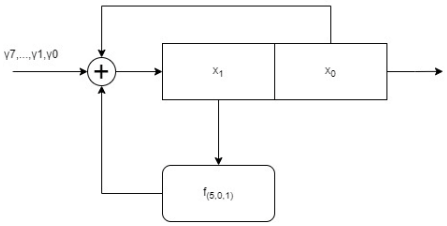
Σχήμα 5 – Μετάθεση sLiSCP-light

2.5.1 Αντικατάσταση μπλοκ (SSb)

Σε αυτό το βήμα το πρώτο και τρίτο μπλοκ (X0 και X2) παραμένουν ίδια και προχωράνε στο επόμενο βήμα. Το δεύτερο και τέταρτο μπλοκ (X1 και X3) τους εφαρμόζεται μία μικρότερη ένθετη μετάθεση που λέγεται Simeck Box (SB-64). Η ένθετη μετάθεση SB-64 εκτελείτε για οχτώ βήματα για κάθε ένα βήμα που εκτελείτε η μετάθεση sLiSCP. Αφού εφαρμοστούν οι λειτουργίες της μετάθεσης SB-64 σε αυτά τα μπλοκ, τότε έξοδος τους θα προχωρήσει στο επόμενο βήμα.

2.5.1.1 Simeck Box

Το Simeck Box είναι μία ένθετη μετάθεση η οποία δέχεται μία είσοδο των 64 bits και παράγει μία έξοδο ίσου μεγέθους. Σε αυτό τον αλγόριθμο εκτελείτε για οχτώ βήματα. Σε κάθε βήμα που εκτελείτε η ένθετη μετάθεση SB-64, προσθέτετε και μία σταθερά τιμή (rc) στο εσωτερικό της κατάστασης. Η σταθερά τιμή rc παράγεται από έναν 7 bit LFSR. Στο σχήμα 6 δίνεται η γραφική αναπαράσταση ενός βήματος Simeck Box που χρησιμοποιείτε στη μετάθεση sLiSCP.



Σχήμα 6 – Simeck Box

2.5.2 Προσθήκη σταθερών (ASc)

Αυτό το βήμα δέχεται τα ενημερωμένα μπλοκς από το προηγούμενο βήμα και εκτελεί περαιτέρω λειτουργίες. Για αρχή στο πρώτο και τρίτο μπλοκ (X0 και X2) προσθέτονται οι σταθερές βήματος (sc) που παράγονται από τον ίδιο 7 bit LFSR που παράγει και τις σταθερές της ένθετης μετάθεσης SB-64. Οι σταθερές αυτές έχουν μέγεθος 8 bits και προσθέτονται στα

τελευταία οχτώ bits της κατάστασης που περιέχουν αυτά τα μπλοκ. Το δεύτερο και τέταρτο μπλοκ (X1 και X3) παραμένουν ίδια και προχωράνε στο επόμενο βήμα.

2.5.3 Αλλαγή θέσεων μπλοκ (MSb)

Σε αυτό το βήμα αρχικά θα εκτελεστεί η πράξη XOR μεταξύ των μπλοκ X0 και X2 με τα μπλοκ X1 και X3 αντίστοιχα. Το αποτέλεσμα των πράξεων θα αποθηκευτεί στα μπλοκ X0 και X2. Στη συνέχεια, θα γίνει αλλαγή των θέσεων των μπλοκ μετακινώντας όλα τα μπλοκ κυκλικά μια θέση αριστερά. (X0→X3, X1→X0, X2→X1, X3→X2)

2.5.4 Παραγωγή σταθερών

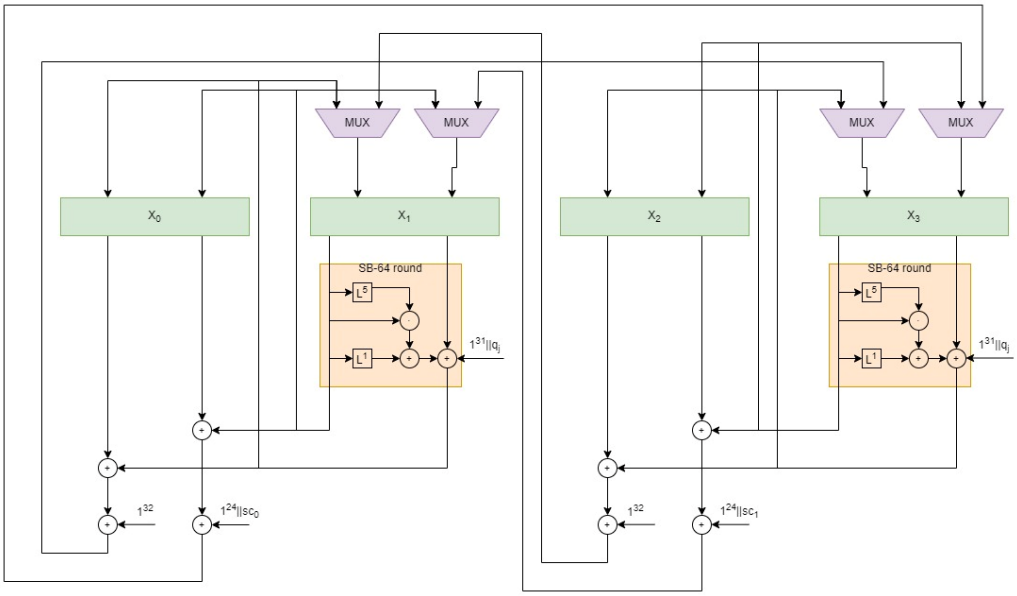
Στη περιγραφή της λειτουργία της μετάθεσης sLiSCP-light και της ένθετης μετάθεσης SB-64, αναφέρθηκαν δύο είδη σταθερών που προσθέτονται κατά τη διάρκεια των λειτουργιών που εκτελούνται από τις μεταθέσεις αυτές. Όπως φαίνεται στο σχήμα 5, στο πρώτο βήμα κατά την εκτέλεση των ένθετων μεταθέσεων SB-64, προσθέτονται σταθερές γύρου (rc), ενώ στο δεύτερο βήμα προσθέτονται σταθερές βήματος (sc) στα μπλοκ X0 και X2. Οι σταθερές αυτές παράγονται από έναν 7 bit LFSR (linear-feedback shift register). Στο σχήμα 7 δίνεται ο πίνακας με τις σταθερές που παράγονται ανά βήμα ρολογιού.

step i	(rc^i_0, rc^i_1)	(sc^i_0, sc^i_1)
0 - 5	(f, 47), (4, b2), (43, b5), (f1, 37), (44, 96), (73, ee)	(8, 64), (86, 6b), (e2, 6f), (89, 2c), (e6, dd), (ca, 99)
6 - 11	(e5, 4c), (b, f5), (47, 7), (b2, 82), (b5, a1), (37, 78)	(17, ea), (8e, 0f), (64, 04), (6b, 43), (6f, f1), (2c, 44)
12 - 17	(96, a2), (ee, b9), (4c, f2), (f5, 85), (7, 23), (82, d9)	(dd, 73), (99, e5), (ea, 0b), (0f, 47), (04, b2), (43, b5)

Σχήμα 7 – Πίνακας σταθερών τιμών αλγορίθμου Spix

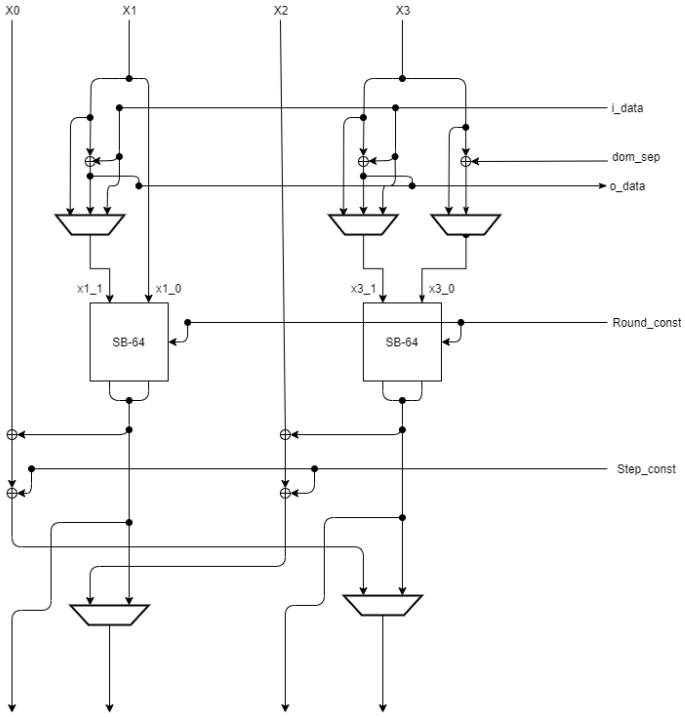
2.6. Υλοποίηση κυκλώματος αλγορίθμου Spix

Σε αυτό το κεφάλαιο θα δοθεί η υλοποίηση του υλικού για τον αλγόριθμο Spix. Ο αλγόριθμος αυτός είναι βελτιστοποιημένος για υλικό και έχει πολύ αποδοτικές ASIC εφαρμογές επειδή η μετάθεση sLiSCP-light που περιέχει, χρησιμοποιείτε σε βήματα. Πιο συγκεκριμένα, τα simeck box, η προσθήκη σταθερών τιμών και η ανάμιξη των μπλοκ εφαρμόζεται στη μισή κατάσταση του αλγορίθμου Spix. Επιπλέον, κάθε ένθετη μετάθεση SB-64 από μόνη της είναι μία πολύ αποδοτική λειτουργία Feistel. Το υλοποιημένο κύκλωμα της μετάθεσης sLiSCP-light απεικονίζεται στο σχήμα 8.



Σχήμα 8 - Κύκλωμα μετάθεσης sLiSCP-light

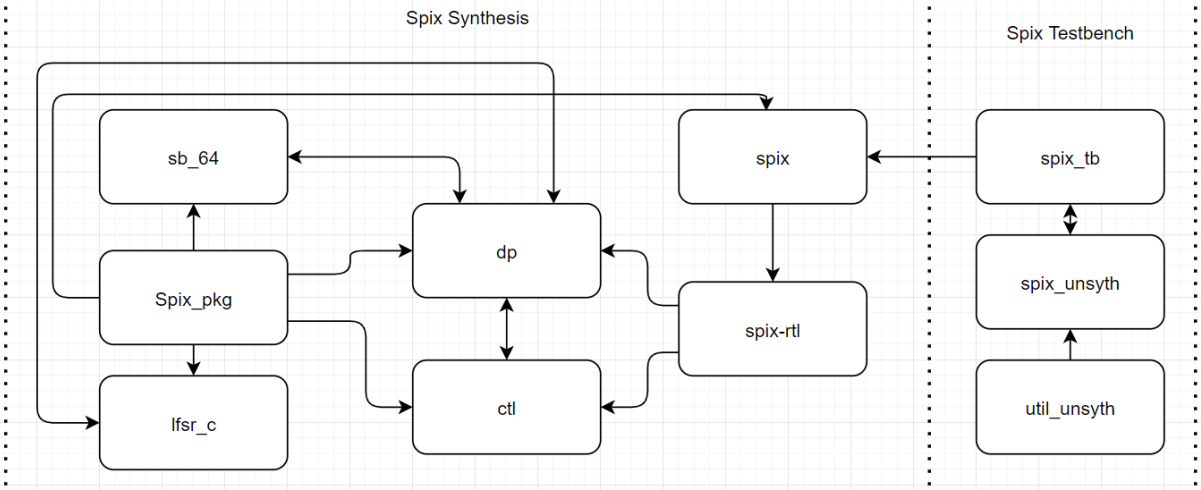
Στο παραπάνω σχήμα φαίνεται το κύκλωμα της μετάθεση sLiSCP-light, η οποία ακολουθεί την ίδια διαδικασία που φαίνεται στο σχήμα 5. Στα πράσινα πλαίσια φαίνεται η είσοδος της κατάστασης S του αλγορίθμου Spix η οποία έχει χωριστεί σε τέσσερα μπλοκ. Η επεξεργασία της κατάστασης γίνεται σε 32 bit κομμάτια. Για αυτό το λόγο κάθε μπλοκ χωρίζεται σε δύο υπομπλοκ όπως φαίνεται στην έξοδο των πράσινων πλαισίων. Στη συνέχεια οι έξοδοι των μπλοκ X0 και X2 δεν επηρεάζονται ενώ οι έξοδοι των μπλοκ X1 και X3 θα τους εφαρμοστεί η ένθετη μετάθεση simeck box και επιπλέον θα τούς προστεθεί η σταθερά γύρων που παράγεται από τον LFSR. Στη συνεχεία η προσθήκη των σταθερών βήματος στα μπλοκ X0 και X2 του βήματος ASc (Προσθήκη σταθερών) γίνεται στο δεύτερο κομμάτι της εξόδου των μπλοκ X0 και X2 (υπομπλοκς) με τη χρήση της εντολής XOR. Τέλος, με τη χρήση πολυπλεκτών, όπως φαίνεται στα μοβ πλαίσια, γίνεται η αριστερόστροφη μετακίνηση των μπλοκ και οι έξοδοι αποθηκεύονται στα πράσινα πλαίσια δηλαδή στην αρχή του κυκλώματος για χρήση σε επόμενο γύρο. Στο επόμενο σχήμα που ακολουθεί απεικονίζεται το κύκλωμα όλου του αλγορίθμου Spix. Στην ουσία είναι το κύκλωμα του σχήματος 8 αλλά του έχουν προστεθεί η γενική είσοδος του αλγορίθμου Spix (i_data), ο domain separator (dom_sep) και η έξοδος του αλγορίθμου Spix (o_data). Στην αρχή του κυκλώματος φαίνεται η είσοδος i_data. Αυτή η είσοδος περιέχει είτε nonce, κλειδί, σχετικά δεδομένα η κείμενο προς κρυπτογράφηση. Αναλόγως την τιμή που έχει παράξει ο domain separator, που είναι επίσης είσοδος του κυκλώματος, τότε η είσοδος i_data μέσω πράξεων και χρήση πολυπλεκτών θα φορτωθεί στην εσωτερική κατάσταση του κυκλώματος με ανάλογο τρόπος, όπως αναφέρθηκαν σε προηγούμενα κεφάλαια κατά την περιγραφή της λειτουργίας του αλγορίθμου. Όταν η λειτουργία του κυκλώματος τελειώνει στην έξοδο o_data περιέχετε το κρυπτογραφημένο η αρχικό κείμενο αναλόγως τι λειτουργία εκτελεί το κύκλωμα. Το υπόλοιπο κύκλωμα τρέχει όπως το κύκλωμα στο σχήμα 8.



Σχήμα 9 - Κύκλωμα αλγορίθμου Spix

2.7. Υλοποίηση αλγορίθμου Spix σε κώδικα VHDL

Η υλοποίηση του αλγορίθμου σε κώδικα vhdl έγινε δημιουργώντας δέκα αρχεία κώδικα τα οποία τα εφτά είναι υπεύθυνα για τη σύνθεση του αλγορίθμου και τα τρία είναι υπεύθυνα για την προσομοίωση του. Τα αρχεία sb_64 και lfsr_c περιέχουν το κύκλωμα για την ένθετη μετάθεση simeck box (SB-64) και λειτουργία LFSR αντίστοιχα. Το αρχείο dp περιέχει την μονάδα εκτέλεσης του κυκλώματος και περιγράφει το ολοκληρωμένο κύκλωμα του αλγορίθμου Spix που απεικονίζεται στο σχήμα 9 και περιέχει ως οντότητες τα κυκλώματα των αρχείων sb_64 και lfsr_c. Επιπλέον, περιέχει και μία διεργασία η οποία περιέχει την βασική λειτουργία του αλγορίθμου Spix. Το αρχείο dp το ελέγχει μία μονάδα ελέγχου η οποία περιγράφεται στο αρχείο ctl . Υπάρχουν δύο ακόμα αρχεία με όνομα spix και spix-rtl τα οποία περιέχουν την οντότητα του αλγόριθμου spix και την αρχιτεκτονική του στο υψηλότερο επίπεδο. Το αρχείο spix_pkg περιέχει στοιχεία, σταθερές, τύπους, υποτύπους και λειτουργίες που χρησιμοποιούνται επανειλημμένα στους κώδικες που υλοποιήθηκαν για τον αλγόριθμο Spix. Τέλος το αρχείο spix_tb και οι βιβλιοθήκες του που είναι τα αρχεία spix_unsyth και util_unsyth και χρησιμοποιούνται για την προσομοίωση του κυκλώματος. Στο σχήμα 10 που ακολουθεί φαίνεται ο τρόπος που συνδέονται τα αρχεία κώδικα που αναφέρθηκαν.



Σχήμα 10 - Διάγραμμα αρχείων VHDL

2.8. Σχεδιαστική λογική αλγορίθμου Spix

Σε αυτό το κεφάλαιο, θα δοθεί το σκεπτικό πίσω από τις σχεδιαστικές επιλογές που παρθήκαν για τον σχεδιασμό κάθε στοιχείου του αλγορίθμου Spix.

2.8.1 Επιλογή της λειτουργίας: Monkey Duplex Sponge Mode

Η λειτουργία που υιοθετήθηκε είναι μια παραλλαγή της κατασκευής Monkey duplex. Η κατασκευή Monkey duplex προτείνεται να χρησιμοποιείται σε λειτουργίες με κλειδί όπου η επαναχρησιμοποίηση της nonce μπορεί να μετριαστεί αποτελεσματικά. Το πιο σημαντικό του πλεονέκτημα είναι ότι χρησιμοποιεί δύο περιπτώσεις της μετάθεσης η οποία καθεμία έχει διαφορετικό αριθμό γύρων. Ένα τέτοιο χαρακτηριστικό ενισχύει την απόδοση του αλγορίθμου Spix επειδή ο αριθμός των γύρων της μετάθεσης κατά την απορρόφηση του μηνύματος μπορεί να βελτιστοποιηθεί μειώνοντάς το έτσι ώστε να αντιστέκεται σε διαφορικές επιθέσεις σε αυτό το συγκεκριμένο σενάριο. Εκτός από τους προηγούμενους λόγους, η λειτουργία που υιοθετήθηκε προσφέρει τα ακόλουθα χαρακτηριστικά:

- Ευελιξία κλειδιού και αντίστροφη αποκρυπτογράφηση. Απλή και ελαφριά λειτουργία όπου δεν απαιτείται ούτε προγραμματισμός κλειδιών ούτε εφαρμογή αλγορίθμου αποκρυπτογράφησης.
- Βασικές φάσεις αρχικοποίησης και οριστικοποίησης. Η ανάκτηση κλειδιού είναι δύσκολη ακόμα και αν ανακτηθεί η εσωτερική κατάσταση. Η πλαστογραφία με γνώση της εσωτερικής κατάστασης δεν είναι πρακτική.
- Ενιαίος domain separator. Ο domain separator εκτελείται για όλους τους γύρους και παράγει τιμή με κάθε νέα μετάβαση επειδή διαπιστώθηκε ότι οδηγεί σε μια πιο αποτελεσματική υλοποίηση σε ASIC. Ένας τέτοιος μηχανισμός έχει αποδειχθεί ότι είναι ασφαλής.

2.8.2 Μέγεθος κατάστασης αλγορίθμου Spix

Ο αλγόριθμος Spix πρέπει να λειτουργεί με κλειδιά μεγέθους 128-bit και να παρέχει ελάχιστη πολυπλοκότητα χρόνου επίθεσης $T = 2^{112}$ με πολυπλοκότητα δεδομένων $D = 2^{50} - 1$ bytes. Για κατάσταση b-bit με b = r + c, όπου r-bit rate bits και c-bit capacity bits. Για να ικανοποιηθούν οι απαιτήσεις ασφαλείας, τα capacity bits θα πρέπει να είναι τουλάχιστον $\log_2 (D^2 + DT)$ όπου $D = 2^{47}$ μπλοκ για r = 64 που σημαίνει $c \geq 160$. Λαμβάνοντας υπόψη τους περιορισμούς της περιοχής υλικού, βρέθηκε ότι το b = 256 και το r = 64 είναι μια αποτελεσματική επιλογή. Πιο συγκεκριμένα, υπήρχε ανάγκη η κατάσταση r και b να είναι πολλαπλάσιο του 64 για να υπάρξει μια αποτελεσματική υλοποίηση του λογισμικού και να ακολουθεί τις προδιαγραφές της δομή της μετάθεσης sLiSCP-light. Αντίστοιχα, οι πρακτικές επιλογές για μεγέθη κατάστασης θα ήταν b = 256 bits και 512 bits, επομένως επιλέχτηκε το μέγεθος b = 256 καθώς παρέχει την καλύτερη ισορροπία μεταξύ των απαιτήσεων υλικού και λογισμικού, ασφάλειας και αποδοτικότητας. Με αυτήν την επιλογή του μεγέθους, ο αλγόριθμος Spix υλοποιείται αποτελεσματικά σε μεγάλο εύρος πλατφορμών. Επιπλέον, με την επιλογή αυτή, παρέχετε ασφάλεια 128 bit με πολυπλοκότητα δεδομένων 260.

2.8.3 Επιλογή Simeck Sbox

Το Simeck sbox, είναι μια παραμετροποιημένη παραλλαγή της συνάρτησης Simon round function [15] η οποία δεν χρειάζεται χρήση κλειδιού. Επιπλέον, έχει σημειώσει νέο ρεκόρ όσον αφορά την αποδοτικότητα του υλικού και τις επιδόσεις σχεδόν σε όλες τις πλατφόρμες. Στη συνέχεια, δίνονται οι λόγοι που οδήγησαν στην υιοθέτηση των Simeck Box στη μετάθεση sLiSCP-light.

- Το Simeck sbox έχει μια φιλική προς το υλικό λειτουργία γύρου που αποτελείται από απλές λειτουργίες XOR, AND και κυκλικής μετατόπισης των bits. Επιπλέον, το αποτύπωμα που προκύπτει αυξάνεται γραμμικά με το μέγεθος εισόδου του.
- Είναι πρακτικό να αξιολογηθεί η μέγιστη (αναμενόμενη) διαφορική πιθανότητα του Sbox και η μέγιστη (αναμενόμενη) γραμμική τετραγωνική συσχέτιση που είναι $2^{-15.8}$ και $2^{-15.6}$, αντίστοιχα. Επιπλέον, παρέχονται αναμενόμενα όρια έναντι της διαφορικής και γραμμικής κρυπτανάλυσης.
- Κάθε sbox Simeck παραμετροποιείται ανεξάρτητα από ένα σύνολο σταθερών που υποδηλώνει ότι τα αναμενόμενα όρια έναντι της διαφορικής και γραμμικής κρυπτανάλυσης δεν είναι στενά, επομένως, αναμένεται καλύτερη ασφάλεια.

2.8.4 Σταθερές γύρου και βήματος

Στον αλγόριθμο Spix γίνεται χρήση σταθερών τιμών ώστε να γίνει μετριασμός της αυτοσυμμετρίας.

- Τρεις μοναδικές σταθερές βήματος των 8 bit (sci0, sci1, sci2). Η σταθερή τιμή των σταθερών βήματος είναι μοναδική σε όλα τα βήματα, επομένως καταστρέφει οποιαδήποτε συμμετρία μεταξύ των βημάτων της μετάθεσης. Επίσης είναι αναγκαίο για οποιοδήποτε βήμα, οι σταθερές τιμές μεταξύ τους πρέπει να διαφέρουν προκειμένου να καταστραφεί οποιαδήποτε συμμετρία μεταξύ της ανακατεύθυνσης λέξεων.

- Τρεις μοναδικές σταθερές γύτου των 8-bit (rci0, rci1, rci2). Ένα bit από κάθε σταθερά γύρου γίνεται XOR με την κατάσταση του Simeck sbox σε κάθε γύρο για να καταστρέψει τη διατήρηση τυχόν περιστροφικών ιδιοτήτων. Επιπλέον, γίνεται πρόσθεση του «1» bit σε κάθε ένα bit σταθερά τιμής που έχει ως αποτέλεσμα πολλές αντιστροφές, και κατά συνέπεια διακόπτει τη διάδοση της περιστροφικής ιδιότητας σε ένα βήμα.

Η επιλογή να χρησιμοποιηθεί ένα πολυώνυμο LFSR διασφαλίζει ότι κάθε πλειάδα τέτοιων σταθερών δεν επαναλαμβάνεται λόγω της περιοδικότητας της ακολουθίας των 8 πλειάδων που κατασκευάζεται από την αποδεκατισμένη m-ακολουθία της περιόδου 127.

2.8.5 Επιλογή αριθμού βημάτων και γύρων

Η λογική στην επιλογή του αριθμού των γύρων u και του αριθμού των βημάτων s του αλγορίθμου Spix βασίζεται στην επίτευξη της καλύτερης ισοροπίας μεταξύ ασφάλειας και αποδοτικότητας. Και στις δύο παραλλαγές της μετάθεσης sLiSCP-light, πρέπει να ελαχιστοποιηθεί η τιμή u×s ενώ πληρούνται οι ακόλουθες δύο προϋποθέσεις:

- Για την παραλλαγή της μετάθεσης sLiSCP-light που χρησιμοποιείται στην αρχικοποίηση και στην οριστικοποίηση, δεν θα πρέπει να διακρίνεται από μια τυχαία μετάθεση.

- Για την παραλλαγή της μετάθεσης sLiSCP-light που χρησιμοποιείται στην επεξεργασία σχετικών δεδομένων, στη λειτουργία κρυπτογράφησης και αποκρυπτογράφησης, απαιτείτε η αναμενόμενη μέγιστη διαφορική πιθανότητα για τη μετάθεση να είναι μικρότερη από 2^−128 και να μην υπάρχει αρχή από τους μεσαίους διαχωριστές για αυτήν.

2.8.5.1. Μετάθεση sLiSCP-light δεκαοχτώ γύρων

Για την μετάθεση sLiSCP-light δεκαοχτώ γύρων, το s πρέπει να είναι τουλάχιστον 3 × 4 όπου απαιτούνται τέσσερα βήματα για να επιτευχθεί πλήρης διάχυση bit στην κατάσταση. Αυτή η επιλογή προσθέτει περιθώριο ασφαλείας 33% έναντι των χαρακτηριστικών meet/miss-in-the-middle, καθώς σε οστώ βήματα, επιτυγχάνεται πλήρης διάχυση bit τόσο προς τις εμπρός κατευθύνσεις όσο και προς τις πίσω κατευθύνσεις.

2.8.5.2. Μετάθεση sLiSCP-light εννέα γύρων

Ακολουθώντας την ορολογία της επίθεσης ανάκαμψης, υπάρχουν διαχωριστές που κατασκευάζονται από είσοδο (αντίστοιχη έξοδος) σε έξοδο (αντίστοιχη είσοδο) από εισερχόμενα διαχωριστικά. Στην κρυπτανάλυση του sLiSCP-light βρέθηκε ότι, ο καλύτερος διαχωριστής που δεν απαιτεί εκκίνηση από τη μεσαία προσέγγιση μπορεί να καλύπτει έως και οχτώ βήματα. Κατά συνέπεια, η επιλογή s > 8 ήταν μια ασφαλής επιλογή για το sLiSCP-light κατά την επεξεργασία σχετικών δεδομένων, λειτουργία κρυπτογράφησης και αποκρυπτογράφησης.

2.8.6. Επιλογή κατάστασης rate

Η απορρόφηση των μπλοκ μηνυμάτων γίνεται στα υπομπλοκ X1 και X3. Τέτοιες θέσεις για την κατάσταση rate επιτρέπει την επεξεργασία των bit εισόδου από τα Simeck sbox το συντομότερο δυνατό, ώστε να επιτυγχάνουμε ταχύτερη διάχυση. Επίσης, η επιλογή αυτή αναγκάζει οποιεσδήποτε εγχυόμενες διαφορές να ενεργοποιήσουν τα Simeck sbox στο πρώτο βήμα, γεγονός που ενισχύει επίσης την αντίσταση του Spix στη διαφορική και γραμμική κρυπτανάλυση.

Κεφάλαιο 3

Αλγόριθμος GIFT-COFB

3.1. Γενική Περιγραφή

Σε αυτό το κεφάλαιο θα γίνει η γενική περιγραφή της λειτουργίας του αλγορίθμου GIFT-COFB. Το όνομα GIFT-COFB υποδηλώνει τη λειτουργία κρυπτογράφησης με βάση COFB (Combined FeedBack) και με μπλοκ κρυπτογράφησης GIFT. Το μπλοκ κρυπτογράφησης COFB εστιάζει κυρίως στο μέγεθος υλοποίησης του υλικού. Σε αυτό τον αλγόριθμο γίνεται χρήση ενός ελαφριού, καλά αναλυμένου, κρυπτογραφημένου μπλοκ. Επιπλέον, γίνεται χρήση ελαχιστοποιημένου μεγέθους κατάστασης κρυπτογράφησης και αποκρυπτογράφησης. Επιπρόσθετα, γίνεται χρήση ανατροφοδότησης με τα δεδομένα της εξόδου του μπλοκ GIFT. Αυτός ο συνδυασμός επιτρέπει την ελαχιστοποίηση του μεγέθους του αλγορίθμου. Τέλος, το μπλοκ κρυπτογράφησης COFB επιτρέπει την κρυπτογράφηση και την αποκρυπτογράφηση του κειμένου χωρίς να χρειάζεται να υλοποιηθεί ο αλγόριθμος αντίστροφα.

3.2. Επισκόπηση GIFT-COFB

Ο αλγόριθμος αυτός για να εκτελέσει κρυπτογράφηση, ο χρήστης θα πρέπει να του φορτώσει ένα κείμενο (M) με σχετικά δεδομένα (A) μία nonce και ένα κλειδί (K). Ο αλγόριθμος δέχεται ένα κλειδί (K) μεγέθους 128 bits, μία nonce μεγέθους 128 bits, σχετικά δεδομένα μεταβαλλόμενου μεγέθους και το κείμενο επίσης μεταβαλλόμενου μεγέθους. Ως έξοδο παράγει ένα κρυπτογραφημένο κείμενο (C) ίσου μεγέθους με το κείμενο και μία ετικέτα για λόγους επαλήθευσης. Αντίστοιχα για την αποκρυπτογράφηση, ο αλγόριθμος δέχεται το κρυπτογραφημένο κείμενο, την nonce, σχετικά δεδομένα, και την ετικέτα που δημιουργήθηκε από την διαδικασία κρυπτογράφησης. Ως έξοδο έχει το αρχικό κείμενο η ένα μήνυμα λάθους στην περίπτωση που η επαλήθευση με την ετικέτα απέτυχε.

3.3. Τρόπος λειτουργίας

Ο αλγόριθμος GIFT-COFB μπορεί να επεξεργάζεται 128 bit δεδομένα προς κρυπτογράφηση κάθε φορά που εκτελείτε. Αν τα σχετικά δεδομένα και το κείμενο είναι μεγαλύτερα από 128 bits σε μέγεθος, τότε θα χωριστούν σε κομμάτια των 128 bits και θα φορτωθούν ένα τη φορά στον αλγόριθμο. Οι λειτουργίες των δύο μπλοκ κρυπτογράφησης τα οποία αποτελείται ο αλγόριθμος, θα αναλυθούν στα επόμενα κεφάλαια.

3.3.1 Μπλοκ κρυπτογράφησης GIFT

Το μπλοκ κρυπτογράφησης GIFT ονομαστικά είναι συντομία του GIFT-128 επειδή δέχεται είσοδο 128 bit δεδομένα με 128 bit κλειδί. Το μπλοκ αυτό περιέχει 40 γύρους λειτουργίας, ο οποίος κάθε γύρος εκτελεί την ίδια λειτουργία. Τα βήματα που αποτελείτε ένας γύρος θα αναλυθούν στη συνέχεια.

3.3.1.1 Αρχικοποίηση εισόδου

Το 128 bit κείμενο προς κρυπτογράφηση φορτώνεται στην εσωτερική κατάσταση S η οποία χωρίζεται σε τέσσερα τμήματα (S0, S1, S2, S3) των 32 bit. Στην προοπτική μιας δισδιάστατης συστοιχίας, η ταξινόμηση των bit γίνεται από πάνω προς τα κάτω και μετά δεξιά προς τα αριστερά. Στο σχήμα που ακολουθεί απεικονίζεται ο τρόπος που φορτώνεται η είσοδος b στα τμήματα.

S=	S_0	←	$b_{124} \dots b_8\ b_4\ b_0$
	S_1		$b_{125} \dots b_9\ b_5\ b_1$
	S_2		$b_{126} \dots b_{10}\ b_6\ b_2$
	S_3		$b_{127} \dots b_{11}\ b_7\ b_4$

Σχήμα 11 – Αρχικοποίηση εισόδου

Αντίστοιχα το 128 bit κρυφό κλειδί φορτώνεται στην εσωτερική κατάσταση κλειδιού KS και χωρίζεται σε οχτώ τμήματα των 16 bit. Στην προοπτική ενός δισδιάστατου πίνακα, η ταξινόμηση των bit γίνεται από δεξιά προς τα αριστερά και στη συνέχεια, από κάτω προς τα πάνω. Στο σχήμα που ακολουθεί απεικονίζεται ο τρόπος που φορτώνεται το κρυφό κλειδί στα τμήματα.

KS =	$W_0 \| W_1$	←	$b_{127} \dots b_{112} \|\| b_{111} \dots b_{98}\ b_{97}\ b_{96}$
	$W_2 \| W_4$		$b_{95} \dots b_{80} \| b_{79} \dots b_{66}\ b_{65}\ b_{64}$
	$W_4 \| W_5$		$b_{63} \dots b_{48} \| b_{47} \dots b_{34}\ b_{33}\ b_{32}$
	$W_6 \| W_7$		$b_{31} \dots b_{16} \|\| b_{15} \dots b_2\ b_1\ b_0$

Σχήμα 12 – Αρχικοποίηση κλειδιού

Η φάση οριστικοποίησης δεν θα αναλυθεί γιατί απλά εκτελεί τις αλλαγές της αρχικοποίησης αντίστροφα ώστε να επαναφέρει την αρχική μορφή της κατάστασης όταν τελειώσει η λειτουργία κρυπτογράφησης. Οι φάσεις αρχικοποίησης και οριστικοποίησης απαιτούν επιπλέον κύκλους ρολογιού. Η ασφάλεια του αλγορίθμου δεν επηρεάζεται από τη χρήση αρχικοποίησης και οριστικοποίησης, για αυτό τον λόγο είναι προαιρετική και στις προσομοιώσεις που έγιναν δεν χρησιμοποιήθηκαν και οι είσοδοι χωρίστηκαν σε τμήματα με τον πιο φυσικό τρόπο.

3.3.1.2 Γύρος GIFT

Κάθε γύρος αποτελείτε από τρία βήματα: SubCells, PermBits και AddRoundKey.

Βήμα SubCells: Η εσωτερική κατάσταση S για αρχή έχει χωριστεί σε τέσσερα τμήματα (S0, S1, S2, S3). Η κατάσταση κατάσταση κάθε τμήματος ενημερώνεται με τη χρήση λογικών πράξεων. Για αρχή η κατάσταση του τμήματος S1 θα ενημερωθεί με την πράξη XOR μεταξύ της κατάστασης S1 και το αποτέλεσμα της πράξης AND μεταξύ των καταστάσεων των τμημάτων S0 και S2. Αφού έχει ενημερωθεί η κατάσταση του τμήματος S1, τότε η κατάσταση του τμήματος S0 θα ενημερωθεί με την πράξη XOR μεταξύ της κατάστασης του τμήματος S0 και το αποτέλεσμα της πράξης AND μεταξύ των καταστάσεων των τμημάτων S1 και S3. Τα τμήματα της εσωτερική κατάσταση S αλλάζει με τις εντολές που απεικονίζονται στο σχήμα 15. Η κατάσταση του τμήματος S2 θα ενημερωθεί με την πράξη XOR μεταξύ της κατάστασης του τμήματος S2 και το αποτέλεσμα της πράξης OR μεταξύ των καταστάσεων των τμημάτων S0 και S1. Η κατάσταση του τμήματος S3 θα ενημερωθεί με την πράξη XOR μεταξύ της κατάστασης του τμήματος S3 και της κατάστασης του τμήματος S2. Ακολουθώντας τις προηγούμενες ενημερώσεις καταστάσεων, η κατάσταση του τμήματος S1 θα ξανά ενημερωθεί με την πράξη XOR μεταξύ της κατάστασης του τμήματος S1 και της κατάστασης του τμήματος S3. Η κατάσταση του τμήματος S3 θα ξανά ενημερωθεί με την πράξη NOT της κατάστασης του τμήματος S3 και η κατάσταση του τμήματος S2 θα ξανά ενημερωθεί με την πράξη XOR μεταξύ της κατάστασης του τμήματος S2 και το αποτέλεσμα της πράξης AND μεταξύ των καταστάσεων των τμημάτων S0 και S1. Τέλος, στη θέση του τμήματος S0 θα αποθηκευτεί η κατάσταση του τμήματος S3, στη θέση του τμήματος S3 θα αποθηκευτεί η κατάσταση του τμήματος S0 και οι άλλες δυο καταστάσεις δεν θα αλλάξουν θέση. Η γραφική αναπαράσταση των πράξεων φαίνονται στο σχήμα που ακολουθεί.

$$S1 \leftarrow S1 \oplus (S0 \,\&\, S2)$$
$$S0 \leftarrow S0 \oplus (S1 \,\&\, S3)$$
$$S2 \leftarrow S2 \oplus (S0 \mid S1)$$
$$S3 \leftarrow S3 \oplus S2$$
$$S1 \leftarrow S1 \oplus S3$$
$$S3 \leftarrow \sim S3$$
$$S2 \leftarrow S2 \oplus (S0 \,\&\, S1)$$
$$\{S0, S1, S2, S3\} \leftarrow \{S3, S1, S2, S0\},$$

Σχήμα 13 – Βήμα SubCells

όπου &, | και ~ είναι λειτουργία AND, OR και NOT αντίστοιχα.

Βήμα PermBits: Διαφορετικές μεταθέσεις των bit εφαρμόζονται σε κάθε τμήμα κατάστασης Si. Ο τρόπος που αλλάζουν θέσεις τα bit δίνεται στο σχήμα 16 που ακολουθεί.

Index	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
S_0	29	25	21	17	13	9	5	1	30	26	22	18	14	10	6	2
S_1	30	26	22	18	14	10	6	2	31	27	23	19	15	11	7	3
S_2	31	27	23	19	15	11	7	3	28	24	20	16	12	8	4	0
S_3	28	24	20	16	12	8	4	0	29	25	21	17	13	9	5	1

Index	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S_0	31	27	23	19	15	11	7	3	28	24	20	16	12	8	4	0
S_1	28	24	20	16	12	8	4	0	29	25	21	17	13	9	5	1
S_2	29	25	21	17	13	9	5	1	30	26	22	18	14	10	6	2
S_3	30	26	22	18	14	10	6	2	31	27	23	19	15	11	7	3

Σχήμα 14 – Βήμα PermBits

Στο σχήμα 17, η σειρά «index» δείχνει την αρχική θέση των 32 bit σε όλα τα Si και η σειρά Si (S0, S1, S2, S3) δείχνει την τελική θέση των bit. Για παράδειγμα, το bit 1 (το 2^o δεξί bit) του S1 μετατοπίζεται 1 θέση της τα δεξιά, στην αρχική θέση του bit 0, ενώ το bit 0 μετατοπίζεται 8 θέσεις της τα αριστερά.

Βήμα AddRoundKey: Αυτό το βήμα αποτελείτε από την προσθήκη του κλειδιού γύρου και της σταθερά τιμής του γύρου. Τα τμήματα U και V έχουν μέγεθος 32 bits και εξάγονται από την εσωτερική κατάσταση κλειδιού που έχει στο κάθε γύρο και ονομάζεται κλειδί γύρου.

$$RK = U||V$$

Η χρήση του κλειδιού γύρου, γίνεται με απορρόφηση των τμημάτων U και V στα τμήματα S2 και S1 με τη χρήση της εντολής XOR.

$$S_2 \leftarrow S_2 \oplus U,$$

$$S_1 \leftarrow S_1 \oplus V.$$

Η προσθήκη σταθεράς γίνεται με την ενημέρωση της κατάστασης του τμήματος S3. Η κατάσταση του τμήματος ενημερώνεται με την εντολή XOR μεταξύ της κατάστασης του τμήματος S3 με τη σταθερά τιμή που έχει δημιουργηθεί από τον LFSR σε αυτό το γύρο.

$$S_3 \leftarrow S_3 \oplus 0x800000XY$$

Όπου XY η σταθερά που έχει δημιουργηθεί στο γύρο που βρίσκεται το μπλοκ GIFT.

3.3.1.3 Key schedule και σταθερά γύρου.

Στο βήμα AddRoundKey αναφέρθηκε ότι τα 32 bit τμήματα U και V εξάγονται από την εσωτερική κατάσταση του κλειδιού. Το τμήμα U εξάγεται από το τρίτο και τέταρτο τμήμα της

κατάστασης κλειδιού (W2 και W3) ενώ το τμήμα V εξάγεται από το έβδομο και όγδοο τμήμα της κατάστασης κλειδιού (W7 και W8). Τα τμήματα U και V την εσωτερική κατάσταση του κλειδιού όπως φαίνεται στη παρακάτω εικόνα.

$$U \leftarrow W_2 \,||\, W_3, \;\; V \leftarrow W_6 \,||\, W_7.$$

Στη συνέχεια η κατάσταση κλειδιού ενημερώνεται με τον τρόπο που δείχνεται στο σχήμα 18, όπου ≫ i είναι μια δεξιόστροφη περιστροφή bit σε ένα τμήμα κατάστασης κλειδιού. Για αρχή στη θέση του τμήματος W0 αποθηκεύεται η κατάσταση του τμήματος W6 αφού έχει γίνει δεξιόστροφη περιστροφή των bits κατά δύο θέσεις. Στη θέση του τμήματος W1 αποθηκεύεται η κατάσταση του τμήματος W7 αφού έχει γίνει δεξιόστροφη περιστροφή των bits κατά δώδεκα θέσεις. Στη συνέχεια, για τις θέσεις W2, W3, W4, W5, W6 και W7 θα αποθηκευτούν οι καταστάσεις των τμημάτων W0, W1, W2, W3, W4 και W5 αντίστοιχα.

$W_0 \,\|\| W_1$	←	$W_6 >>> 2 \,\|\| W_7 >>> 12$
$W_2 \,\|\| W_3$		$W_0 \,\|\| W_1$
$W_4 \,\|\| W_5$		$W_2 \,\|\| W_3$
$W_6 \,\|\| W_7$		$W_4 \,\|\| W_5$

Σχήμα 15 – Ενημέρωση κατάστασης κλειδιού

Οι σταθερές γύρου παράγονται από έναν 6-bit LFSR, όπου η κατάσταση του χαρακτηρίζεται ως c5|c4|c3|c2|c1|c0 και κάθε c χαρακτηρίζει ένα bit. Σε κάθε γύρο GIFT η κατάσταση ενημερώνεται μετακινώντας όλα τα bit αριστερόστροφα και στο τελευταίο τώρα bit εφαρμόζεται η πράξη XOR με το bit που χαρακτηρίζει το c4 και 1. Δηλαδή ενημερώνεται με αυτόν τον τρόπο: c5|c4|c3|c2|c1|c0 ← c4|c3|c2|c1|c0|c5 XOR c4 XOR 1. Τα 6 αυτά bit αρχικοποιούνται με την τιμή 0 πριν την εκκίνηση των γύρων. Στο σχήμα που ακολουθεί δείχνονται οι σταθερές τιμές που παράγονται ανά γύρο. Η σταθερά τιμή 1A αντιπροσωπεύει και τον τελευταίο γύρο του μπλοκ GIFT

Rounds	Constants
1 - 16	01,03,07,0F,1F,3E,3D,3B,37,2F,1E,3C,39,33,27,0E
17 -32	1D,3A,35,2B,16,2C,18,30,21,02,05,0B,17,2E,1C,38
33 - 48	31,23,06,0D,1B,36,2D,1A,34,29,12,24,08,11,22,04

Σχήμα 16 – Σταθερές τιμές ανά γύρο GIFT

3.3.2 Μπλοκ κρυπτογράφησης COFB

Σε αυτό το κεφάλαιο θα αναλυθούνε οι λειτουργίες που περιέχει το μπλοκ COFB οι οποίες εκτελούνται πριν και μετά τις λειτουργίες που περιέχει το μπλοκ GIFT. Το μπλοκ COFB είναι υπεύθυνο να δέχεται τα δεδομένα από το χρήστη και να τα σπάει σε κομμάτια μεγέθους που μπορεί να δεχτεί το μπλοκ GIFT και στη συνέχεια να παίρνει την έξοδο του και μετά από συγκεκριμένες λειτουργίες να δώσει έξοδο πίσω στο χρήστη. Οι λειτουργίες του μπλοκ COFB θα αναλυθούν στα επόμενα κεφάλαια.

3.3.2.1 Λειτουργία Padding

Το μπλοκ GIFT είναι φτιαγμένο έτσι ώστε να δέχεται είσοδο προς επεξεργασία μεγέθους 128 bit. Για αυτό το λόγο το μπλοκ COFB περιέχει μία λειτουργία Padding η οποία είναι υπεύθυνη να μετατρέπει τα δεδομένα προς επεξεργασία σε μέγεθος που δέχεται το μπλοκ GIFT. Σε περίπτωση που το κομμάτι προς επεξεργασία είναι μικρότερο των 128bits τότε προσθέτει στο τέλος μηδενικά ώστε να μετατραπεί στο επιθυμητό μέγεθος.

3.3.2.2 Λειτουργία ανατροφοδότησης

Η λειτουργία ανατροφοδότησης δέχεται ως είσοδο την έξοδο από το μπλοκ GIFT. Την 128 bit είσοδο που δέχεται τη χωρίζει σε δυο κομμάτια των 64 bit. Αν Y η έξοδος του γύρου GIFT τότε η λειτουργία ανατροφοδότησης ορίζεται ως: $G(Y) = (Y[2], Y[1] \ll 1)$. Όπου το X, $X \ll r$ είναι η αριστερόστροφη περιστροφή των κατά r θέσης. Δηλαδή, σε αυτή την περίπτωση αφού χωρίσει η είσοδος Y σε δύο 64 bit κομμάτια και αλλάξουν θέση μεταξύ τους, τα bit του κομματιού Y1 θα περιστραφούν αριστερόστροφα κατά μία θέση.

3.3.2.3 Λειτουργία Delta

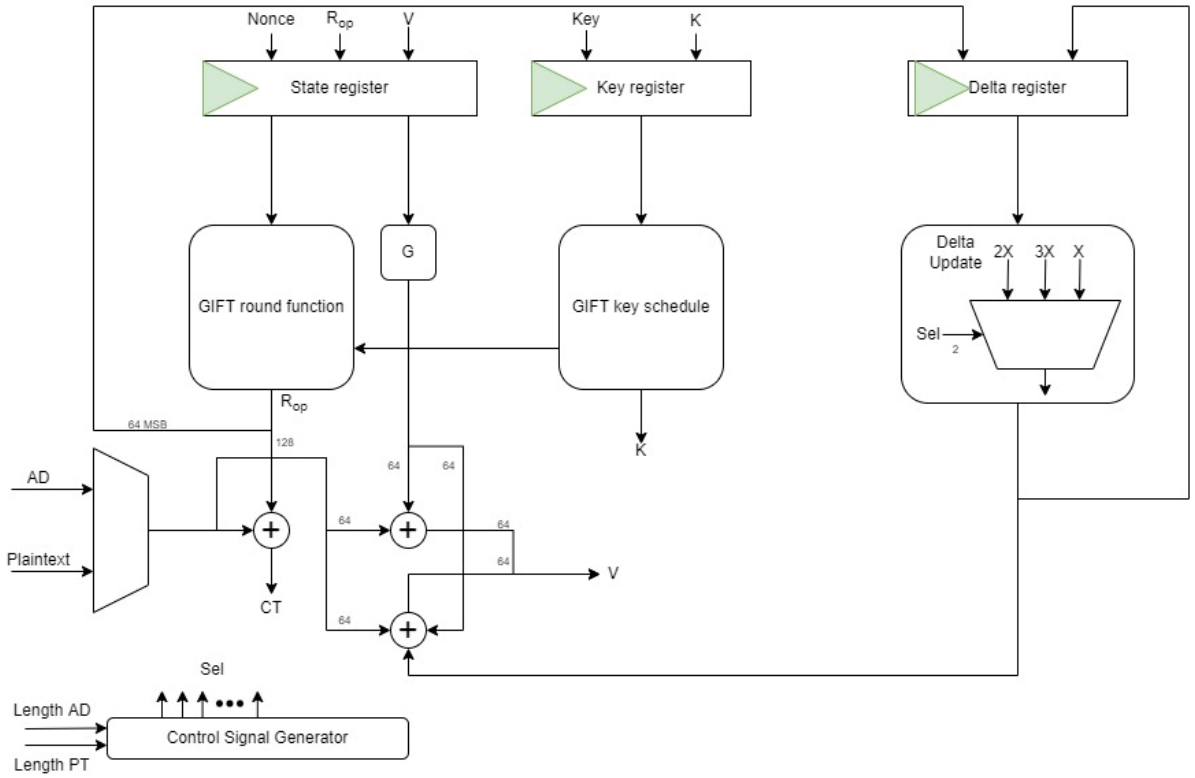
Η λειτουργία delta εκτελεί διαφορετικές λειτουργίες αναλόγως τι είδος μηνύματος επεξεργάζεται το μπλοκ GIFT. Δηλαδή, αν το μπλοκ GIFT επεξεργάζεται μήνυμα nonce τότε παίρνει τα πρώτα 64 bit της εξόδου GIFT και δημιουργεί την κατάσταση delta που θα την επεξεργάζεται αναλόγως τι είναι τα υπόλοιπα μηνύματα. Αν το μήνυμα που επεξεργάζεται το μπλοκ GIFT είναι σχετικά δεδομένα η το κείμενο προς κρυπτογράφηση τότε ο delta διπλασιάζει την κατάσταση delta που έχει δημιουργήσει. Στην περίπτωση που είναι το τελευταίο μήνυμα από σχετικά δεδομένα η κείμενο προς κρυπτογράφηση τότε τριπλασιάζει την κατάσταση delta.

3.4. Χρονισμός αλγορίθμου Spix

Το μπλοκ κρυπτογράφησης GIFT χρειάζεται E = 40 κύκλους ρολογιού για να ολοκληρώσει μία λειτουργία κρυπτογράφησης. Τόσοι κύκλοι ρολογιού είναι απαραίτητοι στην κρυπτογράφηση της nonce. Κάθε κομμάτι σχετικών δεδομένων και κειμένου προς κρυπτογράφηση χρειάζεται E κύκλους ρολογιού να επεξεργαστούν. Πριν από κάθε κομμάτι σχετικών δεδομένων και κειμένου προς κρυπτογράφηση, ο αλγόριθμος ξοδεύει Du = 4 κύκλους ρολογιού για να ενημερώσει την κατάσταση delta. Έτσι, αν Na και Nm ο συνολικός αριθμός κομματιών σχετικών δεδομένων και κειμένου προς κρυπτογράφηση, τότε μια διαδικασία κρυπτογράφησης του αλγορίθμου GIFT-COFB θα χρειαστεί T = E + (Na + Nm)(E + Du) κύκλους ρολογιού να ολοκληρωθεί.

3.5. Υλοποίηση κυκλώματος αλγορίθμου GIFT-COFB

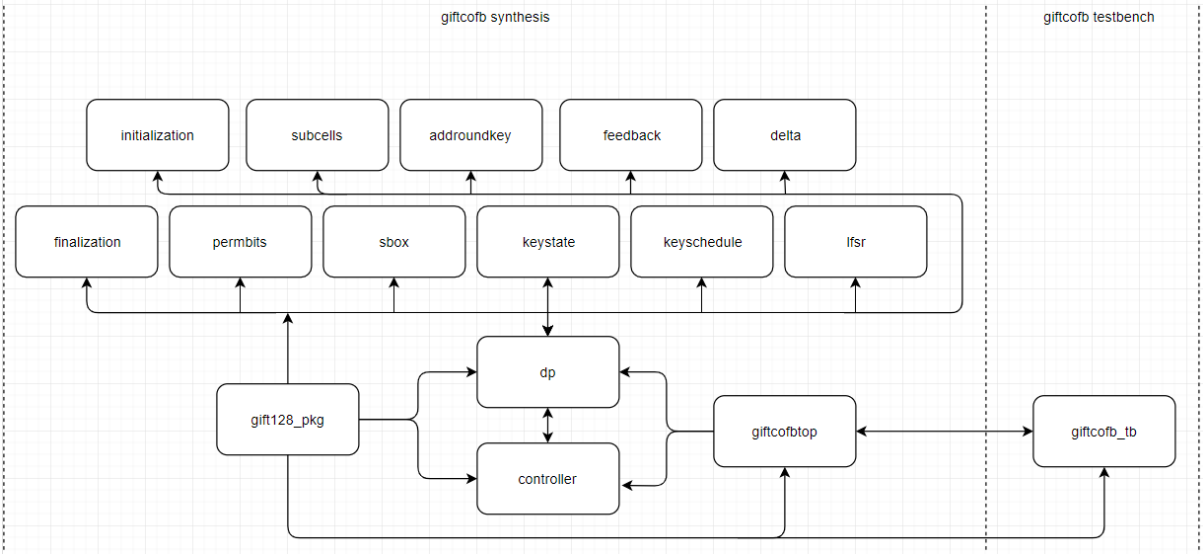
Σε αυτό το κεφάλαιο θα δοθεί η υλοποίηση του υλικού για τον αλγόριθμο GIFT-COFB. Η λειτουργία έχει σχεδιαστεί για να απαιτεί μια επιπλέον κατάσταση 64-bit ξεχωριστή από αυτές που χρησιμοποιούνται στο κύκλωμα κρυπτογράφησης μπλοκ. Έτσι, ο σχεδιασμός απαιτεί ένα πρόσθετο καταχωρητή των 64-bit. Η αρχική nonce (που υποδηλώνεται από το Nonce στο παραπάνω σχήμα) φορτώνεται στη ρουτίνα κρυπτογράφησης και τα σήματα ελέγχου δημιουργούνται εσωτερικά ανάλογα με το μέγεθος του απλού κειμένου και των σχετικών δεδομένων. Ανάλογα με τη φάση λειτουργίας, ο καταχωρητής κατάστασης μπορεί να χρειαστεί να τροφοδοτήσει το μπλοκ GIFT με τη nonce, την έξοδο του προηγούμενο γύρου GIFT (Rop) η το αποτέλεσμα της διαδικασία κρυπτογράφησης από προηγούμενο μήνυμα (V) το οποίο είναι το άθροισμα της λειτουργίας ανατροφοδότησης που έχει γίνει XOR με την κατάσταση delta και το νέο μήνυμα που εισέρχεται στο κύκλωμα. Η έξοδος του γύρου GIFT (Rop) γίνεται XOR με την είσοδο του κυκλώματος (AD / Plaintext) και παράγεται το κρυπτογραφημένο κείμενο. Μέσω του state register η έξοδος του μπλοκ GIFT φορτώνεται στην είσοδο της λειτουργία ανατροφοδότησης (G). Η έξοδος της λειτουργία ανατροφοδότησης στη συνέχεια γίνεται XOR με την έξοδο που έχει η λειτουργία Delta και παράγεται η έξοδος του γύρου GIFT-COFB (V) για χρήση ως είσοδο σε επόμενο γύρο. Στο σχήμα 17 που ακολουθεί περιγράφεται λεπτομερώς το κύκλωμα του υλικού για τον αλγόριθμο GIFT-COFB.



Σχήμα 17 – Κύκλωμα αλγορίθμου GIFT-COFB

3.6. Υλοποίηση αλγορίθμου GIFT-COFB σε κώδικα VHDL

Η υλοποίηση του αλγορίθμου σε κώδικα vhdl έγινε δημιουργώντας δεκαέξι αρχεία κώδικα τα οποία τα δεκαπέντε είναι υπεύθυνα για τη σύνθεση του αλγορίθμου και το ένα είναι υπεύθυνο για την προσομοίωση του. Τα αρχεία initialization και finalization περιέχουν την λειτουργία που αναφέρθηκε στο κεφάλαιο 2.3.1.1. Τα τρία βήματα του γύρου GIFT περιγράφονται στα αρχεία subcells , permbits και addroundkey. Η λειτουργία keyschedule και οι σταθερές που παράγονται, περιγράφονται στα αρχεία keyschedule και lfsr αντίστοιχα. Η λειτουργία ανατροφοδότησης και Delta περιγράφονται στα αρχεία feedback και delta αντίστοιχα. Τα αρχεία sbox και keystate χρησιμοποιούνται για να ενώσουν τις εξόδους της αρχικοποίησης με την είσοδο του κυκλώματος GIFT-COFB. Όλα τα προηγούμενα αρχεία που αναφέρθηκαν χρησιμοποιούνται ως οντότητες στο αρχείο dp, το οποίο περιέχει όλο το κύκλωμα που φαίνεται στο σχήμα 19 και τις βασικές λειτουργίες του σε μία διεργασία. Το αρχείο ctl περιέχει την μονάδα ελέγχου το οποίο ελέγχει το αρχείο dp. Το αρχείο giftcofbtop περιέχει τις εισόδους, εξόδους και αρχιτεκτονική του αλγορίθμου στο υψηλότερο ιεραρχικά επίπεδο. Το αρχείο giftcofb_tb είναι το testbench το οποίο περιέχει το αρχείο giftcofbtop ως οντότητα και το προσομοιώνει αναλόγως τις δοκιμαστικές τιμές που του έχει ορίσει ο χρήστης. Το αρχείο gift128_pkg περιέχει στοιχεία, σταθερές, τύπους, υποτύπους και λειτουργίες που χρησιμοποιούνται επανειλημμένα στους κώδικες που υλοποιήθηκαν για τον αλγόριθμο GIFT-COFB. Επιπλέον, το αρχείο αυτό περιέχει και τη λειτουργία padding η οποία καλείτε από το κύκλωμα όταν είναι αναγκαίο. Στο σχήμα 20 απεικονίζεται ο τρόπος που συνδέονται τα αρχεία VHDL μεταξύ τους.



Σχήμα 18 – Αρχεία VHDL για αλγόριθμο GIFT-COFB

3.7. Σχεδιαστική λογική αλγορίθμου GIFT-COFB

Το μπλοκ COFB περιέχει μια λειτουργία κρυπτογράφησης που χρησιμοποιεί το GIFT 128 ως υποκείμενο μπλοκ κρυπτογράφησης και ο αλγόριθμος GIFT-COFB μπορεί να θεωρηθεί ως μια αποτελεσματική ενοποίηση των COFB και GIFT-128. Το GIFT-128 χρησιμοποιεί μια

κατάσταση μεγέθους 128 bit και ένα κλειδί μεγέθους 128 bit. Για την ακρίβεια, το μπλοκ GIFT ανήκει σε μια οικογένεια μπλοκ κρυπτογράφησης που παραμετροποιούνται από το μέγεθος κατάστασης και το μέγεθος κλειδιού και όλα τα μέλη αυτής της οικογένειας είναι ελαφριά για το υλικό και μπορούν να αναπτυχθούν αποτελεσματικά σε ελαφριές εφαρμογές. Από την άλλη πλευρά, η λειτουργία COFB υπολογίζει τη συνδυασμένη ανάδραση (Combined FeedBack) της εξόδου κρυπτογράφησης μπλοκ και των μπλοκ δεδομένων για να ανεβάσει το επίπεδο ασφάλειας. Αυτό στην πραγματικότητα βοηθά στη σχεδίαση μιας λειτουργίας με χαμηλό μέγεθος κατάστασης και τελικά οδηγεί στην υλοποίηση του με χαμηλό μέγεθος κατάστασης. Επιπλέον, αυτή η τεχνική αντιστέκεται στον εισβολέα για να ελέγξει το μπλοκ εισόδου και την είσοδο κρυπτογράφησης του επόμενου μπλοκ ταυτόχρονα. Συνολικά, ένας συνδυασμός GIFT και COFB θεωρείται να είναι ως μία από τις πιο αποτελεσματικές, ελαφριές, χαμηλής κατάστασης μπλοκ, που η κατασκευή που βασίζεται σε AEAD.

3.7.1 Μπλοκ κρυπτογράφησης GIFT-128

Το GIFT θεωρείται ένα από τα ελαφρύτερα σχέδια που υπάρχουν. Συμβολίζεται ως «Small PRESENT» καθώς η λογική σχεδιασμού που ακολουθεί ο GIFT είναι αυτή του PRESENT [10]. Ωστόσο, το μπλοκ GIFT έχει απαλλαγεί από πολλές γνωστές αδυναμίες που υπάρχουν στο PRESENT όσο αφορά τη γραμμική κρυπτανάλυση. Στο σύνολο, το μπλοκ GIFT υπόσχεται πολύ αυξημένη απόδοση (τόσο ελαφρύτερη όσο και πιο γρήγορη) σε σχέση με το PRESENT. Το μπλοκ GIFT είναι ένας πολύ απλός σχεδιασμός που ξεπερνά ακόμη και τους σχεδιασμούς SIMON και SKINNY σε υλοποιήσεις που χρησιμοποιούν γύρους. Η σχεδίαση είναι κάπως «βέλτιστη», καθώς ένα πιο αδύναμο S-box (από το S-box του GIFT) θα οδηγούσε σε μια πιο αδύναμη σχεδίαση. Το γραμμικό επίπεδο του μπλοκ GIFT, καταναλώνει μικρό χώρο σε υλικό γιατί οι λειτουργίες γύρου γίνονται με μια απλή καλωδίωση των bit, ενώ οι σταθερές δημιουργούνται χάρη σε ένα ελαφρύ LFSR. Η λειτουργία Key schedyle είναι επίσης πολύ ελαφριά για το υλικό, γιατί αποτελείτε από μεταβάσεις.

Παρόλο που δεν υπάρχει σχεδόν κανένα αντίκτυπο στην υλοποίηση υλικού, υπάρχουν πολλά κίνητρα για τη χρήση της υλοποίησης bitslice (μη βασισμένη σε LUT) αντί για την υλοποίηση του GIFT-128 βάσει LUT (Look Up Table), όταν εξετάζουμε την εφαρμογή σε λογισμικό. Εδώ, θα αναφερθούν τα τρία πιο προφανή οφέλη που σχετίζονται με τα τρία βήματα του στην εκτέλεση ενός γύρου.

Πρώτον για το μη γραμμικό επίπεδο, για υλοποίηση που βασίζεται σε LUT, μπορεί να γίνει εξέταση του ενδεχόμενου ενημέρωσης δύο Sbox του GIFT (1 byte) σε μία κλήση μνήμης με 256 καταχωρήσεις LUT. Αυτό θα απαιτούσε 16 αναζητήσεις και χρειάζονται περίπου 16 έως 64 κύκλοι για να εκτελεστούν όλα τα S-box σε έναν γύρο, υποθέτοντας μερικούς επιπλέον κύκλους για πρόσβαση στη μνήμη RAM. Χρησιμοποιώντας την υλοποίηση bitslice, απαιτούνται μόνο 11 βασικές λειτουργίες (ή 10 με λειτουργία XNOR) για τον παράλληλο υπολογισμό όλων των S-box. Και το πιο σημαντικό, η χρήση της εφαρμογής bitslice έχει το καλό χαρακτηριστικό ότι δεν χρειάζεται η χρηση της μνήμης RAM και χρειάζεται σταθερό χρόνο, μετριάζοντας πιθανές επιθέσεις χρονισμού.

Δεύτερον, για το γραμμικό επίπεδο, ενώ είναι βασικά ελαφρύ και γρήγορο σε υλικό, για την εφαρμογή σε λογισμικό είναι εξαιρετικά αργό και πολύπλοκο. Αυτό το φαινόμενο μπορεί να μειωθεί, εκτελώντας πολλά μπλοκ παράλληλα χρησιμοποιώντας τίποτα άλλο παρά μόνο υλοποίηση bitslice. Ακόμη και για ένα μεμονωμένο μπλοκ κρυπτογράφησης, η υλοποίηση bitslice εξακολουθεί να είναι πιο αποτελεσματική από την υλοποίηση που βασίζεται σε LUT λόγω του τρόπου με τον οποίο συσκευάζονται τα bit.

Τρίτο και τελευταίο, η προσθήκη κλειδιού, για υλοποίηση που βασίζεται σε LUT, τα δευτερεύοντα κλειδιά πρέπει γίνουν XOR σε θέσεις bit που απέχουν τρία bit, καθιστώντας την προσθήκη κλειδιού κουραστική και καθόλου εύκολη. Μια επιλογή είναι ο προϋπολογισμός των δευτερευόντων κλειδιών, αλλά ακόμα κι έτσι η προσθήκη κλειδιού θα απαιτούσε αρκετές λειτουργίες XOR για την ενημέρωση της κατάστασης των 128 bit. Χρησιμοποιώντας το bitslice, τα bit που κάποτε απείχαν τρία bit είναι τώρα είναι συσκευασμένα σε τμήματα των 32 bit, κάνοντας την προσθήκη κλειδιού τόσο απλή όσο μόλις δύο λειτουργίες XOR.

3.7.2 Λειτουργία αυθεντικοποιημένης κρυπτρογράφησης: COFB

Το COFB είναι μια ελαφριά λειτουργία AEAD. Η λειτουργία που παρουσιάζεται σε αυτήν την εγγραφή διαφέρει ελαφρώς με την αρχική αναφορά. Οι διαφορές είναι οι εξής:

- Έγινε η αλλαγή του μεγέθους της nonce σε 128 bit.
- Προσαρμόστηκε η λειτουργία feedback (για την ακρίβεια η λειτουργία G στο κύκλωμα) για να γίνει πιο αποδοτική στο υλικό.
- Επειδή τώρα πρέπει να αντιμετωπιστούν κενά δεδομένα, έγινε αλλαγή της λειτουργία της μάσκας για αυτό τον λόγο.
- Έγινε αλλαγή της λειτουργία padding κατά την επεξεργασία σχετικών δεδομένων. Για την ακρίβεια, αν τα σχετικά δεδομένα είναι κενά, τότε η λειτουργία padding θα δημιουργήσει ένα κενό μπλοκ δεδομένων όσο το μέγεθος που μπορεί να επεξεργαστεί.

Οι ενημερώσεις αυτές κάνουν το σχεδιασμό πιο αποδοτικό στην αντιμετώπιση μικρών εισόδων δεδομένων. Ωστόσο, δεν επηρεάζεται η ασφάλεια της λειτουργία αυτής.

Κεφάλαιο 4

Σε αυτό το κεφάλαιο θα δοθούν τα αποτελέσματα τις σύνθεσης και προσομοίωσης των αλγορίθμων που επιλέχτηκαν για την εργασία αυτή. Θα δοθούν οι τελικές αναφορές που θα παράξει το πρόγραμμα ISE Project Navigator και στη συνέχεια θα δειχτούν στιγμιότυπα από τις προσομοιώσεις τα οποία θα ακολουθούνται από επεξηγήσεις.

4.1. Σύνθεση αλγορίθμου Spix

Η σύνθεση του αλγορίθμου SPIX είναι επιτυχής χωρίς να προκύψει κάποιο σφάλμα κατά την εκτέλεση της. Το πρόγραμμα ISE Project Navigator αφού τελειώσει τη λειτουργία σύνθεσης θα παράξει μια αναφορά. Από την αναφορά φαίνεται ότι μεταγλωττίστηκαν όλα τα αρχεία σε κώδικα VHDL:

```
=========================================================================
*                          HDL Compilation                              *
=========================================================================
Compiling vhdl file "/home/ise/vmsharedfolder/spix/spix_pkg.vhd" in Library work.
Architecture spix_pkg of Entity spix_pkg is up to date.
Compiling vhdl file "/home/ise/vmsharedfolder/spix/lfsr_c.vhd" in Library work.
Architecture rtl of Entity lfsr_c is up to date.
Compiling vhdl file "/home/ise/vmsharedfolder/spix/sb_64.vhd" in Library work.
Architecture rtl of Entity sb_64 is up to date.
Compiling vhdl file "/home/ise/vmsharedfolder/spix/spix.vhd" in Library work.
Compiling vhdl file "/home/ise/vmsharedfolder/spix/dp.vhd" in Library work.
Entity <dp> compiled.
Entity <dp> (Architecture <rtl>) compiled.
Compiling vhdl file "/home/ise/vmsharedfolder/spix/ctl.vhd" in Library work.
Architecture rtl of Entity ctl is up to date.
Compiling vhdl file "/home/ise/vmsharedfolder/spix/spix-rtl.vhd" in Library work.
Architecture rtl of Entity spix is up to date.
=========================================================================
```

Στο επόμενο στιγμιότυπο που ακολουθεί είναι η αναφορά του προγράμματος αφού του συνθέσει όλα τα αρχεία κώδικα που μεταγλωττίστηκαν:

```
=========================================================================
HDL Synthesis Report
```

```
Macro Statistics
# Adders/Subtractors                                   : 1
 8-bit adder                                           : 1
# Registers                                            : 15
 1-bit register                                        : 7
 32-bit register                                       : 6
 4-bit register                                        : 1
 8-bit register                                        : 1
# Comparators                                          : 1
 8-bit comparator less                                 : 1
# Multiplexers                                         : 2
 32-bit 4-to-1 multiplexer                             : 2
# Xors                                                 : 25
 1-bit xor2                                            : 19
 32-bit xor2                                           : 2
 32-bit xor4                                           : 2
 64-bit xor2                                           : 2
=========================================================================
```

Στο επόμενο στιγμιότυπο που ακολουθεί είναι η αναφορά του προγράμματος αφού βελτιστοποιήσει τη μηχανή πεπερασμένων καταστάσεων της μονάδας ελέγχου:

```
=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# FSMs                                                 : 1
# Adders/Subtractors                                   : 1
 8-bit adder                                           : 1
# Registers                                            : 211
 Flip-Flops                                            : 211
# Comparators                                          : 1
 8-bit comparator less                                 : 1
```

```
# Multiplexers                                          : 2
 32-bit 4-to-1 multiplexer                             : 2
# Xors                                                  : 25
 1-bit xor2                                            : 19
 32-bit xor2                                           : 2
 32-bit xor4                                           : 2
 64-bit xor2                                           : 2
=========================================================================
```

Στο επόμενο στιγμιότυπο που ακολουθεί είναι η αναφορά του προγράμματος αφού βελτιστοποιήσει όλες τις μονάδες και τις οντότητες που έχουν περιγραφεί στα αρχεία κώδικα:

```
=========================================================================
Final Register Report

Macro Statistics
# Registers                                             : 217
 Flip-Flops                                            : 217
=========================================================================
```

Στο επόμενο στιγμιότυπο που ακολουθεί είναι η τελική αναφορά του προγράμματος αφού έχει τελειώσει η λειτουργία σύνθεσης:

```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : spix.ngr
Top Level Output File Name         : spix
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : No

Design Statistics
# IOs                              : 137

Cell Usage :
```

```
# BELS                              : 1065
#      GND                          : 1
#      LUT2                         : 41
#      LUT2_D                       : 3
#      LUT2_L                       : 1
#      LUT3                         : 410
#      LUT3_D                       : 5
#      LUT3_L                       : 67
#      LUT4                         : 359
#      LUT4_D                       : 10
#      LUT4_L                       : 23
#      MUXF5                        : 145
# FlipFlops/Latches                 : 217
#      FD                           : 6
#      FDE                          : 192
#      FDR                          : 6
#      FDRS                         : 3
#      FDS                          : 3
#      FDSE                         : 7
# Clock Buffers                     : 1
#      BUFGP                        : 1
# IO Buffers                        : 136
#      IBUF                         : 70
#      OBUF                         : 66
=========================================================================
```

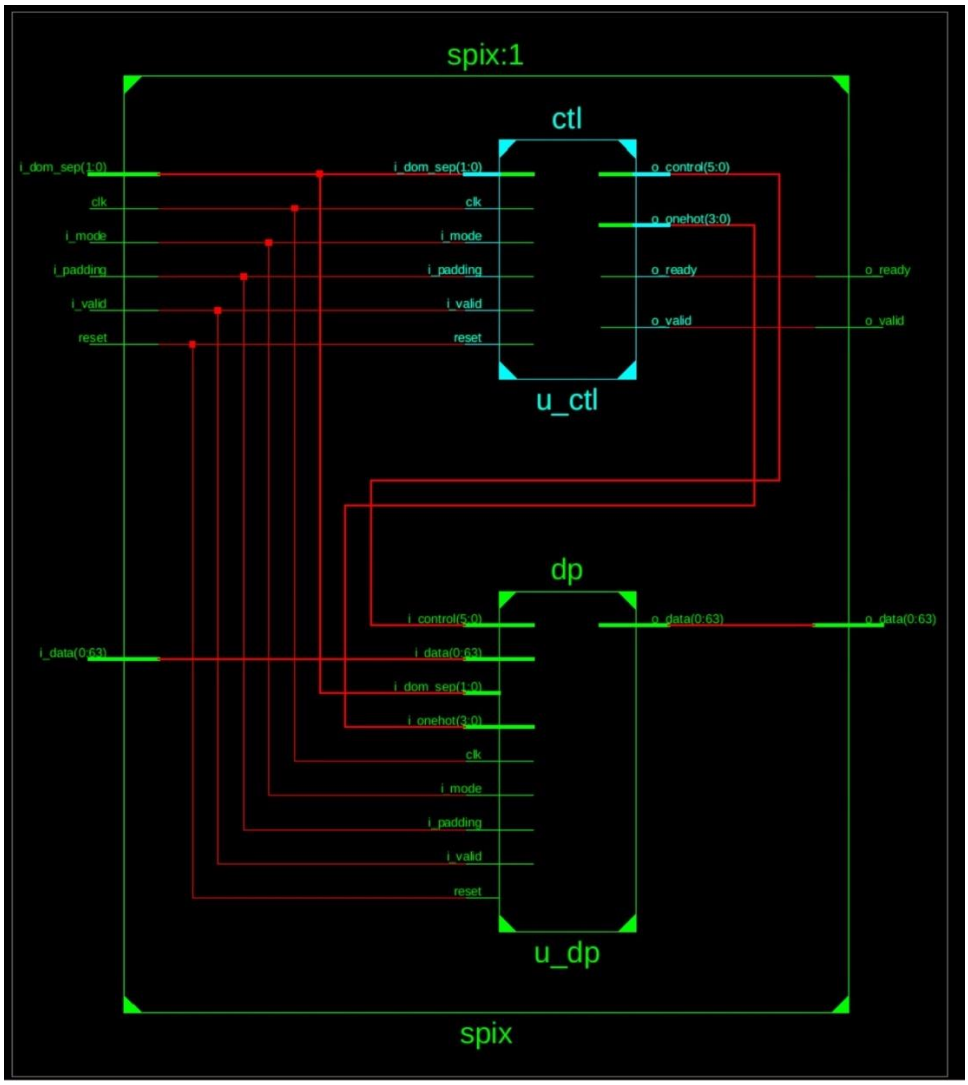
Στο επόμενο στιγμιότυπο που ακολουθεί δίνεται ο χρόνος που χρειάστηκε για τη λειτουργία σύνθεσης, η μνήμη που χρειάζεται και δίνεται αναφορά για σφάλματα, προειδοποιήσεις η πληροφορίες:

```
=========================================================================
Total REAL time to Xst completion: 8.00 secs
Total CPU time to Xst completion: 7.45 secs


-->
```

```
Total memory usage is 534372 kilobytes


Number of errors   :    0 (   0 filtered)
Number of warnings :   22 (   0 filtered)
Number of infos    :    1 (   0 filtered)
================================================================================
```

Η αναλυτική αναφορά βρίσκεται στο αρχείο Spix_Synthesis_Report.txt.

Ταυτόχρονα με την αναφορά, παράγονται και τα κυκλώματα που έχουν σχεδιαστεί στους κώδικες VHDL. Στο σχήμα που ακολουθεί φαίνεται ο αλγόριθμος Spix στο ανώτερο επίπεδο με τις εισόδους και τις εξόδους και ο τρόπος που συνδέονται στη μονάδα ελέγχου και μονάδα εκτέλεσης.



Σχήμα 19 – Εικονικό κύκλωμα SPIX

4.2. Λειτουργική προσομοίωση αλγορίθμου Spix

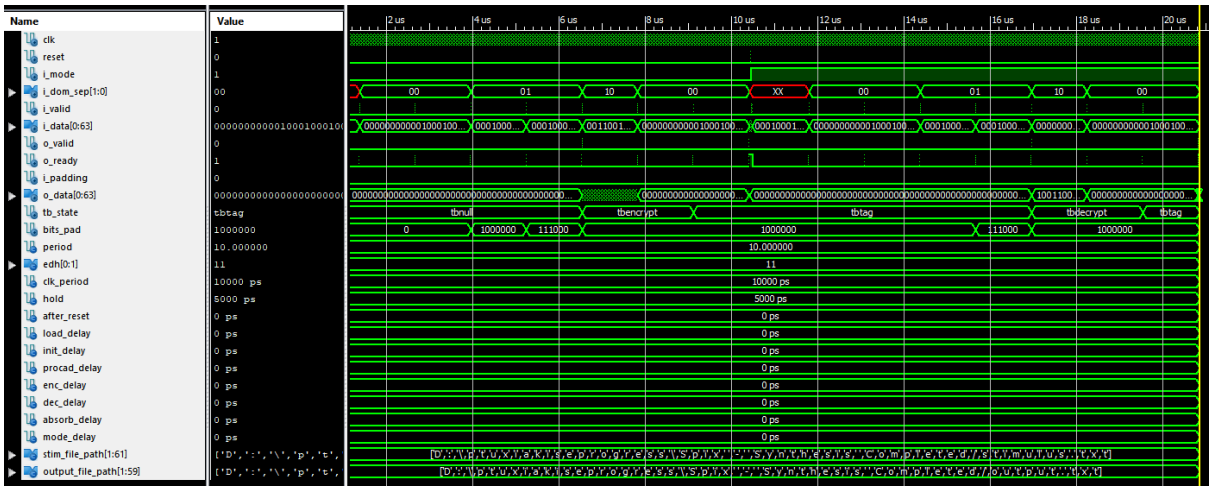
Το αρχείο testbench διαβάζει τα δεδομένα προς προσομοίωση από ένα αρχείο που στην περίπτωση αυτή ονομάζεται stimulus. Το αρχείο stimulus περιέχει όλες τις τιμές απαραίτητες για προσομοίωση του κυκλώματος. Το testbench διαβάζοντας τα αρχικά γράμματα K, N, A, P, C, αναγνωρίζει το διπλανό κείμενο σε τι αντιστοιχεί. Δηλαδή για το γράμμα K αναγνωρίζει ότι το 32 bit κείμενο που ακολουθεί είναι το κλειδί. Αντίστοιχα αναγνωρίζει τη nonce (N), τα σχετικά δεδομένα (A), το κείμενο (P) και το κρυπτογραφημένο κείμενο (C). Στο σχήμα 11 φαίνεται η μορφή ενός stimulus αρχείου.

```
K 00111122335588DD00111122335588DD
N 111122335588DD00111122335588DD00

A 1122335588DD00111122335588DD00
P 335588DD00111122
C 002200E1AA99AA35
```

Σχήμα 20 – Μορφή stimulus

Η λειτουργική παρομοίωση του κυκλώματος πραγματοποιείτε με την επιτυχημένη ολοκλήρωση του ελέγχου συμπεριφοράς (behavioral check). Στο αρχείο testhbench περιέχετε ένα σήμα με όνομα EDH το οποίο αναλόγως την τιμή του εκτελεί προσωμοίωση κρυπτογράφησης, αποκρυπτογράφησης η και τα δύο. Στη λειτουργική προσομοίωση θα γίνει κρυπτογράφηση του κειμένου και στη συνέχεια αποκρυπτογράφηση. Αυτό επιτυγχάνεται με το σήμα EDH να έχει την τιμή 11. Στο σχήμα 12 δίνεται στιγμιότυπο από την προσομοίωση του αλγορίθμου σε ISE Design Suite. Στις κυματομορφές φαίνεται ότι όσο το σήμα tb_state βρίσκεται στην κατάσταση tb_null που σημαίνει ότι δεν έχει ακόμα ξεκινήσει η λειτουργία κρυπτογράφησης. Αυτή η κατάσταση σημαίνει ότι το κύκλωμα βρίσκεται σε φάση αρχικοποίησης, και το testbench διαβάζει τις τιμές της nonce και του κλειδιού από το αρχείο stimulus και τα φορτώνει στο κύκλωμα με τη χρήση της εισόδου του κυκλώματος που είναι το σήμα i_data. Επιπλέον, ο μηχανισμός domain seperator αλλάζει την τιμή του σήματος i_dom_sep από 00 σε 01 που αντιστοιχεί στην αλλαγή του domain από αρχικοποίηση σε επεξεργασία σχετικών δεδομένων. Σε αυτή τη φάση, οι αλλαγές στα σήματα ελέγχου ερμηνεύονται από το κύκλωμα για να κάνει τις αντίστοιχες αλλαγές, ενώ το testbench μέσω του σήματος i_data φορτώνει στο κύκλωμα τα σχετικά δεδομένα. Στη συνέχεια, ο μηχανισμός domain separator αλλάζει την τιμή του σήματος i_dom_sep από 01 σε 10 που αντιστοιχεί στην αλλαγή του domain από επεξεργασία σχετικών δεδομένων σε λειτουργία κρυπτογράφησης. Αντίστοιχα με την επεξεργασία σχετικών δεδομένων έτσι και σε αυτή τη φάση το testbench φορτώνει στο σήμα i_data το κείμενο προς κρυπτογράφηση και κάνει τις αντίστοιχες αλλαγές στα σήματα ελέγχου ώστε το κύκλωμα να ξεκινήσει τη διαδικασία κρυπτογράφησης. Ταυτόχρονα με αυτή την αλλαγή ενημερώνεται και η κατάσταση του σήματος tb_state σε tbencrypt ώστε να ενημερώσει το χρήση ότι γίνεται κρυπτογράφηση. Αφού τελειώσει η διαδικασία της κρυπτογράφησης ο μηχανισμός domain separator αλλάζει την τιμή του σήματος i_dom_sep από 10 πίσω σε 00 που αντιστοιχεί στην αλλαγή του domain απο λειτουργία κρυπτογράφησης σε φάση οριστικοποίησης και η κατάστασης του σήματος tb_state αλλάζει σε tbtag για τη δημιουργία ετικετών. Σε αυτή τη φάση το κύκλωμα δημιουργεί ετικέτα για μετέπειτα επαλήθευση και η κατάσταση αλλάζει σε tbtag για να ενημερώσει το χρήση ότι

παράγεται η ετικέτα. Στη συνέχεια φαίνεται το σήμα i_dom_sep με κόκκινο χρώμα. Αυτό συμβαίνει γιατί το testbnech αρχικοποιείται για να ξεκινήσει τη διαδικασία αποκρυπτογράφησης. Η διαδικασία της αποκρυπτογράφησης είναι ίδια με τη διαφορά ότι η κατάσταση του σήματος tb_state αντί να γίνει tbencrypt, γίνεται tb decrypt και τα σήματα ελέγχου διαφέρουν ώστε το κύκλωμα να ξέρει ότι έχει κρυπτογραφημένο κείμενο και πρέπει να κάνει αποκρυπτογράφηση.



Σχήμα 21 – Προσομοίωση σε ISE Design Suite

Αφού τελειώσει η προσομοίωση, το testbench θα αποθηκεύσει τα αποτελέσματα της κρυπτογράφησης του κειμένου και της αποκρυπτογράφησης του ciphertext, που φορτώθηκε από το αρχείο stimulus, στο αρχείο output. Στο παρακάτω σχήμα φαίνεται η μορφή του αρχείου output πετά από τη λήξη της προσομοίωσης.

```
Ciphertext: 002200E1AA99AA35
Tag:            003311C3
Tag:            99CC22E8
Plaintext:  335588DD00111122
Tag:            003311C3
Tag:            99CC22E8
```

Σχήμα 22 – Μορφή output

4.3. Σύνθεση αλγορίθμου GIFT-COFB

Η σύνθεση του αλγορίθμου GIFT-COFB είναι επιτυχής χωρίς να προκύψει κάποιο σφάλμα κατά την εκτέλεση της. Το πρόγραμμα ISE Project Navigator αφού τελειώσει τη λειτουργία σύνθεσης θα παράξει μια αναφορά[42]. Από την αναφορά φαίνεται ότι μεταγλωττίστηκαν όλα τα αρχεία σε κώδικα VHDL:

```
=========================================================================
*                          HDL Compilation                              *
=========================================================================
```

```
Compiling vhdl file "/home/ise/Documents/giftcofb/gift128_pkg.vhd" in Library work.
Architecture gift128_pkg of Entity gift128_pkg is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/initialization.vhd" in Library work.
Architecture rtl of Entity initialization is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/finalization.vhd" in Library work.
Architecture rtl of Entity finalization is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/sbox.vhd" in Library work.
Architecture rtl of Entity sbox is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/keystate.vhd" in Library work.
Architecture rtl of Entity keystate is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/subcells.vhd" in Library work.
Architecture rtl of Entity subcells is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/permbits.vhd" in Library work.
Architecture rtl of Entity permbits is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/addroundkey.vhd" in Library work.
Architecture rtl of Entity addroundkey is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/keyschedule.vhd" in Library work.
Architecture rtl of Entity keyschedule is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/lfsr.vhd" in Library work.
Architecture rtl of Entity lfsr is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/feedback.vhd" in Library work.
Architecture rtl of Entity feedback is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/delta.vhd" in Library work.
Architecture rtl of Entity delta is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/dp.vhd" in Library work.
Architecture rtl of Entity dp is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/controller.vhd" in Library work.
Architecture rtl of Entity controller is up to date.
Compiling vhdl file "/home/ise/Documents/giftcofb/giftcofbtop.vhd" in Library work.
Architecture rtl of Entity giftcofb is up to date.
=========================================================================
```

Στο επόμενο στιγμιότυπο που ακολουθεί είναι η αναφορά του προγράμματος αφού του συνθέσει όλα τα αρχεία κώδικα που μεταγλωττίστηκαν:

```
=========================================================================
HDL Synthesis Report


Macro Statistics
# Adders/Subtractors                                   : 3
 6-bit adder                                          : 2
 64-bit adder                                         : 1
# Counters                                             : 2
 6-bit up counter                                     : 2
# Registers                                            : 146
 1-bit register                                       : 137
 128-bit register                                     : 5
 2-bit register                                       : 1
 64-bit register                                      : 3
# Latches                                              : 3
 1-bit latch                                          : 1
 3-bit latch                                          : 1
 64-bit latch                                         : 1
# Comparators                                          : 3
 6-bit comparator greatequal                          : 2
 6-bit comparator less                                : 1
# Multiplexers                                         : 1
 128-bit 4-to-1 multiplexer                           : 1
# Xors                                                 : 139
 1-bit xor2                                           : 65
 1-bit xor3                                           : 64
 128-bit xor2                                         : 1
 32-bit xor2                                          : 9
=========================================================================
```

Στο επόμενο στιγμιότυπο που ακολουθεί είναι η αναφορά του προγράμματος αφού βελτιστοποιήσει τη μηχανή πεπερασμένων καταστάσεων της μονάδας ελέγχου:

```
=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 3
 6-bit adder                                          : 2
 64-bit adder                                         : 1
# Counters                                             : 2
 6-bit up counter                                     : 2
# Registers                                            : 971
 Flip-Flops                                           : 971
# Latches                                              : 3
 1-bit latch                                          : 1
 3-bit latch                                          : 1
 64-bit latch                                         : 1
# Comparators                                          : 3
 6-bit comparator greatequal                          : 2
 6-bit comparator less                                : 1
# Multiplexers                                         : 1
 128-bit 4-to-1 multiplexer                           : 1
# Xors                                                 : 139
 1-bit xor2                                           : 65
 1-bit xor3                                           : 64
 128-bit xor2                                         : 1
 32-bit xor2                                          : 9
=========================================================================
```

Στο επόμενο στιγμιότυπο που ακολουθεί είναι η αναφορά του προγράμματος αφού βελτιστοποιήσει όλες τις μονάδες και τις οντότητες που έχουν περιγραφεί στα αρχεία κώδικα:

```
=========================================================================
Final Register Report

Macro Statistics
```

```
# Registers                                          : 983
 Flip-Flops                                         : 983
=========================================================================
```

Στο επόμενο στιγμιότυπο που ακολουθεί είναι η τελική αναφορά του προγράμματος αφού έχει τελειώσει η λειτουργία σύνθεσης:

```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : giftcofb.ngr
Top Level Output File Name         : giftcofb
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : No

Design Statistics
# IOs                              : 517

Cell Usage :
# BELS                             : 1514
#      GND                         : 1
#      INV                         : 4
#      LUT2                        : 334
#      LUT2_D                      : 1
#      LUT3                        : 400
#      LUT4                        : 454
#      LUT4_D                      : 1
#      MUXCY                       : 62
#      MUXF5                       : 193
#      VCC                         : 1
#      XORCY                       : 63
# FlipFlops/Latches                : 1050
```

```
#      FD                          : 256
#      FDE                         : 707
#      FDR                         : 12
#      FDRE                        : 7
#      FDSE                        : 1
#      LD_1                        : 3
#      LDCP_1                      : 64
# Clock Buffers                    : 2
#      BUFG                        : 1
#      BUFGP                       : 1
# IO Buffers                       : 516
#      IBUF                        : 259
#      OBUF                        : 257
=========================================================================
```

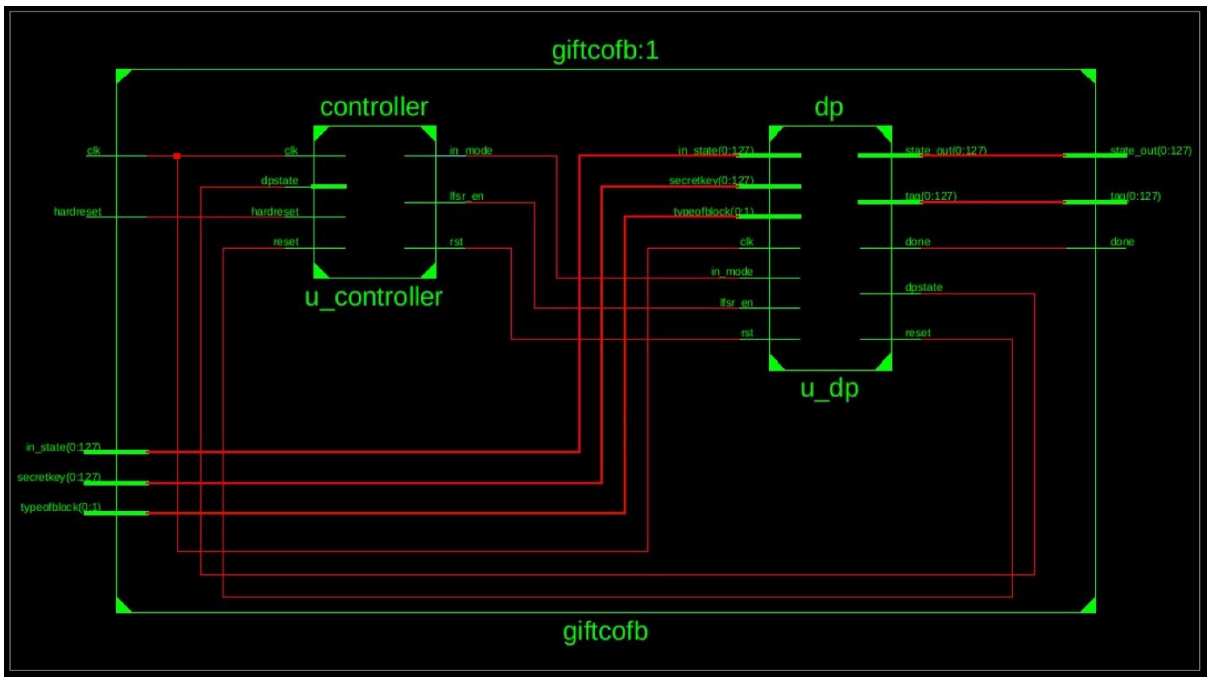
Στο επόμενο στιγμιότυπο που ακολουθεί δίνεται ο χρόνος που χρειάστηκε για τη λειτουργία σύνθεσης, η μνήμη που χρειάζεται και δίνεται αναφορά για σφάλματα, προειδοποιήσεις η πληροφορίες:

```
=========================================================================
Total REAL time to Xst completion: 9.00 secs
Total CPU time to Xst completion: 8.00 secs


-->


Total memory usage is 550500 kilobytes


Number of errors   :    0 (   0 filtered)
Number of warnings :    9 (   0 filtered)
Number of infos    :    3 (   0 filtered)
=========================================================================
```

Η αναλυτική αναφορά βρίσκεται στο αρχείο GIFTCOFB_Synthesis_Report.txt .

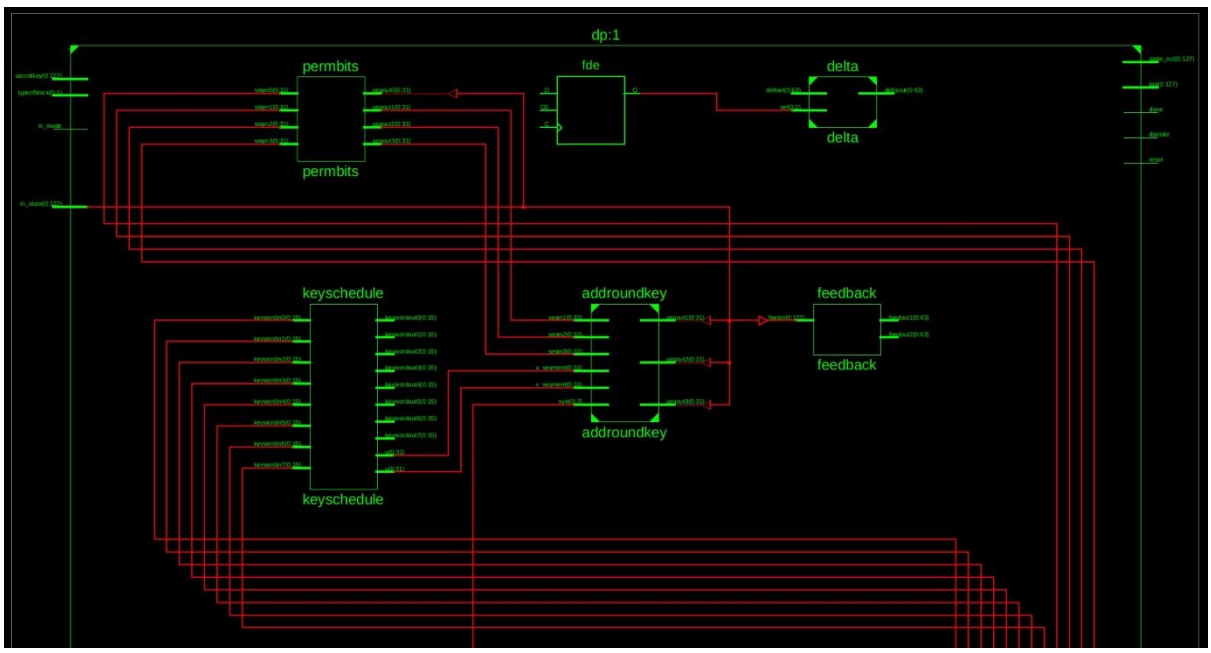
Ταυτόχρονα με την αναφορά, παράγονται και τα κυκλώματα που έχουν σχεδιαστεί στους κώδικες VHDL. Στο σχήμα που ακολουθεί φαίνεται ο αλγόριθμος GIFT-COFB στο ανώτερο

επίπεδο με τις εισόδους και τις εξόδους και ο τρόπος που συνδέονται στη μονάδα ελέγχου και μονάδα εκτέλεσης.

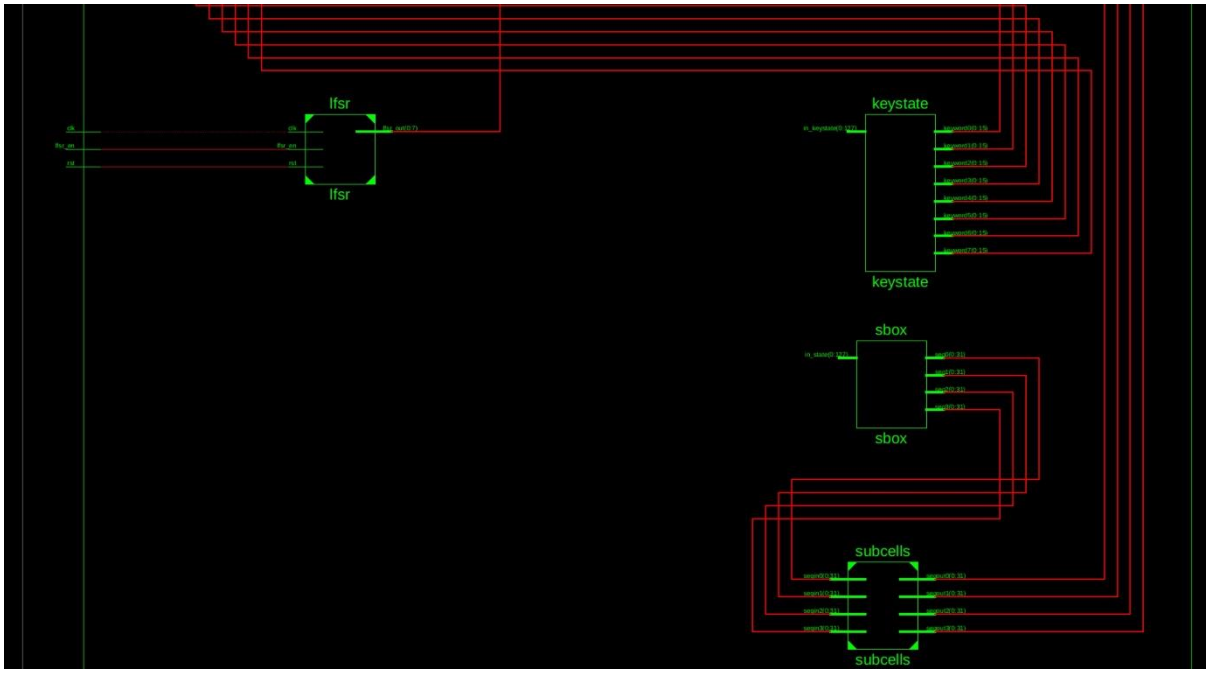


Σχήμα 23 – Εικονικό κύκλωμα GIFT-COFB μέρος 1

Στα σχήματα που ακολουθούν φαίνεται πιο αναλυτικά το κύκλωμα της μονάδας εκτέλεσης με τις οντότητες που περιέχει και το τρόπο που συνδέονται μεταξύ τους.



Σχήμα 24 – Εικονικό κύκλωμα GIFT-COFB μέρος 2



Σχήμα 25 – Εικονικό κύκλωμα GIFT-COFB μέρος 3

4.4. Λειτουργική προσομοίωση αλγορίθμου GIFT-COFB

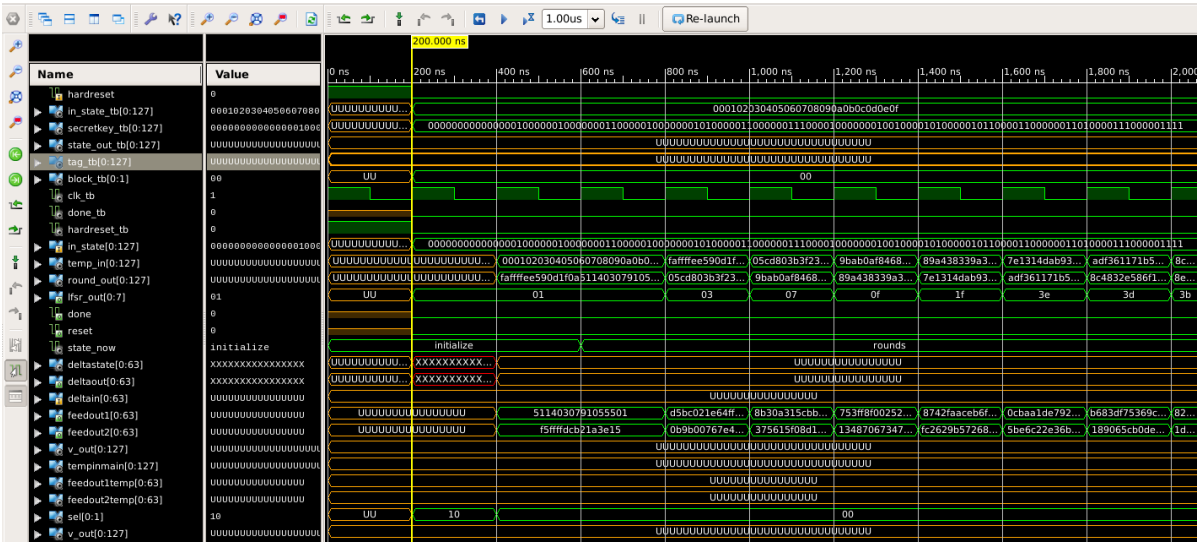
Η λειτουργική προσομοίωση του κυκλώματος πραγματοποιείτε με την επιτυχημένη ολοκλήρωση του ελέγχου συμπεριφοράς (behavioral check). Στην προσομοίωση αυτή κάθε χτύπος ρολογιού χρειάζεται 200ns χρόνο. Θα προσομοιωθεί μία nonce, μια είσοδος σχετικών δεδομένων και ένα μήνυμα προς κρυπτογράφηση. Στις κυματομορφές της προσομοίωσης που φαίνονται στο σχήμα 23, το σήμα hardreset είναι ενεργοποιημένο για τα πρώτα 200ns της προσομοίωσης, το οποίο κρατάει το κύκλωμα σε αδράνεια, και αφού απενεργοποιηθεί τότε τα δεδομένα θα φορτωθούν από το testbench. Οι δοκιμαστικές τιμές που φορτώνονται φαίνονται στο σχήμα 22 και προσομοιώνονται σαν nonce. Το κύκλωμα χρειάζεται 200ns ακόμα, ώστε οι τιμές του testbench να φορτωθούν στο μπλοκ GIFT και να ξεκινήσει η ρουτίνα κρυπρογράφησης. Στη συνέχεια αφού περάσει ένας κύκλος ρολογιού που χρειάζεται η μονάδα ελέγχου να ξεκινήσει να τρέχει τη χρονική στιγμή 400ns με 600ns, ξεκινάνε οι 40 γύροι του μπλοκ GIFT. Σε κάθε γύρο η έξοδος του μπλοκ GIFT, που είναι το round_out, εισέρχεται ως είσοδο πίσω στο μπλοκ στον επόμενο κύκλο ρολογιού. Αυτό σημαίνει ότι το κύκλωμα σωστά τροφοδοτεί το μπλοκ GIFT με την έξοδο του από προηγούμενο γύρο για 40 γύρους. Στο σχήμα 24, τη χρονική στιγμή 8.400 ns ξεκινάει η εκτέλεση του τελευταίου γύρου του μπλοκ GIFT. Το αποτέλεσμα του τελευταίου γύρου συμφωνεί με τις τιμές των test vectors του σχήματος 22. Σε αυτό το γύρο, το σήμα reset στέλνει την τιμή 1 στη μονάδα ελέγχου για να σταματήσει την λειτουργία του, το σήμα done στέλνει την τιμή 1 στο testbench για να ετοιμαστεί να δώσει επόμενα δεδομένα, και οι τιμές (feedout1, feedout2) από τη λειτουργία feedback αποθηκεύονται στα προσωρινά σήματα feedout1temp και feedout2temp αφού εφαρμοστούν οι απαραίτητες αλλαγές που απαιτεί η λειτουργία αυτή. Να σημειωθεί ότι το κύκλωμα δεν εμφανίζει έξοδο σε αυτή τη φάση γιατί δεν έχει γίνει επεξεργασία μηνύματος προς

κρυπτογράφηση. Επιπλέον, σε αυτή τη χρονική στιγμή η λειτουργία delta δημιουργεί την αρχική κατάσταση από τα πρώτα 64 bits της εξόδου του μπλοκ GIFT. Αφού περάσουν 400 ns, απενεργοποιείτε το σήμα reset και done και το κύκλωμα έχει ετοιμαστεί να δεχτεί σχετικά δεδομένα. Ακολουθούν 800 ns καθυστέρηση επειδή η λειτουργία delta χρειάζεται τέσσερεις κύκλους ρολογιού για να ενημερώσει την κατάσταση της. Κατά τη διάρκεια αυτών των τεσσάρων κύκλων ρολογιού, το κύκλωμα φορτώνει τις νέες τιμές από το testbench για προσομοίωση επεξεργασίας σχετικών δεδομένων. Ταυτόχρονα το κύκλωμα αντί να φορτώσει στο μπλοκ GIFT κατευθείαν τις τιμές, αναγνωρίζει ότι έχει δεχτεί σχετικά δεδομένα και παράγει την κατάσταση v_out η οποία παράγεται με τη χρήση της εισόδου, την εξόδο feedout (feedout1temp και feedout2temp) από την προηγούμενη ρουτίνα κρυπτογράφησης, την κατάσταση delta και θα εκτελέσει τις λογικές πράξεις που απαιτεί ο αλγόριθμος όπως φαίνεται στο σχήμα 20. Η κατάσταση v_out στη συνέχεια θα φορτωθεί στην είσοδο του Μπλοκ GIFT και αφού τελειώσει ο υπολογισμός της λειτουργίας delta τότε το κύκλωμα θα ξεκινήσει να τρέχει για σαράντα γύρους που στην προσομοίωση αυτή είναι τη χρονική στιγμή 9.800 ns. Στο σχήμα 25, φαίνεται τη χρονική στιγμή 17,600 ns ότι τελειώνει η επεξεργασία μετά από σαράντα γύρους εκτέλεσης. Σε αυτό το κομμάτι η διαδικασία είναι παρόμοια με πριν. Το σήμα reset στέλνει την τιμή 1 στη μονάδα ελέγχου για να σταματήσει τη λειτουργία του, το σήμα done στέλνει την τιμή 1 στο testbench για να ετοιμαστεί να δώσει επόμενα δεδομένα, και οι τιμές (feedout1, feedout2) από τη λειτουργία feedback αποθηκεύονται στα προσωρινά σήματα feedout1temp και feedout2temp αφού εφαρμοστούν οι απαραίτητες αλλαγές που απαιτεί η λειτουργία αυτή. Το κύκλωμα συνεχίζει να μην εμφανίζει έξοδο γιατί ακόμα δεν έχει γίνει επεξεργασία μηνύματος προς κρυπτογράφηση. Αφού περάσουν 400 ns το οποίο απενεργοποιείτε το σήμα reset και done και το κύκλωμα έχει ετοιμαστεί να δεχτεί σχετικά δεδομένα. Ακολουθούν 800 ns καθυστέρηση επειδή η λειτουργία delta χρειάζεται τέσσερεις κύκλους ρολογιού για να ενημερώσει την κατάσταση της. Κατά τη διάρκεια αυτών των τεσσάρων κύκλων ρολογιού, το κύκλωμα φορτώνει τις νέες τιμές από το testbench για προσομοίωση επεξεργασία σχετικών δεδομένων. Το κύκλωμα αναγνωρίζει ότι έχει δεχτεί μήνυμα και παράγει την κατάσταση v_out η οποία παράγεται με τη χρήση της εισόδου, την εξόδο feedout (feedout1temp και feedout2temp) από την προηγούμενη ρουτίνα κρυπτογράφησης, την κατάσταση delta και θα εκτελέσει τις λογικές πράξεις που απαιτεί ο αλγόριθμος όπως φαίνεται στο σχήμα 20. Η κατάσταση v_out στη συνέχεια θα φορτωθεί στην είσοδο του Μπλοκ GIFT και αφού τελειώσει ο υπολογισμός της λειτουργίας delta τότε το κύκλωμα θα ξεκινήσει να τρέχει για σαράντα γύρους που στην προσομοίωση αυτή είναι την χρονική στιγμή 19.000 ns. Να σημειωθεί, ότι σε αυτή τη φάση ξεκινάει η κρυπτογράφηση του μηνύματος. Στο σχήμα 26, φαίνεται τη χρονική στιγμή 26,800 ns ότι τελειώνει η επεξεργασία μετά από σαράντα γύρους εκτέλεσης. Το κύκλωμα αναγνωρίζει ότι κρυπτογράφησε μήνυμα σε αυτή τη φάση, έτσι στις έξοδο του (state_out) θα εμφανίσει αποτέλεσμα όπως φαίνεται τη χρονική στιγμή 27,000. Το αποτέλεσμα που θα εμφανιστεί, παράγεται από την τελευταία έξοδο του γύρου (round_out) και γίνεται XOR με την είσοδο που είχε το κύκλωμα από το testbench για να παράξει το κρυπτογραφημένο κείμενο.

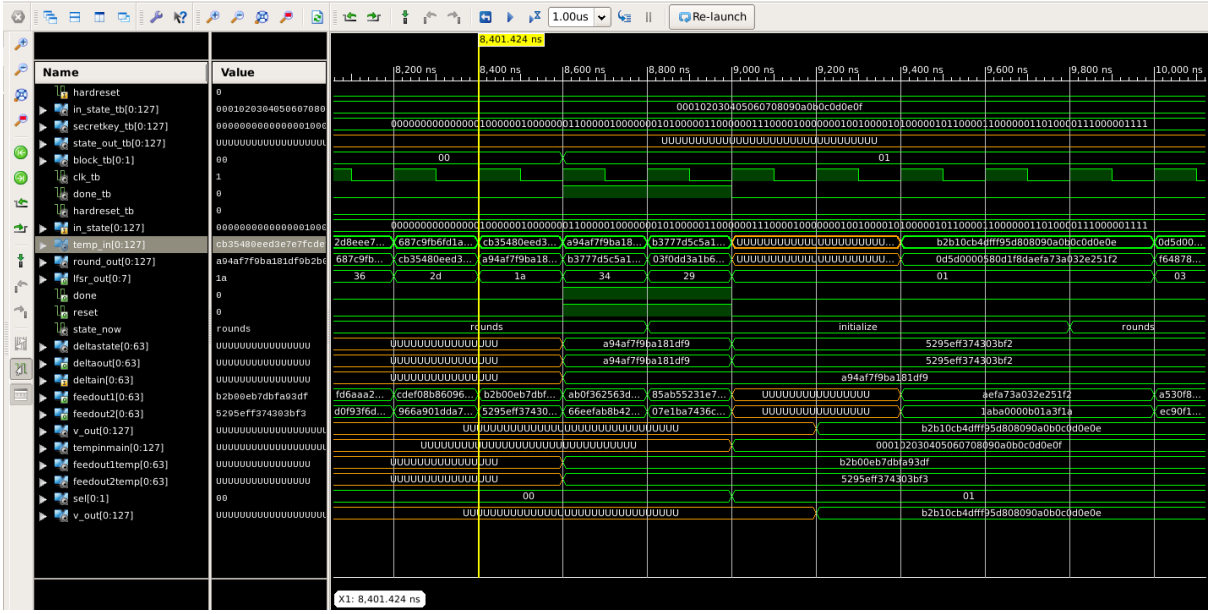
Test Vectors

Key	:	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Plaintext	:	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Ciphertext	:	A9 4A F7 F9 BA 18 1D F9 B2 B0 0E B7 DB FA 93 DF

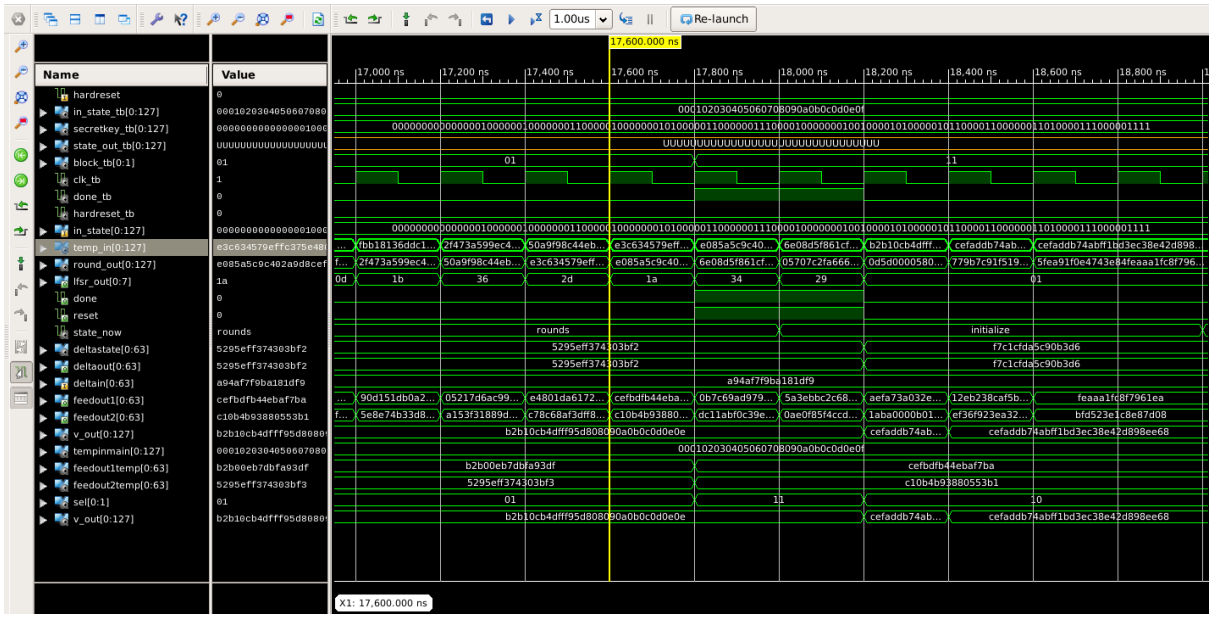
Σχήμα 26 – Test vectors για αλγόριθμο GIFT-COFB



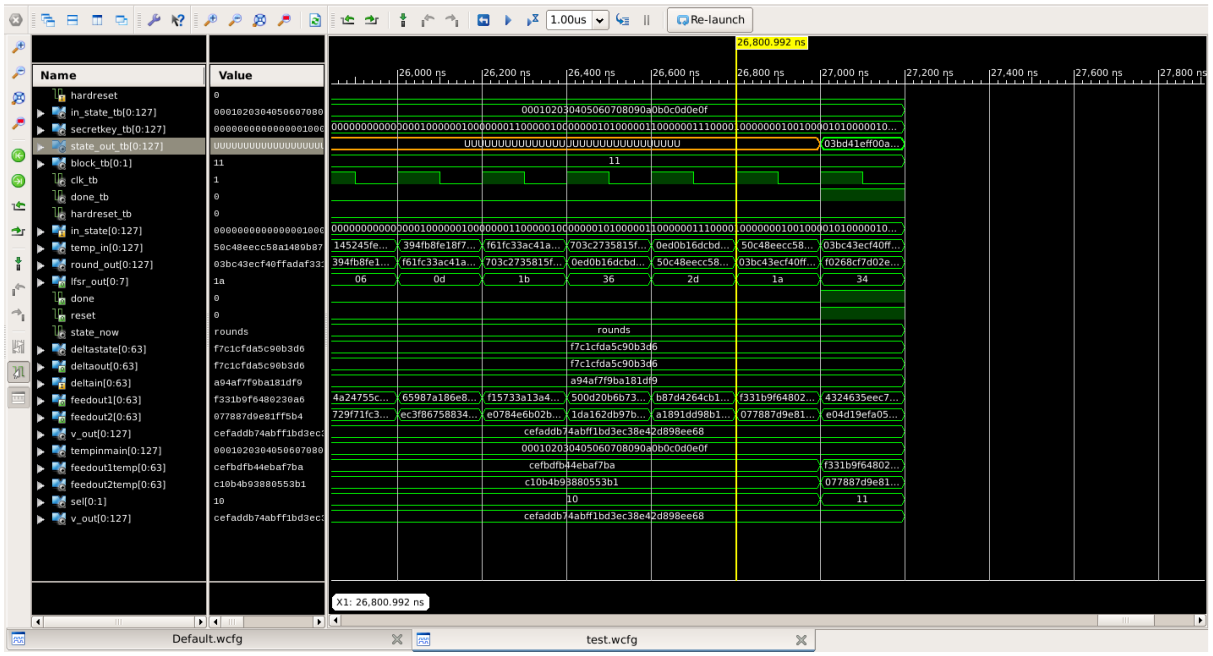
Σχήμα 27 – Προσομοίωση για αλγόριθμο GIFT-COFB μέρος 1



Σχήμα 28 – Προσομοίωση για αλγόριθμο GIFT-COFB μέρος 2



Σχήμα 29 – Προσομοίωση για αλγόριθμο GIFT-COFB μέρος 3



Σχήμα 30 – Προσομοίωση για αλγόριθμο GIFT-COFB μέρος 4

Κεφάλαιο 5

Συμπεράσματα

Σε αυτό το κεφάλαιο θα γίνει αναφορά στα συμπεράσματα που βγήκαν μετά την ολοκλήρωση της παρούσας πτυχιακής εργασίας. Σκοπός της πτυχιακής εργασίας ήταν να υλοποιηθούν δύο από τους αλγόριθμους που προτάθηκαν στο διαγωνισμό που διοργανώθηκε από την NIST (National Institute of Standards and Technology). Έτσι, οι αλγόριθμοι Spix και GIFT-COFB επιλέχτηκαν για την υλοποίηση αυτής της εργασίας. Στα κεφάλαια 2 και 3 έγινε γενική περιγραφή της λειτουργίας αυτών των αλγορίθμων και ακολούθησε ο τρόπος που έγινε η υλοποίηση τους σε κώδικα VHDL. Τέλος έγινε η προσομοίωση των αρχείων με κώδικα που δημιουργήθηκαν με τη χρήση του εργαλείου ISE Design Suite. Οι προσομοιώσεις περιέχονται στο τέλος των κεφαλαίων αυτών.

Ένα από τα πρώτα συμπεράσματα που αμέσως βγήκαν από την εργασία αυτή, είναι ότι τα εγχειρίδια που περιγράφουν τις λειτουργίες των αλγορίθμων κρυπτογράφησης είναι ένα από τα πιο σημαντικά πράγματα που χρειάζονται κατά την υλοποίηση των αλγόριθμων αυτών σε κώδικα VHDL. Το σωστά δομημένο εγχειρίδιο με βαθιά ανάλυση των λειτουργιών των αλγορίθμων βοηθάει τον αναγνώστη να αποκτήσει μεγαλύτερη κατανόηση του αλγορίθμου το οποίο συνεπάγεται στην πιο σωστή και εύκολη υλοποίηση του σε κώδικα VHDL. Επιπλέον, από τη στιγμή που η χρήση του κώδικα VHDL γίνεται για την υλοποίηση κυκλωμάτων, η παροχή αναλυτικών διαγραμμάτων των κυκλωμάτων των αλγορίθμων στα εγχειρίδια παρέχουν πολύ μεγάλη βοήθεια στην υλοποίηση τους.

Δεύτερο συμπέρασμα που βγήκε από την εκπόνηση της εργασίας αυτής, είναι ότι με όσο περισσότερη λεπτομέρεια σχεδιαστούν τα ψηφιακά κυκλώματα, τόσο περισσότερο γίνεται η κατανόηση της λειτουργίας του. Ακόμα, όσο περισσότερος χρόνος καταναλωθεί στη σχεδίαση, τόσο περισσότερες λεπτομέρειες κατανοούνται πάνω στη σχεδίαση αυτή. Αυτό το συμπέρασμα ήταν ιδιαιτέρα αισθητό κατά τη σχεδίαση των αλγορίθμων, γιατί με τον αλγόριθμο Spix έγινε η υλοποίηση του με τη βοήθεια του αλγόριθμου ACE οι οποίοι μοιράζονται παρόμοια αρχιτεκτονική. Η υλοποίηση του αλγορίθμου Spix πρόσφερε μεγάλη κατανόηση στον τρόπο που υλοποιούνται αλγόριθμοι κρυπτογράφησης σε κώδικα VHDL και οι μηχανισμοί που χρησιμοποιούνται. Τη μεγαλύτερη κατανόηση όμως την πρόσφερε η υλοποίηση του αλγόριθμου GIFT-COFB τον οποίο ο σχεδιασμός του έγινε από την αρχή χωρίς κάποια βοήθεια από κάποιον παρόμοιο αλγόριθμο. Η υλοποίηση ενός αλγορίθμου κρυπτογράφησης χωρίς τη βοήθεια ενός προτύπου αφήνει τον σχεδιαστή υπεύθυνο να σχεδιάσει ο ίδιος τον τρόπο που θα γράψει τον κώδικα και θα πρέπει ο ίδιος να δημιουργήσει ένα πλάνο που θα πρέπει να ακολουθήσει κατά τον σχεδιασμό. Η ελευθερία αυτή οδήγησε σε πολλά λάθη και πολλές τροποποιήσεις στα κομμάτια κώδικα μέχρι να επιτευχθεί η επιθυμητή συμπεριφορά. Επιπλέον, τα λάθη και οι συνεχώς τροποποιήσεις του κώδικα οδήγησαν στη χρήση μηχανημάτων και ελέγχων που χρη-

σιμοποιήθηκαν και από σχεδιαστές των άλλων αλγορίθμων που προτάθηκαν σε αυτό τον διαγωνισμό. Αυτό οδήγησε όχι μόνο στην κατανόηση του κώδικα των άλλων αλγορίθμων αλλά και γιατί χρειάστηκε να σχεδιάσουν τους κώδικες με αυτόν τον τρόπο.

Συνεχίζοντας από το προηγούμενο συμπέρασμα, ο σχεδιασμός των αλγορίθμων σε κώδικα VHDL, οδήγησε στη μεγαλύτερη εξοικείωση και γνώση των κανόνων της γλώσσας VHDL. Επιπλέον η υλοποίηση των αλγορίθμων σε πολλά μικρά κομμάτια κώδικα αντί σε λιγότερα και μεγάλα οδηγεί στον πολύ πιο εύκολο σχεδιασμό του αλγορίθμου. Είναι πιο εύκολο να επιβεβαιωθεί η σωστή λειτουργία κάθε κομματιού σε προσομοίωση, σε περίπτωση λάθους είναι πιο εύκολο να εντοπιστεί και είναι πιο εύκολη η μετατροπή του κώδικα αν χρειάζεται να αλλάξει, προστεθεί η να αφαιρεθεί κάποια λειτουργία. Τέλος, γίνεται πιο εύκολη η επαναχρησιμοποίηση κάποιου κώδικα σε κάποιο άλλο σχεδιασμό η εργασία.

Η χρήση των εργαλείων που χρησιμοποιήθηκαν για τον έλεγχο και την προσομοίωση των υποκυκλωμάτων η τη λειτουργία όλου του αλγορίθμου αποδεικνύουν πόσο σημαντικό ρόλο έχουν στο σχεδιασμό κυκλωμάτων. Στην περίπτωση της εργασίας αυτής, η χρήση του εργαλείου ISE Design Suite της Xilinx ήταν πολύ μεγάλη βοήθεια γιατί επέτρεπε τον εντοπισμό συντακτικών η λογικών λαθών, και σε πραγματικό χρόνο μπορούσαν να παρατηρηθούν οι αλλαγές της συμπεριφοράς του κυκλώματος μέσο προσομοίωσης σε τροποποιήσεις που γινόντουσαν στον κώδικα.

Βιβλιογραφία

Βιβλία:

- VOLNEI A. PEDRONI, <<Σχεδιασμός κυκλωμάτων με τη VHDL>>, ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ [1]
- M. MORRIS MANO – MICHAEL D. CILETTI, <<ΨΗΦΙΑΚΗ ΣΧΕΔΙΑΣΗ>>, ΠΑΠΑΣΩΤΗΡΙΟΥ ΕΚΔΟΣΕΙΣ [2]

Ιστότοποι:

- Lightweight Cryptography | CSRC (nist.gov) [3]
- ACE (nist.gov) [4]
- SPIX (nist.gov) [5]
- Sponge function - Wikipedia [6]
- MonkeyDuplex construction | Download Scientific Diagram (researchgate.net) [7]
- GIFT-COFB (nist.gov) [8]
- GIFT-COFB Authenticated Encryption Scheme (isical.ac.in) [9]
- PRESENT: An Ultra-Lightweight Block Cipher | SpringerLink [10]
- LNCS 4727 - PRESENT: An Ultra-Lightweight Block Cipher (springer.com) [11]
- https://stackoverflow.com/ [12]
- eclass.uop.gr | Γλώσσες Περιγραφής Υλικού (HDL) [13]
- sLiSCP-light: Towards Lighter Sponge-specific Cryptographic Permutations (uwaterloo.ca) [14]
- Cryptology ePrint Archive: Report 2013/404 - The SIMON and SPECK Families of Lightweight Block Ciphers (iacr.org) [15]

ΠΑΡΑΡΤΗΜΑ Α

Περιεχόμενα κωδικών VHDL

Αλγόριθμος SPIX

Το περιεχόμενο του αρχείου spix_pkg.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package spix_pkg is

    --for constants
    constant lfsr_c_sz    :  integer := 7;
    subtype lfsr_c_output is std_logic_vector(0 to lfsr_c_sz+2);

    ------------------------------------------------------------

    constant half_word_sz : natural := 32;
    constant word_sz      : natural := 64;

    subtype half_word       is std_logic_vector( 0 to 31 );
    subtype word            is std_logic_vector( 0 to 63 );
    type word_vector        is array( natural range <> ) of word;
    type half_word_vector   is array( natural range <> ) of half_word;

    ------------------------------------------------------------
    -- X0, X1, X2, X3

    constant state_sz          : natural := 256;
    constant word_max_idx      : natural := 4; -- 4 SubBlocks
    constant half_word_max_idx : natural := 8; -- 8 Half SubBlocks

    constant key_sz            : natural := 128;
    constant nonce_sz          : natural := 128;

    subtype word_state_ty is word_vector ( 0 to 4 );
    constant x0_idx : natural := 0;
    constant x1_idx : natural := 1;
    constant x2_idx : natural := 2;
    constant x3_idx : natural := 3;

    subtype half_word_data is  half_word_vector ( 0 to 1 );

    subtype half_word_state_ty is half_word_vector ( 0 to 8 );
    constant x0_0_idx : natural := 1;
    constant x0_1_idx : natural := 0;
    constant x1_0_idx : natural := 3;
    constant x1_1_idx : natural := 2;
    constant x2_0_idx : natural := 5;
    constant x2_1_idx : natural := 4;
    constant x3_0_idx : natural := 7;
    constant x3_1_idx : natural := 6;

    function b2x( b : boolean ) return std_logic;
```

```
    function half_words_to_words( st : half_word_state_ty ) return
word_state_ty;
    function words_to_half_words( st : word_state_ty ) return
half_word_state_ty;

    ------------------------------------------------------
    -- mode

    subtype mode_ty is std_logic;      -- top lvl input
    constant encrypt_mode : mode_ty := '0';
    constant decrypt_mode : mode_ty := '1';

    subtype domsep_ty is std_logic_vector( 1 downto 0 );    -- top lvl input

     -- derived control (from counter and more)
    subtype spix_ctl_ty is std_logic_vector( 5 downto 0 );
    constant replace_idx      : natural := 0;
    constant output_idx       : natural := 1;
    constant endstep_idx      : natural := 2;
    constant permoff_idx      : natural := 3;
    constant lfsr_c_reset_idx : natural := 4;
    constant lfsr_c_en_idx    : natural := 5;

    -- extras cntl for load, init, fin, tag
    subtype onehot_ty is std_logic_vector( 3 downto 0);

    ------------------------------------------------------------

    constant bits_counter : natural := 8;
    subtype count_ty is unsigned( bits_counter - 1 downto 0 );

    ------------------------------------------------------------------------
    -- standard vhdl operators

    function onehot_rotate (a : onehot_ty) return onehot_ty;
    function vector_to_data ( st : half_word_data ) return word;
    function data_to_vector ( st : word ) return half_word_data;

    ------------------------------------------------------------------------

end package;

------------------------------------------------------------------------
--
------------------------------------------------------------------------

package body spix_pkg is

  function onehot_rotate (a : onehot_ty)
    return onehot_ty
  is
    variable z : onehot_ty;
  begin
      z(onehot_ty'high downto 1) := a(onehot_ty'high - 1 downto 0);
      z(0) := a(onehot_ty'high);
    return z;
  end function;
```

```
  function b2x( b : boolean ) return std_logic is
  begin
    if b then
      return '1';
    else
      return '0';
    end if;
  end function;

  function half_words_to_words( st : half_word_state_ty )
    return word_state_ty
  is
    variable i : natural;
    variable z : word_state_ty;
  begin
    main_loop : for i in 0 to word_max_idx loop
     z(i)(0 to half_word_sz - 1)   := st(2*i);
     z(i)(half_word_sz to word_sz - 1)   := st(2*i); --st(2*i+1); error
    end loop;
  return z;
  end function;

  function words_to_half_words( st : word_state_ty )
    return half_word_state_ty
  is
    variable i : natural;
    variable z : half_word_state_ty;
  begin
    main_loop : for i in 0 to word_max_idx loop
      z(2*i)      := st(i)(0 to half_word_sz - 1);
      z(2*i) := st(i)(half_word_sz to word_sz - 1); --z(2*i+1); error
    end loop;
  return z;
  end function;

  function data_to_vector( st : word )
    return half_word_data
  is
    variable z : half_word_data;
  begin
    z(0) := st(0 to half_word_sz - 1);
    z(1) := st(half_word_sz to word_sz - 1);
    return z;
  end function;


  function vector_to_data( st : half_word_data )
    return word
  is
    variable z : word;
  begin
    z(0 to half_word_sz - 1) := st(0);
    z(half_word_sz to word_sz - 1) := st(1);
    return z;
  end function;

end package body;
```

Ακολουθεί το περιεχόμενο του αρχείου sb_64.vhd :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.spix_pkg.all;

entity sb_64 is
  port
    ( i_state : in  word
    ; i_rc    : in  std_logic
    ; o_state : out word
    );
end entity;

architecture rtl of sb_64 is
  signal x0, x1, z0, z1 : half_word;
  signal rc : half_word;
begin

  x1 <= i_state(            0 to half_word_sz-1 );
  x0 <= i_state( half_word_sz to word_sz - 1    );

  rc <= ( 0 to half_word_sz - 2 => '1', half_word_sz - 1 => i_rc );

  z0 <= x1;
  z1 <=    ( ( x1(5 to half_word_sz - 1) & x1 (0 to 4) ) and x1)
        xor (  x1(1 to half_word_sz - 1) & x1 (0)      )
        xor x0
        xor rc;


  o_state <= z1 & z0;

end architecture;
```

Στην συνέχεια ακολουθεί το περιεχόμενο του αρχείου lfsr_c.vhd :

```
library ieee;
use ieee.std_logic_1164.all;
use work.spix_pkg.all;


entity lfsr_c is
  port
    ( clk          : in std_logic
    ; lfsr_c_en    : in std_logic
    ; lfsr_c_reset : in std_logic
    ; o_const      : out lfsr_c_output
   );
end lfsr_c;

architecture rtl of lfsr_c is
  signal sa: std_logic_vector(lfsr_c_sz - 1 downto 0);
  signal xa: std_logic_vector(lfsr_c_sz + 2 downto 0);
begin
```

```
-- 10 output bits for the constants
  o_const <= xa;  -- "to" type <= "downto" type. Index flip intended

-- just rename signal
  xa(lfsr_c_sz-1 downto 0) <= sa(lfsr_c_sz-1 downto 0);

-- for updates and outputs
  xa(lfsr_c_sz + 2 downto lfsr_c_sz) <= xa(3 downto 1) xor xa(2 downto 0);

  lfsr_shift: for i in lfsr_c_sz-1 downto 0 generate
    lfsr_step: process(clk) begin
      if rising_edge(clk) then
        if lfsr_c_reset ='1' then
          sa(i) <= '1';
        elsif lfsr_c_en ='1' then
          sa(i) <= xa(i+3);
        end if;
      end if;
    end process;
  end generate lfsr_shift;

end;
```

Το περιεχόμενο του αρχείου ctl.vhd είναι το ακόλουθο :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.spix_pkg.all;

entity ctl is
  port
    ( clk       : in  std_logic
    ; reset     : in  std_logic
    ; i_mode    : in  mode_ty
    ; i_dom_sep : in  domsep_ty
    ; i_valid   : in  std_logic
    ; i_padding : in  std_logic
    ; o_valid   : out std_logic
    ; o_onehot  : out onehot_ty
    ; o_ready   : out std_logic
    ; o_control : out spix_ctl_ty
   );
end entity;

architecture rtl of ctl is

  type state_ty is (Load, pLoad, iLoad2, iInit, pInit, iProcAD, pProcAD,
pEncrypt, pDecrypt, iFinal, pFinal, Tag);
  signal state, state_next   : state_ty;
  signal count, count_next   : count_ty;
  signal onehot, onehot_next : onehot_ty;
  signal permoff, lfsr_c_reset : std_logic;
  constant onehot_reset      : onehot_ty := ( 0 => '1', others => '0' ); --
0001
  constant count_reset       : count_ty  := ( 7 => '1', others => '0' ); --
1000000
```

```
  constant count_zero        : count_ty  := ( others => '0' );           --
0000000
  constant count_one         : count_ty  := ( 0 => '1', others => '0' ); --
0000001

begin

---------------- STATE UPDATE ----------------

  o_control ( lfsr_c_reset_idx ) <= lfsr_c_reset;

  lfsr_c_reset   <= '1' when reset = '1' or (count(7) = '0' and
count_next(7) = '1' ) --- reset after last ACE permutation
             else '0';

  o_control ( lfsr_c_en_idx )    <= not( permoff ) or lfsr_c_reset;
  o_onehot     <= onehot;

  o_ready      <= '1' when count(7) = '1' and state /= Tag and state_next
/= Tag
             else '0';

  o_valid      <= '1' when ( i_valid = '1' and count(7) = '1' and
                           ( state_next = pEncrypt or state_next = pDecrypt
) )
                            or
                           ( state = Tag or state_next = Tag )
             else '0';


  o_control( replace_idx ) <= '1' when state_next = pDecrypt
                         else '0';

  o_control( output_idx )  <= '1' when state_next = pEncrypt or state_next
= pDecrypt or
                                       state = Tag or state_next = Tag
                         else '0';

  permoff                    <= '0' when ( count(7) = '0' or ( count(7) = '1'
and count_next(0) = '1' ) )
                                       and i_padding = '0'
                         else '1';

  o_control( permoff_idx ) <= permoff;
  o_control( endstep_idx ) <= count(0) and count(1) and count(2);

  process begin

    wait until rising_edge(clk);
    if reset = '1' then
      state  <= Load;
      onehot <= onehot_reset;
      count  <= count_reset;
    else
      state  <= state_next;
      onehot <= onehot_next;
      count  <= count_next;
    end if;
```

```
  end process;

  ------------- NEXT STATE LOGIC ----------------

  process(state, onehot, count, i_valid, i_mode, i_dom_sep, i_padding) is
  begin

    case state is
      when Load =>
        if i_valid = '1' then
          if onehot(3) = '1' then                          -- if onehot =
1000
            state_next  <= pLoad;
            count_next  <= count_zero;                     -- count goes
from 128 to 0
            onehot_next <= onehot_reset;
          else
            state_next  <= Load;
            count_next  <= count_reset;                    -- keep count at
128
            onehot_next <= onehot_rotate( onehot );        -- rotate onehot
          end if;
        else
          count_next  <= count_reset;                      -- keep count at
128
          state_next  <= Load;
          onehot_next <= onehot;                           -- keep onehot
unchanged
        end if;

      when pLoad | iLoad2 =>
        if count < 128 and state = pLoad then
          count_next  <= count + 1;                        -- increment
count if less than 128
          state_next  <= pLoad;
          onehot_next <= onehot_reset;                     -- keep onehot
at 0001
        elsif i_valid = '1' then
          count_next <= count_one;                         -- count goes
from 128 to 1
            onehot_next <= onehot_rotate( onehot );        -- rotate onehot
from 0001 to 0010
            state_next  <= pInit;
        else
        count_next  <= count_reset;                              -- keep
count at 128
        state_next  <= iLoad2;
        onehot_next <= onehot_reset;                       -- keep onehot
at 0001
      end if;

    when pInit =>
      if count < 128 then
        count_next  <= count + 1;                          -- increment
count if less than 128
        state_next  <= pInit;
```

```
        onehot_next <= onehot;                                  -- keep onehot
unchanged
      else
        if onehot(1) = '1' then                                 -- if onehot =
0010 (i.e. K0 init is done, K1 is being sampled)
          if i_valid = '1' then
            onehot_next <= onehot_rotate( onehot );             -- rotate onehot
from 0010 to 0100
            state_next  <= pInit;
            count_next  <= count_one;                           -- count goes
from 128 to 1
          else
            onehot_next <= onehot;                              -- keep onehot
unchanged (0010)
            state_next  <= iInit;
            count_next  <= count_reset;                         -- keep count at
128
          end if;
        else                                                    -- if onehot =
0100
          onehot_next <= onehot_reset;                          -- reset onehot
from 0100 to 0001
          if i_valid = '1' then
            if i_dom_sep(0) = '0' and i_dom_sep(1) = '0' then       --
i_dom_sep = 0
              state_next <= pFinal;
              count_next <= count_one;                          -- count goes
from 128 to 1
            elsif i_dom_sep(0) = '1' and i_dom_sep(1) = '0' then    --
i_dom_sep = 1
              state_next <= pProcAD;
              count_next <= count_one;                          -- count goes
from 128 to 1
            elsif i_dom_sep(0) = '0' and i_dom_sep(1) = '1' then    --
i_dom_sep = 2
              if i_padding = '0' then
                count_next <= count_one;                        -- count goes
from 128 to 1
              else
                count_next <= count_zero;                       -- count goes
from 128 to 0 when padding is needed
              end if;
              if i_mode = '0' then
                state_next <= pEncrypt;
              else                                              -- i.e. if
i_mode = '1'
                state_next <= pDecrypt;
               end if;
            else                                                -- if i_valid =
1 but i_dom_sep has ILLEGAL value = 3
              state_next <= iProcAD;
              count_next <= count_reset;                        -- keep count at
128
            end if;
          else                                                  -- i.e. if
i_valid = '0'
            state_next <= iProcAD;
```

```
            count_next <= count_reset;                    -- keep count at
128
          end if;
        end if;
      end if;

    when iInit =>
      if i_valid = '1' then
        onehot_next <= onehot_rotate( onehot );           -- rotate onehot
from 0010 to 0100
        state_next  <= pInit;
        count_next  <= count_one;                         -- count goes
from 128 to 1
      else
        onehot_next <= onehot;                            -- keep onehot
at 0010 (staying ready for K1)
        state_next  <= iInit;
        count_next  <= count_reset;                       -- keep count at
128
      end if;

    when iProcAD | pProcAD | pEncrypt | pDecrypt =>
      if count < 128 and state /= iProcAD then
        if i_padding = '0' then
          count_next <= count + 1;                        -- increment
count if less than 128 (except if in iProcAD)
        else
          count_next <= count;                            -- stall count
when i_padding = 1
        end if;
        state_next  <= state;
        onehot_next <= onehot_reset;                      -- keep onehot
at 0001
      else
        if i_valid = '1' then
          if i_dom_sep(0) = '0' and i_dom_sep(1) = '0' then          --
i_dom_sep = 0
            onehot_next <= onehot_rotate(onehot);         -- rotate onehot
to 0010
            state_next  <= pFinal;
            count_next  <= count_one;                     -- count goes
from 128 to 1
          elsif i_dom_sep(0) = '1' and i_dom_sep(1) = '0' then       --
i_dom_sep = 1
            state_next  <= pProcAD;
            count_next  <= count_one;                     -- count goes
from 128 to 1
            onehot_next <= onehot_reset;                  -- keep onehot
at 0001
          elsif i_dom_sep(0) = '0' and i_dom_sep(1) = '1' then       --
i_dom_sep = 2
            if i_padding = '0' then
              count_next <= count_one;                    -- count goes
from 128 to 1
            else
              count_next <= count_zero;                   -- count goes
from 128 to 0 when padding is needed
            end if;
```

```
            onehot_next <= onehot_reset;                    -- keep onehot
at 0001
            if i_mode = '0' then
              state_next <= pEncrypt;
            else                                            -- i.e. if
i_mode(0) = '1'
              state_next <= pDecrypt;
            end if;
          else                                              -- if i_valid =
1 but i_dom_sep has ILLEGAL value = 3
            state_next <= iProcAD;
            count_next <= count_reset;                      -- keep count at
128
            onehot_next <= onehot;
          end if;
        else                                                -- i.e. if
i_valid = '0'
          state_next <= iProcAD;
          count_next <= count_reset;                        -- keep count at
128
          onehot_next <= onehot;
        end if;
     end if;

    when pFinal =>
      if count < 128 then
        count_next  <= count + 1;                           -- increment
count if less than 128 (except if in iProcAD)
        state_next  <= pFinal;
        onehot_next <= onehot;                              -- keep onehot
constant (0010 or 0100)
      else
         if onehot(1) = '1' then                            -- if onehot =
0010
          if i_valid = '1' then
            onehot_next <= onehot_rotate( onehot );         -- rotate onehot
from 0010 to 0100
            state_next  <= pFinal;
            count_next  <= count_one;                       -- count goes
from 128 to 1
          else
            onehot_next <= onehot;                          -- keep onehot
unchanged (at 0010)
            state_next  <= iFinal;
            count_next  <= count_reset;                     -- keep count at
128
          end if;
        else                                                -- if onehot =
0100
          state_next  <= Tag;
          onehot_next <= onehot_rotate( onehot );           -- rotate onehot
from 0100 to 1000
          count_next  <= count_reset;                       -- keep count at
128
        end if;
      end if;

    when iFinal =>
```

```
      if i_valid = '1' then
        onehot_next <= onehot_rotate( onehot );            -- rotate onehot
from 0010 to 0100
        state_next  <= pFinal;
        count_next  <= count_one;                          -- count goes
from 128 to 1
      else
        onehot_next <= onehot;                             -- keep onehot
at 0010
        state_next  <= iFinal;
        count_next  <= count_reset;                        -- keep count at
128
      end if;

    when Tag =>
        count_next  <= count_reset;                        -- keep count at
128
        onehot_next <= onehot_rotate( onehot );
        state_next  <= Load;

    when others =>
      state_next  <= Load;
      onehot_next <= onehot_reset;
      count_next  <= count_reset;

  end case;

  end process;

end architecture;
```

Το περιεχόμενο του αρχείου dp.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.spix_pkg.all;

entity dp is
  port
    ( clk       : in  std_logic
    ; reset     : in  std_logic
    ; i_mode    : in  mode_ty
    ; i_control : in  spix_ctl_ty
    ; i_onehot  : in  onehot_ty
    ; i_dom_sep : in  domsep_ty
    ; i_valid   : in  std_logic
    ; i_data    : in  word
    ; i_padding : in  std_logic
    ; o_data    : out word
    );
end entity;

architecture rtl of dp is

  signal permoff, endstep, replace, output,
         lfsr_c_reset, lfsr_c_en      : std_logic;
```

```
  signal spix_state, post_input       : half_word_state_ty;
  signal pre_round, post_round,
         post_xor, post_step_const,
         post_linear, spix_path       : word_state_ty;
  signal dsxor                        : half_word;
  signal i_data_vector, o_data_vector : half_word_data;
  signal ctl_const                    : lfsr_c_output;

Begin

     u_lfsr_c :
    entity work.lfsr_c port map
      ( clk          => clk
      , lfsr_c_reset => lfsr_c_reset
      , lfsr_c_en    => lfsr_c_en
      , o_const      => ctl_const
     );

  i_data_vector <= data_to_vector( i_data );
  o_data        <= vector_to_data( o_data_vector );

  replace       <= i_control( replace_idx );
  output        <= i_control( output_idx );
  endstep       <= i_control( endstep_idx );
  permoff       <= i_control( permoff_idx );
  lfsr_c_reset  <= i_control( lfsr_c_reset_idx );
  lfsr_c_en     <= i_control( lfsr_c_en_idx );

   ------------------------------------------------------------------------
  -- post input:  do input and domain separator and replace

  post_input( x1_1_idx ) <= i_data_vector(0) xor spix_state( x1_1_idx )
when replace = '0'  and i_valid = '1'
                     else i_data_vector(0)
when replace = '1'  and i_valid = '1'
                     else spix_state( x1_1_idx );

  post_input( x3_1_idx ) <= i_data_vector(1) xor spix_state( x3_1_idx )
when replace = '0'  and i_valid = '1'
                     else i_data_vector(1)
when replace = '1'  and i_valid = '1'
                     else spix_state( x3_1_idx );

  post_input( x0_0_idx ) <= spix_state( x0_0_idx );
  post_input( x0_1_idx ) <= spix_state( x0_1_idx );
  post_input( x1_0_idx ) <= spix_state( x1_0_idx );
  post_input( x2_0_idx ) <= spix_state( x2_0_idx );
  post_input( x2_1_idx ) <= spix_state( x2_1_idx );

  dsxor( 0 to half_word_sz - 3 ) <= ( others => '0' );
  dsxor( half_word_sz - 2)       <= i_dom_sep(1);
  dsxor( half_word_sz - 1)       <= i_dom_sep(0);

  post_input( x3_0_idx ) <= dsxor xor spix_state( x3_0_idx ) when (i_valid
= '1')
                     else spix_state( x3_0_idx );

 ------------------------------------------------------------------------
```

```
  --  !!! MDA update pic with output MUXes

  o_data_vector(0) <= i_data_vector(0) xor spix_state( x1_1_idx ) when out-
put = '1' and i_valid = '1'
                  else spix_state( x1_1_idx )                    when out-
put = '1' and i_onehot(2) = '1'  -- first tag
                  else spix_state( x3_1_idx )                    when out-
put = '1'                        -- second tag
                  else ( others => '0' );

  o_data_vector(1) <= i_data_vector(1) xor spix_state( x3_1_idx ) when out-
put = '1' and i_valid = '1'
                  else spix_state( x1_0_idx )                    when out-
put = '1' and i_onehot(2) = '1'              -- first tag
                  else spix_state( x3_0_idx)                     when out-
put = '1'                                    -- second tag
                  else ( others => '0' );

  ------------------------------------------------------------------------
  -- sb 64 ==> post round
pre_round <= half_words_to_words( post_input );

    post_round( x0_idx ) <= pre_round( x0_idx );

  a_sb_64 :
    entity work.sb_64 port map
      ( i_state => pre_round( x1_idx )
      , i_rc    => ctl_const( lfsr_c_sz + 2 )   --rc0
      , o_state => post_round( x1_idx )
     );

    post_round( x2_idx ) <= pre_round( x2_idx );

  c_sb_64 :
    entity work.sb_64 port map
      ( i_state => pre_round( x3_idx )
      , i_rc    => ctl_const( lfsr_c_sz + 1 )   --rc1
      , o_state => post_round( x3_idx )
     );

 ------------------------------------------------------------------------
 -- XORs to the left ==> post xor

  post_xor( x1_idx ) <= post_round( x1_idx );
  post_xor( x3_idx ) <= post_round( x3_idx );

  post_xor( x0_idx ) <= post_round( x0_idx ) xor post_round( x1_idx );
  post_xor( x2_idx ) <= post_round( x2_idx ) xor post_round( x3_idx );

 ------------------------------------------------------------------------
 -- XOR with step constant ==> post step const

  post_step_const( x1_idx )           <= post_xor( x1_idx );
  post_step_const( x3_idx )           <= post_xor( x3_idx );

  post_step_const( x0_idx)( 0 to 55)  <= not post_xor( x0_idx )( 0 to 55
);
```

```
  post_step_const( x0_idx)( 56 to 63) <= post_xor(     x0_idx )( 56 to 63 )
xor ctl_const( 2 to lfsr_c_sz + 2 ); -- sc0

  post_step_const( x2_idx)( 0 to 55)  <= not post_xor( x2_idx )( 0 to 55
);
  post_step_const( x2_idx)( 56 to 63) <= post_xor(     x2_idx )( 56 to 63 )
xor ctl_const( 1 to lfsr_c_sz + 1 ); -- sc1

  ------------------------------------------------------------
  -- post liear layer pi = (1,2,3,0) ==> post linear

  post_linear( x0_idx ) <= post_step_const( x1_idx );
  post_linear( x1_idx ) <= post_step_const( x2_idx );
  post_linear( x2_idx ) <= post_step_const( x3_idx );
  post_linear( x3_idx ) <= post_step_const( x0_idx );

 ------------------------------------------------------------
  -- update state
  spix_path( x0_idx ) <= post_linear( x0_idx ) when endstep = '1'  else
post_round( x0_idx );
  spix_path( x1_idx ) <= post_linear( x1_idx ) when endstep = '1'  else
post_round( x1_idx );
  spix_path( x2_idx ) <= post_linear( x2_idx ) when endstep = '1'  else
post_round( x2_idx );
  spix_path( x3_idx ) <= post_linear( x3_idx ) when endstep = '1'  else
post_round( x3_idx );

    process
    begin

    wait until rising_edge(clk);
    -- spix
    if i_mode    = '0' then
      if permoff = '0' then
        spix_state <= words_to_half_words( spix_path ); -- every clk cycle
      else
        if i_valid = '1' and i_padding = '0' then     -- load
          if i_onehot(0) = '1' then
            spix_state( x1_0_idx ) <= i_data_vector(1);  -- k0
            spix_state( x1_1_idx ) <= i_data_vector(0);
          end if;
          if i_onehot(1) = '1' then
            spix_state( x3_0_idx ) <= i_data_vector(1);  -- k1
            spix_state( x3_1_idx ) <= i_data_vector(0);
          end if;
          if i_onehot(2) = '1' then
            spix_state( x0_0_idx ) <= i_data_vector(1);  -- n0
            spix_state( x0_1_idx ) <= i_data_vector(0);
          end if;
          if i_onehot(3) = '1' then
            spix_state( x2_0_idx ) <= i_data_vector(1);  -- n1
            spix_state( x2_1_idx ) <= i_data_vector(0);
          end if;
        end if;
      end if;
    end if;
  end process;
```

```
  end architecture;
```

Το περιεχόμενο του αρχείου spix.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.spix_pkg.all;

entity spix is
  port
    ( clk       : in  std_logic;
      reset     : in  std_logic;
      i_mode    : in  mode_ty;
      i_dom_sep : in  domsep_ty;
      i_valid   : in  std_logic;
      i_data    : in  word;
      i_padding : in  std_logic;
      o_valid   : out std_logic;
      o_ready   : out std_logic;
      o_data    : out word
      );
end entity;
```

Το περιεχόμενο του αρχείου spix-rtl.vhd είναι το εξής:

```
architecture rtl of spix is
  signal ctl_control    : spix_ctl_ty;
  signal ctl_onehot     : onehot_ty;
  signal ctl_lfsr_en    : std_logic;
  signal ctl_lfsr_reset : std_logic;
begin

  u_dp :
    entity work.dp port map
      ( clk       => clk
      , reset     => reset
      , i_mode    => i_mode
      , i_control => ctl_control
      , i_onehot  => ctl_onehot
      , i_dom_sep => i_dom_sep
      , i_valid   => i_valid
      , i_data    => i_data
      , i_padding => i_padding
      , o_data    => o_data
     );

  u_ctl :
    entity work.ctl port map
      ( clk          => clk
      , reset        => reset
      , i_mode       => i_mode
      , i_dom_sep    => i_dom_sep
      , i_valid      => i_valid
      , i_padding    => i_padding
      , o_valid      => o_valid
      , o_onehot     => ctl_onehot
```

```
      , o_ready      => o_ready
      , o_control    => ctl_control
     );

end architecture;
```

Το περιεχόμενο του αρχείου util_unsynth.vhd είναι το εξής:

```
use std.textio.all;

library ieee;
use ieee.math_real.all;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package util_unsynth is

  function rev ( v : std_logic_vector ) return std_logic_vector;
  function firstn ( v : std_logic_vector; w : natural ) return
std_logic_vector;


  function to_hex_char( v : std_logic_vector ) return character;
  function hex_char_to_up_vector( chr : character ) return std_logic_vec-
tor;
  function hex_char_to_dn_vector( chr : character ) return std_logic_vec-
tor;
  function hex_string_to_up_vector( str : string ) return std_logic_vector;
  function hex_string_to_dn_vector( str : string ) return std_logic_vector;
  ------------------------------------------------------------
  procedure hread_dir
    ( ln       : inout line;
      result   : out   std_logic_vector;
      num_bits : out   natural
    );
  ------------------------------------------------------------
  function to_hex_string( v : std_logic_vector ) return string;

end package;

package body util_unsynth is

  function rev( v : std_logic_vector ) return std_logic_vector is
    variable q_up : std_logic_vector( v'low to v'high );
    variable q_dn : std_logic_vector( v'high downto v'low );
  begin
    if v'ascending then
      for i in v'range loop
        q_up(i) := v( v'high - i + v'low );
      end loop;
      return q_up;
    else
      for i in v'range loop
        q_dn(i) := v( v'high - i + v'low );
      end loop;
      return q_dn;
    end if;
```

```
end function;

-------------------------------------------------------------

function firstn ( v : std_logic_vector; w : natural )
  return std_logic_vector
is
begin
  if v'ascending then
    return v( v'low to v'low + w - 1 );
  else
    return v( v'high downto v'high - w + 1 );
  end if;
end function;

-------------------------------------------------------------

function is01( b : std_logic ) return boolean is
begin
  return b = '0' or b = '1';
end function;

-------------------------------------------------------------

function is01( b : character ) return boolean is
begin
  return b = '0' or b = '1';
end function;

-------------------------------------------------------------

function is01( v : std_logic_vector ) return boolean is
begin
  if v'length > 1 then
    if v'ascending then
      return is01( v( v'low ) ) and is01( v( v'low+1 to v'high) );
    else
      return is01( v( v'high ) ) and is01( v( v'high-1 downto v'low) );
    end if;
  else
    return is01( v( v'low ) );
  end if;
end function;

-------------------------------------------------------------

function to_hex_char( v : std_logic_vector ) return character
is
  variable i : natural;
  variable res : character;
begin
  assert v'length <= 4
    report ( "to_hex_char: length must be <= 4 "&
                     integer'image( v'length) )
    severity error;
  if is01( v ) then
    if v'ascending then
      i := to_integer( unsigned( rev(v) ) );
```

```
      else
        i := to_integer( unsigned( v ) );
      end if;
      if i < 10 then
        res := character'val( character'pos('0') + i );
      else
        res := character'val( character'pos('A') + i - 10 );
      end if;
    else
      if v = "XXXX" then
        res := 'X';
      elsif v = "UUUU" then
        res := 'U';
      else
        res := '?';
      end if;
    end if;
    if v'ascending then
      null;
      -- report( "to_hex_char: ascending  : "& to_string(v) &" --> "& res
);
    else
      null;
      -- report( "to_hex_char: descending : "& to_string(v) &" --> "& res
);
    end if;
    return res;
  end function;

  ------------------------------------------------------------------

  function to_hex_string( v : std_logic_vector ) return string is
  begin
    if v'length <= 4 then
      return to_hex_char( v ) & "";
    else
      if v'ascending then
        -- report( "to_hex_string: ascending" );
        return to_hex_char( v( v'low to v'low+3) )
               & to_hex_string( v(v'low+4 to v'high) );
      else
        -- report( "to_hex_string: descending" );
        return to_hex_char( v( v'high downto v'high-3) )
               & to_hex_string( v(v'high-4 downto v'low) );
      end if;
    end if;
  end function;

  ------------------------------------------------------------------

  function aux_hex_char_to_up_vector( chr : character ) return
std_logic_vector
  is
    variable result : std_logic_vector( 0 to 3 );
  begin

    case chr is
      when '0' => result := "0000";
```

```
      when '1' => result := "1000";
      when '2' => result := "0100";
      when '3' => result := "1100";
      when '4' => result := "0010";
      when '5' => result := "1010";
      when '6' => result := "0110";
      when '7' => result := "1110";
      when '8' => result := "0001";
      when '9' => result := "1001";
      when 'A' => result := "0101";
      when 'B' => result := "1101";
      when 'C' => result := "0011";
      when 'D' => result := "1011";
      when 'E' => result := "0111";
      when 'F' => result := "1111";

      when 'a' => result := "0101";
      when 'b' => result := "1101";
      when 'c' => result := "0011";
      when 'd' => result := "1011";
      when 'e' => result := "0111";
      when 'f' => result := "1111";
      when others => result := "XXXX";
    end case;

    return result;

  end function;

  ------------------------------------------------------------------

  function hex_char_to_up_vector( chr : character ) return std_logic_vector
  is
    variable result : std_logic_vector( 0 to 3 );
  begin

    result := aux_hex_char_to_up_vector( chr );

    assert is01( result )
      report
        "hex_char_to_up_vector: error: read '" & chr &
          "', expected a hex character (0-F)."
      severity error;

    -- report( "hex_char_to_up_vector: " & chr &" -->"& to_string( result )
);

    return result;

  end function;

  ------------------------------------------------------------------

  function hex_char_to_dn_vector( chr : character ) return std_logic_vector
  is
    variable result : std_logic_vector( 3 downto 0 );
  begin
```

```
    result := rev(aux_hex_char_to_up_vector( chr ));

    assert is01( result )
      report
        "hex_char_to_dn_vector: error: read '" & chr &
          "', expected a hex character (0-F)."
      severity error;

    -- report( "hex_char_to_dn_vector: " & chr &" -->"& to_string( result )
);

    return result;

  end function;

  ------------------------------------------------------------------

  function hex_string_to_up_vector( str : string ) return std_logic_vector
  is
    variable result  : std_logic_vector( 0 to str'length*4 - 1);
    variable j       : integer;
  begin
    if str'ascending then
      for i in str'low to str'high loop
        j := i - str'low;
        result( j*4 to (j+1)*4-1 ) := hex_char_to_up_vector( str(i) );
      end loop;
    else
      for i in str'left downto str'right loop
        j := str'high - i;
        result( j*4 to (j+1)*4-1 ) := hex_char_to_up_vector( str(i) );
      end loop;
    end if;
    -- report( "hex_string_to_up_vector: "& str &" --> "& to_string( result
) );
    return result;
  end function;

  ------------------------------------------------------------------

  function hex_string_to_dn_vector( str : string ) return std_logic_vector
  is
    variable result  : std_logic_vector( str'length*4 - 1 downto 0);
    variable j       : integer;
  begin
    if str'ascending then
      for i in str'low to str'high loop
        j := str'high - i;
        result( (j+1)*4-1 downto j*4 ) := hex_char_to_dn_vector( str(i) );
      end loop;
    else
      for i in str'left downto str'right loop
        j := i - str'low;
        result( (j+1)*4-1 downto j*4 ) := hex_char_to_dn_vector( str(i) );
      end loop;
    end if;
    -- report( "hex_string_to_dn_vector: "& str &" --> "& to_string( result
) );
```

```
  return result;
end function;

------------------------------------------------------------------

procedure char_to_vector
  ( c      : in  character;
    result : out std_logic_vector;
    good   : out boolean
  )
is
  variable dn_tmp : std_logic_vector(3 downto 0);
  variable up_tmp : std_logic_vector(0 to     3);
begin

  case c is
    when '0' => dn_tmp :=  x"0"; good := true;
    when '1' => dn_tmp :=  x"1"; good := true;
    when '2' => dn_tmp :=  x"2"; good := true;
    when '3' => dn_tmp :=  x"3"; good := true;
    when '4' => dn_tmp :=  x"4"; good := true;
    when '5' => dn_tmp :=  x"5"; good := true;
    when '6' => dn_tmp :=  x"6"; good := true;
    when '7' => dn_tmp :=  x"7"; good := true;
    when '8' => dn_tmp :=  x"8"; good := true;
    when '9' => dn_tmp :=  x"9"; good := true;
    when 'A' => dn_tmp :=  x"A"; good := true;
    when 'B' => dn_tmp :=  x"B"; good := true;
    when 'C' => dn_tmp :=  x"C"; good := true;
    when 'D' => dn_tmp :=  x"D"; good := true;
    when 'E' => dn_tmp :=  x"E"; good := true;
    when 'F' => dn_tmp :=  x"F"; good := true;

    when 'a' => dn_tmp :=  x"A"; good := true;
    when 'b' => dn_tmp :=  x"B"; good := true;
    when 'c' => dn_tmp :=  x"C"; good := true;
    when 'd' => dn_tmp :=  x"D"; good := true;
    when 'e' => dn_tmp :=  x"E"; good := true;
    when 'f' => dn_tmp :=  x"F"; good := true;
    when others =>
      assert false
        report
          "char_to_vector: error: read '" & c &
             "', expected a hex character (0-F).";
      good := false;
  end case;

  if result'ascending then
    up_tmp := rev( dn_tmp );
    -- report( "char_to_vector: " & c &" -->"& to_string( up_tmp ) );
    result := up_tmp;
  else
    -- report( "char_to_vector: " & c &" -->"& to_string( dn_tmp ) );
    result := dn_tmp;
  end if;

end procedure;
```

```
------------------------------------------------------------

procedure hex_string_to_vector
  ( str    : in string;
    result : out std_logic_vector
  )
is
  constant str_len : natural := str'length;
  constant vec_len : natural := result'length/4;
begin

  assert str_len <= vec_len
    report ("hex_string_to_vector: string longer than vector/4: "&
             integer'image( str_len )
            &" > "& integer'image(vec_len ) )
    severity warning;
  assert vec_len >= str_len
    report ("hex_string_to_vector: vector/4 longer than string : "&
              integer'image(vec_len)
            &" > "&integer'image( str_len )
            )
    severity warning;
  if result'ascending then
    result := hex_string_to_up_vector( str );
  else
    result := hex_string_to_dn_vector( str );
  end if;
end procedure;

------------------------------------------------------------

procedure hread_dir
  ( ln       : inout line;
    result   : out   std_logic_vector;
    num_bits : out   natural
  )
is
  variable ok           : boolean;
  variable chr          : character;
  variable str          : string(1 to result'length/4);
  variable tmp_result   : std_logic_vector(0 to result'length-1);
  variable num_chars,
           tmp_num_bits : natural := 0;
begin

  if result'length mod 4 /= 0 then
    assert false
      report
        "hread_dir error: trying to read vector " &
          "with non multiple of 4 length";
    return;
  end if;

  loop                                        -- skip white space
    read(ln, chr, ok);
    exit when not ok or ((chr /= ' ') and (chr /= CR) and (chr /= HT));
  end loop;
  if not ok then
```

```
      assert false
        report ("hread_dir error: failed" )
        severity error;
      return;
    else
     -- report ("hread_dir: char0="& chr );
    end if;

    num_chars        := 1;
    str( num_chars ) := chr;

    while ok loop
      read(ln, chr, ok);
      if ok then
        num_chars        := num_chars + 1;
        str( num_chars ) := chr;
      end if;
    end loop;

    tmp_num_bits := num_chars * 4;
    tmp_result   := (others => 'U');
    if result'ascending then
      tmp_result( 0 to tmp_num_bits - 1) :=
        hex_string_to_up_vector( str( 1 to num_chars ) );
    else
      tmp_result( 0 to tmp_num_bits - 1) :=
        hex_string_to_dn_vector( str( 1 to num_chars ) );
    end if;

    -- report( "hread_dir:  " & str &" -->"&
    --         to_string( tmp_result( 0 to tmp_num_bits - 1 ) ) );

    result   := tmp_result;
    num_bits := tmp_num_bits;

  end procedure;

  ------------------------------------------------------------

end package body;
```

Το περιεχόμενο του αρχείου spix_unsynth.vhd είναι το εξής:

```
use std.textio.all;

library ieee;
use ieee.math_real.all;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_textio.all;

use work.util_unsynth.all;
use work.spix_pkg.all;

package spix_unsynth is

  constant max_data_sz     : integer := 256;
```

```
  -- ****************************************** --
  type tag_ty is ( KEY_TAG, NONCE_TAG, INPUT_TAG, OUTPUT_TAG, AD_TAG,
PLAINTEXT_TAG, CIPHERTEXT_TAG, EOF_TAG, NULL_TAG );

--  function to_hex_char_normal( v : std_logic_vector ) return character;
  function to_hex_string_normal( v : std_logic_vector ) return string;
  -- ****************************************** --
  function is_tag( c : character ) return boolean;
  function chr_to_tag( chr : character ) return tag_ty;
  -- ****************************************** --
  procedure check_tag
    ( spec, impl : tag_ty );

  procedure read_next_tag_data_line_normal
    ( file f    : text;
      tag       : out tag_ty;
      data      : out std_logic_vector;
      num_bits  : out natural );
  procedure hread_dir_normal
    ( ln        : inout line;
      result    : out   std_logic_vector;
      num_bits  : out   natural
    );

  function pad ( v : std_logic_vector ) return std_logic_vector;

  procedure drive_reset
    ( signal   clk   : in std_logic
    ; signal   reset : out std_logic
    ; constant hold  : in time
    );

  procedure drive_data
    ( signal   clk     : in std_logic
    ; variable data    : in std_logic_vector
    ; constant hold    : in time
    ; signal   i_data  : out std_logic_vector
    ; signal   i_valid : out std_logic
    );

  procedure drive_all
    ( constant target_tag : in tag_ty
    ; stim_file_path      : in string
    ; constant hold       : in time
    ; constant delay      : in time
    ; constant manual_pad : in boolean
    ; signal clk          : in std_logic
    ; signal o_ready      : in std_logic
    ; signal o_data       : in std_logic_vector
    ; signal i_data       : out std_logic_vector
    ; signal i_valid      : out std_logic
    ; signal i_padding    : out std_logic
    ; signal bits_pad     : out natural
    );

  procedure read_key_nonce
    ( key_stim       : out std_logic_vector
    ; nonce_stim     : out std_logic_vector
```

```
    ; stim_file_path : in string
    );

end package;


package body spix_unsynth is

  procedure drive_data
    ( signal   clk      : in std_logic
    ; variable data     : in std_logic_vector
    ; constant hold     : in time
    ; signal   i_data   : out std_logic_vector
    ; signal   i_valid  : out std_logic
    )
  is
  begin

    i_data                <= data;
    i_valid                <= '1';
    wait until rising_edge(clk);
    wait for hold;
    i_valid                <= '0';

  end procedure;

  procedure drive_all
    ( constant target_tag : in tag_ty
    ; stim_file_path      : in string
    ; constant hold       : in time
    ; constant delay      : in time
    ; constant manual_pad : in boolean
    ; signal clk          : in std_logic
    ; signal o_ready      : in std_logic
    ; signal o_data       : in std_logic_vector
    ; signal i_data       : out std_logic_vector
    ; signal i_valid      : out std_logic
    ; signal i_padding    : out std_logic
    ; signal bits_pad     : out natural
    )
  is
    variable tag : tag_ty;
    variable num_bits, num_bits_pad : natural;
    variable data, data_buf : std_logic_vector(0 to word_sz - 1);
    variable data128 : std_logic_vector(0 to 127);
    file     stim_file : text;
    variable break : boolean;
  begin

    break := false;
    i_padding <= '0';
    num_bits_pad := 64;
    bits_pad <= 64;

    file_close(stim_file);
    file_open(stim_file, stim_file_path, read_mode);
```

```
    tag := NULL_TAG;
    while tag /= target_tag and tag /= EOF_TAG loop
      read_next_tag_data_line_normal( stim_file, tag, data128, num_bits );
    end loop;

    while tag /= EOF_TAG and not(break) loop

      report( "Data length = " & natural'image(num_bits) & " bits");

      data := data128(0 to word_sz-1);

      if data(63) /= '1' and data(63) /= '0' and manual_pad = true then
        break := true;
      end if;


      data := pad(data);
      num_bits_pad := num_bits;

      wait for delay;

      if num_bits > 64 then
        bits_pad <= 64;
      else
        bits_pad <= num_bits;
      end if;

      if break then
        i_data <= data;                          -- ciphertext goes here
        i_valid                <= '1';
        i_padding <= '1';                        ---- handle the padding
        wait until rising_edge(clk);
        data_buf := o_data;                       ---- handle the padding
        wait for hold;
        i_valid                <= '0';
      else
        drive_data( clk, data, hold, i_data, i_valid );
        wait until o_ready = '1' and rising_edge(clk);
      end if;

      if not(break) then

        data := data128(word_sz to key_sz - 1);

        if num_bits > 64 then    -- or if num_bits > 64, i.e. second 64-
bits exist

          if data(63) /= '1' and data(63) /= '0' and manual_pad = true then
            break := true;
            num_bits_pad := num_bits;
          end if;

          data := pad(data);
          wait for delay;

          bits_pad <= num_bits - 64;

          if break then
```

```
            i_data <= data;                          -- ciphertext goes here
            i_valid                <= '1';
            i_padding <= '1';                        ---- handle the padding
            wait until rising_edge(clk);
            data_buf := o_data;                       ---- handle the pad-
ding
            wait for hold;
            i_valid                <= '0';
          else
            drive_data( clk, data, hold, i_data, i_valid );
            wait until o_ready = '1' and rising_edge(clk);
          end if;

        end if;

      end if;

      tag := NULL_TAG;
      while tag /= target_tag and tag /= EOF_TAG loop
        read_next_tag_data_line_normal( stim_file, tag, data128, num_bits
);
      end loop;

    end loop;


    if break then
      if num_bits_pad > 64 then num_bits_pad := num_bits_pad - 64; end if;

      report ("last ciphertext block is " & natural'image(num_bits_pad) & "
bits + padding is " & natural'image(64 - num_bits_pad) & " bits");
      report ("I_DATA (ciphertext) is: " & to_hex_string_normal( data ) );
      report ("O_DATA BUF (plaintext) is: " & to_hex_string_normal(
data_buf ) );

      data := data(0 to num_bits_pad-1) & data_buf(num_bits_pad to 63);
      report ("PADDED I_DATA (for replace) is: " & to_hex_string_normal(
data ) );
      wait for 20*hold;
      wait until rising_edge(clk);
      wait for hold;

      i_data     <= data;         --- ciphertext with "replaced" tag goes
here
      i_valid    <= '1';
      i_padding  <= '0';
      wait until rising_edge(clk);
      wait for hold;
      i_valid        <= '0';

      wait until o_ready = '1' and rising_edge(clk);
    end if;

    break := false;
    file_close(stim_file);

  end procedure;
```

```
------------------------------------------------


  procedure drive_reset
    ( signal   clk   : in std_logic
    ; signal   reset : out std_logic
    ; constant hold  : in time
    )
  is
  begin

        wait until rising_edge(clk);
        wait for hold;
        reset <= '1';
        wait until rising_edge(clk);
    wait for hold;
        reset <= '0';

  end procedure;

------------------------------------------------

  procedure read_key_nonce
    ( key_stim          : out std_logic_vector
    ; nonce_stim        : out std_logic_vector
    ; stim_file_path    : in string
    )
  is
    variable tag       : tag_ty;
    file     stim_file : text;
    variable num_bits  : natural;
  begin

    file_close(stim_file);
    file_open(stim_file, stim_file_path, read_mode);

        tag := NULL_TAG;
        while tag /= KEY_TAG and tag /= EOF_TAG loop
          read_next_tag_data_line_normal( stim_file, tag, key_stim,
num_bits );
        end loop;

    if tag = KEY_TAG then
      if num_bits /= 128 then
        file_close(stim_file);
        assert false
        report( "Key length = " & natural'image(num_bits) & " bits, must be
128 bits")
        severity failure;
      else
        report( "Key length = " & natural'image(num_bits) & " bits");
      end if;
    else
      file_close(stim_file);
      assert false
      report ("Key isn't found in file " & stim_file_path)
      severity failure;
    end if;
```

```
        file_close(stim_file);
        file_open(stim_file, stim_file_path, read_mode);

        tag := NULL_TAG;
        while tag /= NONCE_TAG and tag /= EOF_TAG loop
          read_next_tag_data_line_normal( stim_file, tag, nonce_stim,
num_bits );
        end loop;

    if tag = NONCE_TAG then
      if num_bits /= 128 then
        file_close(stim_file);
        assert false
        report( "Nonce length = " & natural'image(num_bits) & " bits, must
be 128 bits")
        severity failure;
      else
        report( "Nonce length = " & natural'image(num_bits) & " bits");
      end if;
    else
      file_close(stim_file);
      assert false
      report ("Nonce isn't found in file " & stim_file_path)
      severity failure;
    end if;

    file_close(stim_file);

  end procedure;

-----------------------------------------------------

  function pad ( v : std_logic_vector ) return std_logic_vector is
    variable z : std_logic_vector(v'low to v'high);
  begin
    for i in v'high downto v'low + 1 loop
      if v(i) /= '1' and v(i) /= '0' then
        if v(i-1) = '1' or v(i-1) = '0' then
          z(i) := '1';
        else
          z(i) := '0';
        end if;
      else
        z(i) := v(i);
      end if;
    end loop;
    z(0) := v(0);
    return(z);
  end function;

  function to_hex_string_normal( v : std_logic_vector ) return string is
  begin
    if v'length <= 4 then
      return to_hex_char( rev ( v ) ) & "";
--<----- added rev (change 1/3)
    else
      if v'ascending then
```

```
        -- report( "to_hex_string: ascending" );
        return to_hex_char( rev (v( v'low to v'low+3)) )                    --
<----- added rev (change 2/3)
               & to_hex_string_normal( v(v'low+4 to v'high) );
      else
        -- report( "to_hex_string: descending" );
        return to_hex_char( rev (v( v'high downto v'high-3)) )
--<----- added rev (change 3/3)
               & to_hex_string_normal( v(v'high-4 downto v'low) );
      end if;
    end if;
  end function;

  function is_tag( c : character ) return boolean is
  begin
    return c = 'K' or c = 'N' or c = 'I' or c = 'O' or c = 'A' or c = 'P'
or c = 'C';
  end function;
  -- ***************************************** --
  -- ***************************************** --
  function chr_to_tag( chr : character ) return tag_ty is
  begin
    case chr is
      when 'K' => return KEY_TAG;
      when 'N' => return NONCE_TAG;
      when 'I' => return INPUT_TAG;
      when 'O' => return OUTPUT_TAG;
      when 'A' => return AD_TAG;
      when 'P' => return PLAINTEXT_TAG;
      when 'C' => return CIPHERTEXT_TAG;
      when others =>
        assert false
          report ("chr_to_tag: not a tag: "& character'image( chr ))
          severity error;
        return EOF_TAG;
    end case;
  end function;
  -- ***************************************** --
  -- ***************************************** --
  procedure check_tag
   ( spec, impl : tag_ty )
  is

  begin
    if spec /= impl then
      assert false
      report( "check_tag: expected tag="& tag_ty'image(spec)
              &", but found tag="& tag_ty'image(impl) )
      severity warning; --error
    else
      report( "else from check_tag" );
    end if;
  end procedure;
  -- ***************************************** --
  procedure read_next_tag_data_line_normal
    ( file f    : text;
      tag       : out tag_ty;
      data      : out std_logic_vector;
```

```
      num_bits : out natural )
  is
    variable found
           , ok             : boolean;
    variable input_line     : line;
    variable chr            : character;
    variable tmp_tag        : tag_ty;
    variable tmp_data       : std_logic_vector( data'range );
    variable tmp_num_bits   : natural;


  begin
    -- read lines until hit one where first non-blank character is a tag
    found := false;
    while not found and not endfile( f ) loop
      readline( f, input_line );
      read( input_line, chr, good => ok );
      -- skip blanks and comments
      found := ok and is_tag(chr);
    end loop;
    if not found then
      tmp_tag  := EOF_TAG;
      tmp_data := (others => 'U');
    else
      tmp_tag  := chr_to_tag( chr );
      hread_dir_normal( input_line, tmp_data, tmp_num_bits );     -- <---
added _normal (change 1/1)
      -- hread( input_line, tmp_data );
    end if;
    ------------------------------------------------------------------------
----------------
    tag      := tmp_tag;
    data     := tmp_data;
    num_bits := tmp_num_bits;
  end procedure;
  -- ***************************************** --
  -- ***************************************** --
 procedure hread_dir_normal
    ( ln       : inout line;
      result   : out   std_logic_vector;
      num_bits : out   natural
    )
  is
    variable ok           : boolean;
    variable chr          : character;
    variable str          : string(1 to result'length/4);
    variable tmp_result   : std_logic_vector(0 to result'length-1);
    variable num_chars,
             tmp_num_bits : natural := 0;
  begin

    if result'length mod 4 /= 0 then
      assert false
        report
          "hread_dir error: trying to read vector" &
            "with non multiple of 4 length";
      return;
    end if;
```

```
    loop                                        -- skip white space
      read(ln, chr, ok);
      exit when not ok or ((chr /= ' ') and (chr /= CR) and (chr /= HT));
    end loop;
    if not ok then
      assert false
        report ("hread_dir error: failed" )
        severity error;
      return;
    else
     -- report ("hread_dir: char0="& chr );
    end if;

    num_chars        := 1;
    str( num_chars ) := chr;

    while ok loop
      read(ln, chr, ok);
      if ok then
        num_chars        := num_chars + 1;
        str( num_chars ) := chr;
      end if;
    end loop;

    tmp_num_bits := num_chars * 4;
    tmp_result   := (others => 'U');
    if result'ascending then
      tmp_result( 0 to tmp_num_bits - 1) :=
        hex_string_to_dn_vector( str( 1 to num_chars ) );    -- <--- changed
from up to dn (change 1/2)
                                                             -- HEX(0)
should be treated MSB, HEX(3) should be treated as LSB
                                                             -- for correct
interpretation of HEX
    else
      tmp_result( 0 to tmp_num_bits - 1) :=
        hex_string_to_up_vector( str( 1 to num_chars ) );    -- <--- changed
from up to dn (change 2/2)
                                                             -- HEX(0)
should be treated MSB, HEX(3) should be treated as LSB
                                                             -- for correct
interpretation of HEX
    end if;

    -- report( "hread_dir:  " & str &" -->"&
    --         to_string( tmp_result( 0 to tmp_num_bits - 1 ) ) );

    result   := tmp_result;
    num_bits := tmp_num_bits;
  end procedure;

end package body;
```

Το περιεχόμενο του αρχείου spix_tb.vhd είναι το εξής:

```
use std.textio.all;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;
use ieee.numeric_std.all;

use work.util_unsynth.all;
use work.spix_unsynth.all;
use work.spix_pkg.all;

entity spix_tb is
  generic (
        period            : real    := 10.0
  );
end entity;

architecture main of spix_tb is

  constant EDH          : std_logic_vector(0 to 1) := "11"; -- ENCRYPTION /
DECRYPTION: '0' = don't do, '1' = do
  constant stim_file_path   : string :=
"/home/ise/vmsharedfolder/spix/stimulus.txt";
  constant output_file_path : string := "/home/ise/vmsharedfolder/spix/out-
put.txt";



 --constant period : real (error)

  constant clk_period   : time := period * 1 ns;
  constant hold         : time := clk_period / 2;
  constant after_reset  : time := 0 * clk_period;        -- delay between
reset and loading
  constant load_delay   : time := 0 * clk_period;        -- delay between
K, N data
  constant init_delay   : time := 0 * clk_period;        -- delay between
load permutation and initialization
  constant procad_delay : time := 0 * clk_period;        -- AD data delay
  constant enc_delay    : time := 0 * clk_period;        -- encryption data
delay
  constant dec_delay    : time := 0 * clk_period;        -- decryption data
delay
  constant absorb_delay : time := 0 * clk_period;        -- delay between
absorb data
  constant mode_delay   : time := 0 * clk_period;        -- delay between
two modes

  signal clk            : std_logic := '0';
  signal reset          : std_logic := '0';

  signal i_mode         : mode_ty;

  signal i_dom_sep      : domsep_ty;
  signal i_valid        : std_logic := '0';
  signal i_data         : word;
```

```
  signal o_valid,
         o_ready,
         i_padding      : std_logic;
  signal o_data         : word;

  file output_file      : text open write_mode is output_file_path;

  type tb_state_ty is (tbEncrypt, tbDecrypt, tbTag, tbHash, tbNull);
  signal tb_state : tb_state_ty;

  signal bits_pad       : natural;

begin

  uut : entity work.spix port map
      ( clk       => clk
      , reset     => reset
      , i_mode    => i_mode
      , i_dom_sep => i_dom_sep
      , i_valid   => i_valid
      , i_data    => i_data
      , i_padding => i_padding
      , o_valid   => o_valid
      , o_ready   => o_ready
      , o_data    => o_data
      );

  clk <= not clk after clk_period / 2;

  reading_proc : process
    variable msg  : line;
    variable done : boolean;
  begin

    done := false;
    while not(done) loop
      wait until rising_edge(clk);
        done := (o_valid = '1');
      end loop;

      if tb_state = tbEncrypt then
        write(  msg, "Ciphertext: " & to_hex_string_normal( o_data( 0 to
bits_pad-1 ) ) );
      elsif tb_state = tbDecrypt then
        write(  msg, "Plaintext:  " & to_hex_string_normal( o_data( 0 to
bits_pad-1 ) ) );
      elsif tb_state = tbTag then
        write(  msg, "Tag:            " & to_hex_string_normal( o_data ) );
      elsif tb_state = tbHash then
        write(  msg, "Hash:       " & to_hex_string_normal( o_data ) );
      else
        write(  msg, "N/A:      " & to_hex_string_normal( o_data ) );
      end if;

    writeline( output_file, msg);

  end process;
```

```
  stimulus_proc : process
    variable num_bits   : natural;
    variable k0,k1,n0,
             n1,ad,m,
             m_next,iv  : std_logic_vector(0 to word_sz -1);
    variable key_stim   : std_logic_vector(0 to key_sz -1);
    variable nonce_stim : std_logic_vector(0 to nonce_sz -1);
    variable done       : boolean;
    variable i, j       : natural;
    variable data_buf   : word;
    variable data128    : std_logic_vector(0 to 127);
    variable tag        : tag_ty;
  begin

    ------------------------ SIGNAL DEFAULTS --------------------------
    i_padding <= '0';
    i_data    <= (others => 'X');
    i_dom_sep <= (others => 'X');
    tb_state  <= tbNull;

    if EDH(0) = '1' then

      ------------------------------------------------------------------------
      ---------------------------------- Encryption ----------------------
      ------------------------------------------------------------------------

      -------------------------------- Reset -----------------------------

      wait for clk_period;
      drive_reset( clk, reset, hold );
      wait until rising_edge(clk);

      -------------------------------- Load ------------------------------
-

      report( "LOADING PHASE" );
      read_key_nonce( key_stim, nonce_stim, stim_file_path );

      k0 := key_stim(0 to word_sz - 1);
      k1 := key_stim(word_sz to key_sz - 1);
      n0 := nonce_stim(0 to word_sz - 1);
      n1 := nonce_stim(word_sz to key_sz - 1);

      i_mode <= encrypt_mode;


      wait for hold;
      drive_data( clk, k0, hold, i_data, i_valid );    wait for load_delay;
      drive_data( clk, k1, hold, i_data, i_valid );    wait for load_delay;
      drive_data( clk, n0, hold, i_data, i_valid );    wait for load_delay;
      drive_data( clk, n1, hold, i_data, i_valid );

      ------------------------------- LoadPerm --------------------------
---
      report( "PERMUTATION AFTER LOAD" );
      wait until o_ready = '1' and rising_edge(clk);
      wait for init_delay;
```

```
      ---------------------------------- Init ----------------------------
---
      report( "INITIALIZATION PHASE" );

      i_dom_sep <= "00";

      drive_data( clk, k0, hold, i_data, i_valid );
      wait until o_ready = '1' and rising_edge(clk);
      drive_data( clk, k1, hold, i_data, i_valid );
      wait until o_ready = '1' and rising_edge(clk);

      ---------------------------------- ProcAD --------------------------
-----
      report( "AD PROCESSING PHASE" );

      i_dom_sep <= "01";
      drive_all(AD_TAG, stim_file_path, hold, procad_delay, false, clk,
o_ready, o_data, i_data, i_valid, i_padding, bits_pad);

      ---------------------------------- Encrypt -------------------------
----
      report( "ENCRYPTION PHASE" );

      tb_state <= tbEncrypt;

      i_dom_sep <= "10";
      drive_all(PLAINTEXT_TAG, stim_file_path, hold, enc_delay, false, clk,
o_ready, o_data, i_data, i_valid, i_padding, bits_pad);

      ---------------------------------- Final ---------------------------
----
      report( "FINALIZATION PHASE" );
      i_dom_sep <= "00";

      drive_data( clk, k0, hold, i_data, i_valid );
      wait until o_ready = '1' and rising_edge(clk);
      drive_data( clk, k1, hold, i_data, i_valid );

      ---------------------------------- Tag -----------------------------
----
      report( "TAG PHASE" );
      tb_state <= tbTag;
      wait until o_ready = '1' and rising_edge(clk);

    end if;

      ------------------------------------------------------------------------
      ------------------------------------------------------------------------
      ------------------------------------------------------------------------
      ---------------------------------- Decryption ----------------------
      ------------------------------------------------------------------------
      ------------------------------------------------------------------------

    if EDH(1) = '1' then
      if EDH(0) = '1' then wait for mode_delay; end if;
      ------------------------- SIGNAL DEFAULTS --------------------------
-
```

```
      i_data <= (others => 'X');
      i_dom_sep   <= (others => 'X');

      ------------------------ READING STIM FILES  -----------------------
-

      ------------------------------- Reset ------------------------------
-

      wait for clk_period;
      drive_reset( clk, reset, hold );
      wait until rising_edge(clk);

      ------------------------------- Load -------------------------------
-

    report( "LOADING PHASE" );
    read_key_nonce( key_stim, nonce_stim, stim_file_path );

    k0 := key_stim(0 to word_sz - 1);
    k1 := key_stim(word_sz to key_sz - 1);
    n0 := nonce_stim(0 to word_sz - 1);
    n1 := nonce_stim(word_sz to key_sz - 1);

    i_mode <= decrypt_mode;

    wait for hold;
    drive_data( clk, k0, hold, i_data, i_valid );      wait for load_delay;
    drive_data( clk, k1, hold, i_data, i_valid );      wait for load_delay;
    drive_data( clk, n0, hold, i_data, i_valid );      wait for load_delay;
    drive_data( clk, n1, hold, i_data, i_valid );


   ------------------------------ LoadPerm ------------------------------
    report( "LOADING PERMUTATION" );
    wait until o_ready = '1' and rising_edge(clk);
    wait for init_delay;

    ------------------------------- Init -------------------------------
-

    report( "INITIALIZATION PHASE" );

    i_dom_sep <= "00";

    drive_data( clk, k0, hold, i_data, i_valid );
    wait until o_ready = '1' and rising_edge(clk);
    drive_data( clk, k1, hold, i_data, i_valid );
    wait until o_ready = '1' and rising_edge(clk);

   ------------------------------- ProcAD -------------------------------
--

    report( "AD PROCESSING PHASE" );

    i_dom_sep <= "01";
```

```
    drive_all(AD_TAG, stim_file_path, hold, procad_delay, false, clk,
o_ready, o_data, i_data, i_valid, i_padding, bits_pad);


    ---------------------------------- Decrypt ----------------------------
--

    report( "DECRYPTION PHASE" );

    tb_state <= tbDecrypt;

    i_dom_sep <= "10";
    drive_all(CIPHERTEXT_TAG, stim_file_path, hold, enc_delay, true, clk,
o_ready, o_data, i_data, i_valid, i_padding, bits_pad);

    ---------------------------------- Final ----------------------------
--

    report( "FINALIZATION PHASE" );
    i_dom_sep <= "00";

    drive_data( clk, k0, hold, i_data, i_valid );
    wait until o_ready = '1' and rising_edge(clk);
    drive_data( clk, k1, hold, i_data, i_valid );

    ---------------------------------- Tag ----------------------------
--

    report( "TAG PHASE" );
    tb_state <= tbTag;
    wait until o_ready = '1' and rising_edge(clk);



  end if;


    wait until o_ready = '1' and rising_edge(clk);

    assert false
    report ("SIMULATION IS FINISHED")
    severity failure;


  end process;

end architecture main;
```

Αλγόριθμος GIFT-COFB

Το περιεχόμενο του αρχείου gift128.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package gift128_pkg is
    subtype state   is std_logic_vector(0 to 127); -- state size 128 bits
    subtype segment is std_logic_vector(0 to 31 ); -- state segment 32 bits
    subtype keyword is std_logic_vector(0 to 15 ); -- keyword 16bits
    subtype word    is std_logic_vector(0 to 63 ); -- 64bit word
    subtype xy_1    is std_logic_vector(0 to 7  ); -- 8 bit for round con-
stants
    function pad128(x: std_logic_vector) return std_logic_vector;
    subtype typeofblock is std_logic_vector(0 to 1);
end package;

package body gift128_pkg is

    function pad128(x: std_logic_vector) return std_logic_vector is
        constant ZERO : std_logic_vector(0 to 127) := (others => '0');
    begin
        if (x'length < 128) then
            return ZERO(x'length to 127) & x;
        else
            return x(0 to 127);
        end if;
    end function;

end package body;
```

Το περιεχόμενο του αρχείου initialization.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity initialization is
    port
        (in_state,in_keystate   : in  state --128 bit input
        ;out_state,out_keystate : out state --128 bit outputs
        );
end entity;

architecture rtl of initialization is

begin
--------------------FIRST_SEGMENT_SEG0---------------------------
    out_state(0  )  <= in_state(124);
    out_state(1  )  <= in_state(120);
    out_state(2  )  <= in_state(116);
    out_state(3  )  <= in_state(112);
    out_state(4  )  <= in_state(108);
    out_state(5  )  <= in_state(104);
```

```
    out_state(6  )  <= in_state(100);
    out_state(7  )  <= in_state(96 );
    out_state(8  )  <= in_state(92 );
    out_state(9  )  <= in_state(88 );
    out_state(10 )  <= in_state(84 );
    out_state(11 )  <= in_state(80 );
    out_state(12 )  <= in_state(76 );
    out_state(13 )  <= in_state(72 );
    out_state(14 )  <= in_state(68 );
    out_state(15 )  <= in_state(64 );
    out_state(16 )  <= in_state(60 );
    out_state(17 )  <= in_state(56 );
    out_state(18 )  <= in_state(52 );
    out_state(19 )  <= in_state(48 );
    out_state(20 )  <= in_state(44 );
    out_state(21 )  <= in_state(40 );
    out_state(22 )  <= in_state(36 );
    out_state(23 )  <= in_state(32 );
    out_state(24 )  <= in_state(28 );
    out_state(25 )  <= in_state(24 );
    out_state(26 )  <= in_state(20 );
    out_state(27 )  <= in_state(16 );
    out_state(28 )  <= in_state(12 );
    out_state(29 )  <= in_state(8  );
    out_state(30 )  <= in_state(4  );
    out_state(31 )  <= in_state(0  );
--------------------SECOND_SEGMENT_SEG1----------------------------
    out_state(32 )  <= in_state(125);
    out_state(33 )  <= in_state(121);
    out_state(34 )  <= in_state(117);
    out_state(35 )  <= in_state(113);
    out_state(36 )  <= in_state(109);
    out_state(37 )  <= in_state(105);
    out_state(38 )  <= in_state(101);
    out_state(39 )  <= in_state(97 );
    out_state(40 )  <= in_state(93 );
    out_state(41 )  <= in_state(89 );
    out_state(42 )  <= in_state(85 );
    out_state(43 )  <= in_state(81 );
    out_state(44 )  <= in_state(77 );
    out_state(45 )  <= in_state(73 );
    out_state(46 )  <= in_state(69 );
    out_state(47 )  <= in_state(65 );
    out_state(48 )  <= in_state(61 );
    out_state(49 )  <= in_state(57 );
    out_state(50 )  <= in_state(53 );
    out_state(51 )  <= in_state(49 );
    out_state(52 )  <= in_state(45 );
    out_state(53 )  <= in_state(41 );
    out_state(54 )  <= in_state(37 );
    out_state(55 )  <= in_state(33 );
    out_state(56 )  <= in_state(29 );
    out_state(57 )  <= in_state(25 );
    out_state(58 )  <= in_state(21 );
    out_state(59 )  <= in_state(17 );
    out_state(60 )  <= in_state(13 );
    out_state(61 )  <= in_state(9  );
    out_state(62 )  <= in_state(5  );
```

```
out_state(63 )  <= in_state(1  );
--------------------THIRD_SEGMENT_SEG2---------------------------
    out_state(64 )  <= in_state(126);
    out_state(65 )  <= in_state(122);
    out_state(66 )  <= in_state(118);
    out_state(67 )  <= in_state(114);
    out_state(68 )  <= in_state(110);
    out_state(69 )  <= in_state(106);
    out_state(70 )  <= in_state(102);
    out_state(71 )  <= in_state(98 );
    out_state(72 )  <= in_state(94 );
    out_state(73 )  <= in_state(90 );
    out_state(74 )  <= in_state(86 );
    out_state(75 )  <= in_state(82 );
    out_state(76 )  <= in_state(78 );
    out_state(77 )  <= in_state(74 );
    out_state(78 )  <= in_state(70 );
    out_state(79 )  <= in_state(66 );
    out_state(80 )  <= in_state(62 );
    out_state(81 )  <= in_state(58 );
    out_state(82 )  <= in_state(54 );
    out_state(83 )  <= in_state(50 );
    out_state(84 )  <= in_state(46 );
    out_state(85 )  <= in_state(42 );
    out_state(86 )  <= in_state(38 );
    out_state(87 )  <= in_state(34 );
    out_state(88 )  <= in_state(30 );
    out_state(89 )  <= in_state(26 );
    out_state(90 )  <= in_state(22 );
    out_state(91 )  <= in_state(18 );
    out_state(92 )  <= in_state(14 );
    out_state(93 )  <= in_state(10 );
    out_state(94 )  <= in_state(6  );
    out_state(95 )  <= in_state(2  );
--------------------FOURTH_SEGMENT_SEG3---------------------------
    out_state(96 )  <= in_state(127);
    out_state(97 )  <= in_state(123);
    out_state(98 )  <= in_state(119);
    out_state(99 )  <= in_state(115);
    out_state(100)  <= in_state(111);
    out_state(101)  <= in_state(107);
    out_state(102)  <= in_state(103);
    out_state(103)  <= in_state(99 );
    out_state(104)  <= in_state(95 );
    out_state(105)  <= in_state(91 );
    out_state(106)  <= in_state(87 );
    out_state(107)  <= in_state(83 );
    out_state(108)  <= in_state(79 );
    out_state(109)  <= in_state(75 );
    out_state(110)  <= in_state(71 );
    out_state(111)  <= in_state(67 );
    out_state(112)  <= in_state(63 );
    out_state(113)  <= in_state(59 );
    out_state(114)  <= in_state(55 );
    out_state(115)  <= in_state(51 );
    out_state(116)  <= in_state(47 );
    out_state(117)  <= in_state(43 );
    out_state(118)  <= in_state(39 );
```

```
out_state(119)  <= in_state(35 );
out_state(120)  <= in_state(31 );
out_state(121)  <= in_state(27 );
out_state(122)  <= in_state(23 );
out_state(123)  <= in_state(19 );
out_state(124)  <= in_state(15 );
out_state(125)  <= in_state(11 );
out_state(126)  <= in_state(7  );
out_state(127)  <= in_state(3  );

---------keystate ini-------------

out_keystate(0  ) <= in_keystate(127);
out_keystate(1  ) <= in_keystate(126);
out_keystate(2  ) <= in_keystate(125);
out_keystate(3  ) <= in_keystate(124);
out_keystate(4  ) <= in_keystate(123);
out_keystate(5  ) <= in_keystate(122);
out_keystate(6  ) <= in_keystate(121);
out_keystate(7  ) <= in_keystate(120);
out_keystate(8  ) <= in_keystate(119);
out_keystate(9  ) <= in_keystate(118);
out_keystate(10 ) <= in_keystate(117);
out_keystate(11 ) <= in_keystate(116);
out_keystate(12 ) <= in_keystate(115);
out_keystate(13 ) <= in_keystate(114);
out_keystate(14 ) <= in_keystate(113);
out_keystate(15 ) <= in_keystate(112);
out_keystate(16 ) <= in_keystate(111);
out_keystate(17 ) <= in_keystate(110);
out_keystate(18 ) <= in_keystate(109);
out_keystate(19 ) <= in_keystate(108);
out_keystate(20 ) <= in_keystate(107);
out_keystate(21 ) <= in_keystate(106);
out_keystate(22 ) <= in_keystate(105);
out_keystate(23 ) <= in_keystate(104);
out_keystate(24 ) <= in_keystate(103);
out_keystate(25 ) <= in_keystate(102);
out_keystate(26 ) <= in_keystate(101);
out_keystate(27 ) <= in_keystate(100);
out_keystate(28 ) <= in_keystate(99 );
out_keystate(29 ) <= in_keystate(98 );
out_keystate(30 ) <= in_keystate(97 );
out_keystate(31 ) <= in_keystate(96 );
out_keystate(32 ) <= in_keystate(95 );
out_keystate(33 ) <= in_keystate(94 );
out_keystate(34 ) <= in_keystate(93 );
out_keystate(35 ) <= in_keystate(92 );
out_keystate(36 ) <= in_keystate(91 );
out_keystate(37 ) <= in_keystate(90 );
out_keystate(38 ) <= in_keystate(89 );
out_keystate(39 ) <= in_keystate(88 );
out_keystate(40 ) <= in_keystate(87 );
out_keystate(41 ) <= in_keystate(86 );
out_keystate(42 ) <= in_keystate(85 );
out_keystate(43 ) <= in_keystate(84 );
out_keystate(44 ) <= in_keystate(83 );
out_keystate(45 ) <= in_keystate(82 );
```

```
out_keystate(46 ) <= in_keystate(81 );
out_keystate(47 ) <= in_keystate(80 );
out_keystate(48 ) <= in_keystate(79 );
out_keystate(49 ) <= in_keystate(78 );
out_keystate(50 ) <= in_keystate(77 );
out_keystate(51 ) <= in_keystate(76 );
out_keystate(52 ) <= in_keystate(75 );
out_keystate(53 ) <= in_keystate(74 );
out_keystate(54 ) <= in_keystate(73 );
out_keystate(55 ) <= in_keystate(72 );
out_keystate(56 ) <= in_keystate(71 );
out_keystate(57 ) <= in_keystate(70 );
out_keystate(58 ) <= in_keystate(69 );
out_keystate(59 ) <= in_keystate(68 );
out_keystate(60 ) <= in_keystate(67 );
out_keystate(61 ) <= in_keystate(66 );
out_keystate(62 ) <= in_keystate(65 );
out_keystate(63 ) <= in_keystate(64 );
out_keystate(64 ) <= in_keystate(63 );
out_keystate(65 ) <= in_keystate(62 );
out_keystate(66 ) <= in_keystate(61 );
out_keystate(67 ) <= in_keystate(60 );
out_keystate(68 ) <= in_keystate(59 );
out_keystate(69 ) <= in_keystate(58 );
out_keystate(70 ) <= in_keystate(57 );
out_keystate(71 ) <= in_keystate(56 );
out_keystate(72 ) <= in_keystate(55 );
out_keystate(73 ) <= in_keystate(54 );
out_keystate(74 ) <= in_keystate(53 );
out_keystate(75 ) <= in_keystate(52 );
out_keystate(76 ) <= in_keystate(51 );
out_keystate(77 ) <= in_keystate(50 );
out_keystate(78 ) <= in_keystate(49 );
out_keystate(79 ) <= in_keystate(48 );
out_keystate(80 ) <= in_keystate(47 );
out_keystate(81 ) <= in_keystate(46 );
out_keystate(82 ) <= in_keystate(45 );
out_keystate(83 ) <= in_keystate(44 );
out_keystate(84 ) <= in_keystate(43 );
out_keystate(85 ) <= in_keystate(42 );
out_keystate(86 ) <= in_keystate(41 );
out_keystate(87 ) <= in_keystate(40 );
out_keystate(88 ) <= in_keystate(39 );
out_keystate(89 ) <= in_keystate(38 );
out_keystate(90 ) <= in_keystate(37 );
out_keystate(91 ) <= in_keystate(36 );
out_keystate(92 ) <= in_keystate(35 );
out_keystate(93 ) <= in_keystate(34 );
out_keystate(94 ) <= in_keystate(33 );
out_keystate(95 ) <= in_keystate(32 );
out_keystate(96 ) <= in_keystate(31 );
out_keystate(97 ) <= in_keystate(30 );
out_keystate(98 ) <= in_keystate(29 );
out_keystate(99 ) <= in_keystate(28 );
out_keystate(100) <= in_keystate(27 );
out_keystate(101) <= in_keystate(26 );
out_keystate(102) <= in_keystate(25 );
out_keystate(103) <= in_keystate(24 );
```

```
    out_keystate(104) <= in_keystate(23 );
    out_keystate(105) <= in_keystate(22 );
    out_keystate(106) <= in_keystate(21 );
    out_keystate(107) <= in_keystate(20 );
    out_keystate(108) <= in_keystate(19 );
    out_keystate(109) <= in_keystate(18 );
    out_keystate(110) <= in_keystate(17 );
    out_keystate(111) <= in_keystate(16 );
    out_keystate(112) <= in_keystate(15 );
    out_keystate(113) <= in_keystate(14 );
    out_keystate(114) <= in_keystate(13 );
    out_keystate(115) <= in_keystate(12 );
    out_keystate(116) <= in_keystate(11 );
    out_keystate(117) <= in_keystate(10 );
    out_keystate(118) <= in_keystate(9  );
    out_keystate(119) <= in_keystate(8  );
    out_keystate(120) <= in_keystate(7  );
    out_keystate(121) <= in_keystate(6  );
    out_keystate(122) <= in_keystate(5  );
    out_keystate(123) <= in_keystate(4  );
    out_keystate(124) <= in_keystate(3  );
    out_keystate(125) <= in_keystate(2  );
    out_keystate(126) <= in_keystate(1  );
    out_keystate(127) <= in_keystate(0  );
end architecture;
```

Το περιεχόμενο του αρχείου finalization.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity finalization is
    port
        (in_state  : in  state --128 bit input
        ;out_state : out state --128 bit outputs
        );
end entity;

architecture rtl of finalization is

begin
---------------------FIRST_SEGMENT_SEG0----------------------------
    out_state(124)  <= in_state(0  );
    out_state(120)  <= in_state(1  );
    out_state(116)  <= in_state(2  );
    out_state(112)  <= in_state(3  );
    out_state(108)  <= in_state(4  );
    out_state(104)  <= in_state(5  );
    out_state(100)  <= in_state(6  );
    out_state(96 )  <= in_state(7  );
    out_state(92 )  <= in_state(8  );
    out_state(88 )  <= in_state(9  );
    out_state(84 )  <= in_state(10 );
    out_state(80 )  <= in_state(11 );
    out_state(76 )  <= in_state(12 );
```

```
    out_state(72 )  <= in_state(13 );
    out_state(68 )  <= in_state(14 );
    out_state(64 )  <= in_state(15 );
    out_state(60 )  <= in_state(16 );
    out_state(56 )  <= in_state(17 );
    out_state(52 )  <= in_state(18 );
    out_state(48 )  <= in_state(19 );
    out_state(44 )  <= in_state(20 );
    out_state(40 )  <= in_state(21 );
    out_state(36 )  <= in_state(22 );
    out_state(32 )  <= in_state(23 );
    out_state(28 )  <= in_state(24 );
    out_state(24 )  <= in_state(25 );
    out_state(20 )  <= in_state(26 );
    out_state(16 )  <= in_state(27 );
    out_state(12 )  <= in_state(28 );
    out_state(8  )  <= in_state(29 );
    out_state(4  )  <= in_state(30 );
    out_state(0  )  <= in_state(31 );
--------------------SECOND_SEGMENT_SEG1----------------------------
    out_state(125)  <= in_state(32 );
    out_state(121)  <= in_state(33 );
    out_state(117)  <= in_state(34 );
    out_state(113)  <= in_state(35 );
    out_state(109)  <= in_state(36 );
    out_state(105)  <= in_state(37 );
    out_state(101)  <= in_state(38 );
    out_state(97 )  <= in_state(39 );
    out_state(93 )  <= in_state(40 );
    out_state(89 )  <= in_state(41 );
    out_state(85 )  <= in_state(42 );
    out_state(81 )  <= in_state(43 );
    out_state(77 )  <= in_state(44 );
    out_state(73 )  <= in_state(45 );
    out_state(69 )  <= in_state(46 );
    out_state(65 )  <= in_state(47 );
    out_state(61 )  <= in_state(48 );
    out_state(57 )  <= in_state(49 );
    out_state(53 )  <= in_state(50 );
    out_state(49 )  <= in_state(51 );
    out_state(45 )  <= in_state(52 );
    out_state(41 )  <= in_state(53 );
    out_state(37 )  <= in_state(54 );
    out_state(33 )  <= in_state(55 );
    out_state(29 )  <= in_state(56 );
    out_state(25 )  <= in_state(57 );
    out_state(21 )  <= in_state(58 );
    out_state(17 )  <= in_state(59 );
    out_state(13 )  <= in_state(60 );
    out_state(9  )  <= in_state(61 );
    out_state(5  )  <= in_state(62 );
    out_state(1  )  <= in_state(63 );
--------------------THIRD_SEGMENT_SEG2-----------------------------
    out_state(126)  <= in_state(64 );
    out_state(122)  <= in_state(65 );
    out_state(118)  <= in_state(66 );
    out_state(114)  <= in_state(67 );
    out_state(110)  <= in_state(68 );
```

```
    out_state(106)  <= in_state(69 );
    out_state(102)  <= in_state(70 );
    out_state(98 )  <= in_state(71 );
    out_state(94 )  <= in_state(72 );
    out_state(90 )  <= in_state(73 );
    out_state(86 )  <= in_state(74 );
    out_state(82 )  <= in_state(75 );
    out_state(78 )  <= in_state(76 );
    out_state(74 )  <= in_state(77 );
    out_state(70 )  <= in_state(78 );
    out_state(66 )  <= in_state(79 );
    out_state(62 )  <= in_state(80 );
    out_state(58 )  <= in_state(81 );
    out_state(54 )  <= in_state(82 );
    out_state(50 )  <= in_state(83 );
    out_state(46 )  <= in_state(84 );
    out_state(42 )  <= in_state(85 );
    out_state(38 )  <= in_state(86 );
    out_state(34 )  <= in_state(87 );
    out_state(30 )  <= in_state(88 );
    out_state(26 )  <= in_state(89 );
    out_state(22 )  <= in_state(90 );
    out_state(18 )  <= in_state(91 );
    out_state(14 )  <= in_state(92 );
    out_state(10 )  <= in_state(93 );
    out_state(6  )  <= in_state(94 );
    out_state(2  )  <= in_state(95 );
--------------------FOURTH_SEGMENT_SEG3---------------------------
    out_state(127)  <= in_state(96 );
    out_state(123)  <= in_state(97 );
    out_state(119)  <= in_state(98 );
    out_state(115)  <= in_state(99 );
    out_state(111)  <= in_state(100);
    out_state(107)  <= in_state(101);
    out_state(103)  <= in_state(102);
    out_state(99 )  <= in_state(103);
    out_state(95 )  <= in_state(104);
    out_state(91 )  <= in_state(105);
    out_state(87 )  <= in_state(106);
    out_state(83 )  <= in_state(107);
    out_state(79 )  <= in_state(108);
    out_state(75 )  <= in_state(109);
    out_state(71 )  <= in_state(110);
    out_state(67 )  <= in_state(111);
    out_state(63 )  <= in_state(112);
    out_state(59 )  <= in_state(113);
    out_state(55 )  <= in_state(114);
    out_state(51 )  <= in_state(115);
    out_state(47 )  <= in_state(116);
    out_state(43 )  <= in_state(117);
    out_state(39 )  <= in_state(118);
    out_state(35 )  <= in_state(119);
    out_state(31 )  <= in_state(120);
    out_state(27 )  <= in_state(121);
    out_state(23 )  <= in_state(122);
    out_state(19 )  <= in_state(123);
    out_state(15 )  <= in_state(124);
    out_state(11 )  <= in_state(125);
```

```
    out_state(7  )  <= in_state(126);
    out_state(3  )  <= in_state(127);

end architecture;
```

Το περιεχόμενο του αρχείου sbox.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity sbox is
    port
        (in_state            : in  state --128 bit input
        ;seg0,seg1,seg2,seg3 : out segment --32 bit outputs
        );
end entity;

architecture rtl of sbox is

begin
    seg0 <= in_state(0   to 31 );
    seg1 <= in_state(32  to 63 );
    seg2 <= in_state(64  to 95 );
    seg3 <= in_state(96  to 127);
end architecture;
```

Το περιεχόμενο του αρχείου keystate.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity keystate is
    port
        ( in_keystate                          : in  state --128 bit input
        ; keyword7,keyword6,keyword5,keyword4 : out keyword --16 bit out-
puts
        ; keyword3,keyword2,keyword1,keyword0 : out keyword --16 bit out-
puts
        );
end entity;

architecture rtl of keystate is

begin
    keyword0 <= in_keystate(0   to 15 );
    keyword1 <= in_keystate(16  to 31 );
    keyword2 <= in_keystate(32  to 47 );
    keyword3 <= in_keystate(48  to 63 );
    keyword4 <= in_keystate(64  to 79 );
    keyword5 <= in_keystate(80  to 95 );
    keyword6 <= in_keystate(96  to 111);
    keyword7 <= in_keystate(112 to 127);
```

```
end architecture;
```

Το περιεχόμενο του αρχείου subcells.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity subcells is
    port
        (segin0,segin1,segin2,segin3    : in  segment -- 32 bit inputs
        ;segout0,segout1,segout2,segout3: out segment -- 32 bit outputs
        );
end entity;

architecture rtl of subcells is

signal temp0,temp1,temp2,temp3,temp10,temp30,temp20 : segment; -- temporary
32 bit signals

begin

    temp1  <= segin1 xor (segin0 and segin2);   -- S1 <-- S1 XOR (S0 AND
S2)
    temp0  <= segin0 xor (temp1  and segin3);   -- S0 <-- S0 XOR (S1 AND
S3)
    temp2  <= segin2 xor (temp0  or  temp1 );   -- S2 <-- S2 XOR (S0 OR
S1)
    temp3  <= segin3 xor temp2;                 -- S3 <-- S3 XOR S2
    temp10 <= temp1  xor temp3;                 -- S1 <-- S1 XOR S3
    temp30 <= not temp3;                        -- S3 <-- NOT S3
    temp20 <= temp2  xor (temp0 and temp10  );  -- S2 <-- S2 XOR (S0 AND
S1)

--{S0,S1,S2,S3} <-- {S3,S1,S2,S0}--
    segout0 <= temp30;
    segout1 <= temp10;
    segout2 <= temp20;
    segout3 <= temp0;
end architecture;
```

Το περιεχόμενο του αρχείου permbits.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity permbits is
    port
        (segin0,segin1,segin2,segin3    : in  segment -- 32 bit inputs
        ;segout0,segout1,segout2,segout3: out segment -- 32 bit outputs
        );
end entity;
```

```
architecture rtl of permbits is

begin
--------Permutations_of_S0--------
    segout0(0 ) <= segin0(2 );
    segout0(1 ) <= segin0(6 );
    segout0(2 ) <= segin0(10);
    segout0(3 ) <= segin0(14);
    segout0(4 ) <= segin0(18);
    segout0(5 ) <= segin0(22);
    segout0(6 ) <= segin0(26);
    segout0(7 ) <= segin0(30);
    segout0(8 ) <= segin0(1 );
    segout0(9 ) <= segin0(5 );
    segout0(10) <= segin0(9 );
    segout0(11) <= segin0(13);
    segout0(12) <= segin0(17);
    segout0(13) <= segin0(21);
    segout0(14) <= segin0(25);
    segout0(15) <= segin0(29);
    segout0(16) <= segin0(0 );
    segout0(17) <= segin0(4 );
    segout0(18) <= segin0(8 );
    segout0(19) <= segin0(12);
    segout0(20) <= segin0(16);
    segout0(21) <= segin0(20);
    segout0(22) <= segin0(24);
    segout0(23) <= segin0(28);
    segout0(24) <= segin0(3 );
    segout0(25) <= segin0(7 );
    segout0(26) <= segin0(11);
    segout0(27) <= segin0(15);
    segout0(28) <= segin0(19);
    segout0(29) <= segin0(23);
    segout0(30) <= segin0(27);
    segout0(31) <= segin0(31);
--------Permutations_of_S1--------
    segout1(0 ) <= segin1(1 );
    segout1(1 ) <= segin1(5 );
    segout1(2 ) <= segin1(9 );
    segout1(3 ) <= segin1(13);
    segout1(4 ) <= segin1(17);
    segout1(5 ) <= segin1(21);
    segout1(6 ) <= segin1(25);
    segout1(7 ) <= segin1(29);
    segout1(8 ) <= segin1(0 );
    segout1(9 ) <= segin1(4 );
    segout1(10) <= segin1(8 );
    segout1(11) <= segin1(12);
    segout1(12) <= segin1(16);
    segout1(13) <= segin1(20);
    segout1(14) <= segin1(24);
    segout1(15) <= segin1(28);
    segout1(16) <= segin1(3 );
    segout1(17) <= segin1(7 );
    segout1(18) <= segin1(11);
    segout1(19) <= segin1(15);
    segout1(20) <= segin1(19);
```

```
    segout1(21) <= segin1(23);
    segout1(22) <= segin1(27);
    segout1(23) <= segin1(31);
    segout1(24) <= segin1(2 );
    segout1(25) <= segin1(6 );
    segout1(26) <= segin1(10);
    segout1(27) <= segin1(14);
    segout1(28) <= segin1(18);
    segout1(29) <= segin1(22);
    segout1(30) <= segin1(26);
    segout1(31) <= segin1(30);
--------Permutations_of_S2--------
    segout2(0 ) <= segin2(0 );
    segout2(1 ) <= segin2(4 );
    segout2(2 ) <= segin2(8 );
    segout2(3 ) <= segin2(12);
    segout2(4 ) <= segin2(16);
    segout2(5 ) <= segin2(20);
    segout2(6 ) <= segin2(24);
    segout2(7 ) <= segin2(28);
    segout2(8 ) <= segin2(3 );
    segout2(9 ) <= segin2(7 );
    segout2(10) <= segin2(11);
    segout2(11) <= segin2(15);
    segout2(12) <= segin2(19);
    segout2(13) <= segin2(23);
    segout2(14) <= segin2(27);
    segout2(15) <= segin2(31);
    segout2(16) <= segin2(2 );
    segout2(17) <= segin2(6 );
    segout2(18) <= segin2(10);
    segout2(19) <= segin2(14);
    segout2(20) <= segin2(18);
    segout2(21) <= segin2(22);
    segout2(22) <= segin2(26);
    segout2(23) <= segin2(30);
    segout2(24) <= segin2(1 );
    segout2(25) <= segin2(5 );
    segout2(26) <= segin2(9 );
    segout2(27) <= segin2(13);
    segout2(28) <= segin2(17);
    segout2(29) <= segin2(21);
    segout2(30) <= segin2(25);
    segout2(31) <= segin2(29);
--------Permutations_of_S3--------
    segout3(0 ) <= segin3(3 );
    segout3(1 ) <= segin3(7 );
    segout3(2 ) <= segin3(11);
    segout3(3 ) <= segin3(15);
    segout3(4 ) <= segin3(19);
    segout3(5 ) <= segin3(23);
    segout3(6 ) <= segin3(27);
    segout3(7 ) <= segin3(31);
    segout3(8 ) <= segin3(2 );
    segout3(9 ) <= segin3(6 );
    segout3(10) <= segin3(10);
    segout3(11) <= segin3(14);
    segout3(12) <= segin3(18);
```

```
    segout3(13) <= segin3(22);
    segout3(14) <= segin3(26);
    segout3(15) <= segin3(30);
    segout3(16) <= segin3(1 );
    segout3(17) <= segin3(5 );
    segout3(18) <= segin3(9 );
    segout3(19) <= segin3(13);
    segout3(20) <= segin3(17);
    segout3(21) <= segin3(21);
    segout3(22) <= segin3(25);
    segout3(23) <= segin3(29);
    segout3(24) <= segin3(0 );
    segout3(25) <= segin3(4 );
    segout3(26) <= segin3(8 );
    segout3(27) <= segin3(12);
    segout3(28) <= segin3(16);
    segout3(29) <= segin3(20);
    segout3(30) <= segin3(24);
    segout3(31) <= segin3(28);

end architecture;
```

Το περιεχόμενο του αρχείου addroundkey.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity addroundkey is
    port(u_segment,v_segment,segin1,segin2,segin3 : in segment  -- UV in-
puts 32 bit each/32 bit segment inputs
        ;xyin                                     : in xy_1     -- 8 bit
input for round constants
        ;segout1,segout2,segout3 : out segment                  -- 32 bit
segments outputs
        --;roundkey               : out word                   -- 64 bit
output
        );
end entity;

architecture rtl of addroundkey is

    signal temp3 : segment;

begin

    --roundkey(0  to 31) <= u_segment(0 to 31);          --RK = U||V
    --roundkey(32 to 63) <= v_segment(0 to 31);          --RK = U||V

    segout1 <= segin1 xor v_segment;                -- S1 ← S1 ⊕ V
    segout2 <= segin2 xor u_segment;                -- S2 ← S2 ⊕ U

    temp3(0  to 23) <= X"800000";
    temp3(24 to 31) <= xyin(0 to 7);                --00c5c4c3c2c1c0

    segout3 <= segin3 xor temp3;                    --S3 ← S3 ⊕ 0x800000XY
```

```
end architecture;
```

Το περιεχόμενο του αρχείου keyschedule.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity keyschedule is
    port(keywordin7,keywordin6,keywordin5,keywordin4     : in  keyword --16
bit inputs
        ;keywordin3,keywordin2,keywordin1,keywordin0     : in  keyword --16
bit inputs
        ;keywordout7,keywordout6,keywordout5,keywordout4 : out keyword --16
bit outputs
        ;keywordout3,keywordout2,keywordout1,keywordout0 : out keyword --16
bit outputs
        ;u,v                                             : out segment --32
bit outputs
        );
end entity;

architecture rtl of keyschedule is

begin
    --U ← W2||W3
    u(0  to 15) <= keywordin2;
    u(16 to 31) <= keywordin3;
    --V ← W6||W7
    v(0  to 15) <= keywordin6;
    v(16 to 31) <= keywordin7;

    keywordout0 <= std_logic_vector(unsigned(keywordin6) ror 2 ); -- rotate
right 2 bits
    keywordout1 <= std_logic_vector(unsigned(keywordin7) ror 12); -- rotate
right 12 bits

    keywordout2 <= keywordin0;
    keywordout3 <= keywordin1;
    keywordout4 <= keywordin2;
    keywordout5 <= keywordin3;
    keywordout6 <= keywordin4;
    keywordout7 <= keywordin5;

end architecture;
```

Το περιεχόμενο του αρχείου lfsr.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity lfsr is
```

```
    port( rst,clk,lfsr_en : in  std_logic
        ;lfsr_out         : out xy_l
        );
end entity;

architecture rtl of lfsr is

signal lfsr : xy_l;

    begin

lfsr_out(0) <= lfsr(7);
lfsr_out(1) <= lfsr(6);
lfsr_out(2) <= lfsr(5);
lfsr_out(3) <= lfsr(4);
lfsr_out(4) <= lfsr(3);
lfsr_out(5) <= lfsr(2);
lfsr_out(6) <= lfsr(1);
lfsr_out(7) <= lfsr(0);

process(clk)

    begin
        if rising_edge(clk) then
            if rst = '1' then
                lfsr <= "10000000";
            elsif lfsr_en = '1' then
                lfsr(7) <= '0';
                lfsr(6) <= '0';
                lfsr(5) <= lfsr(4);
                lfsr(4) <= lfsr(3);
                lfsr(3) <= lfsr(2);
                lfsr(2) <= lfsr(1);
                lfsr(1) <= lfsr(0);
                lfsr(0) <= lfsr(5) xor lfsr(4) xor '1';
            end if;
        end if;

end process;

end architecture;
```

Το περιεχόμενο του αρχείου delta.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;
use ieee.NUMERIC_STD.all;

entity delta is
    port
        (deltain  : in  word --64 bit input
        ;sel      : in  std_logic_vector( 0 to 1)
        ;deltaout : out word --64 bit output
        );
end entity;
```

```
architecture rtl of delta is
signal deltastate : word;
begin

deltaout <= deltastate;

    process(sel,deltain) is

        begin

            if sel = "01" then --AD/messages
                deltastate <= std_logic_vector(unsigned(deltastate) sll 1);
            elsif sel = "10" then --last messages
                deltastate <= std_logic_vector((unsigned(deltastate) sll 1)
+ unsigned(deltastate));
            elsif sel = "00" then --Nonce
                deltastate <= deltain;
            elsif sel = "01" then --
                deltastate <= deltastate;
            end if;

    end process;

end architecture;
```

Το περιεχόμενο του αρχείου feedback.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity feedback is
    port
        (feedin            : in  state --128 bit input
        ;feedout1,feedout2 : out word  --64 bit outputs
        );
end entity;

architecture rtl of feedback is
signal temp : word;
begin
temp        <= feedin(0  to 63 );
feedout1    <= feedin(64 to 127);
feedout2    <= std_logic_vector(unsigned(temp) rol 1 ); -- rotate left 1
bit
end architecture;
```

Το περιεχόμενο του αρχείου controller.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;
use ieee.std_logic_unsigned.all;
```

```
entity controller is
    port( reset, clk, dpstate, hardreset : in std_logic
    ; rst, lfsr_en, in_mode : out std_logic
        );
end entity;

architecture rtl of controller is

type state is (initialize, rounds);
signal state_now, state_next : state;
signal timer: integer range 0 to 40;

begin

    process
        variable countlow: integer range 0 to 40;
    begin
        wait until rising_edge(clk);
        if reset = '1' or hardreset = '1' then
            state_now <= initialize;
            countlow := 0;
        else
            countlow := countlow + 1;
            if countlow >= timer then
                state_now <= state_next;
                countlow := 0;
            end if;
        end if;
    end process;
--------------------------------------------------------------------------------
    process(state_now, dpstate)
    begin
        case state_now is
            when initialize =>
                in_mode <= '1';
                lfsr_en <= '0';
                rst <= '1';
                if dpstate = '1' then
                    timer <= 1;
                    state_next <= rounds;
                else
                    timer <= 4;
                    state_next <= rounds;
                end if;
            when rounds =>
                rst <='0';
                lfsr_en <='1';
                in_mode <= '0';
        end case;
    end process;

end architecture;
```

Το περιεχόμενο του αρχείου dp.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity dp is
    port(secretkey,in_state          : in  state
    ;clk,rst,lfsr_en,in_mode         : in  std_logic
    ;typeofblock                     : in typeofblock
    ;state_out, tag                  : out state
    ;dpstate, done ,reset            : out std_logic
    );
end entity;

architecture rtl of dp is

signal segment0,segment1,segment2,segment3,segmentout0,segmentout1,segmen-
tout2,segmentout3 : segment;
signal permbitsout0,permbitsout1,permbitsout2,permbitsout3,u_seg-
ment_out,v_segment_out : segment;
signal addroundkeyseg1,addroundkeyseg2,addroundkeyseg3 : segment;
signal
keystateout7,keystateout6,keystateout5,keystateout4,keystateout3,keystateou
t2,keystateout1,keystateout0 : keyword;
signal keyscheduleout7,keyscheduleout6,keyscheduleout5,keysched-
uleout4,keyscheduleout3,keyscheduleout2,keyscheduleout1,keyscheduleout0 :
keyword;
signal lfsr_out : xy_l;
signal feedout1,feedout2,deltaout,deltatemp,feedout1temp,feedout2temp:
word;
signal round_out, temp_in, temp_secretkey, v_out, temp_state_out, keysched-
uleouttemp, tempinmain, keyhold : state;
signal sel : std_logic_vector(0 to 1);

begin

--fragmentation-------------------------------------------------------------
    sbox :
    entity work.sbox port map
        (in_state => temp_in
        ,seg0 => segment0
        ,seg1 => segment1
        ,seg2 => segment2
        ,seg3 => segment3
        );

    keystate :
    entity work.keystate port map
      ( in_keystate => temp_secretkey
      , keyword7 => keystateout7
      , keyword6 => keystateout6
      , keyword5 => keystateout5
      , keyword4 => keystateout4
      , keyword3 => keystateout3
      , keyword2 => keystateout2
      , keyword1 => keystateout1
```

```
      , keyword0 => keystateout0
     );

--GIFT-------------------------------------------------------------------
  subcells :
    entity work.subcells port map
      ( segin0  => segment0
      , segin1  => segment1
      , segin2  => segment2
      , segin3  => segment3
      , segout0 => segmentout0
      , segout1 => segmentout1
      , segout2 => segmentout2
      , segout3 => segmentout3
     );

    permbits :
    entity work.permbits port map
      ( segin0  => segmentout0
      , segin1  => segmentout1
      , segin2  => segmentout2
      , segin3  => segmentout3
      , segout0 => permbitsout0  --round segment0 output
      , segout1 => permbitsout1
      , segout2 => permbitsout2
      , segout3 => permbitsout3
     );

    addroundkey :
    entity work.addroundkey port map
      ( u_segment => u_segment_out
      , v_segment => v_segment_out
      , segin1    => permbitsout1
      , segin2    => permbitsout2
      , segin3    => permbitsout3
      , xyin      => lfsr_out
      , segout1   => addroundkeyseg1 --round segment1 output
      , segout2   => addroundkeyseg2 --round segment2 output
      , segout3   => addroundkeyseg3 --round segment3 output
     );
--keyscedule_and_constants-----------------------------------------------
---
    keyschedule :
    entity work.keyschedule port map
      ( keywordin7  => keystateout7
      , keywordin6  => keystateout6
      , keywordin5  => keystateout5
      , keywordin4  => keystateout4
      , keywordin3  => keystateout3
      , keywordin2  => keystateout2
      , keywordin1  => keystateout1
      , keywordin0  => keystateout0
      , keywordout7 => keyscheduleout7
      , keywordout6 => keyscheduleout6
      , keywordout5 => keyscheduleout5
      , keywordout4 => keyscheduleout4
      , keywordout3 => keyscheduleout3
      , keywordout2 => keyscheduleout2
```

```
      , keywordout1 => keyscheduleout1
      , keywordout0 => keyscheduleout0
      , u           => u_segment_out
      , v           => v_segment_out
     );

        lfsr :
    entity work.lfsr port map
      ( rst => rst
      ,lfsr_en => lfsr_en
      , clk => clk
      , lfsr_out => lfsr_out
     );
--giftcofb_round_output-------------------------------------------------------
------
        keyscheduleouttemp(0   to 15 ) <= keyscheduleout0;
        keyscheduleouttemp(16  to 31 ) <= keyscheduleout1;
        keyscheduleouttemp(32  to 47 ) <= keyscheduleout2;
        keyscheduleouttemp(48  to 63 ) <= keyscheduleout3;
        keyscheduleouttemp(64  to 79 ) <= keyscheduleout4;
        keyscheduleouttemp(80  to 95 ) <= keyscheduleout5;
        keyscheduleouttemp(96  to 111) <= keyscheduleout6;
        keyscheduleouttemp(112 to 127) <= keyscheduleout7;

        round_out(0  to  31) <= permbitsout0;
        round_out(32 to  63) <= addroundkeyseg1;
        round_out(64 to  95) <= addroundkeyseg2;
        round_out(96 to 127) <= addroundkeyseg3;
--COFB-------------------------------------------------------------------------
-----
        feedback :
    entity work.feedback port map
         (feedin => round_out
        ,feedout1 => feedout1
        ,feedout2 => feedout2
        );

        delta :
    entity work.delta port map
        (deltain(0 to 63) => deltatemp
        ,sel => sel
        ,deltaout => deltaout
        );

        temp_state_out <= round_out xor tempinmain;
-------------------------------------------------------------------------------
----

    process
        variable count: integer range 0 to 40;
    begin

    wait until rising_edge(clk);

    if in_mode = '0' then     --in_mode 0 = rounds \ 1 = initialization
        temp_in <= round_out;
        temp_secretkey <= keyscheduleouttemp;
        count := count + 1;
```

```
            if count >= 40 then
                feedout1temp <= feedout1;
                feedout2temp <= feedout2;
                keyhold      <= keyscheduleouttemp;
                if typeofblock = "00" then --NONCE
                    deltatemp <= round_out(0 to 63);
               elsif typeofblock = "01" then --AD
                    sel(0 to 1) <= "11";
                elsif typeofblock = "10" then --M
                    sel(0 to 1) <= "11";
                    state_out <= temp_state_out;
                else --Last
                    sel(0 to 1) <= "11";
                    state_out <= temp_state_out;
                    tag <= round_out;
                end if;
                reset <= '1';
                done <= '1';
                count := 0;
            end if;
    elsif in_mode ='1' then
                count := 0;
                reset <= '0';
                done  <= '0';
            if typeofblock = "00" then --NONCE
                sel(0 to 1) <= "00";
                temp_in <= in_state;
                temp_secretkey <= secretkey;
                dpstate <= '1';
            elsif typeofblock = "01" then --AD
                sel(0 to 1) <= "01";
                tempinmain <= in_state;
                temp_secretkey <= keyhold;
                v_out(0  to  63) <= tempinmain(0  to  63) xor feedout1temp;
                v_out(64 to 127) <= tempinmain(64 to 127) xor feedout2temp
xor deltaout;
                temp_in <= v_out;
                dpstate <= '0';
            elsif typeofblock = "10" then --M
                sel(0 to 1) <= "01";
                tempinmain <= in_state;
                temp_secretkey <= keyhold;
                v_out(0  to  63) <= tempinmain(0  to  63) xor feedout1temp;
                v_out(64 to 127) <= tempinmain(64 to 127) xor feedout2temp
xor deltaout;
                temp_in <= v_out;
                dpstate <= '0';
            else --Last
                sel(0 to 1) <= "10";
                tempinmain <= in_state;
                temp_secretkey <= keyhold;
                v_out(0  to  63) <= tempinmain(0  to  63) xor feedout1temp;
                v_out(64 to 127) <= tempinmain(64 to 127) xor feedout2temp
xor deltaout;
                temp_in <= v_out;
                dpstate <= '0';
        end if;
    end if;
```

```
    end process;
end architecture;
```

Το περιεχόμενο του αρχείου giftcofbtop.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity giftcofb is
    port
    (in_state    :in  state
    ;secretkey   :in  state
    ;typeofblock:in  typeofblock
    ;hardreset   :in  std_logic
    ;clk         :in  std_logic
    ;done        :out std_logic
    ;state_out   :out state
    ;tag         :out state
    );
end entity;

architecture rtl of giftcofb is

signal ctl_rst, ctl_lfsr_en, ctl_in_mode, ctl_dpstate, reset_ctl :
std_logic;

begin

    u_dp :
    entity work.dp port map
    (in_state    => in_state
    ,secretkey   => secretkey
    ,typeofblock=> typeofblock
    ,dpstate     => ctl_dpstate
    ,done        => done
    ,reset       => reset_ctl
    ,clk         => clk
    ,rst         => ctl_rst
    ,lfsr_en     => ctl_lfsr_en
    ,in_mode     => ctl_in_mode
    ,state_out   => state_out
    ,tag         => tag
    );

    u_controller :
    entity work.controller port map
      (reset     => reset_ctl
      ,hardreset=> hardreset
      ,clk       => clk
      ,rst       => ctl_rst
      ,dpstate   => ctl_dpstate
      ,lfsr_en   => ctl_lfsr_en
      ,in_mode   => ctl_in_mode
     );
```

```
end architecture;
```

Το περιεχόμενο του αρχείου giftcofb_tb.vhd είναι το εξής:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.gift128_pkg.all;

entity giftcofb_tb is
end entity;

architecture testbench of giftcofb_tb is

signal in_state_tb, secretkey_tb, state_out_tb, tag_tb: state;
signal block_tb: typeofblock;
signal clk_tb : std_logic := '1';
signal done_tb: std_logic;
signal hardreset_tb: std_logic;

begin

testbench : entity work.giftcofb port map
        (in_state   => in_state_tb
        ,secretkey  => secretkey_tb
        ,typeofblock=> block_tb
        ,hardreset  => hardreset_tb
        ,clk        => clk_tb
        ,done       => done_tb
        ,state_out  => state_out_tb
        ,tag        => tag_tb
);

clk_tb <= not clk_tb after 100ns;

process
begin
--test
-- 00 -- nonce
-- 01 -- AD
-- 10 -- M
-- 11 -- Last
    hardreset_tb <= '1';
    wait for 200ns;
    hardreset_tb <= '0';

        in_state_tb <= pad128(X"000102030405060708090A0B0C0D0E0F");
        secretkey_tb <= X"000102030405060708090A0B0C0D0E0F";
    --in_state_tb <= pad128(X"A94AF7F9BA181DF9B2B00EB7DBFA93DF");
    --secretkey_tb <= X"10000302504007069080 0b0ad0c00f0e";
    block_tb <= "00";
   wait until done_tb = '1';
    in_state_tb <= pad128(X"000102030405060708090A0B0C0D0E0F");
    block_tb <= "01";
   wait until done_tb = '1';
    in_state_tb <= pad128(X"000102030405060708090A0B0C0D0E0F");
```

```
    block_tb <= "11";
   wait until done_tb = '1';
   wait for 200ns;
   assert false report "!!! Simulation Finished !!!" severity failure;
   wait;
end process;

end architecture;
```