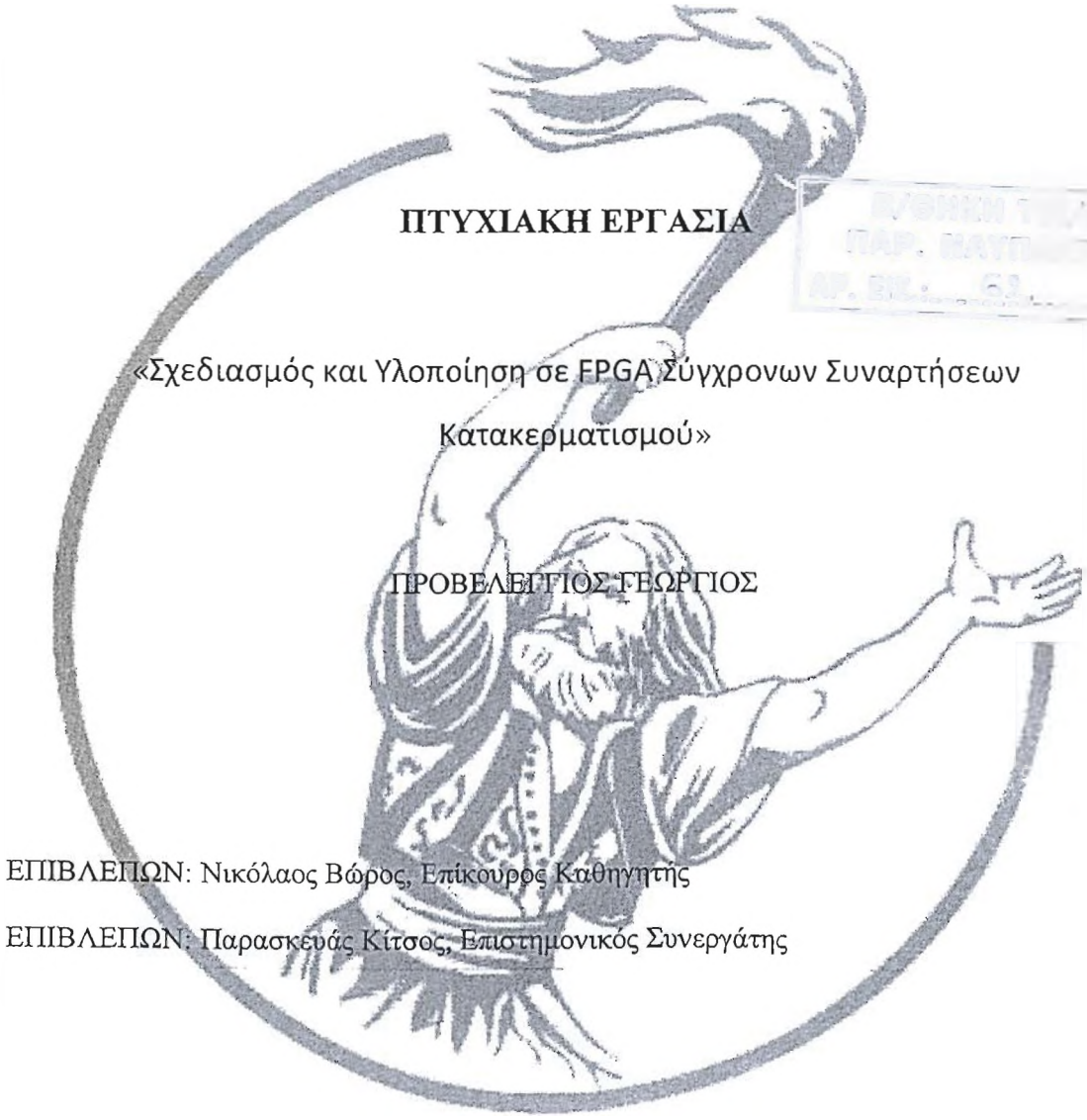
ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΜΕΣΟΛΟΓΓΙΟΥ

ΤΜΗΜΑ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ & ΔΙΚΤΥΩΝ



ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΒΥΘΙΝΗ ΤΕΧΝΗ
ΠΑΡ. ΒΑΥΠΑΙΟΥ
ΑΡ. ΕΙΣ. ... 63 ...

«Σχεδιασμός και Υλοποίηση σε FPGA Σύγχρονων Συναρτήσεων Κατακερματισμού»

ΠΡΟΒΕΛΕΓΓΙΟΣ ΓΕΩΡΓΙΟΣ

ΕΠΙΒΛΕΠΩΝ: Νικόλαος Βώρος, Επίκουρος Καθηγητής

ΕΠΙΒΛΕΠΩΝ: Παρασκευάς Κίτσος, Επιστημονικός Συνεργάτης

ΝΑΥΠΑΚΤΟΣ 2011

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Ναύπακτος,

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1.

2.

3.

ΠΕΡΙΛΗΨΗ ΕΛΛΗΝΙΚΑ

Η κρυπτογραφία στις μέρες μας χρησιμοποιείται σε μια πληθώρα εφαρμογών για την εξασφάλιση της ασφαλής μεταφοράς δεδομένων. Οι συναρτήσεις κατακερματισμού ως υποκατηγορία των αλγορίθμων κρυπτογράφησης χρησιμοποιούνται κυρίως για την ανίχνευση αλλαγών σε μεταδιδόμενα δεδομένα αλλά και για την πιστοποίηση της ταυτότητας του συντάκτη του εκάστοτε μηνύματος. Καθώς ο τομέας των υπολογιστικών συστημάτων και των τηλεπικοινωνιών αναπτύσσεται ραγδαία γίνεται επιτακτική η ανάγκη για την ανάπτυξη πολυπλοκότερων αλγορίθμων που θα παρέχουν υψηλότερα επίπεδα ασφάλειας. Δύο συνιστώσες του προβλήματος που προκύπτει καθώς αυξάνεται η πολυπλοκότητα είναι η μείωση της ταχύτητας εκτέλεσης των αλγορίθμων καθώς και η αύξηση της απαιτούμενης ισχύος του συστήματος. Για το λόγο αυτό, συχνά προτιμάται, η υλοποίηση των αλγορίθμων αυτών να γίνεται απευθείας σε υλικό κάτι το οποίο προσφέρει υψηλότερες ταχύτητες εκτέλεσης από την υλοποίηση μέσω λογισμικού. Για τη μείωση της απαιτούμενης ισχύος μπορούν να εφαρμοστούν διάφορες τεχνικές σχεδίασης οι οποίες θα μειώνουν την κατανάλωση του συστήματος.

Στα πλαίσια της παρούσας εργασίας επιλέχτηκαν δύο συναρτήσεις κατακερματισμού οι οποίες αρχικά υλοποιήθηκαν σε υλικό και στη συνέχεια εφαρμόστηκαν διάφορες τεχνικές σχεδίασης για τη μείωση της κατανάλωσης ισχύος. Οι συναρτήσεις αυτές είναι οι Fugue και JH οι οποίες και συμμετέχουν στο δημόσιο διαγωνισμό συναρτήσεων κατακερματισμού του Εθνικού Ιδρύματος Προτύπων και Τεχνολογίας (National Institute of Standards and Technology - NIST). Για την υλοποίηση των συναρτήσεων χρησιμοποιήθηκε μια ειδική κατηγορία ολοκληρωμένων κυκλωμάτων που ονομάζεται «Δομές Πίνακα Πυλών Επαναδιατάξιμης Λογικής (Field Programmable Gate Arrays - FPGA)». Αρχικά σχεδιάστηκαν δύο αρχιτεκτονικές που πραγματοποιούν τις λειτουργίες των παραπάνω αλγορίθμων και στη συνέχεια σχεδιάστηκαν παραλλαγές αυτών των αρχιτεκτονικών οι οποίες χρησιμοποιούν μια πληθώρα τεχνικών σχεδίασης χαμηλής κατανάλωσης. Επίσης όσων αφορά τη συνάρτηση Fugue σχεδιάστηκαν και δύο επιπλέον αρχιτεκτονικές που σκοπό έχουν την αύξηση της ταχύτητας εκτέλεσης του αλγορίθμου. Για τη περιγραφή όλων των παραπάνω αρχιτεκτονικών χρησιμοποιήθηκε η γλώσσα περιγραφής υλικού VHDL. Τέλος, γίνεται παρουσίαση των αποτελεσμάτων των μετρήσεων που πραγματοποιήθηκαν όσων αφορά τη κατανάλωση ισχύος, τη ταχύτητα εκτέλεσης αλλά και τις απαιτήσεις σε υλικό.

ΠΕΡΙΛΗΨΗ ΑΓΓΛΙΚΑ - ABSTRACT

Cryptography nowadays used in a variety of applications to ensure secure data transfer. The hash functions as a subset of cryptographic algorithms used primarily to detect changes in the transmitted data and for authentication of the originator of each message. As the field of computing and telecommunications is growing rapidly is an urgent need to develop complex algorithms to provide higher levels of security. Two components of the problem arising from increasing complexity is the reduced speed performance of the algorithms and the increasing power requirements of the system. For this reason, often preferred the solution of directly hardware implementation these algorithms something that offer higher speed performance than software implementation. To reduce the power requirements could applied different design techniques that will reduce the consumption of the system.

For the purpose of this study selected two hash functions which are initially implemented in hardware and then applied various design techniques to reduce power consumption. These functions are the Fugue and JH hush functions which are also participating in the hash algorithm public competition of the National Institute of Standards and Technology - NIST. The targeted device for those implementations is a special category of integrated circuits called «Field Programmable Gate Arrays - FPGA». Originally designed two architectures that implement the basics operations of those algorithms and then designed variations which use a variety of low-power design techniques. Also, only for Fugue hash function designed two additional architectures to increase the speed performance of the algorithm. The hardware description language which used for the implementation of above architectures is VHDL. Finally, in this study presented the results of measurements about power consumption, the speed performance and the hardware requirements for each architecture.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Στόχος αυτής της εργασίας είναι η μελέτη και η ανάλυση δύο συναρτήσεων κατακερματισμού καθώς και η υλοποίηση τους σε ολοκληρωμένα κυκλώματα. Οι αλγόριθμοι που θα μελετηθούν είναι οι Fugue και JH οι οποίοι κατατέθηκαν στον δημόσιο διαγωνισμό συναρτήσεων κατακερματισμού του Εθνικού Ιδρύματος Προτύπων και Τεχνολογίας (National Institute of Standards and Technology - NIST) το 2009. Η περιγραφή τους θα γίνει χρησιμοποιώντας τη γλώσσα περιγραφής υλικού VHDL ενώ τα ολοκληρωμένα κυκλώματα που θα χρησιμοποιηθούν για την υλοποίηση ανήκουν στη κατηγορία των Field Programmable Gate Array (FPGAs) .

Η πολυπλοκότητα των σημερινών αλγορίθμων αλλά και η ταχύτητα που καλούνται να εκτελούν τις λειτουργίες τους κάνει επιτακτική την ανάγκη υλοποίησης τους σε επίπεδο υλικού. Επίσης σημαντικό ρόλο παίζει και η εφαρμογή στην οποία προορίζεται να χρησιμοποιηθούν αυτοί οι αλγόριθμοι. Ένας λόγος που το κάνει τόσο σημαντικό είναι ο διαχωρισμός των εφαρμογών σε συσκευές που προορίζονται να λειτουργούν με συνεχή ή μη συνεχή παροχή τροφοδοσίας ρεύματος. Από αυτό προκύπτει η σημαντικότητα της χαμηλής κατανάλωσης ισχύος των συσκευών που προορίζονται για μη συνεχή τροφοδοσία. Έτσι λοιπόν ένας επιπλέον στόχος αυτής της εργασίας είναι η μελέτη των αλγορίθμων ως προς τη κατανάλωση αλλά και η σχεδίαση αρχιτεκτονικών οι οποίες θα έχουν χαμηλή κατανάλωση σε ισχύ.

Αρχικά θα υλοποιηθούν δύο κυκλώματα των αλγορίθμων τα οποία θα πραγματοποιούν τις λειτουργίες αυτών. Στη συνέχεια πάνω σε αυτά τα δύο κυκλώματα «αναφοράς» θα εφαρμοστούν τεχνικές χαμηλής κατανάλωσης με σκοπό τη μείωση της κατανάλωσης των δύο αρχικών κυκλωμάτων.

Όπως αναφέρθηκε και πριν η περιγραφή των κυκλωμάτων θα γίνει με τη γλώσσα περιγραφής υλικού VHDL. Το λογισμικό που χρησιμοποιήθηκε για την εξομοίωση των κυκλωμάτων αλλά και την εξομοίωση της κατανάλωση είναι το ModelSim XE III 6.2c . Το περιβάλλον που χρησιμοποιήθηκε για την ανάπτυξη του κώδικα είναι το λογισμικό ISE 12.3. Επίσης τα εργαλεία που χρησιμοποιήθηκαν για τις διαδικασίες σύνθεσης, υλοποίησης, ανάλυσης της κατανάλωσης και ο εικονικός λογικός αναλυτής ανήκουν στην ίδια οικογένεια λογισμικού.

Διάρθρωση της εργασίας

Η εργασία χωρίζεται σε δύο βασικά μέρη. Το πρώτο μέρος αποτελείται από τα τρία επόμενα κεφάλαια όπου και γίνεται μια εισαγωγή σε θέματα κρυπτογραφίας, παρουσιάζονται τεχνολογίες ολοκληρωμένων κυκλωμάτων και τέλος εξετάζονται οι τεχνικές χαμηλής κατανάλωσης. Το δεύτερο μέρος το οποίο αποτελείται από τα κεφάλαια 5 έως 7 όπου παρουσιάζονται οι αλγόριθμοι, οι υλοποίησης τους και τα αποτελέσματα των μετρήσεων που έγιναν πάνω σε αυτές. Ποιο αναλυτικά:

Το Κεφάλαιο 2 παρουσιάζει τις βασικές αρχές της κρυπτογραφίας, τα είδη των αλγορίθμων, την εξέλιξη της μέσω μιας ιστορικής αναδρομής αλλά και τις εφαρμογές στις οποίες χρησιμοποιείται.

Στο Κεφάλαιο 3 γίνεται μια εισαγωγή στις τεχνολογίες των ολοκληρωμένων κυκλωμάτων και την εξέλιξη τους. Ιδιαίτερη βαρύτητα δίνεται στην οικογένεια των προγραμματιζόμενων ολοκληρωμένων κυκλωμάτων στην οποία ανήκουν τα FPGA που χρησιμοποιήθηκαν στην εργασία αυτή. Επίσης γίνεται μια περεταίρω ανάλυση στα δύο FPGA του εμπορίου που χρησιμοποιήθηκαν για τις υλοποιήσεις των κυκλωμάτων της πτυχιακής.

Στο Κεφάλαιο 4 παρουσιάζονται βασικά στοιχεία κατανάλωσης στα ολοκληρωμένα κυκλώματα και αναλύονται οι βασικότερες σχεδιαστικές τεχνικές μείωσης της.

Στο Κεφάλαιο 5 παρουσιάζονται οι αλγόριθμοι Fugue και JH των οποίων γίνεται ανάλυση του τρόπου λειτουργίας τους. Αρχικά γίνεται μια εισαγωγή στο μαθηματικό υπόβαθρο των αλγορίθμων στη συνέχεια περιγράφονται αναλυτικά οι τεχνικές προδιαγραφές τους. Η παραπάνω ανάλυση περιορίζεται σε θέματα που αφορούν μόνο τη δομή τους και όχι το παρεχόμενο επίπεδο ασφάλειας.

Στο Κεφάλαιο 6 δίνονται οι υλοποίησης των αλγορίθμων και η περιγραφή το τρόπου σχεδίασης τους. Αρχικά αναλύεται πλήρως το κύκλωμα που πραγματοποιεί την λειτουργία του κάθε αλγορίθμου και στη συνέχεια παρουσιάζονται οι διαφοροποίησης αυτού κυκλώματος βάση των τεχνικών χαμηλής κατανάλωσης. Για τη περίπτωση μόνο του αλγορίθμου Fugue παρουσιάζονται και δύο επιπλέον αρχιτεκτονικές που στόχο έχουν τη μείωση της ταχύτητας εκτέλεσης.

Στο Κεφάλαιο 7 παρουσιάζονται και αναλύονται τα αποτελέσματα των μετρήσεων της κάθε αρχιτεκτονικής που υλοποιήθηκε. Επίσης δίνονται και τα αποτελέσματα των

εξομοιώσεων που έγιναν με σκοπό την διερεύνηση της ορθής λειτουργίας του κάθε κυκλώματος.

Στο τελευταίο κεφάλαιο της εργασίας γίνεται μια σύνοψη των συμπερασμάτων που βγήκαν μέσω της εργασίας καθώς και προτάσεις για περαιτέρω μελέτη.

Στο Παράρτημα Α δίνονται ένας πίνακας ο οποίος περιέχει όλα τα σχήματα που χρησιμοποιούνται στη παρούσα εργασία ενώ στο Παράρτημα Β δίνεται ο αντίστοιχος «πίνακας πινάκων». Στο Παράρτημα Γ δίνονται συνοπτικά τα δεδομένα εισόδου αλλά και τα αντίστοιχα αποτελέσματα για το κάθε αλγόριθμο τα οποία χρησιμοποιήθηκαν κατά τις εξομοιώσεις της εργασίας αυτής. Τα δεδομένα αυτά προέρχονται από τις επίσημες προδιαγραφές που κατέθεσαν οι σχεδιαστές τον αλγορίθμων στο διαγωνισμό συναρτήσεων κατακερματισμού του NIST. Τέλος, στο Παράρτημα Δ δίνεται ο κώδικας περιγραφής υλικού VHDL των δύο κυκλωμάτων αναφοράς κάθε αλγορίθμου καθώς επίσης σε ηλεκτρονική μορφή (CD) παρέχεται ολόκληρος ο κώδικας όλων των αρχιτεκτονικών που σχεδιάστηκαν.

ΚΕΦΑΛΑΙΟ 2

ΚΡΥΠΤΟΓΡΑΦΙΑ

2.1 Ιστορική αναδρομή

Η κρυπτογραφία έχει εντυπωσιακή ιστορία που φτάνει χιλιάδες χρόνια πίσω και χρησιμοποιείται για την ασφαλή ανταλλαγή μηνυμάτων μεταξύ δύο μερών. Η λέξη ***κρυπτογραφία*** (cryptography) προέρχεται από τις ελληνικές λέξεις «κρυφή γραφή» και ιστορικά χωρίζεται σε τρείς περιόδους [1][2].

Πρώτη περίοδος 1900 π.Χ. – 1900 μΧ.. Κατά την διάρκεια αυτής της περιόδου αναπτύχθηκε πλήθος μεθόδων και αλγορίθμων κρυπτογράφησης, που βασίζονταν κυρίως σε απλές αντικαταστάσεις γραμμάτων. Διάσημα κρυπτοσυστήματα της εποχής εκείνης ήταν η «Σκυτάλη των Λακεδαιμονίων» και μετέπειτα η μετατόπιση του Λατινικού αλφαβήτου κατά τρείς θέσεις των μεταδιδόμενων μηνυμάτων από τον Ιούλιο Καίσαρα.

Δεύτερη περίοδος 1900 μ.Χ. – 1950 μ.Χ.. Στην οποία αναπτύχθηκαν πιο πολύπλοκα κρυπτοσυστήματα τα οποία υλοποιούταν από μηχανικές και ηλεκτρομηχανικές κατασκευές, οι οποίες ονομάζονται «κρυπτομηχανές». Ένα τέτοιο διάσημο σύστημα ήταν η μηχανή «Αίνιγμα» (Enigma) η οποία χρησιμοποιήθηκε ευρέως από την Γερμανία κατά το δεύτερο παγκόσμιο πόλεμο.

Τρίτη περίοδος 1950 μ.Χ. – Σήμερα. Η έξαρση της ανάπτυξης των επιστημονικών κλάδων των μαθηματικών, της μικροηλεκτρονικής και των υπολογιστικών συστημάτων διαμόρφωσε την σύγχρονη κρυπτογραφία όπως τη γνωρίζουμε σήμερα. Στα μέσα της δεκαετίας του '70 έγινε η δημοσίευση του σχεδίου προτύπου κρυπτογράφησης DES (Data Encryption Standard) από την IBM στην πρόσκληση ανοικτού ενδιαφέροντος του Εθνικού Ιδρύματος Προτύπων και Τεχνολογίας (National Institute of Standards and Technology - NIST) για την ανάπτυξη ασφαλών ηλεκτρονικών εγκαταστάσεων επικοινωνίας. Ύστερα από κάποιες τροποποιήσεις το 1977 υιοθετήθηκε και δημοσιεύθηκε ως το πρώτο ομοσπονδιακό τυποποιημένο πρότυπο επεξεργασίας πληροφοριών.

Από την αρχαιότητα μέχρι και τη δεκαετία του 70 το σημαντικότερο ρόλο στη διαμόρφωση της κρυπτογραφίας είχε ο στρατός. Χάρη όμως στην απελευθέρωση της αποκλειστικής έρευνας της κρυπτογραφίας από το στρατό, προκλήθηκε τεράστιο δημόσιο και ακαδημαϊκό ενδιαφέρον για τα συστήματα κρυπτογραφίας.

2.2 Εισαγωγή στη κρυπτογραφία

Ως ***κρυπτογραφία*** μπορεί να ορισθεί ο επιστημονικός κλάδος που ασχολείται με τη μετατροπή των πληροφοριών, με σκοπό τη διαφύλαξη του απορρήτου τους [3]. Σκοπός της είναι να διασφαλίσει την ιδιωτικότητα ενός μηνύματος με το να κρατά την πληροφορία «κρυφή» από οποιοδήποτε άτομο, το οποίο δεν έχει ορισθεί ως αποδέκτης του μηνύματος ακόμα και εάν έχει πρόσβαση στα κρυπτογραφημένα δεδομένα.

Στην ορολογία της κρυπτογραφίας, το αρχικό μήνυμα που προορίζεται για κρυπτογράφηση ονομάζεται ***απλό κείμενο*** (plaintext ή cleartext). Η διαδικασία μετατροπής του περιεχομένου του μηνύματος σε μορφή τέτοια που να είναι μη κατανοητή για μη εξουσιοδοτημένους αποδέκτες ονομάζεται ***κρυπτογράφηση*** (encryption) ενώ η αντίστροφη διαδικασία κατά την οποία το κρυπτογραφημένο μήνυμα μετατρέπεται στο αρχικό ή απλό μήνυμα ονομάζεται ***αποκρυπτογράφηση*** (decryption). Το αποτέλεσμα της κρυπτογράφησης του αρχικού μηνύματος ονομάζεται ***κρυπτογράφημα*** (ciphertext). Η κρυπτογράφηση και η αποκρυπτογράφηση ενός μηνύματος ή ενός κρυπτογραφήματος πραγματοποιούνται με τη χρήση μαθηματικών συναρτήσεων και αλγορίθμων που ονομάζονται ***αλγόριθμοι κρυπτογράφησης*** οι οποίοι κατά τους μετασχηματισμούς, για την ολοκλήρωση του αποτελέσματος χρησιμοποιούν επιπλέον πληροφορία η οποία ονομάζεται ***κλειδί*** (key).

Τα κύρια μέρη που λαμβάνουν χώρα κατά τη διαδικασία της κρυπτογράφησης και αποκρυπτογράφησης είναι:

- Ο αλγόριθμος κρυπτογράφησης.
- Τα κλειδιά που χρησιμοποιούνται από τον αλγόριθμο για να καθοριστεί το πώς κωδικοποιούνται και αποκωδικοποιούνται τα δεδομένα.
- Το μήκος του κλειδιού, το οποίο όσο μεγαλύτερο είναι τόσο πιο δύσκολο είναι να βρεθεί μέσω μιας εξαντλητικής αναζήτησης, αφού υπάρχουν περισσότεροι συνδυασμοί κλειδιών για να ελεγχθούν.
- Το απλό κείμενο που θα κρυπτογραφηθεί.
- Το κρυπτογραφημένο κείμενο.

Οι υπηρεσίες που πρέπει να παρέχονται από τους αλγόριθμους κρυπτογράφησης είναι η εμπιστευτικότητα, η ακεραιότητα, η αυθεντικοποίηση, η προστασία από απάρνηση παραλαβής ή αποστολής πληροφοριών, ο έλεγχος πρόσβασης και διαθεσιμότητα δεδομένων και υπηρεσιών.

Εκτός από την κρυπτογραφία που έχει στόχο τη διατήρηση της μυστικότητας των μηνυμάτων υπάρχει και η ***κρυπτανάλυση*** (cryptanalysis) η οποία ορίζεται ως η «τεχνική» της παραβίασης του κρυπτογραφημένου μηνύματος, χωρίς να είναι απαραίτητα γνωστό το κλειδί της αποκρυπτογράφησης. Η συνένωση της κρυπτογραφίας και της κρυπτανάλυσης είναι γνωστή ως ***κρυπτολογία*** (cryptology).

2.3 Είδη αλγορίθμων

Η κρυπτογράφηση δεδομένων μπορεί να χρησιμοποιηθεί σε μια πληθώρα εφαρμογών όπως:

- Προστασία δεδομένων, αποθηκευμένων στον υπολογιστή, από μη εξουσιοδοτημένη πρόσβαση.
- Προστασία δεδομένων κατά τη μεταφορά τους ανάμεσα σε δύο υπολογιστικά συστήματα.
- Ανίχνευση τυχαίας ή εσκεμμένης αλλαγής σε δεδομένα.
- Πιστοποίηση της ταυτότητας του συντάκτη/εκδότη ενός κειμένου ή μηνύματος.

Για την επίτευξη των παραπάνω στόχων έχουν αναπτυχθεί διάφορα είδη αλγορίθμων κρυπτογράφησης όπως:

- Συμμετρικοί αλγόριθμοι ή ιδιωτικού κλειδιού (symmetric or secret key).
- Ασύμμετροι αλγόριθμοι ή δημόσιου κλειδιού (asymmetric or public key).
- Αλγόριθμοι σύνοψης μηνύματος (message digest) ή συναρτήσεις κατακερματισμού (hash functions).

Οι συμμετρικοί αλγόριθμοι χρησιμοποιούν το ίδιο κλειδί για την κρυπτογράφηση και αποκρυπτογράφηση το οποίο ονομάζεται ***ιδιωτικό κλειδί*** και χωρίζονται σε δύο κατηγορίες, στους αλγόριθμους τμήματος (Block Ciphers) και αλγόριθμους ροής (Stream Ciphers). Οι αλγόριθμοι ροής συνδυάζουν μια ψευδοτυχαία ροή δεδομένων (κλειδοροή) με το καθαρό κείμενο για τη δημιουργία του κρυπτοκειμένου. Οι αλγόριθμοι τμήματος επεξεργάζονται το αρχικό μήνυμα σε τμήματα (blocks) και η διαδικασία ολοκληρώνεται μετά από κάποιες επαναλήψεις μιας βασικής δομής που ονομάζεται γύρος (round). Στην οικογένεια των αλγορίθμων τμήματος υπάρχουν τρόποι λειτουργίας για τη διασύνδεση των αλγορίθμων τμήματος που έχει ως αποτέλεσμα την αύξηση της πολυπλοκότητας και της κρυπτογραφικής δύναμης.

Στην κρυπτογραφία δημόσιου κλειδιού ο κάθε χρήστης έχει ένα ζευγάρι κλειδιών εκ των οποίων το ένα χρησιμοποιείται για την κρυπτογράφηση και το άλλο για την αποκρυπτογράφηση. Τα κλειδιά αυτά ονομάζονται ***δημόσιο*** (public key) και ***ιδιωτικό κλειδί*** (private key) αντίστοιχα. Σε αυτή τη περίπτωση ο αποστολέας κρυπτογραφεί το μήνυμα με το δημόσιο κλειδί του παραλήπτη και στη συνέχεια ο παραλήπτης αποκρυπτογραφεί το μήνυμα με το αντίστοιχο ιδιωτικό του κλειδί. Οι αλγόριθμοι δημόσιου κλειδιού λύνουν κυρίως το πρόβλημα της ανταλλαγής του ιδιωτικού κλειδιού στη συμμετρική κρυπτογράφηση.

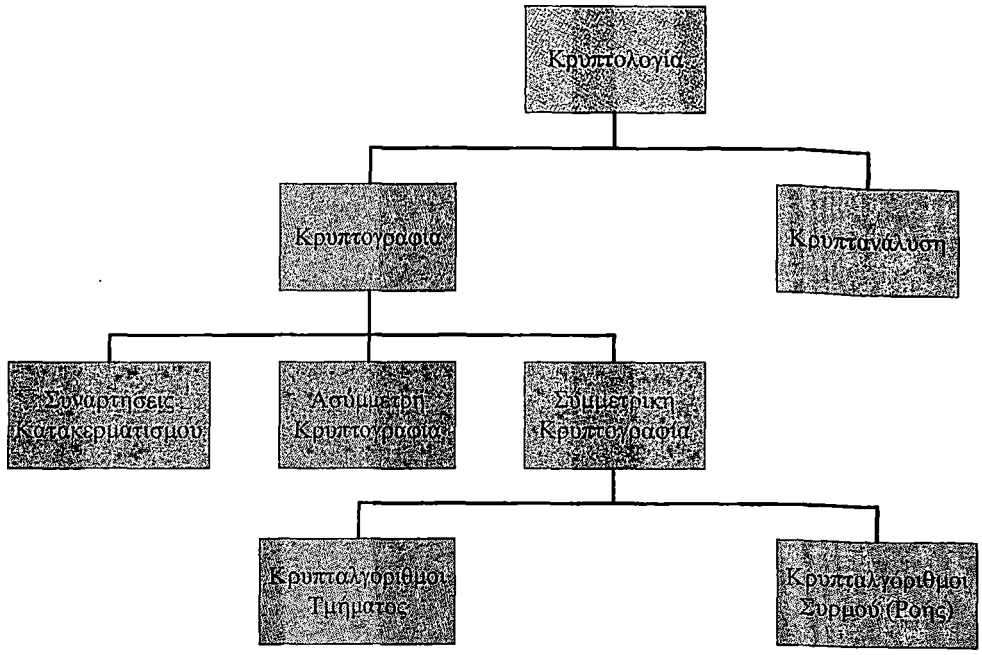
Οι συναρτήσεις κατακερματισμού χρησιμοποιούνται κυρίως για την ανίχνευση αλλαγών σε δεδομένα αλλά και για τη πιστοποίηση της ταυτότητας του συντάκτη. Αυτό που εξασφαλίζουν παράγοντας τη σύνοψη του αρχικού μηνύματος είναι ότι κατά τη διάρκεια της μετάδοσης αν αλλάξει τυχαία ή εσκεμμένα το περιεχόμενο του μηνύματος ο παραλήπτης θα είναι σε θέση να το γνωρίζει παράγοντας και ο ίδιος τη σύνοψη του αρχικού μηνύματος που θα λάβει.. Η παραπάνω διαδικασία προϋποθέτει τη κρυπτογράφηση μόνο της σύνοψης και όχι όλου του αρχικού μηνύματος. Στο Σχήμα 2.1 δίνεται η σχέση μεταξύ των παραπάνω ειδών αλγορίθμων κρυπτογράφησης.

2.4 Εφαρμογές και υλοποιήσεις αλγορίθμων κρυπτογράφησης

Η κρυπτογραφία στις μέρες μας χρησιμοποιείται σε μια πληθώρα εφαρμογών. Κρυπτογράφηση δεδομένων συναντάμε στις κινητές και δορυφορικές επικοινωνίες όπου τα δεδομένα πριν μεταφερθούν στο ασύρματο κανάλι επικοινωνίας κρυπτογραφούνται για την εξασφάλιση της ασφαλής επικοινωνίας μεταξύ ατόμων ή συστημάτων.

Στη προστασία πνευματικών δικαιωμάτων έχει παίξει σημαντικό ρόλο η ανάπτυξη της κρυπτογραφίας καθώς παρέχει κωδικοποίηση υδατογραφημάτων, οπτικών δίσκων, ψηφιακές υπογραφές κ.α. Στον ευαίσθητο τομέα των ηλεκτρονικών συναλλαγών και ηλεκτρονικού εμπορίου η κρυπτογραφία συναντάται σε συστήματα αυτόματων αναλήψεων, σε χρηματοπιστωτικές κάρτες έως και συστήματα συναγερμών. Επίσης έχει πολύ σημαντικό ρόλο και μεγάλη προσφορά στην ασφάλεια των δικτύων υπολογιστών και υπολογιστικών συστημάτων. Στα δίκτυα υπολογιστών παρέχει μηχανισμούς ασφάλειας σε επίπεδο πρωτοκόλλων είτε σε ενσύρματα είτε σε ασύρματα δίκτυα καθώς επίσης στα υπολογιστικά συστήματα προστασίας των βάσεων δεδομένων και των δικτυακών εφαρμογών.

Από τις παραπάνω εφαρμογές καθίσταται σαφές ότι οι αλγόριθμοι κρυπτογράφησης πρέπει να παρέχουν υψηλό ποσοστό ασφάλειας στους εκάστοτε χρήστες. Επίσης οι



Σχήμα 2.1: *Το μοντέλο της κρυπτογραφίας.*

υλοποιήσεις των αλγορίθμων κρυπτογράφησης πρέπει να συμβαδίζουν με τις τεχνολογίες των παραπάνω εφαρμογών, λόγω του ότι ο τομέας των υπολογιστών και τηλεπικοινωνιών αναπτύσσεται ραγδαία παρέχοντας ολοένα και μεγαλύτερες ταχύτητες μετάδοσης και επεξεργασίας δεδομένων.

Παραδοσιακά οι αλγόριθμοι κρυπτογράφησης υλοποιούνται σε λογισμικό το οποίο εγκαθίσταται είτε σε ειδικά σχεδιασμένους μικροεπεξεργαστές είτε σε επεξεργαστές γενικού σκοπού. Αυτή η λύση είναι οικονομική αλλά έχει το μειονέκτημα της μικρής ταχύτητας εκτέλεσης των συστημάτων καθώς αυξάνεται η πολυπλοκότητα των αλγορίθμων και το μέγεθος των κλειδιών που χρησιμοποιούνται. Η λύση της υλοποίησης των αλγορίθμων απευθείας σε υλικό μοιάζει πιο ελκυστική αφού προσφέρει πολύ μεγαλύτερες ταχύτητες εκτέλεσης σε σχέση με το λογισμικό. Σε αυτό έχει συμβάλει η ανάπτυξη και βελτιστοποίηση των προγραμματιζόμενων ολοκληρωμένων κυκλωμάτων (Programmable Integrated Circuits - PIC) τη τελευταία δεκαετία, καθώς μειώνεται το κόστος του υλικού το οποίο γενικά είναι μεγαλύτερο σε σχέση με το λογισμικό.

ΑΝΑΦΟΡΕΣ

[1] Andrew S. Tanenbaum, Δίκτυα Υπολογιστών (Τέταρτη Αμερικάνικη Έκδοση).

[2] http://el.wikipedia.org/wiki/Κρυπτογραφία.

[3] Πομπόσης Ανδρέας, Παπαδημητρίου Γεώργιος, Ασφάλεια Δικτύων Υπολογιστών.

ΚΕΦΑΛΑΙΟ 3

ΤΕΧΝΟΛΟΓΙΕΣ ΟΛΟΚΛΗΡΟΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ

3.1 Εισαγωγή

Από τα μέσα του προηγούμενου αιώνα, όπου ξεκινάει η ανάπτυξη των ***ολοκληρωμένων κυκλωμάτων*** (Integrated Circuit - IC) μέχρι και σήμερα έχουν αναπτυχθεί ποικίλες τεχνολογίες υλοποίησης αυτών των κυκλωμάτων που ανταποκρίνονται σε ένα ευρύ φάσμα εφαρμογών. Τα ολοκληρωμένα κυκλώματα κατηγοριοποιούνται κυρίως από το επίπεδο ολοκλήρωσης τους, δηλαδή τον αριθμό των πυλών που τα αποτελούν και από τη τεχνολογία υλοποίησης των τρανζίστορ τους. Τα επίπεδα ολοκλήρωσης διακρίνονται σε:

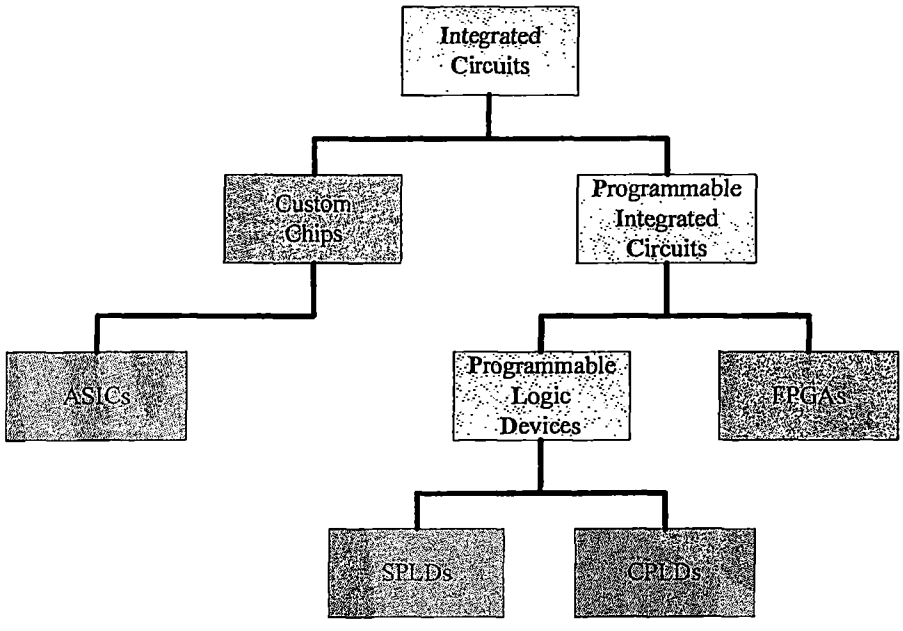
- Μικρό επίπεδο ολοκλήρωσης (Small Scale Integration - SSI) χαρακτηρίζεται το κύκλωμα με λιγότερες από 10 πύλες.
- Μεσαίο επίπεδο ολοκλήρωσης (Medium Scale Integration - MSI) χαρακτηρίζεται το κύκλωμα που περιέχει από 10 έως 100 πύλες.
- Μεγάλο επίπεδο ολοκλήρωσης (Large Scale Integration - LSI) χαρακτηρίζεται το κύκλωμα που περιέχει από 100 έως 10,000 πύλες.
- Πολύ μεγάλο επίπεδο ολοκλήρωσης (Very Large Scale Integration - VLSI) χαρακτηρίζεται το κύκλωμα που περιέχει από 10,000 έως 100,000 πύλες.
- Υπέρ-μεγάλο επίπεδο ολοκλήρωσης (Ultra Large Scale Integration - ULSI) χαρακτηρίζεται το κύκλωμα που περιέχει από 100,000 πύλες και άνω.

Οι οικογένειες τεχνολογιών των τρανζίστορ χωρίζονται σε τρείς βασικές, τα MOSFET, TTL και τα BiCMOS που είναι ο συνδυασμός των προηγούμενων [1]. Τα MOSFET τρανζίστορ υποδιαιρούνται και αυτά σε δύο κατηγορίες, στα NMOS και PMOS τρανζίστορ τα οποία συνδυάζοντας τα υλοποιείται το ευρέος διαδεδομένο CMOS τρανζίστορ. Τα SSI και MSI κυκλώματα μπορούν να υλοποιηθούν είτε από CMOS είτε από TTL τρανζίστορ αλλά οι κατηγορίες LSI,VLSI και ULSI υλοποιούνται από CMOS ή NMOS τρανζίστορ και ο λόγος είναι ότι αυτές οι τεχνολογίες απαιτούν λιγότερο χώρο και κατανάλωση ενέργειας πάνω στο ολοκληρωμένο κύκλωμα.

Μία άλλη διάκριση που γίνεται μεταξύ των ολοκληρωμένων κυκλωμάτων είναι σε αυτά που κατά τη κατασκευή τους σχεδιάζονται οι λειτουργίες που θα εκτελούν και μετέπειτα ο εκάστοτε χρήστης δεν μπορεί να αλλάξει τη λειτουργία του ολοκληρωμένου κυκλώματος και ονομάζονται ***ολοκληρωμένα κυκλώματα ειδικού σκοπού*** (Custom Chips ή Fixed

Function Logic) [2]. Η άλλη μεγάλη κατηγορία είναι τα κυκλώματα στα οποία ο χρήστης καθορίζει τον τρόπο λειτουργίας τους και ονομάζονται ***προγραμματιζόμενα ολοκληρωμένα κυκλώματα*** (Programmable Integrated Circuits - PIC).

Στο Σχήμα 3.1 φαίνονται συνοπτικά αυτές οι δύο μεγάλες οικογένειες τεχνολογιών και οι υποκατηγορίες τους οι οποίες θα παρουσιαστούν στα επόμενα κεφάλαια.



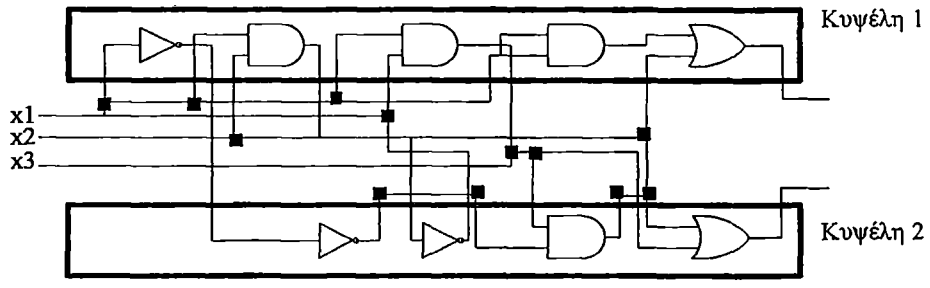
Σχήμα 3.1: *Τεχνολογίες ολοκληρωμένων κυκλωμάτων.*

3.2 Ολοκληρωμένα κυκλώματα ειδικού σκοπού

Τα ειδικά ολοκληρωμένα κυκλώματα αποτελούν τη πιο κλασσική εκδοχή ολοκληρωμένων. Δημιουργούνται έτσι ώστε ο σχεδιαστής του είναι αυτός που θα αποφασίσει για το μέγεθος του κυκλώματος, τον αριθμό των τρανζίστορ που θα περιέχει, τη τοποθέτησης του κάθε τρανζίστορ μέσα στο κύκλωμα και το τρόπο σύνδεσης των τρανζίστορ. Λόγω του ότι το κύκλωμα μπορεί να περιέχει περισσότερα από ένα εκατομμύριο τρανζίστορ απαιτείται πολύ μεγάλη σχεδιαστική προσπάθεια. Μια καλύτερη εκδοχή αυτών των ολοκληρωμένων, όσον αφορά το στάδιο της σχεδίασης, είναι τα ***ολοκληρωμένα κυκλώματα ειδικού σκοπού*** (Application Specific Integrated Circuits - ASIC). Σε αυτή τη τεχνολογία χρησιμοποιούνται τυποποιημένες κυψέλες (Standard Cells) πυλών και ο σχεδιαστής ασχολείται με τη κατάλληλη χρήση αυτών για την υλοποίηση των λογικών συναρτήσεων. Ένα πλεονέκτημα

που προκύπτει από αυτή τη τεχνολογία είναι ότι η οργάνωση των κυψελών μπορεί να πραγματοποιηθεί αυτόματα με τη χρήση προγραμμάτων σχεδίασης.

Στο σχήμα 3.2 φαίνονται δύο γραμμές κυψελών διασυνδεμένες για την υλοποίηση κάποιας λογικής συνάρτησης. Αυτή η οργάνωση χρησιμοποιείται επειδή καθιστά δυνατή την τοποθέτηση γραμμών σύνδεσης σε πολλαπλά επίπεδα και επομένως μπορούν δύο γραμμές να διασταυρώνονται χωρίς να προκαλείται βραχυκύκλωμα. Για παράδειγμα στο Σχήμα 3.2 βραχυκύκλωμα στις γραμμές διασύνδεσης, όταν διασταυρώνονται, υπάρχει μόνο στα τετράγωνα.



Σχήμα 3.2: *Δύο γραμμές κυψελών συνδεδεμένες σε ASIC.*

Έτσι μπορούν να τοποθετηθούν πολλαπλά επίπεδα κυψελών ακόμα και πάνω από τα τρανζίστορ κάνοντας πιο αποδοτική την οργάνωση του κυκλώματος.

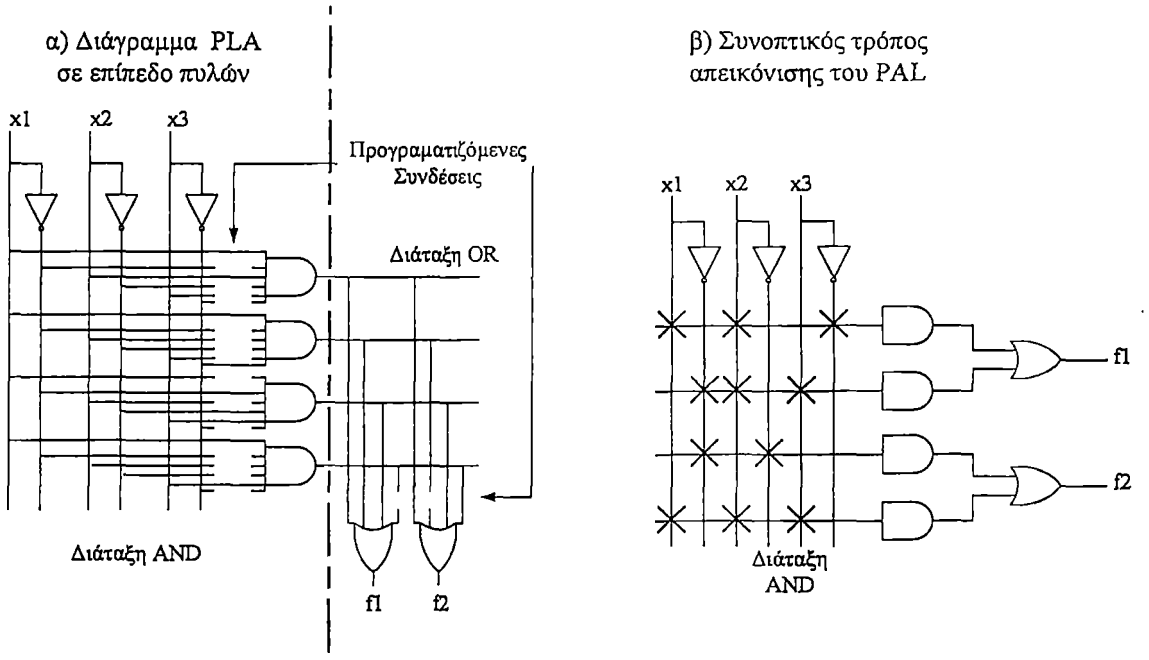
Όπως και στα απλά ειδικά ολοκληρωμένα κυκλώματα έτσι και τα κυκλώματα ASIC δημιουργούνται από την αρχή σύμφωνα με τις ανάγκες του χρήστη και χωρίς τη δυνατότητα της μετέπειτα τροποποίησης τους. Επομένως η υλοποίηση κάποιου κυκλώματος σε τεχνολογία ειδικών ολοκληρωμένων κυκλωμάτων γενικότερα, είναι χρονοβόρα και δαπανηρή οικονομικά διαδικασία. Είναι εύλογο λοιπόν το ότι τέτοιες τεχνικές χρησιμοποιούνται μόνο στις περιπτώσεις όπου απαιτείται πάρα πολύ μεγάλος αριθμός τρανζίστορ και πολύ υψηλές ταχύτητες που δεν μπορούν να επιτύχουν οι άλλες δύο οικογένειες ολοκληρωμένων κυκλωμάτων που θα εξεταστούν στα επόμενα κεφάλαια, τα PLDs και FPGAs.

3.3 Διατάξεις προγραμματιζόμενης λογικής

Οι *συσκευές προγραμματιζόμενης λογικής* (Programmable Logic Devices - PLD) είναι ολοκληρωμένα κυκλώματα γενικής χρήσης που μπορούν να υλοποιήσουν διάφορα λογικά κυκλώματα. Αποτελούνται από λογικά στοιχεία που ανάλογα τις απαιτήσεις του χρήστη μπορούν να οργανώνονται με διάφορους τρόπους. Ουσιαστικά τα PLDs είναι ένα σύνολο διατάξεων λογικών πυλών και προγραμματιζόμενων διακοπτών που ανάλογα το κύκλωμα που επιθυμεί ο χρήστης συνδέονται κατάλληλα για να το υλοποιήσουν. Στα σύγχρονα PLDs η υλοποίηση των προγραμματιζόμενων διακοπτών γίνεται με τη χρήση μνημών μόνο ανάγνωσης τύπου EEPROM (Electrically Erasable Programmable Read Only Memory). Κάτι το οποίο τους δίνει το πλεονέκτημα να συμπεριφέρονται μη πτητικά δηλαδή να διατηρούν τις διατάξεις πυλών και χωρίς τη τροφοδοσία ρεύματος. Τα PLDs διακρίνονται σε δύο κατηγορίες ανάλογα με τη πολυπλοκότητα τους, στις απλές συσκευές προγραμματιζόμενης λογικής (Simple Programmable Logic Devices - SPLD) και στις σύνθετες συσκευές προγραμματιζόμενης λογικής (Complex Programmable Logic Devices - CPLD).

Η πρώτη μορφή SPLD που αναπτύχθηκε ήταν η προγραμματιζόμενη λογική πίνακα (Programmable Logic Array - PLA) και λίγο αργότερα λόγω της δυσκολίας κατασκευής του αναπτύχθηκε άλλο ένα SPLD παρόμοιο με το PLA το οποίο ονομάζεται προγραμματιζόμενος πίνακας λογικής (Programmable Array Logic - PAL). Τα δύο αυτά ολοκληρωμένα διαφέρουν μόνο στο ότι στα PAL ο χρήστης έχει τη δυνατότητα να προγραμματίζει ένα μέρος των πυλών που του παρέχονται και το άλλο μένει σταθερό.

Γενικότερα η τεχνολογία των PLD εκμεταλλεύεται το γεγονός ότι όλες οι λογικές συναρτήσεις μπορούν να υλοποιηθούν ως αθροίσματα γινομένων. Όπως φαίνεται στο Σχήμα 3.3 το PLA αποτελείται από δύο διατάξεις πυλών. Στην αριστερή διάταξη πυλών υπάρχουν πύλες AND οι οποίες συνδέονται με τις εισόδους του PLA και τα συμπληρώματα τους. Στη δεξιά διάταξη υπάρχουν πύλες OR που ως είσοδο δέχονται την έξοδο της διάταξης AND. Στο στάδιο πριν το προγραμματισμό του κυκλώματος όλες οι πύλες συνδέονται μεταξύ τους. Μετά τον προγραμματισμό των διακοπτών που υπάρχουν σε κάθε διασταύρωση γραμμών προκύπτει το αποτέλεσμα του Σχήματος 3.3(α) στο οποίο οι διακόπτες που είναι κλειστοί υλοποιούν τη συνάρτηση $f1 = x1\,x2 + x1\,\overline{x3} + \overline{x1}\,\overline{x\,2}\,x3$ και $f2 = \chi1\,x2 + \overline{x1}\,\overline{\chi2}\,\chi3 + \chi1\,x3$. Στο Σχήμα 3.3(β) φαίνεται ένα παράδειγμα κυκλώματος PAL με συνοπτικό τρόπο απεικόνισης. Οι διασταυρώσεις που σημειώνονται με «Χ» σε κάθε γραμμή είναι οι είσοδοι

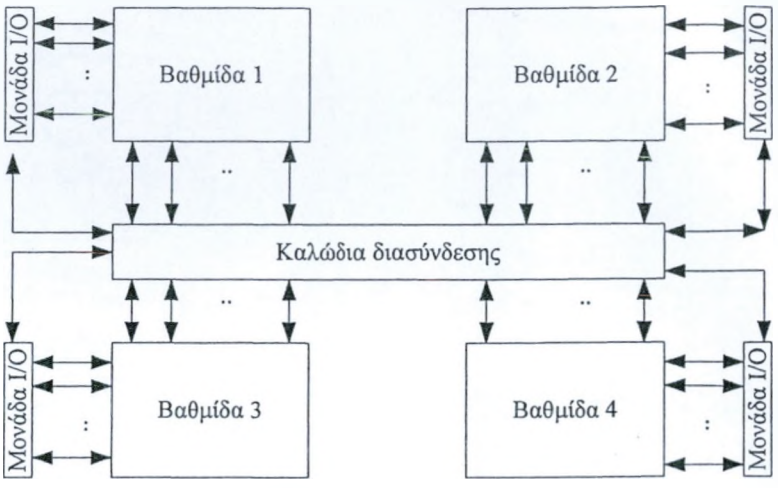


Σχήμα 3.3: *Λογικά διαγράμματα των PLA και PAL. α) PLA αριστερά, β) PAL δεξιά.*

για τη κάθε πύλη AND της αντίστοιχης γραμμής. Το συγκεκριμένο PAL υλοποιεί τη συνάρτηση $f1 = x1\,x2\,\overline{\chi 3} + \overline{x1}\,\chi 2\,\chi 3$ και $f2 = \overline{\chi 1}\,\overline{\chi 2} + \chi 1\,\chi 2\,\chi 3$. Επίσης στα σχήματα φαίνεται και η διαφορά των δύο ολοκληρωμένων, στο PLA διατίθενται προς προγραμματισμό και οι δύο διατάξεις σε αντίθεση με το PAL που προγραμματίζεται μόνο η διάταξη AND.

Μια ακόμα διαφορά που έχουν είναι ότι αρκετές φορές οι έξοδοι των διατάξεων OR των PAL συνδέονται με ένα στοιχείο μνήμης και δίνεται η δυνατότητα είτε για απευθείας έξοδο είτε για έξοδο μέσω του στοιχείου μνήμης καθώς επίσης η έξοδος ανατροφοδοτείται στη διάταξη AND. Αυτό προσφέρει ευελιξία στο κύκλωμα και καθίσταται δυνατή η υλοποίηση κυκλωμάτων πολλαπλών σταδίων.

Τα SPLDs κατασκευάζονται εύκολα και είναι χρήσιμα σε εφαρμογές μικρής πολυπλοκότητας. Για πιο σύνθετα κυκλώματα χρησιμοποιούνται τα CPLDs τα οποία αποτελούνται από αρκετές λογικές βαθμίδες που μοιάζουν με PAL με τη διαφορά ότι κάθε βαθμίδα αποτελείται από αρκετές κυψέλες τύπου PAL αλλά και ότι κάθε κυψέλη μπορεί να λειτουργεί ως είτε είσοδος είτε ως έξοδος. Στο Σχήμα 3.4 φαίνεται η αρχιτεκτονική ενός CPLD το οποίο αποτελείται από τέσσερις βαθμίδες που συνδέονται μεταξύ τους με εσωτερικές καλωδιώσεις. Κάθε τέτοια βαθμίδα συνδέεται επίσης με ένα υποκύκλωμα που ονομάζεται βαθμίδα εισόδου/εξόδου και προσαρμόζεται σε έναν αριθμό ακροδεκτών εισόδου



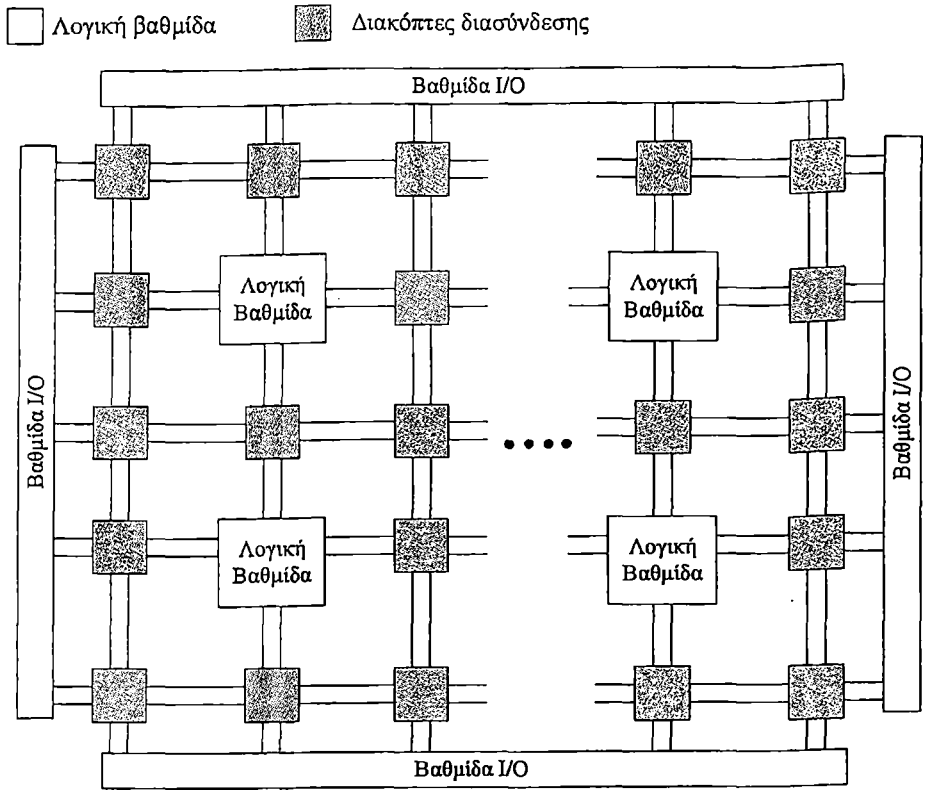
Σχήμα 3.4: *Αρχιτεκτονική ενός CPLD.*

και εξόδου του ολοκληρωμένου κυκλώματος. Οι εσωτερικές καλωδιώσεις περιέχουν προγραμματιζόμενους διακόπτες, οι οποίοι χρησιμοποιούνται για να συνδέσουν μεταξύ τους τις βαθμίδες. Κάθε ένα από τα οριζόντια καλώδια μπορεί να συνδεθεί με κάποια από τα κατακόρυφα καλώδια, με τα οποία τέμνεται, αλλά όχι με όλα.

Τα CPLD του εμπορίου έχουν μεγέθη που κυμαίνονται μεταξύ δύο βαθμίδων και περισσότερων από εκατό και χρησιμοποιούνται σε μια ευρεία γκάμα ψηφιακών κυκλωμάτων. Γενικότερα προτιμούνται στις περισσότερες των περιπτώσεων σε αντίθεση με τα SPLDs που χρησιμοποιούνται σε περιορισμένες εφαρμογές.

3.4 Δομές πίνακα πυλών επαναδιατάξιμης λογικής

Τα κυκλώματα προγραμματιζόμενης λογικής που έχουν αναφερθεί μέχρι στιγμής είναι ικανοποιητικά για την υλοποίηση μικρού και μεσαίου μεγέθους εφαρμογές. Όταν υπάρχει απαίτηση για μεγαλύτερες εφαρμογές τότε χρησιμοποιείται άλλη οικογένεια ολοκληρωμένων κυκλωμάτων η οποία ονομάζεται **δομές πίνακα πυλών επαναδιατάξιμης λογικής** (Field Programmable Gate Arrays - FPGA). Τα FPGAs διαφέρουν σημαντικά από τα CPLD και SPLD επειδή δεν περιέχουν πύλες AND και OR. Αντίθετα, τα FPGAs περιέχουν λογικές βαθμίδες για την υλοποίηση των ζητουμένων συναρτήσεων. Η γενική δομή ενός FPGA φαίνεται στο Σχήμα 3.5. Αυτή περιέχει τέσσερα είδη πόρων υλικού: λογικές βαθμίδες, βαθμίδες εισόδου/εξόδου για την διασύνδεση με τους ακροδέκτες της συσκευασίας, γραμμές

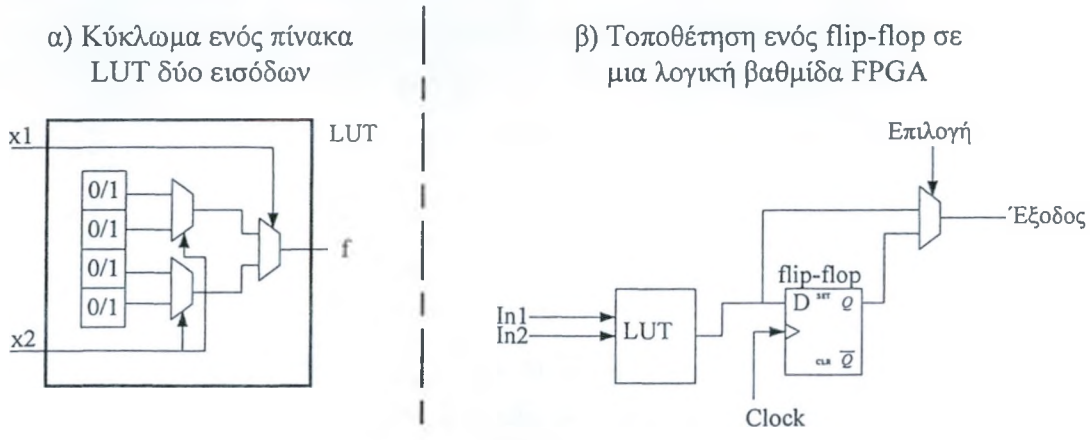


Σχήμα 3.5: *Γενική δομή ενός FPGA.*

για την εσωτερική διασύνδεση και διακόπτες. Οι λογικές βαθμίδες οργανώνονται με τη μορφή δισδιάστατης σειράς και οι γραμμές διασύνδεσης οργανώνονται ως οριζόντια και κατακόρυφα κανάλια δρομολόγησης ανάμεσα στις γραμμές και στήλες των λογικών βαθμίδων. Τα κανάλια αυτά εμπεριέχουν καλώδια και προγραμματιζόμενους διακόπτες που επιτρέπουν τις λογικές βαθμίδες να διασυνδέονται με πολλούς τρόπους. Στο Σχήμα 3.5 τα τετράγωνα που βρίσκονται δίπλα στις λογικές βαθμίδες περιέχουν διακόπτες που συνδέουν τους ακροδέκτες εισόδου και εξόδου των λογικών βαθμίδων με τα κανάλια διασύνδεσης και τα τετράγωνα που βρίσκονται διαγώνια μεταξύ των λογικών βαθμίδων συνδέουν ένα καλώδιο διασύνδεσης με ένα άλλο. Υπάρχουν επίσης προγραμματιζόμενες συνδέσεις ανάμεσα στις βαθμίδες εισόδου και εξόδου και τα καλώδια διασύνδεσης.

Υπάρχουν διάφορα είδη λογικών βαθμίδων αλλά η πιο ευρέως χρησιμοποιημένη λογική βαθμίδα είναι ο πίνακας αναφοράς (LookUp Table - LUT), ο οποίος περιέχει κυψέλες αποθήκευσης που χρησιμοποιούνται για την υλοποίηση μιας μικρής συνάρτησης. Κάθε κυψέλη μπορεί να κρατήσει μια λογική τιμή, 0 ή 1. Η αποθηκευμένη τιμή μεταφέρεται στην

έξοδο της κυψέλης αποθήκευσης. Το μέγεθος του πίνακα LUT είναι θέμα της κατασκευάστριας εταιρίας. Στο Σχήμα 3.6(α) φαίνεται η δομή ενός μικρού LUT. Αυτός



Σχήμα 3.6: *Η δομή μιας λογικής βαθμίδας. α) Αριστερά, κύκλωμα ενός πίνακα LUT, β) δομή μιας λογικής βαθμίδας.*

διαθέτει δύο εισόδους, x1 και x2 και μια έξοδο f, και έχει τη δυνατότητα να υλοποιεί οποιαδήποτε λογική συνάρτηση δύο μεταβλητών. Οι μεταβλητές εισόδου x1 και x2 χρησιμοποιούνται ως είσοδοι επιλογής τριών πολυπλεκτών, οι οποίοι ανάλογα με τις τιμές των x1 και x2, επιλέγουν το περιεχόμενο μιας από τις τέσσερις κυψέλες ως έξοδο του πίνακα LUT.

Στα FPGAs σε κάθε λογική βαθμίδα εκτός από το πίνακα LUT περιλαμβάνονται επιπλέον κυκλώματα. Αυτά είναι στοιχεία μνήμης και στο Σχήμα 3.6(β) φαίνεται ο τρόπος που μπορεί να περιληφθεί ένα flip-flop στη λογική βαθμίδα. Ο σκοπός και εδώ είναι να μπορεί να αποθηκεύεται η τιμή της εξόδου του LUT υπό την οδήγηση ενός ωρολογιακού σήματος.

Για την υλοποίηση ενός κυκλώματος θα πρέπει η κάθε λογική συνάρτηση του κυκλώματος να είναι αρκετά μικρή, ώστε να χωρά μέσα σε μια μοναδική λογική βαθμίδα. Στη πράξη, το κύκλωμα ενός χρήστη μεταφράζεται αυτόματα στη ζητούμενη μορφή με τη βοήθεια των εργαλείων σχεδίασης. Όταν ένα κύκλωμα υλοποιείται με τη βοήθεια ενός FPGA τότε οι λογικές βαθμίδες προγραμματίζονται για να υλοποιούν τις αναγκαίες συναρτήσεις και τα κανάλια δρομολόγησης προγραμματίζονται για να εκτελούν τις απαραίτητες διασυνδέσεις ανάμεσα στις λογικές βαθμίδες. Οι κυψέλες αποθήκευσης των πινάκων LUT

είναι πτητικές, που σημαίνει ότι χάνουν τα δεδομένα που περιέχουν εάν διακοπεί η τροφοδοσία του ολοκληρωμένου κυκλώματος. Κάτι το οποίο οφείλεται στο ότι τα FPGA για την υλοποίηση των διακοπτών τους χρησιμοποιών *στατική μνήμη τυχαίας προσπέλασης* (Static Random Access Memory - SRAM), όπου το κάθε στοιχείο μνήμης ελέγχει τον ακροδέκτη πύλης ενός τρανζίστορ. Επίσης στοιχεία μνήμης τύπου SRAM χρησιμοποιούνται και για την υλοποίηση των πινάκων LUT. Επομένως το FPGA πρέπει να προγραμματίζεται κάθε φορά που εφαρμόζεται τροφοδοσία στο κύκλωμα ή να υπάρχει κάποια εξωτερική μνήμη για τη μόνιμη αποθήκευσης των δεδομένων διάταξης.

3.5 Ανάλυση δύο FPGAs του εμπορίου

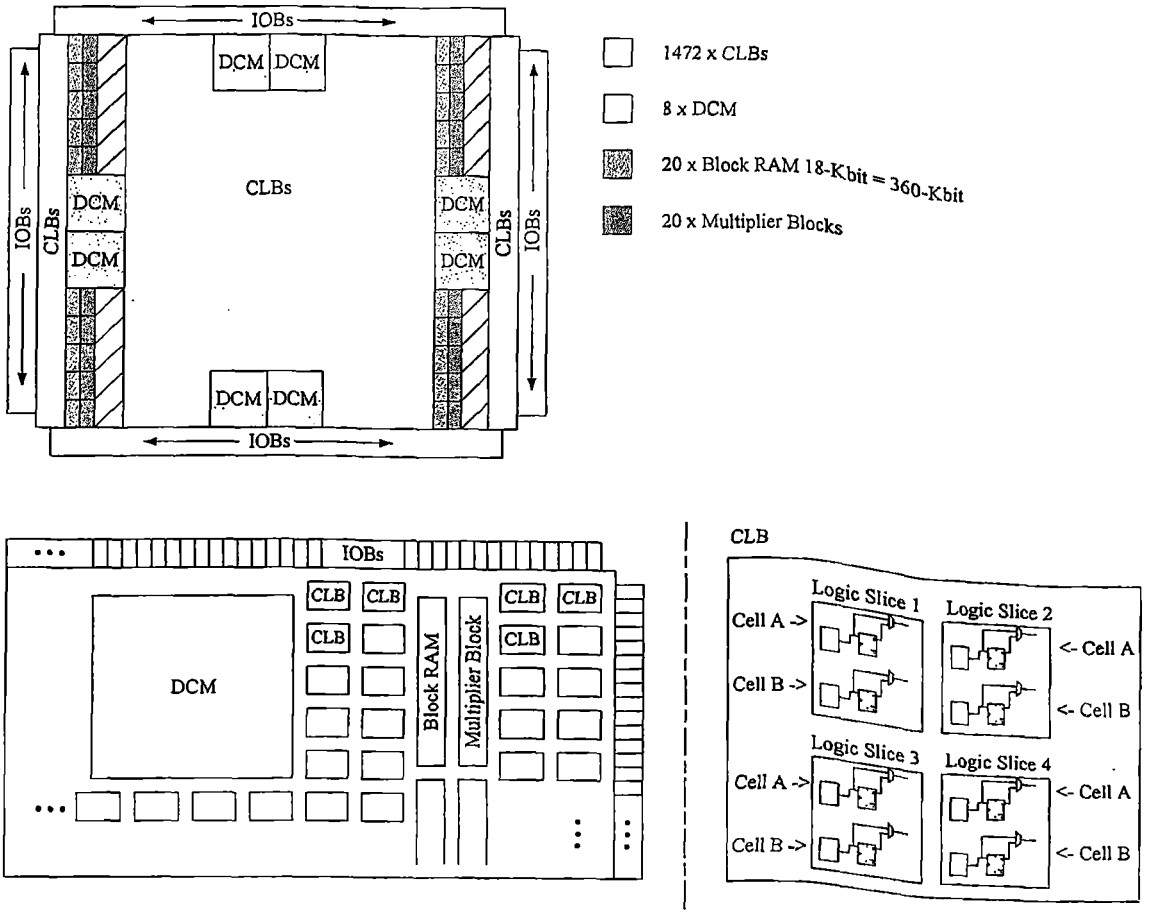
Για την υλοποίηση των κυκλωμάτων της παρούσας εργασίας χρησιμοποιήθηκαν δύο συγκεκριμένα FPGA κυκλώματα, τα οποία κατασκευάζονται από την εταιρία Xilinx και ανήκουν σε δύο μεγάλες οικογένειες FPGA. Το πρώτο ανήκει στην οικογένεια Spartan3A και είναι το μοντέλο XC3S700A-4FG484 [3] και το δεύτερο ανήκει στην οικογένεια Virtex5 και είναι το μοντέλο XC5VLX330-2FF1760 [4].

Spartan3A XC3S700A-4FG484

Η δομή και η οργάνωση ενός FPGA του εμπορίου διαφοροποιείται λίγο από το Σχήμα 3.5 . Λόγω του ότι το κάθε FPGA εμπεριέχει χιλιάδες λογικές βαθμίδες αυτές οργανώνονται σε *λογικά τμήματα* (Logical Slices). Το κάθε λογικό τμήμα στο συγκεκριμένο μοντέλο αποτελείται από δύο λογικές βαθμίδες τεσσάρων εισόδων η κάθε μια. Στη συνέχεια τέσσερα λογικά τμήματα υλοποιούν τη πιο αφηρημένη μονάδα του FPGA, το διαμορφώμενο δομικό τμήμα (Configurable Logic Block - CLB). Στο Σχήμα 3.7 φαίνεται η διάταξη του συγκεκριμένου μοντέλου καθώς και κάποια επιπλέον κυκλώματα που κάνουν το FPGA πιο ευέλικτο, όπως τμήματα μνήμης (Block RAM), τμήματα πολλαπλασιαστών (Multiplier Blocks), τμήματα διαχείρισης ψηφιακών ωρολογιακών σημάτων (Digital Clock Manager - DCM blocks) και βαθμίδες εισόδου/εξόδου.

Από το όνομα του συγκεκριμένου FPGA προκύπτει ότι μπορεί να υλοποιήσει κυκλώματα μεγέθους εφτακοσίων χιλιάδων πυλών (700K). Η παύλα και ο αριθμός τέσσερα (-4) στο όνομα του συμβολίζει τη ταχύτητα του συγκεκριμένου FPGA. Αυτός ο συμβολισμός στο όνομα χρησιμοποιείται από την εταιρία Xilinx και πρακτικά σημαίνει ότι όσο μεγαλύτερος είναι ο αριθμός που έπεται της παύλας τόσο πιο γρήγορο είναι το FPGA. Και τέλος το fg484 συμβολίζει το τύπο της συσκευασίας και τον αριθμών των εισόδων/εξόδων.

Η δομή του Spartan3A XC3S700A-4FG484



Σχήμα 3.7: *Η δομή του XC3S700A-4FG484 και ένα CLB.*

Ο αριθμός των εισόδων/εξόδων είναι 484 εκ των οποίων ο χρήστης μπορεί να χρησιμοποιήσει τα 372 καθώς τα υπόλοιπα χρησιμοποιούνται από το FPGA για τη μεταφορά σημάτων τροφοδοσίας, προγραμματισμού και ελέγχου του FPGA.

Ο Πίνακας 3.1 συνοψίζει τα κύρια χαρακτηριστικά του συγκεκριμένου FPGA. Τα 92K κατανεμημένης μνήμης που φαίνονται στο πίνακα υλοποιούνται από τους πίνακες αναφοράς των CLBs. Τα 360K κύριας μνήμης, οργανώνονται όπως φαίνεται στο Σχήμα 3.7 σε δύο στήλες που η κάθε μια έχει 10 τμήματα μνήμης των 18Kbit το καθένα. Με τον ίδιο τρόπο οργανώνονται και τα τμήματα πολλαπλασιαστών. Όπου ο κάθε πολλαπλασιαστής επεξεργάζεται δύο αριθμούς των 18-bit.

Πίνακας 3.1: *Τα κύρια χαρακτηριστικά του Spartan3A XC3S700A-4FG484.*

Συσκευή	Πύλες	Πίνακας CLB Ένα CLB=4 Slices				Flip-flop	Κατανεμημένη Μνήμη - bits	Τμήματα Μνήμης - bits	Τμήματα Πολ/στων	DCMs	I/O Χρήστη
		Γραμμές	Στήλες	CLB	Slices						
xc3s700a -4fg484	700K	48	32	1472	5888	11776	92K	360K	20	8	372

Virtex5 XC5VLX330-2FF1760

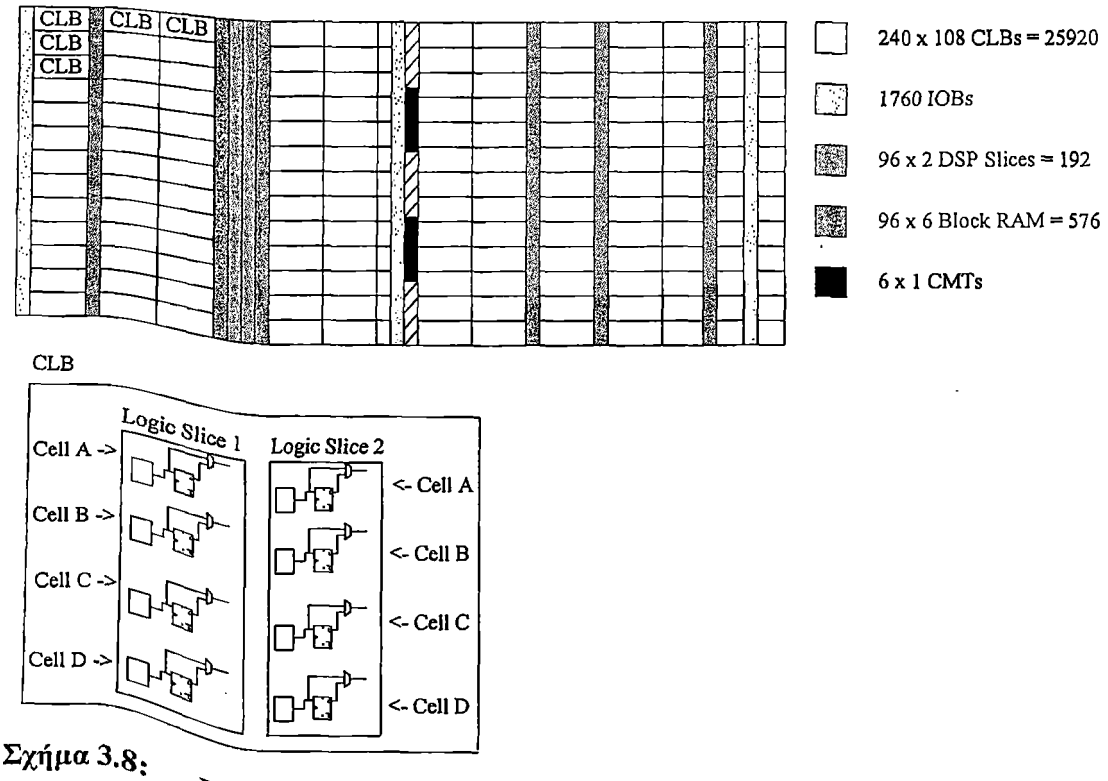
Το δεύτερο FPGA που χρησιμοποιήθηκε ανήκει στην οικογένεια Virtex5 και διαφέρει αρκετά από το προηγούμενο ως προς το μέγεθος του. Ως προς τη δομή και το τύπο συσκευασίας μοιάζει, ωστόσο το Virext5 παρέχει επιπλέον κυκλώματα τα οποία ονομάζονται DSPs (Digital Signal Processing) τμήματα και χρησιμοποιούνται σε εφαρμογές που αποσκοπούν στην επεξεργασία σημάτων. Οι λογικές βαθμίδες έχουν έξι εισόδους και οργανώνονται σε ομάδες των τεσσάρων για να σχηματίσουν ένα λογικό τμήμα και στη συνέχεια δύο λογικά τμήματα σχηματίζουν ένα CLB. Συνολικά παρέχονται 25920 CLBs, 3420 Kbit κατανεμημένης μνήμης, 576 τμήματα κύριας μνήμης των 18 Kbit το καθένα δηλαδή 10368 Kbit μνήμη και 1760 pin εισόδου/εξόδου εκ των οποίων από το χρήστη μπορούν να χρησιμοποιηθούν τα 1200. Μερικές από τις διαφορές από το Spartan3A είναι ότι παρέχονται 192 DSP τμήματα για επεξεργασία σήματος. Κάθε DSP τμήμα υλοποιείται από 25 x 18 πολλαπλασιαστές, έναν αθροιστή και έναν καταχωρητή που έπεται του αθροιστή και χρησιμοποιείται για την επανατροφοδότηση του αθροιστή, ο οποίος ονομάζεται accumulator. Τέλος για τον έλεγχο των ωρολογιακών σημάτων χρησιμοποιούνται 6 CMT (Clock Management Tiles) τα οποία αποτελούνται από δύο DCM και ένα επιπλέον τμήμα το οποίο ονομάζεται Phase Locked Loops - PLL. Ο ρόλος των DCM είναι ο ίδιος με αυτόν του DCM στο Spartan3A ενώ το PLL χρησιμοποιείται για την περαιτέρω επεξεργασία του ωρολογιακού σήματος.

Στο Πίνακα 3.2 συνοψίζονται τα κύρια χαρακτηριστικά του Virtex5 XC5VLX330-2FF1760 και στο Σχήμα 3.8 φαίνεται η γενική δομή του.

Πίνακας 3.2: *Τα κύρια χαρακτηριστικά του Virtex5 XC5VLX330-2FF1760.*

Συσκευή	Πίνακας CLB Ένα CLB = 4 Slices				Flip-flop	Κατανεμημένη Μνήμη - bits	Τμήματα Μνήμης - bits	DSP Blocks	CMTs	I/O Χρήστη
	Γραμμές	Στήλες	CLB	Slices						
xc5vlx330-2ff1760	240	108	25920	51840	207360	3420 K	10368 K	192	6	1200

Η δομή του Virtex5 XC5VLX330-2FF1760



Σχήμα 3.8: *Η δομή του XC5VLX330-2FF1760 και ένα CLB.*

ΑΝΑΦΟΡΕΣ

[1] Floyd, Digital Fundamentals (Ninth Edition).

[2] Stephen Brown, Zvonko Vranesic, Σχεδίαση Ψηφιακών Συστημάτων με τη Γλώσσα VHDL.

[3] DS529 August 19, 2010, Spartan-3A FPGA Family: Data Sheet.

[4] DS100 (v5.0) February 6, 2009, Virtex-5 Family Overview: Product Specification.

ΚΕΦΑΛΑΙΟ 4

ΤΕΧΝΙΚΕΣ ΧΑΜΗΛΗΣ ΚΑΤΑΝΑΛΩΣΗΣ

4.1 Εισαγωγή

Στις μέρες μας η χαμηλή κατανάλωση ισχύος των κυκλωμάτων, αποτελεί μια από τις μεγαλύτερες σχεδιαστικές προκλήσεις στο τομέα των ολοκληρωμένων κυκλωμάτων. Η συνεχώς αυξανόμενη ζήτηση για συσκευές που δε θα τροφοδοτούνται συνεχώς με παροχή ρεύματος, δίνοντας τη δυνατότητα της κινητικότητας στο χρήστη ή γενικότερα τη μη συνεχή τροφοδοσία συσκευών, έχει καταστήσει τη σχεδίαση ολοκληρωμένων κυκλωμάτων χαμηλής κατανάλωσης όσο πιο απαραίτητη από ποτέ. Για το σκοπό αυτό έχουν αναπτυχθεί διάφορες τεχνικές σχεδίασης εκ των οποίων οι σημαντικότερες θα παρουσιαστούν στη συνέχεια καθώς και θα χρησιμοποιηθούν στο σχεδιασμό των κυκλωμάτων της παρούσας εργασίας.

4.2 Είδη κατανάλωσης ισχύος

Η μέθοδος για την μέτρηση της κατανάλωσης στα ολοκληρωμένα κυκλώματα βασίζεται στην διάσπαση της ολικής κατανάλωσης του ολοκληρωμένου κυκλώματος σε επιμέρους καταναλώσεις. Τα δύο βασικότερα είδη κατανάλωσης που υπάρχουν στα ολοκληρωμένα κυκλώματα είναι η δυναμική και στατική κατανάλωση [1]. Το ποσοστό συνεισφοράς της κάθε συνιστώσας της κατανάλωσης στην ολική εξαρτάται από τη τεχνολογία του ολοκληρωμένου. Στα FPGA παρατηρείται ότι η δυναμική κατανάλωση έχει μεγαλύτερη συνεισφορά από ότι η στατική.

Η στατική κατανάλωση δημιουργείται στα τρανζίστορ του κυκλώματος από τα ρεύματα διαρροής, σε στατική λειτουργία, όταν υπάρχει αγώγιμο μονοπάτι από τη τροφοδοσία στη γείωση και υπολογίζεται από το παρακάτω σχέση.

$$P_{\text{Στατικής ισχύος}} = V \times I \tag{4.1}$$

Ουσιαστικά η στατική κατανάλωση είναι ο συνδυασμός της τάσης που εφαρμόζεται στη τροφοδοσία του κυκλώματος και το ρεύμα διαρροής του κάθε τρανζίστορ στο

ολοκληρωμένο. Συνεπώς η στατική κατανάλωση εξαρτάται εξολοκλήρου από τη τεχνολογία και τον τρόπο κατασκευής των τρανζίστορ στο ολοκληρωμένο κύκλωμα.

Αντιθέτως, η δυναμική κατανάλωση συσχετίζεται με την εκάστοτε λειτουργία του κυκλώματος, καθώς ουσιαστικά μετράει το ρεύμα που απαιτείται για την φόρτιση και εκφόρτιση του χωρητικού φορτιού των εξόδων των τρανζίστορ. Αυτό περικλείει και τη συχνότητα λειτουργία του κυκλώματος επειδή αυξάνοντας τη συχνότητα αυξάνονται και ,μεταβάσεις των σημάτων του κυκλώματος. Συνεπώς η δυναμική κατανάλωση εξαρτάται από το χωρητικό φορτίου του κυκλώματος, τη τάση τροφοδοσίας και τη συχνότητα λειτουργίας του. Ο υπολογισμός της δυναμικής κατανάλωσης δίνεται από τη παρακάτω σχέση.

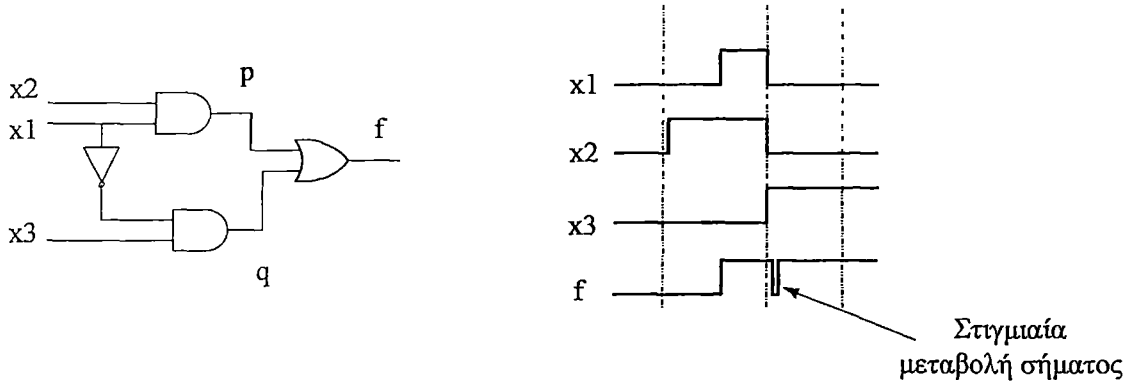
(4.2) $$P_{\Delta υναμικής\ ισχύος} = \frac{1}{2} \times C \times V^2 \times f$$

Από τα παραπάνω προκύπτει ότι την ευθύνη για χαμηλή στατική κατανάλωση την έχει ο κατασκευαστής του FPGA ενώ την ευθύνη για τη χαμηλή δυναμική κατανάλωση την έχει ο σχεδιαστής. Για το σκοπό αυτό έχουν αναπτυχθεί πολλές σχεδιαστικές τεχνικές εκ των οποίων οι σημαντικότερες θα παρουσιαστούν στο επόμενο υποκεφάλαιο.

4.3 Σχεδιαστικές τεχνικές χαμηλής κατανάλωσης

Pipeling

Ένα από τα μεγαλύτερα προβλήματα στη σχεδίαση κυκλωμάτων για FPGA, όσων αφορά τη δυναμική κατανάλωση είναι ο περιορισμός των ανεπιθύμητων μεταβολών των σημάτων οι οποίες ονομάζονται ακίδες ή βυθίσεις (**glitches**). Οι μεταβολές αυτές συμβαίνουν κυρίως σε μεγάλα τμήματα που αποτελούνται από κυκλώματα μόνο συνδυαστικής λογικής. Εκεί μπορούν να προκληθούν στιγμιαίες μεταβολές σημάτων είτε λόγω των πολλών επιπέδων συνδυαστικής λογικής είτε από το μη συγχρονισμό των σημάτων στην είσοδο μιας πύλης. Στο Σχήμα 4.1 φαίνεται ένα απλό παράδειγμα δημιουργίας ανεπιθύμητης μεταβολής σήματος. Στο συγκεκριμένο παράδειγμα η μεταβολή προκύπτει λόγω της επιπλέον καθυστέρησης που δημιουργεί στο δεύτερο κλάδο η πύλη NOT. Έτσι ο χρόνος διάδοσης του σήματος x2 είναι μεγαλύτερος από τους υπόλοιπους χρόνους ως αποτέλεσμα τη στιγμιαία μεταβολή της εξόδου f της πύλης OR από λογικό 1 σε λογικό 0.

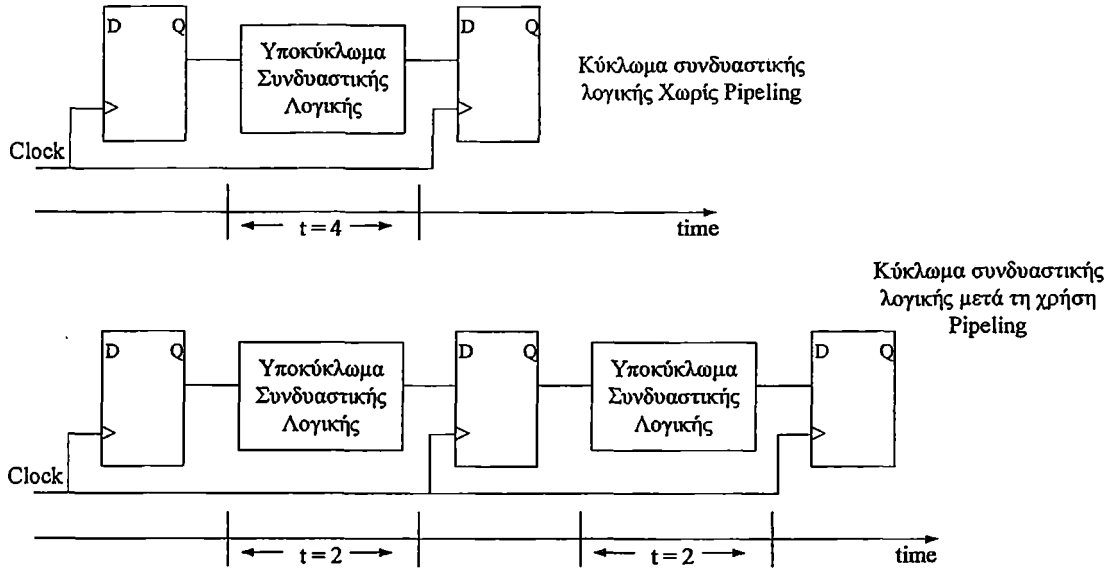


Σχήμα 4.1: *Κύκλωμα που δημιουργεί ανεπιθύμητες μεταβολές.*

Έχει αποδειχθεί ότι στα FPGA μεγάλο ποσοστό της κατανάλωσης οφείλεται σε αυτές τις μεταβολές αφού τα glitches μπορούν να φτάσουν να αποτελούν μέχρι και το 80% της συνολικής δραστηριότητας στο κύκλωμα [2][3]. Μια αποτελεσματική τεχνική που χρησιμοποιείται για το περιορισμό έως ένα βαθμό των glitches ονομάζεται **pipeling**. Στη τεχνική αυτή ο σχεδιαστής χωρίζει το κύκλωμα σε μικρότερα υποκυκλώματα και τοποθετεί ενδιάμεσα καταχωρητές. Το pipeling είναι κυρίως δημοφιλή για την επιτάχυνση των κυκλωμάτων καθώς αυξάνει τη συχνότητα λειτουργίας ωστόσο είναι και εξαιρετικά αποτελεσματική τεχνική για το περιορισμό των ακίδων στα ολοκληρωμένα κυκλώματα.

Η μείωση των glitches μέσω pipeling επιτυγχάνεται γιατί μειώνεται το βάθος της συνδυαστικής λογικής στο κύκλωμα μέσω των καταχωρητών που προστίθενται ανά ίσα χρονικά διαστήματα. Οι καταχωρητές δεν μεταδίδουν τα glitches στο επόμενο επίπεδο λόγω του ότι λειτουργούν σε συγκεκριμένες χρονικές στιγμές. Επίσης λόγω αυτής της ιδιότητας τα σήματα που μεταδίδονται στο επόμενο επίπεδο είναι συγχρονισμένα ως αποτέλεσμα τα συνδυαστικά κυκλώματα που έπονται των καταχωρητών να έχουν λιγότερες πιθανότητες να εμφανίσουν glitches. Στο Σχήμα 4.2 φαίνεται ένα απλό παράδειγμα pipeling.

Ένα μειονέκτημα της χρήσης αυτής τη μεθόδου είναι ότι αυξάνονται οι κύκλοι ρολογιού που απαιτούνται για την εκτέλεση του συνολικού κυκλώματος. Βέβαια αυτό ισοσταθμίζεται έως ένα βαθμό με το γεγονός ότι τα pipeling αυξάνει και τη συχνότητα λειτουργίας του κυκλώματος. Ένα άλλο μειονέκτημα είναι ότι αν χρησιμοποιηθούν



Σχήμα 4.2: *Το ίδιο συνδυαστικό κύκλωμα πριν και μετά από τη χρήση pipeling.*

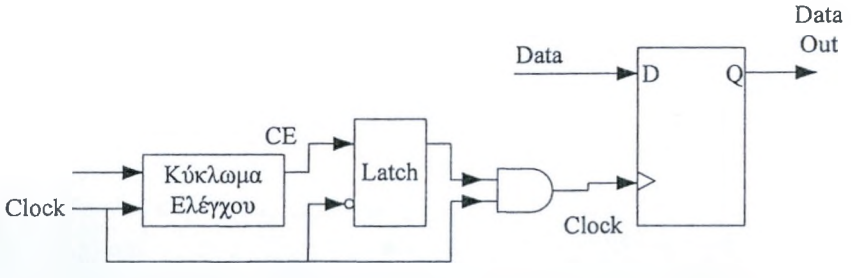
καταχωρητές σε κυκλώματα που δεν έχουν τη τάση να παρουσιάζουν glitches τότε επέρχεται το τελείως αντίθετο από το επιθυμητό αποτέλεσμα, δηλαδή η αύξηση της δυναμικής κατανάλωσης. Αυτό συμβαίνει γιατί και οι καταχωρητές που θα χρησιμοποιηθούν αποτελούν κομμάτι του κυκλώματος οπότε συμβάλουν και αυτοί στη επιπλέον κατανάλωση.

Gated-Clock

Πέρα από τα glitches, όσα σήματα έχουν συνεχείς μεταβάσεις είναι κύριες αιτίες αύξησης της δυναμικής κατανάλωσης στο κύκλωμα. Ένα τέτοιο σήμα στα ολοκληρωμένα κυκλώματα είναι το ωρολογιακό σήμα το οποίο ανάλογα με τη συχνότητα λειτουργίας που έχει καθοριστεί πραγματοποιεί συνεχείς μεταβάσεις με μεγάλη ταχύτητα. Έχει αποδειχθεί ότι τα ωρολογιακά σήματα καταναλώνουν από 15% έως και 45% της ολικής ισχύος ενός κυκλώματος [4]. Μια ακόμα πολύ διαδεδομένη τεχνική μείωσης της δυναμικής κατανάλωσης, είναι η προσπάθεια τα ωρολογιακά σήματα να απενεργοποιούνται στις περιοχές του κυκλώματος οι οποίες κατά τη διάρκεια λειτουργίας του είναι ανενεργές. Αυτή η τεχνική ονομάζεται **Gated-Clock** και έχει διαφορετικούς τρόπους υλοποίησης ανάλογα το ολοκληρωμένο κύκλωμα που προορίζεται.

Η τεχνική του clock-gating [5] αποσκοπεί στη μείωση της δραστηριότητας των υποκυκλωμάτων εκείνων που δε χρησιμοποιούνται συνέχεια. Επίσης, αυτό που επιτυγχάνεται

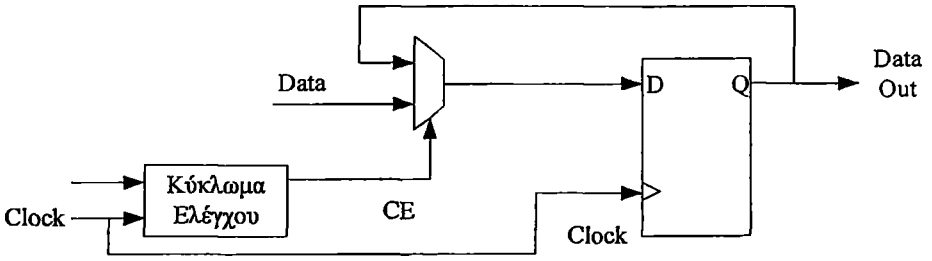
είναι ότι εκτός από τη μείωση της δραστηριότητας σε κάποια τμήματα δε διαδίδεται σε αυτά το ωρολογιακό σήμα σε όλη τη διάρκεια λειτουργίας του κυκλώματος. Κάτι το οποίο συνεπάγεται περαιτέρω μείωση της δυναμικής κατανάλωσης καθώς όπως έχει ήδη αναφερθεί το ωρολογιακό σήμα έχει τεράστια συμβολή στην αύξηση της. Τα υποκυκλώματα που κυρίως αλληλεπιδρούν με το ωρολογιακό σήμα είναι οι καταχωρητές και διάφορα στοιχεία μνήμης. Για το λόγο αυτό εφαρμόζεται στην είσοδο του ωρολογιακού σήματος κάθε καταχωρητή μια πύλη AND για να μπορεί μέσω κάποιο κυκλώματος ελέγχου να απενεργοποιείται το σήμα ρολογιού στο καταχωρητή. Στο Σχήμα 4.3 φαίνεται ένα τέτοιο παράδειγμα εφαρμογής. Ο λόγος ύπαρξης του επιπλέον μανδαλωτή είναι απλά για την αποφυγή glitches στην έξοδο της πύλης AND.



Σχήμα 4.3: *Clock-gating με χρήση πύλης AND.*

Το κύκλωμα του Σχήματος 4.3 λειτουργεί πολύ αποδοτικά σε τεχνολογίες ολοκληρωμένων κυκλωμάτων ειδικού σκοπού. Στα FPGA όμως το συγκεκριμένο κύκλωμα έχει προβληματική συμπεριφορά λόγω του τρόπου διαμοιρασμού του ωρολογιακού σήματος. Για τη μη ύπαρξη πολλών ξεχωριστών ρολογιών τα περισσότερα FPGA ξεχωρίζουν τη λειτουργία του clock-gating από το ωρολογιακό σήμα χρησιμοποιώντας έναν πολυπλέκτη στην είσοδο των δεδομένων του κάθε flip-flop. Η έξοδος του flip-flop ανατροφοδοτείται στο flip-flop μέσω ενός πολυπλέκτη. Στο Σχήμα 4.4 φαίνεται αυτή η μεθοδολογία, η οποία ουσιαστικά εξομοιώνει τη λειτουργία του clock-gating μέσω του σήματος ελέγχου στο πολυπλέκτη ο οποίος ελέγχει τη διάδοση των δεδομένων ή την ανατροφοδότηση του καταχωρητή. Το σήμα ρολογιού φτάνει στην είσοδο του πολυπλέκτη χωρίς να έχει προηγηθεί η πύλη AND του Σχήματος 4.3 κάτι το οποίο αν συνέβαινε θα μπορούσε να προκαλέσει αποσυγχρονισμό (clock skew) των στοιχείων που αλληλεπιδρούν με το ωρολογιακό σήμα στα FPGA, λόγω της διαμοίρασης ενός και μόνο σήματος στο κύκλωμα. Το κύκλωμα του

Σχήματος 4.4 εγγυάται μικρό αποσυγχρονισμό των στοιχείων αλλά παρόλα αυτά, αυτό που επιτυγχάνεται από πλευράς κατανάλωσης είναι μόνο η μη συνεχείς λειτουργία των καταχωρητών, καθώς το ωρολογιακό σήμα συνεχίζει να διαμοιράζεται σε όλο το κύκλωμα σε όλη τη διάρκεια λειτουργίας του.



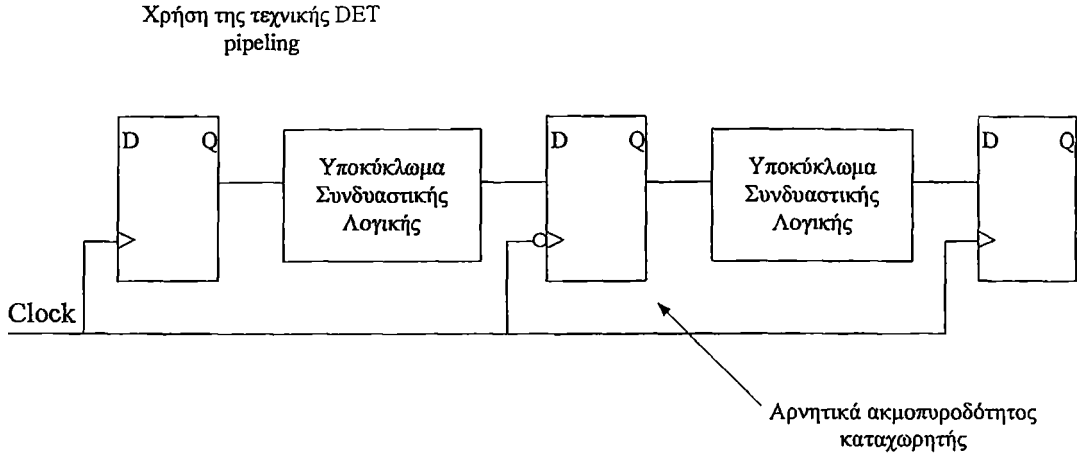
Σχήμα 4.4: *Η υλοποίηση του Clock-gating σε flip-flop ενός FPGA.*

Double Edge Trigger pipeling

Μια άλλη τεχνική που χρησιμοποιείται για τη μείωση της δυναμικής κατανάλωσης αλλά και για να εξισορροπήσει τη καθυστέρηση της μεθόδου pipeling είναι η χρήση επιπλέον καταχωρητών οι οποίοι είναι αρνητικά ακμοπυροδότητοι (**Double Edge Trigger - DET** pipeling) [6]. Ουσιαστικά σε αυτή εφαρμόζεται η τεχνική pipeling με τη διαφορά ότι οι καταχωρητές που χρησιμοποιούνται εναλλάσσονται σε θετικά και αρνητικά ακμοπυροδότητους. Το πλεονέκτημα αυτής της τεχνικής είναι ότι με τη χρήση αυτών των δύο καταχωρητών δεν αυξάνονται οι κύκλοι ρολογιού που χρειάζονται για τον υπολογισμό μιας λογικής συνάρτησης σε αντίθεση με τη τεχνική του απλού pipeling. Επιπροσθέτως, τα πλεονεκτήματα του απλού pipeling σε σχέση με τη δυναμική κατανάλωση για το περιορισμό των glitches ισχύουν και σε αυτή τη μέθοδο. Στο Σχήμα 4.5 φαίνεται η εφαρμογή αυτής της μεθόδου στο παράδειγμα του Σχήματος 4.2 με τη διαφορά ότι οι κύκλοι επεξεργασίας έχουν μειωθεί στο μισό.

BRAM

Ένας άλλος τρόπος για τη μείωση της δυναμικής κατανάλωσης στα FPGA είναι η χρησιμοποίηση των ενσωματωμένων κυκλωμάτων που παρέχονται αντί των λογικών βαθμίδων. Τα FPGA, όπως αναφέρθηκε και στο κεφάλαιο τρία, εμπεριέχουν τμήματα



Σχήμα 4.5: *DET pipeling, χρήση καταχωρητών που διαδοχικά εναλλάσσονται σε θετικά και αρνητικά ακμοπυροδότητοι .*

κυκλωμάτων μνήμης (Block RAM), αθροιστών, πολλαπλασιαστών κ.α. Επειδή αυτά τα τμήματα δεν περιέχουν επιπλέον κυκλώματα για τον προγραμματισμό τους έχει αποδειχθεί ότι μπορών να μειώσουν τη δυναμική κατανάλωση έως και 23% [7].

Επίσης πρέπει να αναφερθεί ότι μπορεί να γίνει και συνδυασμός των παραπάνω τεχνικών για την επίτευξη του επιθυμητού αποτελέσματος. Τέλος, δεν υπάρχει κανόνας για το πια τεχνική αποδίδει καλύτερα από την άλλη επειδή η κάθε μια από τις παραπάνω εξαρτάται άμεσα από την εφαρμογή στην οποία πρόκειται να υλοποιηθεί.

ΑΝΑΦΟΡΕΣ

[1] Paris Kitsos, Nikolas Sklavos, Athanasios N. Skodras, Low Power FPGA Implementations of 256-bit Luffa Hash Function (DSD 2010).

[2] Gustavo Sutter Elias Todorovich, Eduardo Boemo, Design of Power Aware FPGA-based Systems.

[3] Mouzam Khan, Power Optimization in FPGA Designs.

[4] Yan Zhang, Jussi Roivainen, Aarne Mammela, Clock-Gating in FPGAs: A Novel and Comparative Evaluation.

[5] Synplicity, Gated Clock Conversion with Synplicity's Synthesis Products.

[6] Tomasz S. Czajkowski, Stephen D. Brown, Using Negative Edge Triggered FFs to Reduce Glitching Power in FPGA Circuits

[7] David Kenney, Energy Efficiency Analysis and Implementation of AES on an FPGA.

ΚΕΦΑΛΑΙΟ 5

ΠΕΡΙΓΡΑΦΗ ΑΛΓΟΡΙΘΜΩΝ FUGUE ΚΑΙ JH

5.1 Πεδία πεπερασμένων αριθμών

Οι μαθηματικές πράξεις που υλοποιούνται στους αλγορίθμους Fugue και JH βασίζονται στην ειδική κατηγορία αριθμητικών πεδίων τα πεδία Galois (Galois Field - GF) [1]. Τα πεδία που θα χρησιμοποιηθούν κυρίως στους αλγόριθμους είναι τα $GF(2^8)$ και $GF(2^4)$ όπου το 2^8 και το 2^4 υποδηλώνουν το εύρος του αριθμητικού πεδίου. Και στις περιπτώσεις των αλγορίθμων Fugue και JH τα αριθμητικά πεδία αποτελούνται από 256 και 16 ακέραιων στοιχεία αντίστοιχα. Στον αλγόριθμο Fugue τα στοιχεία του πεδίου $GF(2^8)$ αναπαριστούν byte και μπορούν να εκφραστούν και ως πολυώνυμα εβδόμου βαθμού.

Ο αλγόριθμος Fugue χρησιμοποιεί τις μαθηματικές πράξεις της πρόσθεσης και του πολλαπλασιασμού ενώ ο αλγόριθμος JH χρησιμοποιεί μόνο τη πράξη της πρόσθεσης. Στα πεδία πεπερασμένων αριθμών η πράξη της πρόσθεσης αλλά και της αφαίρεσης υλοποιείται με τη πράξη αποκλειστικού-Η (exclusive-or - **XOR**) και το σύμβολο που θα υποδηλώνει αυτή τη πράξη στα σχήματα του κεφαλαίου αυτού θα είναι το « $\oplus$ ».

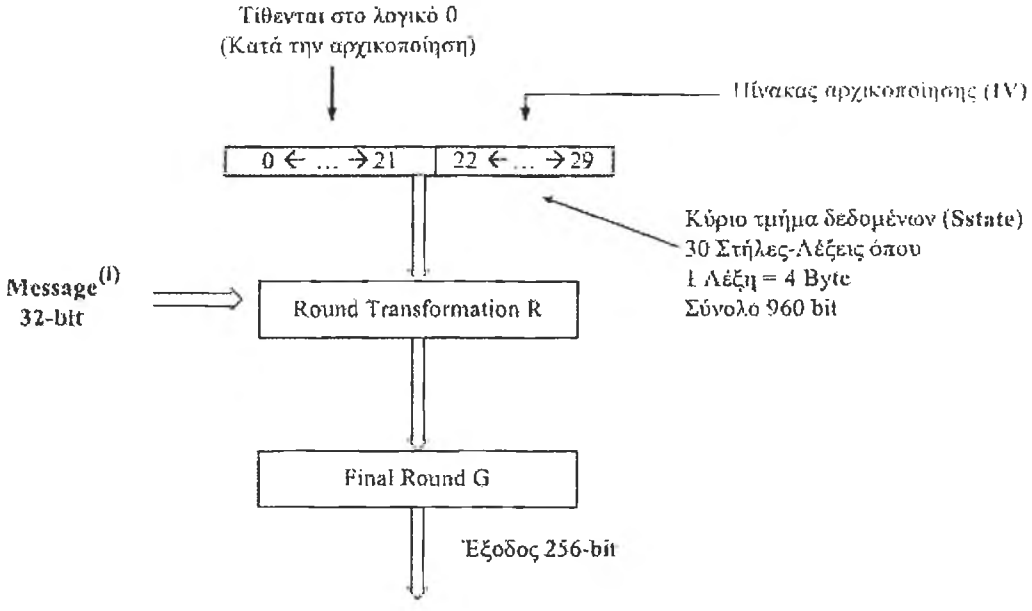
Η πράξη του πολλαπλασιασμού υλοποιείται με το πολλαπλασιασμό μεταξύ των πολυωνύμων και στη συνέχεια το αποτέλεσμα προκύπτει από το υπόλοιπο της ακεραίας διαίρεσης (modulo) με ένα συγκεκριμένο πολυώνυμο το οποίο χαρακτηρίζει το αριθμητικό πεδίο. Βάση των προδιαγραφών του αλγορίθμου Fugue το πολυώνυμο που χρησιμοποιείται για τη μαθηματική πράξη του πολλαπλασιασμού είναι το $x^8 + x^4 + x^3 + x + 1$. Το οποίο στη δυαδική μορφή του εκφράζεται ως 100011011 . Τα σύμβολα που θα χρησιμοποιούνται για να δηλώσουν πολλαπλασιασμό είναι τα ίδια με αυτά που χρησιμοποιούνται και στους πραγματικούς αριθμούς.

5.2 Fugue

Η συνάρτηση κατακερματισμού Fugue σχεδιάστηκε από τους Shai Halevi, William E. Hall και Charanjit S. Jutla στο ερευνητικό κέντρο της IBM T.J Watchon και κατατέθηκε [2] το 2009 στον διαγωνισμό συναρτήσεων κατακερματισμού του NIST (National Institute of Standards and Technology).

Στο διαγωνισμό αυτό κατατέθηκαν πέντε εκδοχές του αλγορίθμου οι οποίες διαφοροποιούνται κυρίως στο μέγεθος της εξόδου. Οι εκδοχές αυτές είναι οι : Fugue-224, Fugue-256, Fugue-384,Fugue-512 και μια πιο αδύναμη έκδοση της Fugue-256. Επίσης έχει κατατεθεί ένα σχέδιο για την παραμετρική υλοποίηση των τεσσάρων βασικών εκδόσεων. Οι παράμετροι που χρησιμοποιούνται διαχωρίζονται στο μέγεθος της εξόδου, στον αριθμό των γύρων που συμβαίνουν σε κάθε στάδιο του αλγορίθμου, στο μέγεθος του κυρίως τμήματος που επεξεργάζεται, στο αριθμό των γύρων του πρώτου σταδίου και στον αριθμό των γύρων του τελευταίου σταδίου. Στη παρούσα εργασία επιλέχτηκε να υλοποιηθεί σε υλικό η έκδοση Fugue-256 η οποία και περιγράφεται αναλυτικότερα παρακάτω.

Ο αλγόριθμος Fugue ανήκει στη κατηγορία των αλγορίθμων τμήματος. Δέχεται μηνύματα μήκους $\leq 2^{64} - 1$ και επεξεργάζεται μηνύματα μήκους πολλαπλάσια των 32-bit τα οποία στη συνέχεια συνδυάζονται με το αρχικοποιημένο τμήμα του αλγορίθμου μήκους 960-bit. Στο Σχήμα 5.1 δίνεται η βασική δομή του αλγορίθμου Fugue-256. Το κύριο τμήμα δεδομένων ονομάζεται Sstate και αποτελείται από 30 στήλες-λέξεις των 4-byte η κάθε μια. Κατά το στάδιο της αρχικοποίησης οι στήλες 0 έως 21 τίθενται στο λογικό 0 και οι υπόλοιπες στήλες αρχικοποιούνται από το πίνακα αρχικοποίησης (Initialization Vector - **IV**). Τα δύο κύρια τμήματα του αλγορίθμου είναι τα Round Transformation R και Final Round G. Το πρώτο τμήμα επαναλαμβάνεται όσο υπάρχουν μηνύματα (**Message**$^{(i)}$) προς επεξεργασία ενώ το δεύτερο επαναλαμβάνεται για 23 γύρους.



Σχήμα 5.1: *Η δομή του αλγορίθμου Fugue.*

Round Transformation R

Το επίπεδο Round Transformation R αποτελείται από τέσσερις συναρτήσεις και ο αριθμός των γύρων που πραγματοποιούνται εξαρτάται από τον αριθμό των μηνυμάτων που υπάρχουν για επεξεργασία. Οι συναρτήσεις αυτές είναι οι TIX, ROR3, CMIX και SMIX. Παρακάτω δίνεται ο τρόπος εκτέλεσης των συναρτήσεων στο επίπεδο Round Transformation R.

(5.1)

1. Για i = 1 έως N (πλήθος μηνυμάτων)
   {
   TIX (Message$^{(i)}$)
2. Για i = 1 έως 2 { ROR3, CMIX, SMIX }
   }

Ο αριθμός των επαναλήψεων του συγκεκριμένου επιπέδου όπως προαναφέρθηκε εξαρτάται από το πλήθος των μηνυμάτων. Σε κάθε επανάληψη το βήμα 2 εκτελείται από δύο φορές.

Final Round G

Το επίπεδο Final Round G αποτελείται από τις συναρτήσεις ROR3, CMIX, SMIX, ROR15, ROR14 και πράξεις αποκλειστικού-Η (XOR) μεταξύ των στηλών S_0,S_4,S_{15} και S_{16} ενώ ο αριθμός των επαναλήψεων είναι προκαθορισμένος στις 26 επαναλήψεις για τη Fugue-256. Παρακάτω δίνονται τα στάδια του επιπέδου Final Round G.

(5.2)

1. Για i = 1 έως 10
   {
   ROR3,CMIX,SMIX
   }

2. Για i = 1 έως 13
   {
   $S_4 = S_4 \oplus S_0$, $S_{15} = S_{15} \oplus S_0$, ROR15, CMIX, SMIX
   $S_4 = S_4 \oplus S_0$, $S_{16} = S_{16} \oplus S_0$, ROR15, CMIX, SMIX
   }

3. $S_4 = S_4 \oplus S_0$, $S_{15} = S_{15} \oplus S_0$

Τέλος , όταν ολοκληρωθεί το επίπεδο Final Round G επιλέγονται από το Sstate οκτώ λέξεις των 4 byte η κάθε μια οι οποίες αποτελούν την έξοδο του αλγορίθμου Fugue-256. Οι οκτώ στήλες οι οποίες επιλέγονται είναι οι : $\mathbf{S_1, S_2, S_3, S_4, S_{15}, S_{16}, S_{17}}$ και $\mathbf{S_{18}}$.

TIX

Η συνάρτηση TIX λαμβάνει χώρα μόνο στο επίπεδο Round Transformation R. Η κύρια λειτουργία της συνάρτησης είναι να μετασχηματίζει το Sstate εισάγοντας και χρησιμοποιώντας το αρχικό μήνυμα. Η λειτουργία του TIX περιγράφεται αλγοριθμικά παρακάτω και σχηματικά από το Σχήμα 5.2 όπου ουσιαστικά πρόκειται για τη πράξη του αποκλειστικού-Η μεταξύ των στηλών S_0, S_1, S_8, S_{10}, S_{24} και του Message$^{(i)}$.

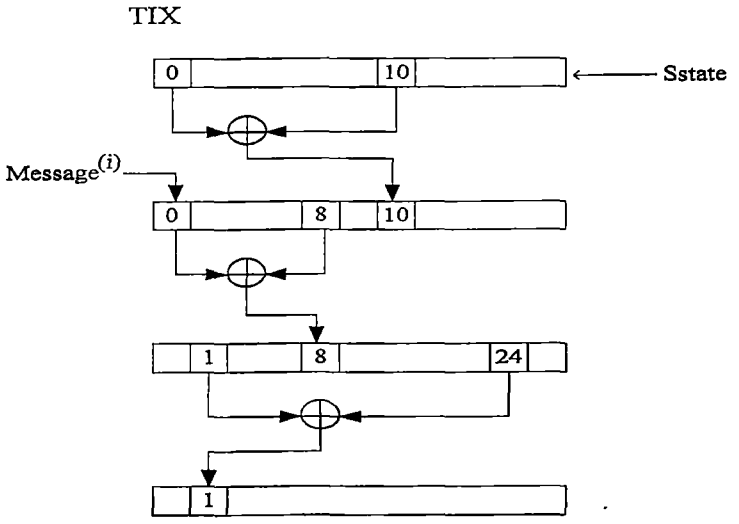
(5.3)

1. S_{10} = S_{10} XOR S_0

2. $S_0 = \text{Message}^{(i)}$

3. $S_8 = S_8 \text{ XOR } S_0$

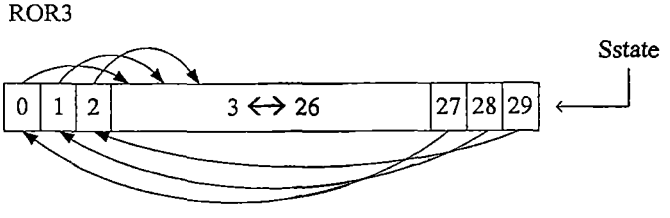
4. $S_1 = S_1 \text{ XOR } S_{24}$



Σχήμα 5.2: *Σχηματικό της λειτουργίας της συνάρτησης TIX.*

ROR

Η διαδικασία της δεξιάς μετατόπισης (ROR) χρησιμοποιείται και στα δύο επίπεδα του αλγορίθμου. Στο επίπεδο Round Transformation R συμβαίνουν δύο δεξιές μετατοπίσεις κατά τρείς στήλες (ROR3) κάθε φόρα που εκτελείται. Στο επίπεδο Final Round G συμβαίνουν συνολικά 10 μετατοπίσεις ROR3. Στο Σχήμα 5.2 δίνεται ένα παράδειγμα δεξιάς μετατόπισης κατά τρείς στήλες. Επίσης στο επίπεδο Final Round G συμβαίνουν 13 δεξιές μετατοπίσεις κατά 15 στήλες (ROR15) και 13 δεξιές μετατοπίσεις κατά 14 στήλες (ROR14).



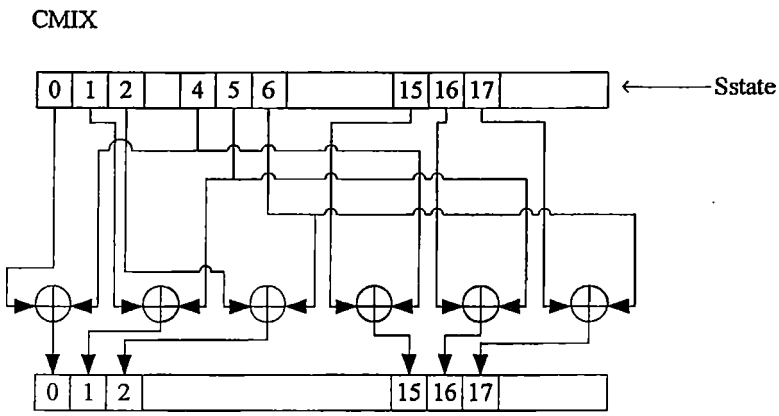
Σχήμα 5.3: *Παράδειγμα δεξιάς μετατόπισης κατά τρείς στήλες (ROR3).*

CMIX

Ήδη έχουν περιγραφεί τρείς πράξεις αποκλειστικού-Η οι οποίες συμβαίνουν στη συνάρτηση TIX. Στη συνάρτηση CMIX συμβαίνουν άλλες έξι πράξεις αποκλειστικού-Η μεταξύ των στηλών του Sstate S_0, S_1, S_2, S_4, S_5, S_6, S_{15}, S_{16}, και S_{17}. Οι έξι πράξεις αποκλειστικού-Η που πραγματοποιούνται στη συνάρτηση CMIX είναι οι παρακάτω και απεικονίζονται στο Σχήμα 5.4 :

(5.4)

1. $S_0 = S_0$ XOR S_4
2. $S_1 = S_1$ XOR S_5
3. $S_2 = S_2$ XOR S_6
4. $S_{15} = S_{15}$ XOR S_4
5. $S_{16} = S_{16}$ XOR S_5
6. $S_{17} = S_{17}$ XOR S_6



Σχήμα 5.4: *Σχηματικό της λειτουργίας της συνάρτησης CMIX.*

SMIX

Η συνάρτηση SMIX αποτελεί ένα από τα βασικότερα κομμάτια του αλγορίθμου Fugue καθώς χρησιμοποιείται και στα δύο επίπεδα και παίζει καθοριστικό ρόλο στο μετασχηματισμό του αρχικού μηνύματος. Στο επίπεδο Round Transformation R χρησιμοποιείται δύο φορές κάθε φορά που εκτελείται και στο επίπεδο Final Round G συνολικά χρησιμοποιείται 36 φορές. Η λειτουργία του εφαρμόζεται μόνο στις τέσσερις πρώτες στήλες-λέξεις του τμήματος Sstate (S_0, S_1, S_2, S_4) οι οποίες διαχειρίζονται ως ένας πίνακας 4x4. Συνολικά λοιπόν κάθε φορά που χρησιμοποιείται μετασχηματίζει 128-bit. Η υλοποίηση του SMIX πραγματοποιείται από δύο μέρη τα οποία είναι η αντικατάσταση των

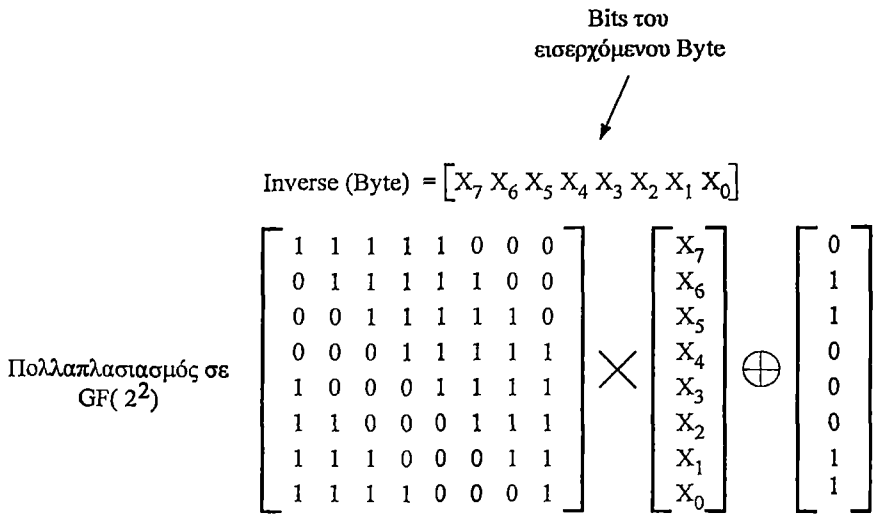
εισερχόμενων byte με τα προκαθορισμένα byte ενός S-box και ο αριστερός πολλαπλασιασμός πινάκων μεταξύ του αποτελέσματος της αντικατάστασης από τα S-boxes και ενός προκαθορισμένου πίνακα 16x16. Το βήμα αυτό ονομάζεται **Super-Mix**.

Όπως προαναφέρθηκε το πρώτο στάδιο είναι η αντικατάσταση των byte με τα byte του S-box. Οι τιμές του S-box είναι προκαθορισμένες και δίνονται από το Πίνακα 5.1. Κάθε εισερχόμενο byte χωρίζεται στη μέση και τα πρώτα 4 bit καθορίζουν τη γραμμή του S-box ενώ τα τέσσερα τελευταία bit καθορίζουν τη στήλη του S-box. Έτσι επιλέγεται το στοιχείο που θα αντικαταστήσει το εισερχόμενο byte. Πρέπει να αναφερθεί ότι το S-box του αλγορίθμου Fugue είναι το ίδιο S-box που χρησιμοποιεί ο AES κάτι το οποίο καθιστά τον αλγόριθμο Fugue ευέλικτο και δίνει τη δυνατότητα για παράλληλες υλοποιήσεις του αλγορίθμου Fugue με τον αλγόριθμο AES.

Πίνακας 5.1: *Κουτί αντικατάστασης του αλγορίθμου Fugue.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
A	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
B	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
C	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
D	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
E	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
F	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Επίσης πρέπει να αναφερθεί ότι τα στοιχεία του S-box μπορούν να υπολογιστούν και δυναμικά κατά τη διάρκεια εκτέλεσης του αλγορίθμου. Αυτό γίνεται εάν στο εισερχόμενο byte υπολογιστεί ο αντίστροφος του και στη πραγματοποιηθεί ο πολλαπλασιασμός και η πρόσθεση πινάκων που δείχνει το Σχήμα 5.5 .



Σχήμα 5.5: *Πολλαπλασιασμός και πρόσθεση του αντίστροφου εισερχόμενου byte για τον υπολογισμό του byte αντικατάστασης.*

Μόλις ολοκληρωθεί η αντικατάσταση των byte ακολουθεί το δεύτερο στάδιο το οποίο είναι το **Super-Mix**. Σε αυτό το στάδιο τα 16 byte διαχειρίζονται ως ένας 4x4 πίνακας ο οποίος ουσιαστικά πολλαπλασιάζεται με ένα προκαθορισμένο πίνακα ο οποίος ονομάζεται M και δίνεται από το Πίνακα 5.2 .

Πίνακας 5.2: *Πίνακας M.*

$$M = \begin{bmatrix} 01 & 04 & 07 & 01 \\ 01 & 01 & 04 & 07 \\ 07 & 01 & 01 & 04 \\ 04 & 07 & 01 & 01 \end{bmatrix}$$

Ο υπολογισμός του βήματος Super-Mix δίνεται από τη σχέση 5.5 όπου με M συμβολίζεται ο παραπάνω πίνακας, με U ο πίνακας που δεδομένων που εισέρχεται στο βήμα Super-Mix και με W το αποτέλεσμα. Η σχέση περιγράφει τη διαδικασία όπου για τον υπολογισμό κάθε στοιχείου του πίνακα W πραγματοποιείται ο πολλαπλασιασμός μεταξύ του M και του U και στη συνέχεια προστίθεται το αποτέλεσμα του πολλαπλασιασμού μεταξύ του ανάστροφου πίνακα M και το άθροισμα των στοιχείων της γραμμής i του πίνακα U εξαιρουμένου του στοιχείου που βρίσκεται στη θέση k = i .

(5.5) $$W_j^i = \sum_k (M_k^i \cdot U_j^k) + M_i^j \cdot (\sum_{k \in [0..3],\, k \neq i} U_k^i)$$

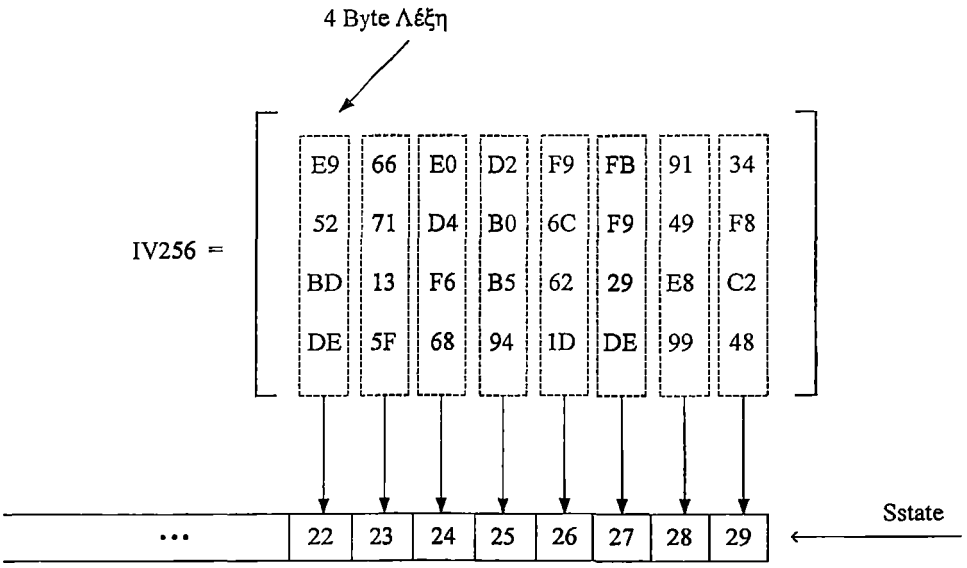
Ο υπολογισμός του Super-Mix μπορεί να προκύψει και πιο απλά, χωρίς όλη τη παραπάνω διαδικασία. Αν ο εισερχόμενος πίνακας U θεωρηθεί ως ένας μονοδιάστατος πίνακας 16 γραμμών και μιας στήλης (16x1), τότε υπολογίζοντας το αποτέλεσμα του αριστερού πολλαπλασιασμού με το πίνακα N (Πίνακα 5.3) προκύπτει η έξοδος του βήματος Super-Mix. Κάτι το οποίο ισούται με τον υπολογισμό της σχέσης 5.5 .

Πίνακας 5.3: *Πίνακας N 16x16.*

$$N = \begin{bmatrix}
1 & 4 & 7 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 4 & 7 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 7 & 1 & 1 & 4 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 4 & 7 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 4 & 7 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 7 & 1 & 0 & 4 \\
4 & 7 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 6 & 4 & 7 & 1 & 7 & 0 & 0 & 0 \\
0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 1 & 6 & 4 & 7 \\
7 & 1 & 6 & 4 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 0 & 7 & 4 & 7 & 1 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 5 & 4 & 7 & 1 \\
1 & 5 & 4 & 7 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 \\
0 & 0 & 4 & 0 & 7 & 1 & 5 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 \\
0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 & 4 & 7 & 1 & 5 & 0 & 0 & 0 & 0
\end{bmatrix}$$

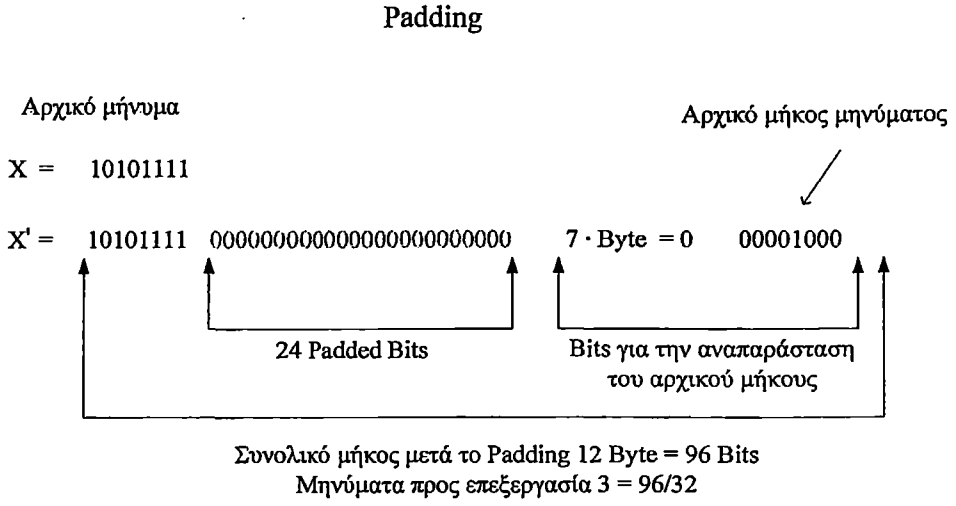
Padding και αρχικοποίηση.

Όπως φαίνεται και στο Σχήμα 5.1 κατά το στάδιο της αρχικοποίησης οι στήλες του τμήματος Sstate από την 0 έως την 21 τίθενται στο λογικό 0 ενώ οι στήλες από 22 έως 29 αρχικοποιούνται βάση ενός προκαθορισμένου πίνακα ο οποίος δίνεται στο Σχήμα 5.6 . Κάθε στήλη του πίνακα IV256 αποτελεί μια λέξη αρχικοποίησης για τις παραπάνω στήλες του τμήματος Sstate. Ο πίνακας αυτός αρχικοποιεί το τμήμα Sstate μόνο της Fugue-256. Για τις άλλες εκδοχές της Fugue υπάρχουν αντίστοιχοι πίνακες.



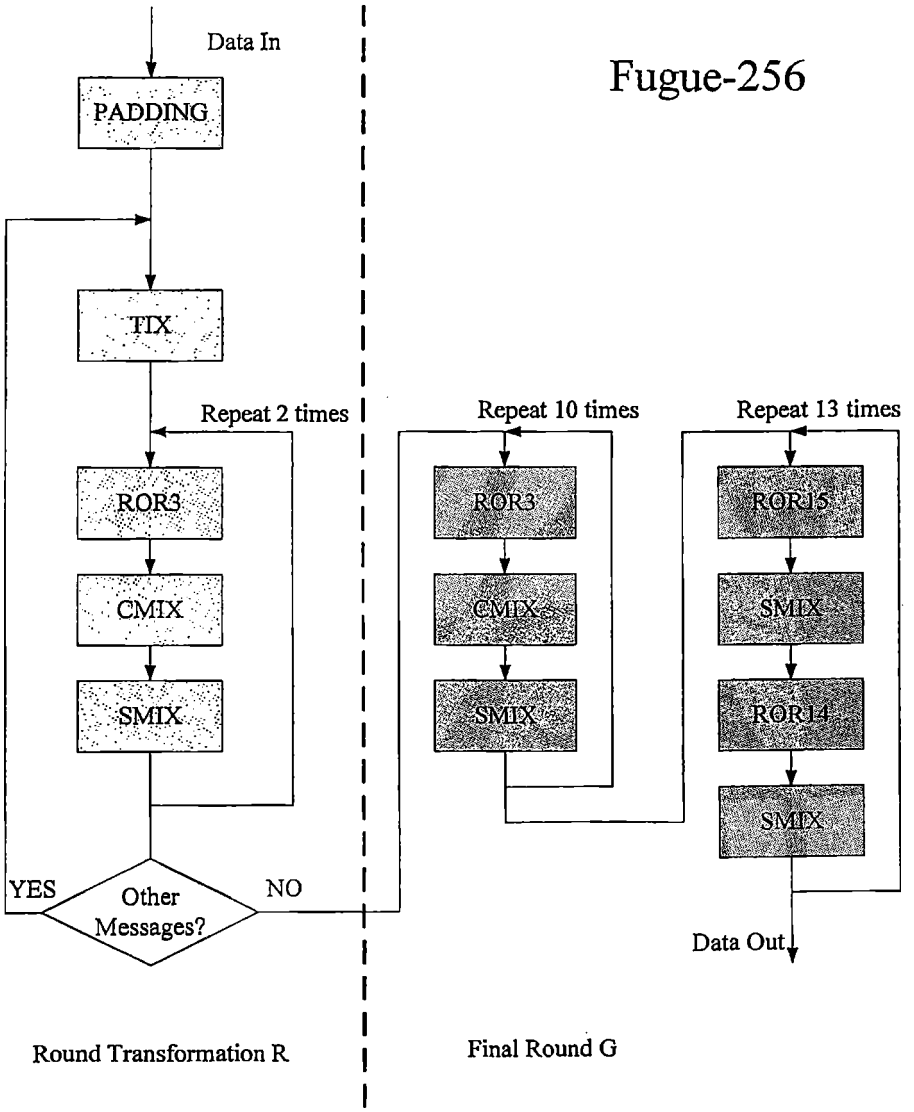
Σχήμα 5.6: *Αρχικοποίηση στη Fugue-256.*

Το αρχικό μήνυμα προκειμένου να χρησιμοποιηθεί πρέπει να είναι μήκους πολλαπλάσιου των 32-bit. Στη περίπτωση που δε πληρείται η προϋπόθεση αυτή το μήνυμα στο τέλος του συμπληρώνεται με λογικά 0 για να σχηματιστούν ομάδες των 32-bit. Μέτα τη συμπλήρωση με λογικά 0 προστίθενται άλλα 8 byte τα οποία αναπαριστούν σε δυαδική μορφή το αρχικό μήκος του μηνύματος. Ένα απλό παράδειγμα της διαδικασίας του padding φαίνεται στο Σχήμα 5.7 .



Σχήμα 5.7: *Ένα παράδειγμα padding στη Fugue-256.*

Στο Σχήμα 5.8 παρουσιάζεται ένα διάγραμμα ροής με τη συνολική λειτουργία του αλγορίθμου Fugue-256 στο οποίο φαίνεται αναλυτικά η ροή των δεδομένων και με ποιο τρόπο επεξεργάζονται από τον αλγόριθμο.

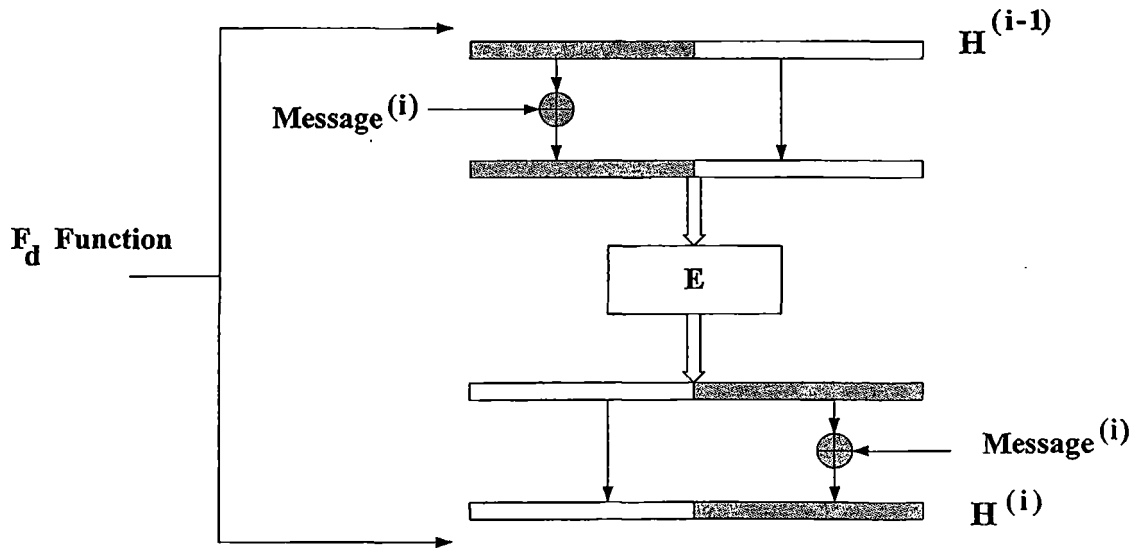


Σχήμα 5.8: *Fugue-256.*

5.3 JH

Η συνάρτηση κατακερματισμού JH σχεδιάστηκε από τον Hongjun Wu στο ινστιτούτο Infocomm Research, Sigapore και κατατέθηκε [3] το 2009 στον διαγωνισμό συναρτήσεων κατακερματισμού του NIST (National Institute of Standards and Technology).

Ο JH αλγόριθμος ανήκει και αυτός στη κατηγορία των αλγορίθμων τμήματος και επεξεργάζεται μηνύματα μήκους πολλαπλάσιου των 512-bit τα οποία συνδυάζονται με το αρχικοποιημένο τμήμα του αλγορίθμου μήκους 1024-bit. Μπορεί να λειτουργήσει σε τέσσερις διαφορετικές καταστάσεις στις οποίες παράγει εξόδους μήκους 224-bit, 256-bit, 384-bit και των 512-bit. Η εναλλαγή αυτών των καταστάσεων υλοποιείται πολύ εύκολα καθώς η απόφαση για το μήκος της εξόδου του αλγορίθμου πραγματοποιείται στην αρχικοποίηση και στο τελευταίο στάδιο της διαδικασίας μόνο. Για την ολοκλήρωση του αλγορίθμου απαιτούνται 35.5 γύροι. Στο Σχήμα 5.9 φαίνεται η δομή του αλγορίθμου JH.



Σχήμα 5.9: *Η δομή του αλγορίθμου JH.*

Στο παραπάνω σχήμα με το γράμμα $\mathbf{H}^{(i)}$ συμβολίζεται το κύριο τμήμα των 1024-bit που διαχειρίζεται ο αλγόριθμος JH και με τη λέξη $\mathbf{Message}^{(i)}$ συμβολίζεται το εκάστοτε μήνυμα προς επεξεργασία το οποίο είναι μήκους 512-bit. Όπως προκύπτει από το σχήμα στη κύρια συνάρτηση του αλγορίθμου, η οποία συμβολίζεται με το γράμμα **E** περνάει το αποτέλεσμα της πράξης αποκλειστικού-Η (XOR), ανάμεσα στο μήνυμα και στο πρώτο μισό

κομμάτι του τμήματος H, και το δεύτερο μισό του τμήματος H. Στη συνέχεια στο δεύτερο μισό της εξόδου της συναρτήσεως E πραγματοποιείται πάλι η πράξη XOR με το αρχικό μήνυμα. Το αποτέλεσμα αυτό είναι το δεύτερο μισό του τελικού αποτελέσματος του αλγορίθμου ενώ το πρώτο μισό είναι το πρώτο μισό κομμάτι της εξόδου της συναρτήσεως E. Η διαδικασία αυτή ονομάζεται F_d συνάρτηση και περιγράφεται από τη παρακάτω σχέση και η αλγοριθμικά από τη (5.7).

(5.6) $$H^{(i)} = F_d(H^{(i-1)}, \quad M^{(i)})$$

(5.7)

1. $A^J = H^{(i-1),j} \oplus M^{(i),j}$ Για $0 \leq j \leq 511$
   $A^j = H^{(i-1),j}$ Για $512 \leq j \leq 1023$
2. $B = E_d(A)$
3. $H^{(i),j} = B^j$ Για $0 \leq j \leq 511$
   $H^{(i),j} = B^j \oplus M^{(i),j-512}$ Για $512 \leq j \leq 1023$

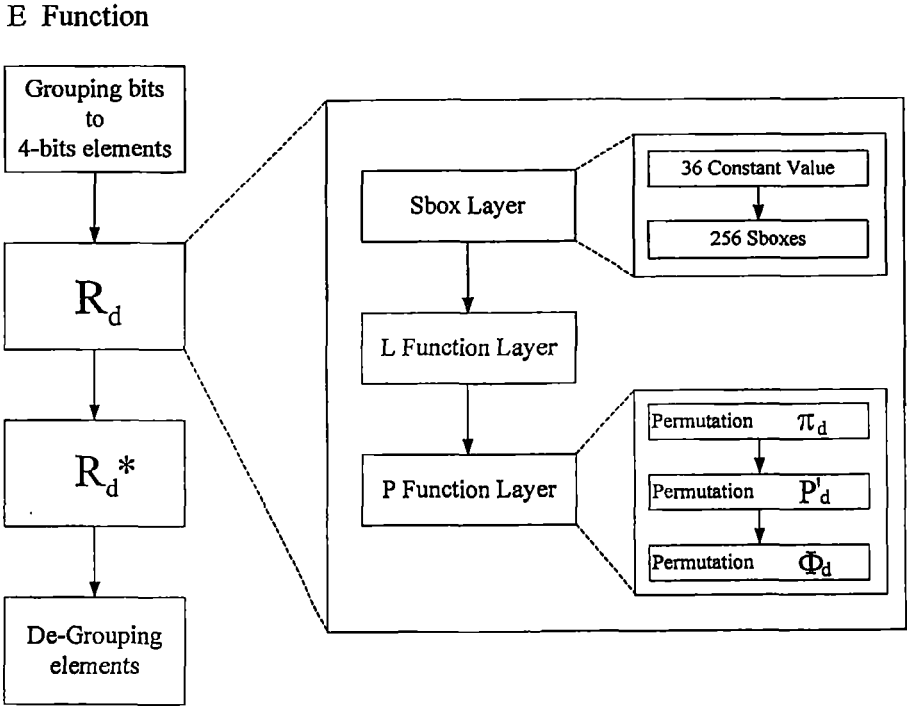
Ο δείκτης d με αγγλικούς χαρακτήρες συμβολίζει το πώς διαχειρίζονται ο αλγόριθμος και οι συναρτήσεις τα bit. Οι τιμές που παίρνει ο δείκτης d εκφράζονται ως είναι δυνάμεις του 2.

Η συνάρτηση E

Η συνάρτηση E είναι το βασικό κομμάτι του αλγόριθμου JH και στο Σχήμα 5.10 φαίνονται οι υποσυναρτήσεις από τις οποίες αποτελείται. Τα στάδια της συνάρτησης είναι τέσσερα τα οποία και δίνονται παρακάτω. Στο πρώτο στάδιο γίνεται μια ομαδοποίηση στην οποία σχηματίζονται 256 τμήματα των 4-bit στη συνέχεια ακολουθούν 35 γύροι της συναρτήσεως R_d και ένας γύρος της συναρτήσεως $R_d *$ ο οποίος ουσιαστικά είναι ένας γύρος R_d μόνο με το επίπεδο Sbox και τέλος η αντίστροφη διαδικασία του grouping το de-grouping.

(5.8)

1. Ομαδοποίηση (**Grouping**) των bits σε 256 τμήματα των 4-bit τα οποία το συνθέτουν Q_0.
2. $Q_{r+1} = R_d(Q_r, C_r^{(d)})$ Για r = 0 έως 5(d-1)-1,
3. $Q_{5(d-1)+1} = R_d^* (Q_{5(d-1)}, C_{5(d-1)}^d)$
4. Από-ομαδοποίηση (**De-Grouping**) των τμημάτων του $Q_{5(d-1)+1}$ για την σύνθεση του B.



Σχήμα 5.10: *Η δομή του τμήματος E.*

Grouping – De-Grouping

Η διαδικασία του Grouping περιγράφεται από το Σχήμα 5.11 ενώ η αντίστροφη διαδικασία περιγράφεται από το Σχήμα 5.12. Με q_i συμβολίζονται τα τμήματα των 4-bit και με τα A^i, B^i συμβολίζονται τα ανεξάρτητα bit πριν το Grouping και μετά το De-Grouping.

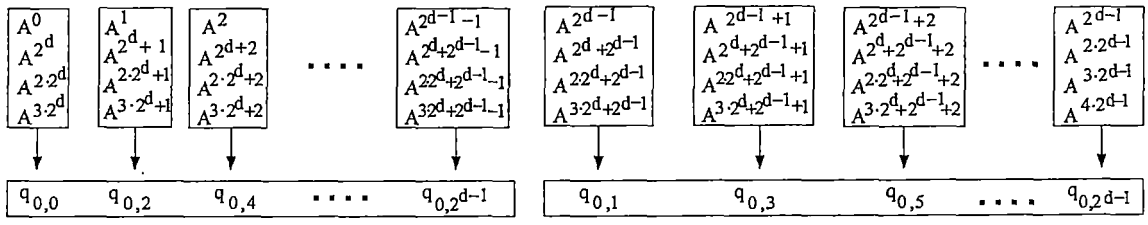
1. Για $i = 0$ έως $2^{d-1} - 1$,

{

$q_{0,2i} = A^{i}, \quad A^{i+2^{d}}, \quad A^{i+2\cdot 2^{d}}, \quad A^{i+3\cdot 2^{d}}$ ← *Τμήμα 2i*

$q_{0,2i+1} = A^{i+2^{d-1}}, \quad A^{i+2^{d-1}+2^{d}}, \quad A^{i+2^{d-1}+2\cdot 2^{d}}, \quad A^{i+2^{d-1}+3\cdot 2^{d}}$ ← *Τμήμα 2i + 1*

}



Σχήμα 5.11: *Grouping.*

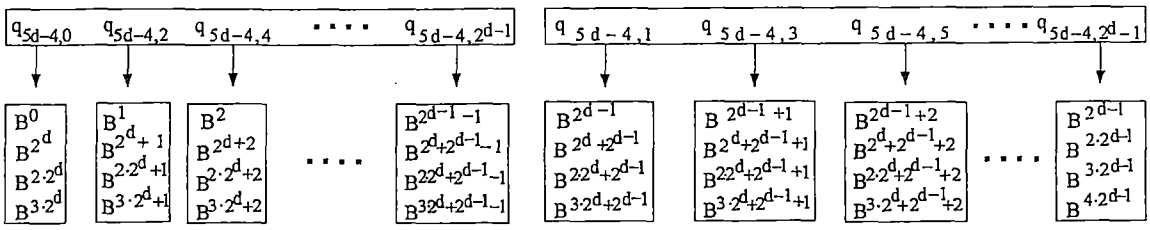
1. Για $i = 0$ έως $2^{d-1} - 1$,

{

$B^{i}, \quad B^{i+2^{d}}, \quad B^{i+2\cdot 2^{d}}, \quad B^{i+3\cdot 2^{d}} = q_{5(d-1)+1,2i}$

$B^{i+2^{d-1}}, \quad B^{i+2^{d-1}+2^{d}}, \quad B^{i+2^{d-1}+2\cdot 2^{d}}, \quad B^{i+2^{d-1}+3\cdot 2^{d}} = q_{5(d-1)+1,2i+1}$

}



Σχήμα 5.12: *De-Grouping.*

S–box

Ο JH χρησιμοποιεί ένα ζευγάρι δύο 64-bit S-box των 16 στοιχείων το καθένα. Ύστερα από την ομαδοποίηση των bits σε στοιχεία των 4-bit, το κάθε στοιχείο κατευθύνεται σε ένα ζευγάρι S-boxes. Ανάλογα από τον γύρο που βρίσκεται η συνάρτηση R_d χρησιμοποιείται μια συγκεκριμένη σταθερά (Round Constant Value) η οποία αποφασίζει ποιό από τα δύο S-box του ζευγαριού θα χρησιμοποιηθεί για την αντικατάσταση του στοιχείου. Εφόσον το κύριο τμήμα των δεδομένων αποτελείται από 1024-bit που συνεπάγεται 256 στοιχεία των 4-bit είναι απαραίτητα 256 ζευγάρια S-boxes, δηλαδή ένα για το κάθε στοιχείο στη περίπτωση που η αντικατάσταση απαιτείται να ολοκληρώνεται σε ένα γύρο. Στο Πίνακα 5.4 δίνονται τα S-boxes του αλγορίθμου JH.

Πίνακας 5.4: *Τα δύο S-boxes του αλγορίθμου JH.*

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S0(x)	9	0	4	B	D	C	3	F	1	A	2	6	7	5	8	E
S1(x)	3	C	6	D	5	7	1	9	F	2	0	4	B	A	E	8

L επίπεδο

Το L επίπεδο αποτελεί ένα γραμμικό (Linear) μετασχηματισμό μεταξύ κάθε δύο συνεχόμενων 4-bit στοιχείων. Ο μετασχηματισμός που πραγματοποιείται είναι ο πολλαπλασιασμός και η πρόσθεση δύο στοιχείων με δύο σταθερούς αριθμούς. Ο μετασχηματισμός L δίνεται από τη παρακάτω σχέση.

$$(C,D) = L(A,B) = (5 \cdot A + 2 \cdot B,\ 2 \cdot A + B) \tag{5.9}$$

Οι πράξεις του πολλαπλασιασμού και της πρόσθεσης όπως έχει ήδη προαναφερθεί στο υποκεφάλαιο 5.1 συμβαίνουν σε πεπερασμένα πεδία αριθμών και πιο συγκεκριμένα στον αλγόριθμο JH σε $GF(2^4)$. Αυτό συνεπάγεται ότι η πράξη της πρόσθεσης της παραπάνω σχέσης ορίζεται ως η πράξη αποκλειστικού-Η και η πράξη του πολλαπλασιασμού ως ο πολλαπλασιασμός modulo δυαδικών πολυωνύμων με πρωτεύων πολυώνυμο το $\chi^4 + \chi + 1$. Έτσι η σχέση (5.9) απλοποιείται και ουσιαστικά χρησιμοποιείται μόνο η πράξη

αποκλειστικού-Η. Για παράδειγμα αν τα Α και Β είναι δύο συνεχόμενες 4-bit λέξεις όπου τα A^0, A^1, A^2, A^3 είναι τα bit της λέξης Α και τα $B^0B^1B^2B^3$ τα bit της λέξης Β τότε ο μετασχηματισμός L υπολογίζεται από τα παρακάτω βήματα.

(5.10)

$$D^0 = B^0 \oplus A^1 \qquad D^1 = B^1 \oplus A^2$$

$$D^2 = B^2 \oplus A^3 \oplus A^0 \qquad D^3 = B^3 \oplus A^0$$

$$C^0 = A^0 \oplus D^1 \qquad C^1 = A^1 \oplus D^2$$

$$C^2 = A^2 \oplus D^3 \oplus D^0 \qquad C^3 = A^3 \oplus D^0$$

Ρ επίπεδο

Το Ρ επίπεδο αποτελείται από τρείς συναρτήσεις οι οποίες διαδοχικά αλλάζουν τη σειρά των bit της εισόδου του επιπέδου Ρ. Οι συναρτήσεις αυτές είναι οι π_d, P'_d και Φ_d. Ο υπολογισμός της συναρτήσεως π_d δίνεται παρακάτω όπου a_i είναι η είσοδος της συναρτήσεως και b_i η έξοδος της συναρτήσεως.

(5.11)

$$b_{4i+0} = a_{4i+0} \qquad \text{Για } i = 0 \text{ έως } 2^{d-2} - 1$$

$$b_{4i+1} = a_{4i+1} \qquad \text{Για } i = 0 \text{ έως } 2^{d-2} - 1$$

$$b_{4i+2} = a_{4i+3} \qquad \text{Για } i = 0 \text{ έως } 2^{d-2} - 1$$

$$b_{4i+3} = a_{4i+2} \qquad \text{Για } i = 0 \text{ έως } 2^{d-2} - 1$$

Η έξοδος της συναρτήσεως π_d είναι η είσοδος της συναρτήσεως P'_d και η υλοποίηση της δίνεται παρακάτω.

(5.12)

$$b_i = a_{2i} \qquad \text{Για } i = 0 \text{ έως } 2^{d-1} - 1$$

$$b_{i+2^{d-1}} = a_{2i+1} \qquad \text{Για } i = 0 \text{ έως } 2^{d-1} - 1$$

Το τελευταίο στάδιο του επιπέδου P_d είναι το Φ_d το οποίο και υπολογίζεται από τα παρακάτω βήματα.

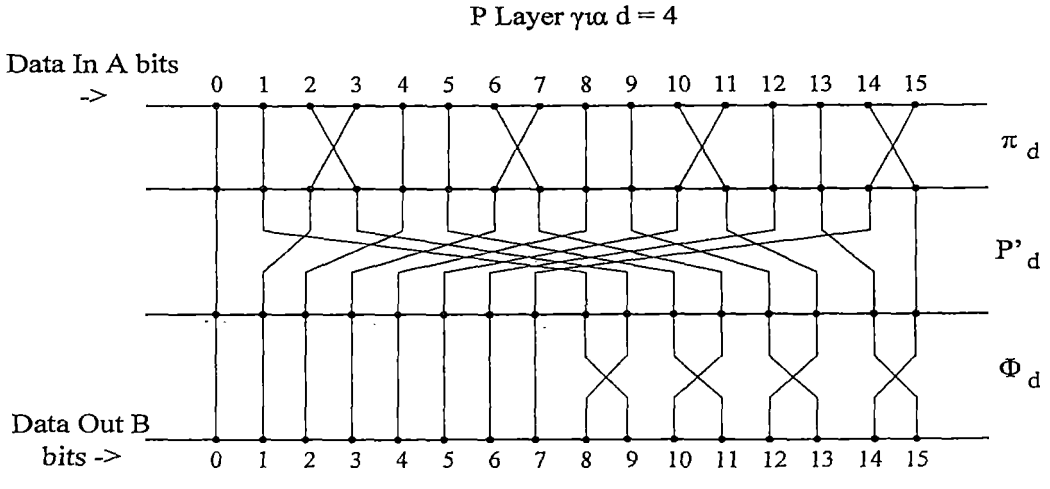
(5.13)

$$b_i = a_i \qquad \text{Για } i = 0 \text{ έως } 2^{d-1} - 1$$

$$b_{2i+0} = a_{2i+1} \qquad \text{Για } i = 2^{d-2} \text{ έως } 2^{d-1} - 1$$

$$b_{2i+1} = a_{2i+0} \qquad \text{Για } i = 2^{d-2} \text{ έως } 2^{d-1} - 1$$

Στο Σχήμα 5.13 δίνεται ένα παράδειγμα της λειτουργίας του P επιπέδου για d = 4, δηλαδή για 16 στοιχεία.



Σχήμα 5.13: *Το P επίπεδο για d = 4.*

Η συνάρτηση R_d

Η συνάρτηση R_d αποτελείται από τα προαναφερθέντα τρία επίπεδα και ολοκληρώνεται σε 35 γύρους. Όπου εάν η είσοδος στην R_d είναι A και $C_r^{(d)}$ η εκάστοτε σταθερά τότε $B = R_d(A, C_r^{(d)})$. Παρακάτω δίνονται τα βήματα εκτέλεσης της συνάρτησης R_d.

(5.14)

1. Για $i = 0$ έως $2^d - 1$

   {

   Αν $C_{round}^{(d),i} = 0$, τότε $u_i = S_{0(a_i)}$

   Αν $C_{round}^{(d),i} = 1$, τότε $u_i = S_{1(a_i)}$

   }

2. $(w_{2i}, w_{2i+1}) = L(u_{2i}, u_{2i+1})$ Για $0 \le i \le 2^{d-1} - 1$

3. $(b_0, b_1, \ldots, b_{2^d-1}) = P_d(w_0, w_1, \ldots, w_{2^d-1})$

Η συνάρτηση $C_r^{(d),i}$ στο βήμα 1 δίνει τη σταθερά του κάθε γύρου R_d που χρειάζεται για να επιλεγεί το σωστό S-box. Για τη συνάρτηση R_d υπολογίζονται 35 σταθερές των 256-bit η κάθε μια. Σε κάθε γύρο χρησιμοποιείται μια σταθερά και αντιστοιχούνται τα 256-bit κάθε σταθεράς σε κάθε ένα τμήμα των 4-bit της εξόδου του επιπέδου Grouping. Όπως προκύπτει και από το βήμα 1 του αλγορίθμου, εάν αυτό το bit ισούται με λογικό 0 τότε επιλέγεται το S-box S_0 αλλιώς εάν ισούται με λογικό 1 τότε επιλέγεται το S-box S_1.

Συνολικά υπολογίζονται 35 + 1 σταθερές. Η τριακοστή έκτη σταθερά που υπολογίζεται χρησιμοποιείται στη συνάρτηση R_d* του Σχήματος 5.10 η οποία περιλαμβάνει μόνο το επίπεδο S-box και πραγματοποιείται ένας γύρος μόνο ο οποίος έπεται του 35 γύρου της συναρτήσεως R_d .

Για τον υπολογισμό των 36 σταθερών χρησιμοποιείται η συνάρτηση R_{d-2}. Ο υπολογισμός της πρώτης σταθεράς προκύπτει από τη συνάρτηση R_{d-2} με είσοδο σταθεράς το

λογικό 0. Ο υπολογισμός των επόμενων σταθερών προκύπτει από τις ακριβώς προηγούμενες σταθερές και ως είσοδο σταθεράς στη συνάρτηση R_{d-2} δίνεται πάλι το λογικό 0. Ο υπολογισμός των 36 σταθερών δίνεται από το παρακάτω αλγόριθμο.

(5.15)

$$C^{(d)}_{\text{round } 0} = R_{d-2}(0,0)$$

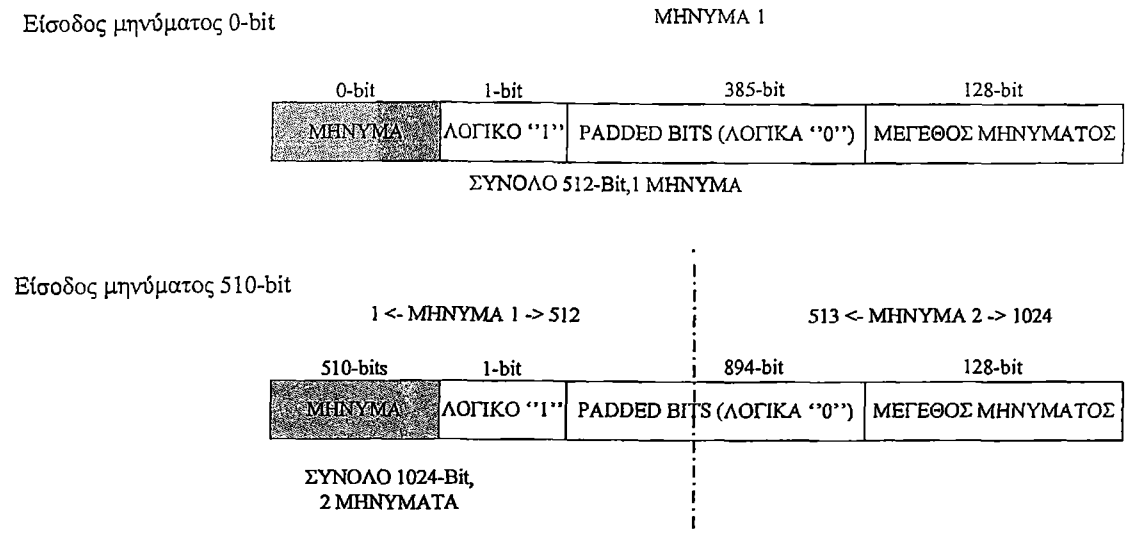
$$C^{(d)}_{\text{round}} = R_{d-2}(C^{(d)}_{\text{round-1}},0) \qquad \text{Για } 1 \leq \text{round} \leq 5(d-1)$$

Padding

Το αρχικό μήνυμα προκειμένου να χρησιμοποιηθεί από τον αλγόριθμο JH πρέπει να είναι πολλαπλάσιο των 512-bit και να φέρει τη πληροφορία του αρχικού του μεγέθους. Πιο συγκεκριμένα, μετά το τέλος του μηνύματος προστίθεται το λογικό 1 και στη συνέχεια ακολουθούν 384 - 1 + (- μήκος μηνύματος modulo 512) λογικά 0 και τέλος ακολουθεί ένα τμήμα 128-bit το οποίο είναι ίσο με τη δυαδική αναπαράσταση του αρχικού μήκους του μηνύματος. Στη περίπτωση που είσοδος του αλγορίθμου είναι μήκους 0 bit τότε παράγεται μόνο ένα μήνυμα μετά το padding αλλιώς για μήκη εισόδου από 1 έως 511-bit τότε παράγονται δύο μηνύματα. Συνεπώς, μετά τη διαδικασία του padding στο αρχικό μήνυμα θα έχουν προστεθεί τουλάχιστον 512-bit. Στο Σχήμα 5.14 δίνονται δύο παραδείγματα για τη διαδικασία του padding στον αλγόριθμο JH.

Αρχικοποίηση

Το κύριο τμήμα δεδομένων που διαχειρίζεται ο αλγόριθμος JH, το οποίο συμβολίζεται με αγγλικό γράμμα H, έχει 4 διαφορετικές αρχικοποιήσεις οι οποίες εξαρτούνται από το μέγεθος της εξόδου που έχει επιλεχτεί να δημιουργηθεί από τον αλγόριθμο (JH-224, JH-256, JH-384 και JH-512). Από την σχέση 5.6 προκύπτει η σχέση 5.15 όπου από τη υπολογίζεται η αρχικοποίηση του τμήματος H.



Σχήμα 5.14: *Δύο παραδείγματα padding.*

(5.16) $$H^{(0)} = F_d(H^{(-1)}, M^{(0)})$$

Τα στοιχεία του $M^{(0)}$ τίθενται όλα στο λογικό 0. Στο τμήμα $H^{(-1)}$ τα δύο πρώτα byte αντιπροσωπεύουν το μέγεθος της εξόδου που έχει επιλεχτεί για τον αλγόριθμο. Δηλαδή, 0x00E0,0x0100,0x0180 και 0x0200 για τις εξόδους JH-224, JH-256, JH-384 και JH-512 αντίστοιχα. Τα υπόλοιπα byte του $H^{(-1)}$ τίθενται στο λογικό 0.

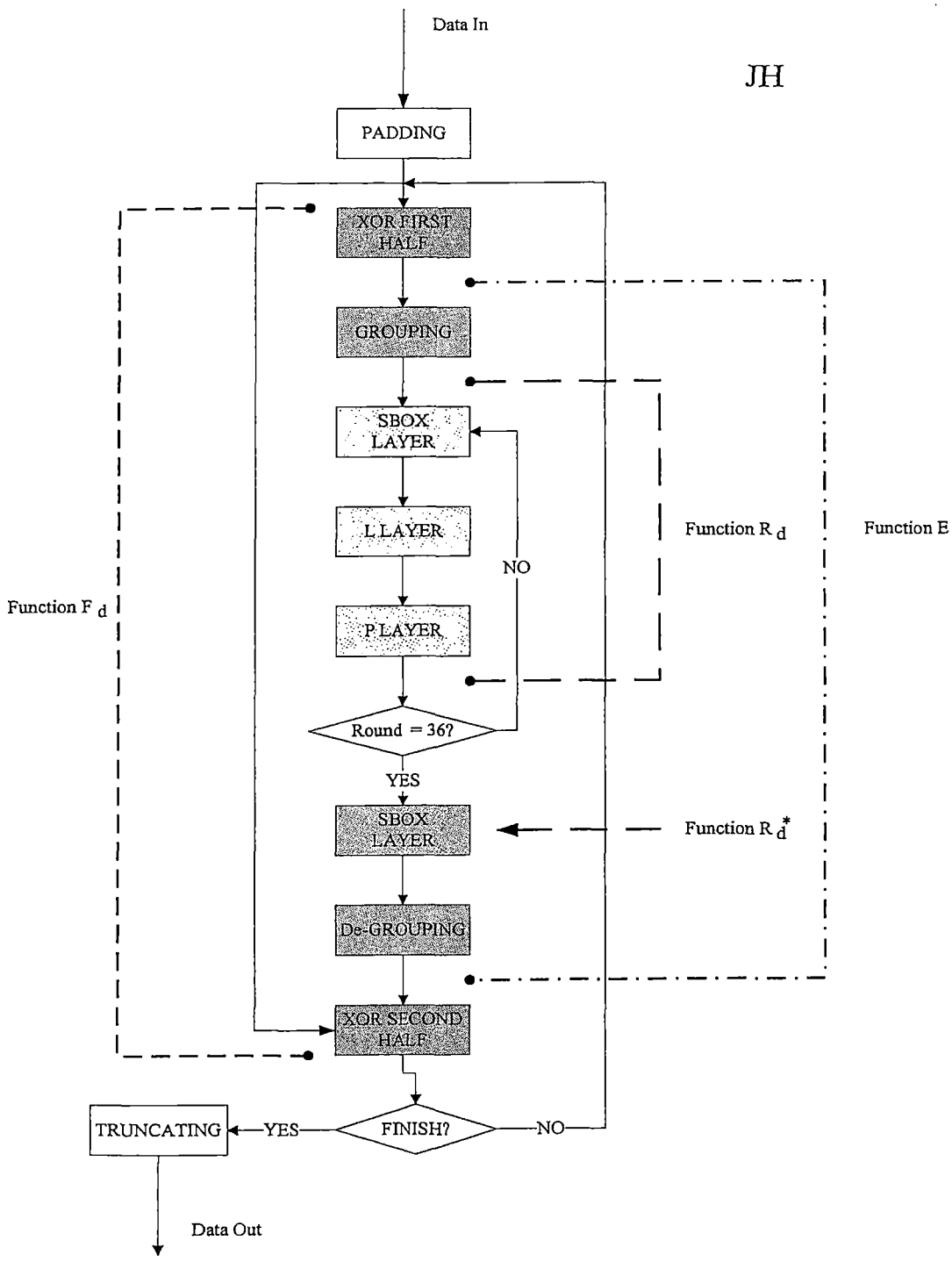
Truncating

Όπως έχει ήδη προαναφερθεί ο αλγόριθμος JH υποστηρίζει 4 μεγέθη εξόδου. Όταν φτάσει στο τελικό στάδιο η διαδικασία υπολογισμού της σύνοψης, όλων των εισερχόμενων μηνυμάτων τότε ανάλογα την έξοδο που έχει επιλεχτεί αφαιρούνται τα bit που δε χρειάζονται από την αρχή του τμήματος H. Αυτή η διαδικασία ονομάζεται truncating. Πιο συγκεκριμένα, εάν έχει επιλεχτεί έξοδος μεγέθους 224-bit τότε από τα 1024-bit του τμήματος H αφαιρούνται τα bit από 0 έως 799 και τα 224-bit (λιγότερα σημαντικά bits) που μένουν είναι η έξοδος του αλγορίθμου.

Η παραπάνω διαδικασία είναι ίδια και για τις υπόλοιπες εξόδους που υποστηρίζει ο αλγόριθμος JH. Στη περίπτωση της εξόδου 256-bit θα εξαιρόντουσαν από τμήμα H τα bit 0

έως 767, στη περίπτωση 384-bit θα εξαιρόντουσαν από τμήμα Η τα bit 0 έως 639 και τέλος στη περίπτωση της εξόδου 512-bit θα εξαιρόντουσαν από το τμήμα Η τα bit 0 έως 511.

Στο Σχήμα 5.15 παρουσιάζεται ένα διάγραμμα ροής με τη συνολική λειτουργία του αλγορίθμου JH καθώς και ο τρόπος με τον οποίο επεξεργάζονται τα δεδομένα.



Σχήμα 5.15: *Ο αλγόριθμος JH σε διάγραμμα ροής.*

ΑΝΑΦΟΡΕΣ

[1] http://en.wikipedia.org/wiki/Finite_field_arithmetic.

[2] Shai Halevi, William E. Hall, Charanjit S. Jutla, The Hush Function «Fugue» .

[3] Hongjun Wu, The Hush Function «JH» .

ΚΕΦΑΛΑΙΟ 6
ΥΛΟΠΟΙΗΣΗ ΤΩΝ ΑΛΓΟΡΙΘΜΩΝ FUGUE ΚΑΙ JH

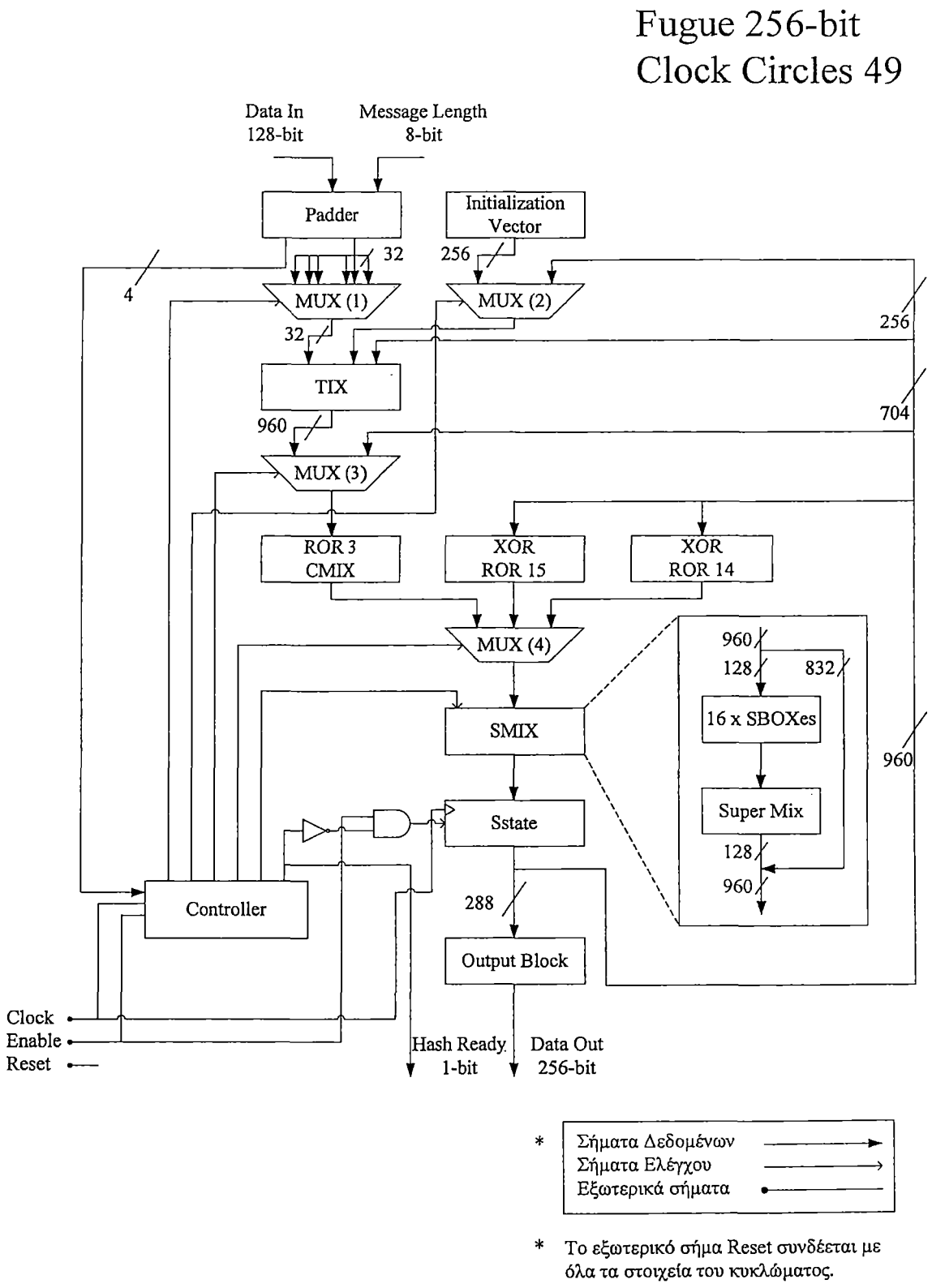
6.1 Fugue συνάρτηση κατακερματισμού

Στα πλαίσια της υλοποιήσεως του αλγορίθμου Fugue στη παρούσα εργασία σχεδιάστηκε ένα κύκλωμα που υλοποιεί τον αλγόριθμο. Επίσης σχεδιάστηκαν άλλες 13 παραλλαγές του βασικού κυκλώματος που στόχο έχουν λιγότερους επεξεργαστικούς κύκλους ή τη μείωση της κατανάλωσης του κυκλώματος χρησιμοποιώντας τις τεχνικές που παρουσιάστηκαν στο κεφάλαιο 4. Το βασικό κύκλωμα θα παρουσιαστεί και θα αναλυθεί πρώτο και στη συνέχεια θα παρουσιαστούν οι διαφοροποιήσεις του οι οποίες αποτελούν τις 13 παραλλαγές.

Fugue 256 Clock Circles 49

Στο σχήμα 6.1 δίνεται το σχεδιάγραμμα του βασικού κυκλώματος Fugue το οποίο ονομάζεται Fugue 256 Clock Circles 49. Η είσοδος του κυκλώματος είναι 128-bit ενώ η έξοδος 256-bit. Στο κύκλωμα υπολογίζεται η σύνοψη του μηνύματος στη χειρότερη περίπτωση σε 49 ωρολογιακούς παλμούς. Οι κύκλοι ρολογιού που θα χρειαστούν εξαρτούνται άμεσα από το μέγεθος του εισερχόμενου μηνύματος. Για εισόδους 0-bit απαιτούνται 41 κύκλοι, για εισόδους από 1 έως 32 bit απαιτούνται 43 κύκλοι, για εισόδους από 33 έως 64 bit απαιτούνται 45 κύκλοι, για εισόδους από 65 έως 96 bit απαιτούνται 47 κύκλοι και τέλος για εισόδους από 97 έως 128 bit απαιτούνται 49 κύκλοι ρολογιού. Για αυτή τη παραμετρική συμπεριφορά του κυκλώματος χρειάζεται μια επιπλέον είσοδος στο κύκλωμα που ονομάζεται Message Length και αποτελείται από 8–bit. Το Message Length φέρει τη πληροφορία για το μήκος του εισερχόμενου μηνύματος. Επίσης μαζί με την έξοδο των 256-bit υπάρχει και ένα επιπλέον σήμα ενός bit το οποίο οδηγείται σε λογικό 1 μετά την ολοκλήρωση του υπολογισμού της σύνοψης του αρχικού μηνύματος και ονομάζεται Hash Ready. Τέλος στο κύκλωμα εισέρχονται 3 ακόμα σήματα ενός bit τα οποία είναι το ρολόι (Clock), ένα σήμα για την ασύγχρονη αρχικοποίηση του κυκλώματος (Reset) και ένα σήμα για την ασύγχρονη παύση της λειτουργίας του κυκλώματος (Enable).

Όπως αναφέρθηκε στο κεφάλαιο 5.2 ο αλγόριθμος Fugue διαχειρίζεται ένα τμήμα δεδομένων 30 στηλών όπου η κάθε στήλη αποτελείται από 4 byte δηλαδή συνολικά 960-bit. Για τη μεταφορά αυτού του τμήματος 30 στηλών στο Σχήμα 6.1 χρησιμοποιείται ένα κοινό



Σχήμα 6.1: *Fugue 256-bit, Clock Circles 49.*

σύνολο καλωδίων (Bus) πλάτους 960-bit. Το εννιακοσιοστό εξηκοστό bit του Bus συμβολίζει το πρώτο bit της στήλης S0 ενώ το τελευταίο bit του bus συμβολίζει και το τελευταίο bit της στήλης S29 . Ουσιαστικά οι 30 στήλες του τμήματος έχουν τοποθετηθεί σε σειρά στο bus των 960-bit με τη στήλη S0 να βρίσκεται στις θέσεις 959-928 του bus ενώ η στήλη S29 να βρίσκεται στις θέσεις 31-0 του bus.

Το πρώτο συνδυαστικό υποκύκλωμα που συναντάται στο Σχήμα 6.1 είναι ο Padder και υλοποιεί τη διαδικασία του Padding που περιγράφτηκε στη παράγραφο «Padding και αρχικοποίηση» του κεφαλαίου 5.2 . Έχει δύο εισόδους, μία για το εισερχόμενο μήνυμα μήκους 128-bit και μια για το μέγεθος του μηνύματος μήκους 8-bit. Επίσης έχει δύο εξόδους, μία για τα δεδομένα μετά το Padding μήκους 192-bit και μια έξοδος 4-bit η οποία περιέχει πληροφορία για τους κύκλους ρολογιού που θα χρειαστούν και κατευθύνεται στο κύκλωμα ελέγχου, τον Controller.

Η έξοδος του Padder στη συνέχεια διαχωρίζεται σε 6 ομάδες των 32-bit καθώς βάση του αλγορίθμου το κύκλωμα επεξεργάζεται μηνύματα μήκους 32-bit. Υπεύθυνος για το ποια ομάδα των 32-bit θα επεξεργαστεί είναι ο πολυπλέκτης 6 σε 1 (MUX (1)) που έπεται του Padder.

Στη συνέχεια ο πολυπλέκτη 6x1 συνδέεται με το κύκλωμα TIX το οποίο περιγράφτηκε στη παράγραφο «TIX» του κεφαλαίου 6.2 . Το κύκλωμα αυτό έχει 3 εισόδους, μια των 32-bit για την έξοδο του πολυπλέκτη 6x1, μια των 256-bit η οποία συνδέεται με το πίνακα αρχικοποίησης (Initialization Vector) του Σχήματος 5.6 ή με την έξοδο του καταχωρητή Sstate και τέλος μια είσοδος των 704-bit η οποία συνδέεται και αυτή με την έξοδο του καταχωρητή Sstate. Το κύκλωμα TIX συνδέεται με το πίνακα αρχικοποίησης μόνο στο πρώτο κύκλου ρολογιού ενώ στους υπόλοιπους κύκλους συνδέεται μόνο με την έξοδο του καταχωρητή Sstate. Υπεύθυνος για τον αν θα εισαχθεί ο πίνακας αρχικοποίησης είναι ο πολυπλέκτης 2 σε 1 (MUX (2)) που έπεται του κυκλώματος Initialization Vector.

Όπως περιγράφηκε στις παραγράφους «Round Transformation R, Final Round G και TIX» του κεφαλαίου 5.2 το κύκλωμα TIX δε χρησιμοποιείται σε όλη τη διάρκεια λειτουργίας του κυκλώματος ούτε σε όλους τους γύρους του αλγορίθμου Fugue και την απόφαση για το πότε θα χρησιμοποιηθεί τη παίρνει ο πολυπλέκτης 2 σε 1 (MUX (3)) που βρίσκεται μετά το κύκλωμα TIX. Ο παραπάνω πολυπλέκτης συνδέεται με τις εξόδους του TIX και του καταχωρητή Sstate. Η έξοδος του πολυπλέκτη συνδέεται με το επόμενο συνδυαστικό κύκλωμα που υλοποιεί τη δεξιά περιστροφή κατά 3 στήλες της είσοδος του και τις λογικές

πράξεις αποκλειστικού-Η. Το κύκλωμα αυτό ονομάζεται ROR3 CMIX και περιγράφτηκε στις παραγράφους «ROR και CMIX» του κεφαλαίου 5.2 .

Στο Σχήμα 6.1 μαζί με το κύκλωμα ROR3 CMIX φαίνονται στο ίδιο επίπεδο και άλλα δύο κυκλώματα τα οποία είναι παρόμοια με το ROR3 CMIX και ονομάζονται XOR ROR15 και XOR ROR14. Οι πράξεις αποκλειστικού-Η των δύο κυκλωμάτων περιγράφονται από τη Σχέση 5.2 ενώ η δεξιά περιστροφή στη μια περίπτωση συμβαίνει κατά 15 στήλες ενώ στην άλλη κατά 14 στήλες. Οι είσοδοι και οι έξοδοι και των τριών παραπάνω κυκλωμάτων έχουν μήκος 960-bit.

Όπως αναφέρθηκε και στις παραγράφους «Round Transformation R και Final Round G» του κεφαλαίου 5.2 τα κυκλώματα ROR3 CMIX, XOR ROR15 και XOR ROR14, σε κάθε γύρο του αλγορίθμου κάθε φορά μόνο ένα από τα παραπάνω χρησιμοποιείται. Την απόφαση για το ποιό θα χρησιμοποιηθεί τη λαμβάνει ο πολυπλέκτης 3 σε 1 (MUX (4)) που έπεται των κυκλωμάτων αυτών.

Η έξοδος του πολυπλέκτη 3 σε 1 συνδέεται με το κύκλωμα SMIX που περιγράφηκε στη παράγραφο «SMIX» του κεφαλαίου 5.2 . Το κύκλωμα SMIX δέχεται δύο εισόδους. Η μια είναι η έξοδος του πολυπλέκτη 3 σε 1 μήκους 960-bit και η άλλη είναι ένα εσωτερικό σήμα αρχικοποίησης των SBOXes. Η έξοδος του SMIX είναι μήκους 960-bit. Όπως έχει ήδη αναφερθεί στο κεφάλαιο 5.2 η διαδικασία του SMIX πραγματοποιείται μόνο στις 4 πρώτες στήλες του τμήματος δεδομένων 960-bit που επεξεργάζεται ο αλγόριθμος Fugue. Αυτό συνεπάγεται το διαχωρισμό της εισόδου μήκους 960-bit σε δύο τμήματα. Το πρώτο τμήμα περιλαμβάνει τις 4 πρώτες στήλες και έχει μήκος 128-bit και το δεύτερο τμήμα έχει τις υπόλοιπες στήλες που δεν επεξεργάζονται από το κύκλωμα SMIX και έχει μήκος 832-bit. Το τμήμα των 832-bit οδηγείται κατευθείαν στην έξοδο του SMIX ενώ το τμήμα των 128-bit οδηγείται στο υποκύκλωμα εκείνο που περιέχει τα 16 SBOXes που απαιτούνται για την αντικατάσταση των 128-bit σε ένα κύκλο ρολογιού. Στη συνέχεια η έξοδος των SBOXes συνδέεται με το κύκλωμα Super Mix που υλοποιεί τον αριστερό πολλαπλασιασμό πινάκων μεταξύ της εξόδου των SBOXes και του Πίνακα 5.3. Τέλος η έξοδος του SMIX γίνεται από την επανένωση του τμήματος 832-bit και της εξόδου του υποκυκλώματος Super Mix που έχει μήκος 128-bit.

Για το συγχρονισμό των δεδομένων χρησιμοποιείται ένας καταχωρητής μεγέθους 960-bit που ονομάζεται Sstate. Ο State είναι θετικά ακμοπυροδότητος από το ωρολογιακό σήμα και παρέχεται και η δυνατότητα παύσης της λειτουργίας του από το εξωτερικό σήμα

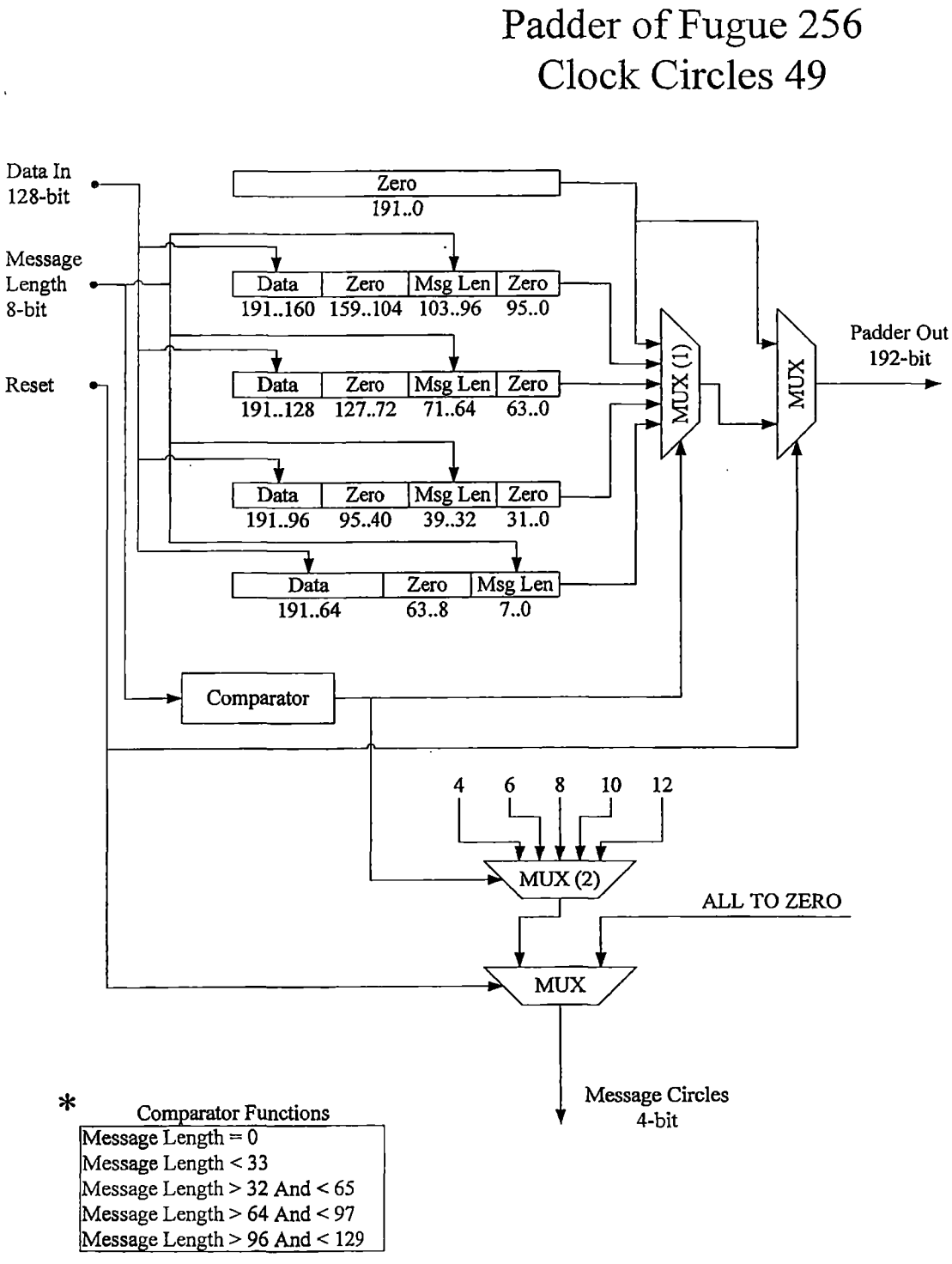
Enable. Επίσης ο καταχωρητής σταματάει τη λειτουργία του όταν ολοκληρωθεί ο υπολογισμός της σύνοψης μέσω ενός σήματος ελέγχου που προέρχεται από κύκλωμα Control και ονομάζεται Hash Ready. Για το λόγο αυτό στην είσοδο του σήματος ελέγχου του καταχωρητή υλοποιείται η πράξη ENABLE REGISTER = External Enable AND $\overline{\text{Hash Ready}}$.

Μετά το καταχωρητή Sstate ακολουθεί το κύκλωμα Output Block που υλοποιεί τις δύο πράξεις αποκλειστικού-Η του βήματος 3 της Σχέσης 5.2 και επιλέγονται οι 8 στήλες που θα αποτελέσουν την έξοδο του κυκλώματος Fugue 256. Η είσοδος του κυκλώματος είναι 288-bit και η έξοδος είναι 256-bit.

Τέλος το κύκλωμα που είναι υπεύθυνο για τη κατάσταση όλων των πολυπλεκτών του Σχήματος 6.1 αλλά και του σήματος Hash Ready όπως και του σήματος αρχικοποίησης των SBOXes ονομάζεται Controller. Έχει 3 εισόδους για τα εξωτερικά σήματα και μια ακόμα είσοδο για το σήμα που περιέχει τη πληροφορία για τους κύκλους επεξεργασίας το οποίο προέρχεται από το κύκλωμα Padder. Επίσης έχει 4 εξόδους για τον έλεγχο των 4 πολυπλεκτών μια έξοδο για την αρχικοποίηση των SBOXes και μια ακόμα έξοδο για το σήμα Hash Ready.

Padder

Στο Σχήμα 6.2 δίνεται το κύκλωμα του Padder. Όπως έχει αναφερθεί το κύκλωμα Fugue 256 είναι παραμετρικό όσων αφορά τους κύκλους ρολογιού που χρειάζονται για την επεξεργασία του μηνύματος. Βάση του αλγορίθμου οι κύκλοι ρολογιού που χρειάζονται εξαρτούνται από το μέγεθος του μηνύματος. Για το σκοπό αυτό λοιπόν κάθε φορά που εισέρχεται κάποιο μήνυμα ανεξάρτητα από το μέγεθος του στο κύκλωμα Padder γίνονται 5 padding. Στη συνέχεια ένας πολυπλέκτης 5 σε 1 (MUX (1)) επιλέγει το σωστό padding για έξοδο βασιζόμενο στο μέγεθος του μηνύματος. Ο πολυπλέκτης στην είσοδο του σήματος επιλογής δέχεται το αποτέλεσμα ενός συγκριτή (Comparator) ο οποίος πραγματοποιεί τις συγκρίσεις που φαίνονται στο κάτω μέρος του Σχήματος 6.2. Βάση των αποτελεσμάτων αυτών των συγκρίσεων ο πολυπλέκτης επιλέγει τη σωστή έξοδο του Padder. Ακριβώς με την ίδια λογική ο δεύτερος πολυπλέκτης 5 σε 1 (MUX (2)) επιλέγει τη σωστή προκαθορισμένη τιμή για το σήμα Message Circle. Το σήμα Message Circle όπως θα αναφερθεί και στη παράγραφο για τον Controller χρησιμοποιείται από τον μετρητή του κυκλώματος για να



Σχήμα 6.2: *Το υποκύκλωμα Padder του κυκλώματος Fugue 256 Clock Circles 49.*

μπορέσουν να μειωθούν οι κύκλοι ρολογιού όταν το εισερχόμενο μήνυμα είναι μικρότερο από 96-bit.

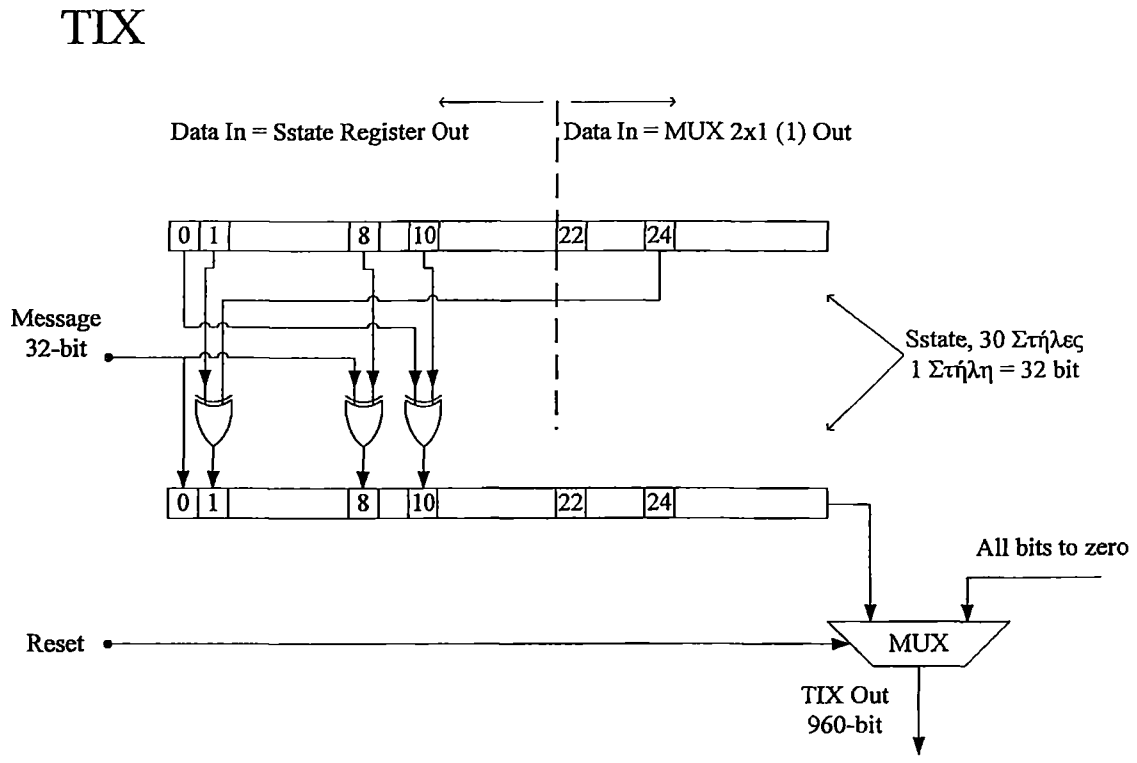
Οι δύο πολυπλέκτες 2 σε 1 (MUX) του κυκλώματος χρησιμεύουν στην ασύγχρονη αρχικοποίηση του κυκλώματος μέσω του εξωτερικού σήματος Reset.

Η έξοδος του padding έχει μήκος 192-bit και αυτό οφείλεται στο ότι απαιτούνται, βάση των προδιαγραφών, 8 byte (64-bit) για τη δυαδική αναπαράσταση του μεγέθους του αρχικού μηνύματος. Αυτό δίνει το συνολικό αποτέλεσμα των 192-bit εφόσον η είσοδος έχει μήκος 128-bit. Για τη δυαδική αναπαράσταση του αριθμού 128 απαιτούνται 8-bit. Για το λόγο αυτό τα πρώτα 56-bit των padded bits είναι προκαθορισμένα στο λογικό 0 ενώ τα υπόλοιπα 8 bit παίρνουν τιμές ανάλογα του μεγέθους του μηνύματος.

Στη πρώτη περίπτωση padding όπου η είσοδος και το μέγεθος του κυκλώματος είναι 0 όλα τα bits τίθενται στο λογικό 0 ενώ στις υπόλοιπες περιπτώσεις στο πρώτο μέρος τοποθετείται το μήνυμα και στη συνέχεια ανάλογα του μεγέθους του μηνύματος τοποθετούνται τα padded bits. Η τοποθέτηση των padded bits γίνεται βάση της προϋπόθεσης ότι το μήνυμα πρέπει να συμπληρώνεται με λογικά 0 έτσι ώστε να σχηματίζονται ομάδες πολλαπλάσιες των 32-bit.

TIX

Η υλοποίηση του TIX κυκλώματος δίνεται από το Σχήμα 6.3 . Το κύκλωμα υλοποιεί τη διαδικασία που περιγράφηκε στη παράγραφο «TIX» του κεφαλαίου 5.2 . Όπως φαίνεται και στο Σχήμα 6.1 το κύκλωμα TIX έχει 3 εισόδους. Η μια είναι η έξοδος του πολυπλέκτη 6 σε 1 (MUX (1)) που έχει μήκος 32-bit και αποτελεί το εκάστοτε μήνυμα προς επεξεργασία η άλλη είναι η έξοδος του καταχωρητή Sstate η οποία έχει μήκος 704-bit και τέλος η έξοδος του πολυπλέκτη 2 σε 1 (MUX (2)) ο οποίος ορίζει αν θα εισαχθεί στο TIX κύκλωμα ο πίνακας αρχικοποίησης ή όχι με μήκος 256-bit. Οι 3 πράξεις αποκλειστικού-Η που φαίνονται στο Σχήμα 6.3 πραγματοποιούνται μεταξύ των στηλών των τμήματος δεδομένων Sstate όπου η κάθε στήλη αποτελείται από 32-bit. Τέλος η έξοδος του TIX αποτελείται από 960-bit και δίνεται από την έξοδο του πολυπλέκτη 2 σε 1 οποίος είναι υπεύθυνος για την ασύγχρονη αρχικοποίηση TIX.

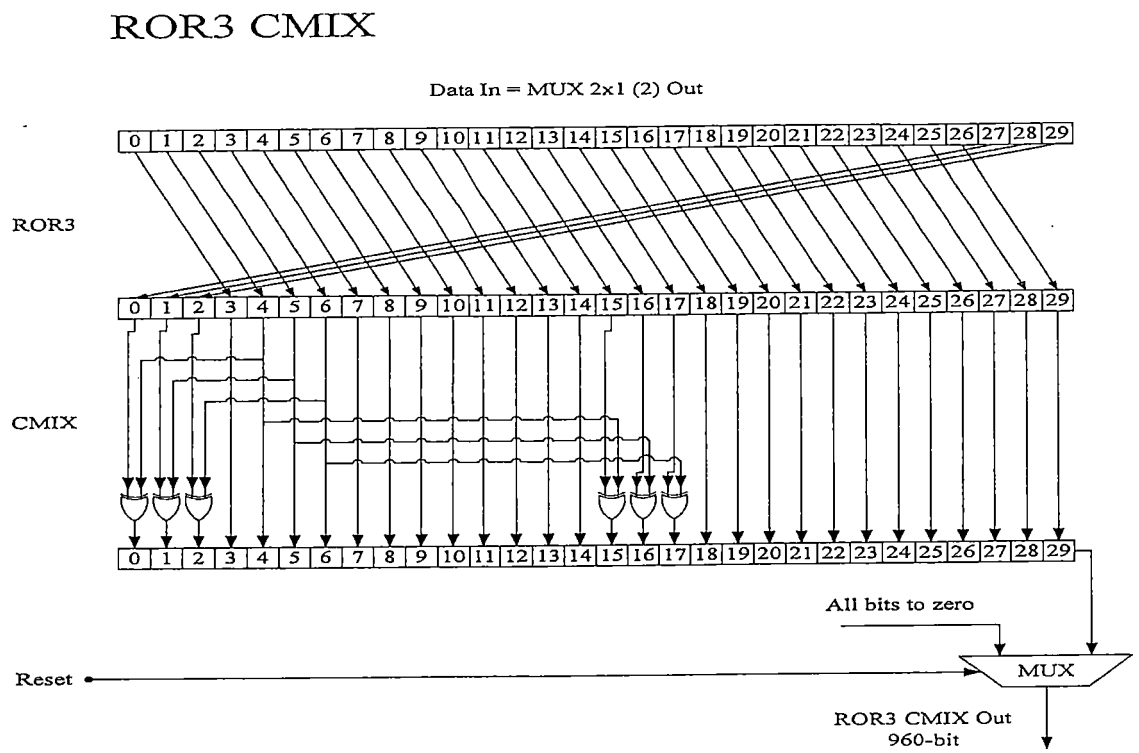


Σχήμα 6.3: *Το κύκλωμα του TIX.*

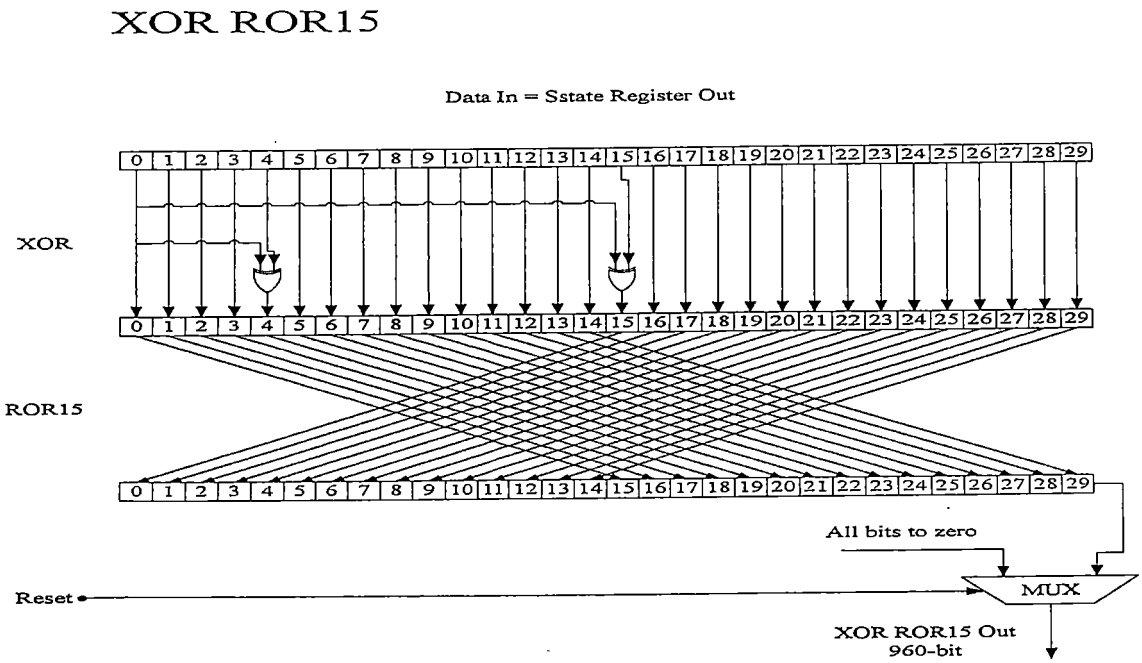
ROR3 CMIX, XOR ROR15 και XOR ROR14

Τα κυκλώματα ROR3 CMIX, XOR ROR15 και XOR ROR14 είναι παρόμοια μεταξύ τους με μοναδικές διαφορές τη σειρά με την οποία πραγματοποιούνται οι πράξεις του αποκλειστικού-Η και το πλήθος των δεξιών μετατοπίσεων. Στο Σχήμα 6.4 φαίνεται η υλοποίηση του κυκλώματος ROR3 CMIX. Στο οποίο αρχικά γίνεται μια δεξιά μετατόπιση κατά 3 θέσεις και στη συνέχεια οι πράξεις αποκλειστικού που περιγράφηκαν στη παράγραφο CMIX του κεφαλαίου 5.2 . Η είσοδος στο κύκλωμα είναι η έξοδος του πολυπλέκτη 2 σε 1 (MUX (3)) του Σχήματος 6.1 που έχει μήκος 960-bit όπως επίσης έχει και η έξοδος του κυκλώματος ROR3 CMIX.

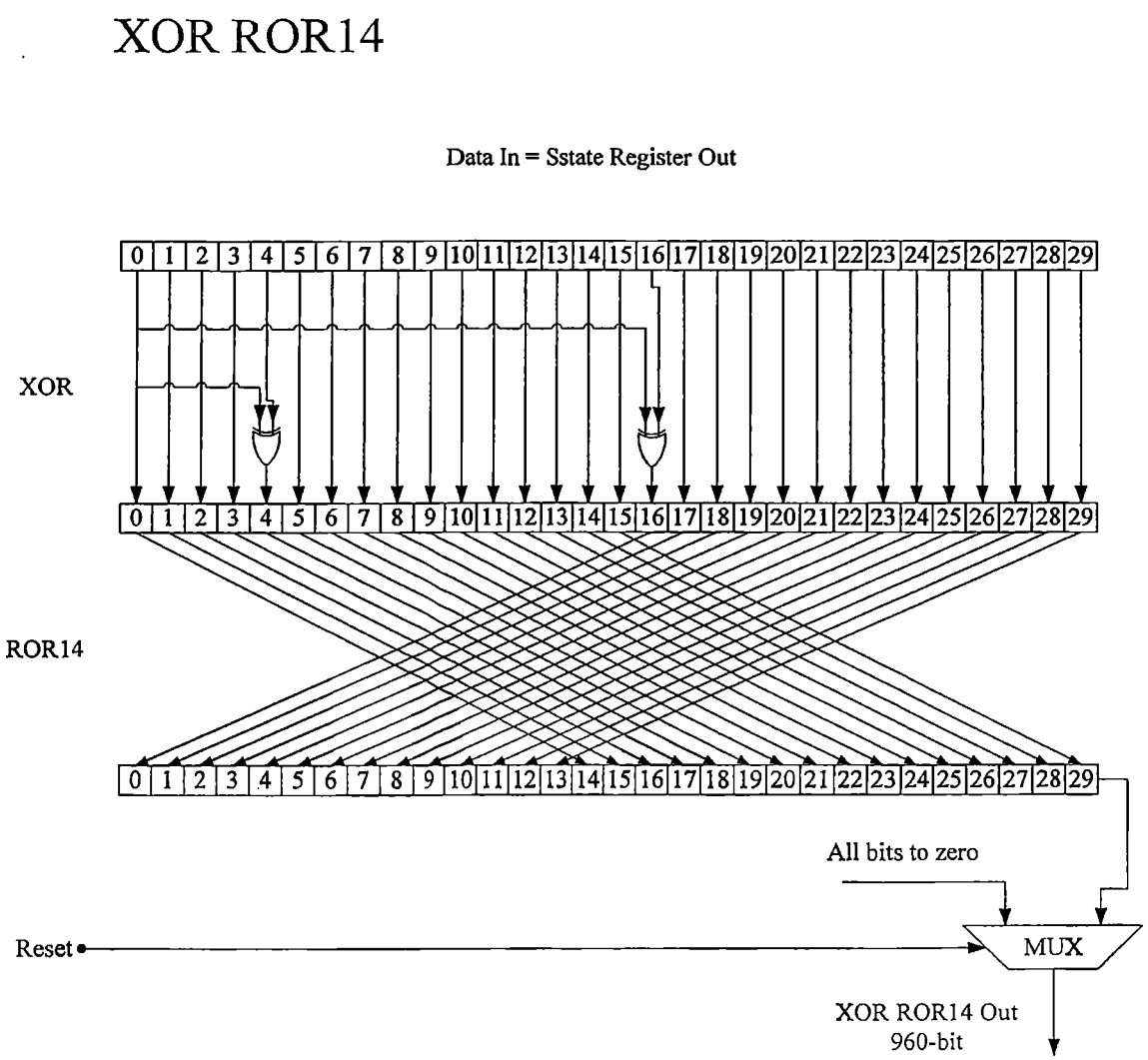
Στα Σχήματα 6.5 και 6.6 δίνονται και τα κυκλώματα των XOR ROR15 και XOR ROR14 αντίστοιχα. Στο πρώτο πραγματοποιούνται δύο πράξεις αποκλειστικού-Η και μια μετατόπιση κατά 15 θέσεις και στο δεύτερο πάλι συμβαίνουν δύο πράξεις αποκλειστικού-Η και μια μετατόπιση κατά 14 θέσεις. Οι είσοδοι και οι έξοδοι των και των δύο κυκλωμάτων έχουν μήκος 960-bit. Τέλος και στα τρία παραπάνω κυκλώματα παρέχεται η δυνατότητα ασύγχρονης αρχικοποίησης μέσω του εξωτερικού σήματος Reset.



Σχήμα 6.4: *Το κύκλωμα ROR3 CMIX.*



Σχήμα 6.5: *Το κύκλωμα XOR ROR15.*

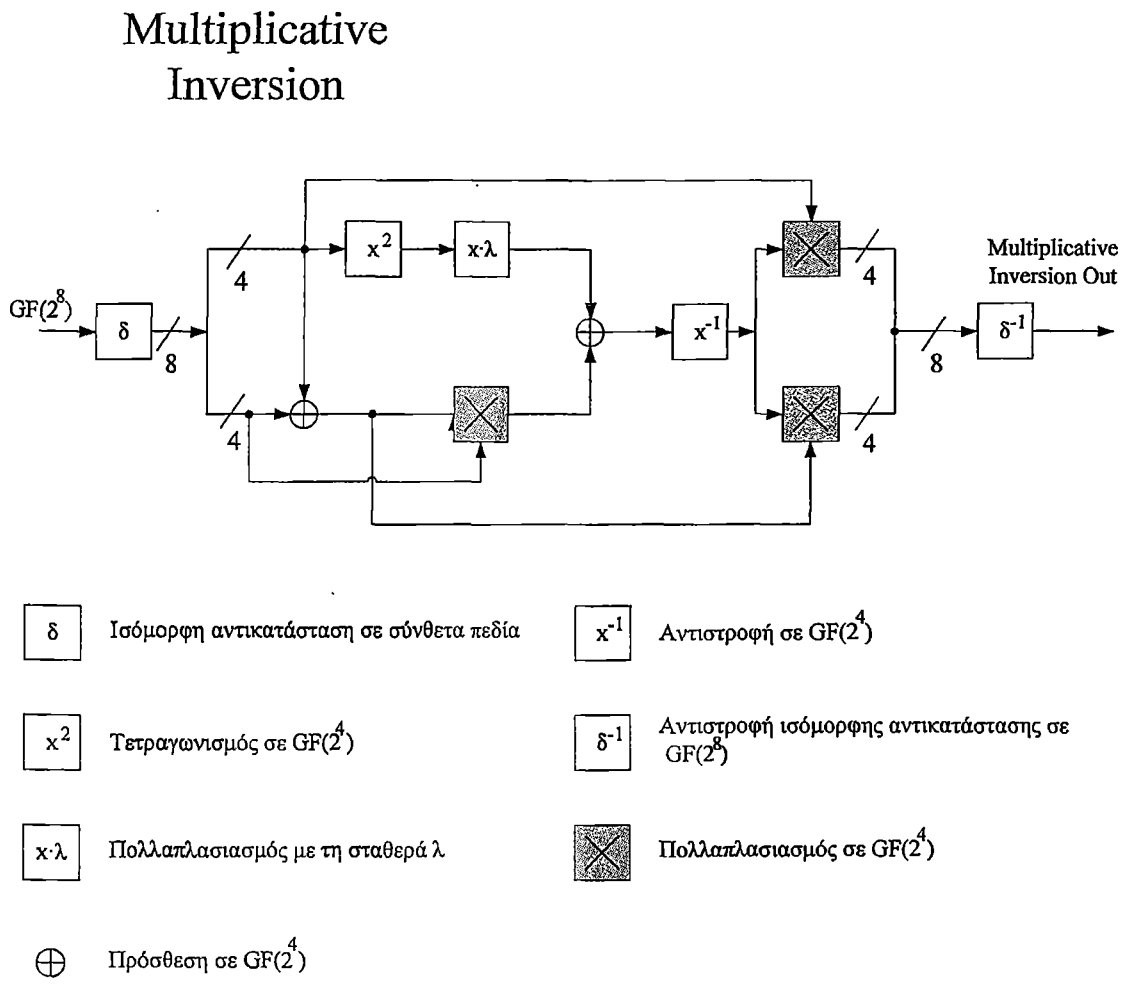


Σχήμα 6.6: *Το κύκλωμα XOR ROR14.*

SMIX SBOX

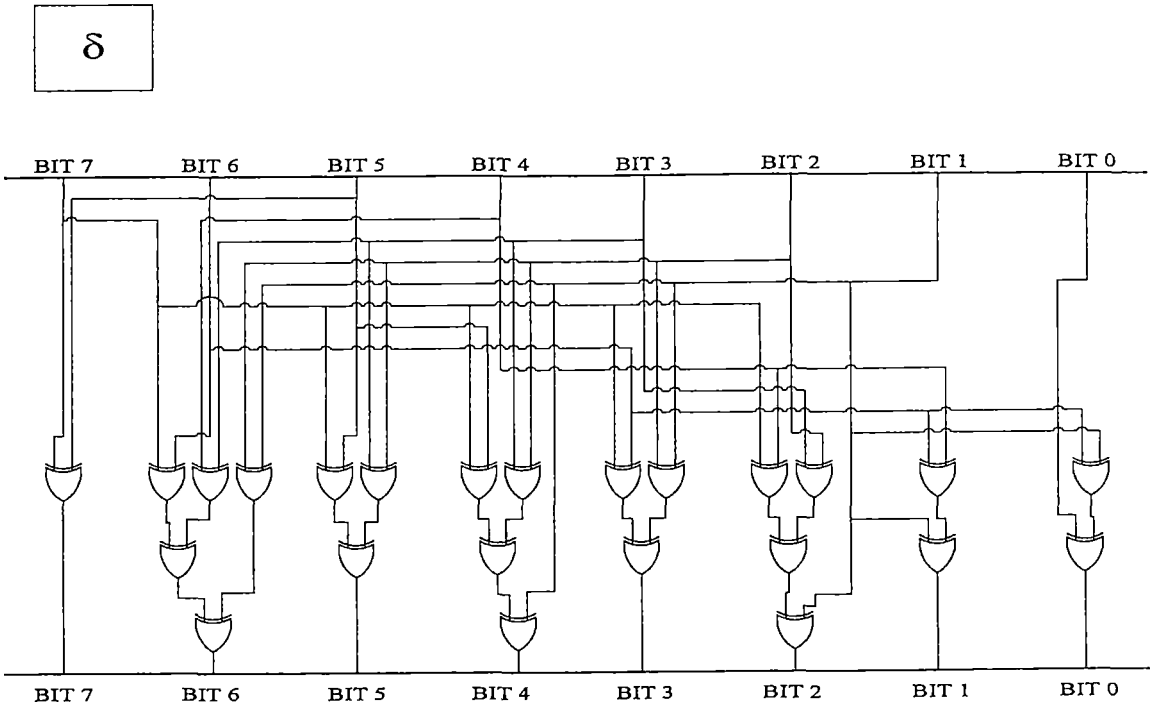
Όπως προκύπτει και από το Σχήμα 6.1 το κύκλωμα SMIX χωρίζεται σε δύο υποκυκλώματα. Το πρώτο είναι το κύκλωμα που υλοποιεί την αντικατάσταση των πρώτων 4 στηλών με τα byte του SBOX του Πίνακα 5.1 και το δεύτερο κύκλωμα υλοποιεί τον αριστερό πολλαπλασιασμό της εξόδου του πρώτου κυκλώματος με το Πίνακα 5.3 . Εφόσον υπάρχουν 128-bit προς αντικατάσταση θα πρέπει να υπάρξουν 16 παράλληλα SBOXes αν απαιτείται η αντικατάσταση να ολοκληρώνεται σε ένα παλμό ρολογιού από τη στιγμή που το κάθε SBOX αντικαθιστά ένα byte. Η υλοποίηση των SBOXes μπορεί να γίνει είτε με χρήση μνήμης είτε

με χρήση συνδυαστικού κυκλώματος που υλοποιεί τη διαδικασία του Σχήματος 5.5. Στο κύκλωμα Fugue 256 Clock Circles 49 τα SBOXes έχουν υλοποιηθεί με το δεύτερο τρόπο. Για τον υπολογισμό του αντιστρόφου ενός byte έχει χρησιμοποιηθεί το κύκλωμα της αναφοράς [1] το οποίο και δίνεται από τα Σχήματα 6.7 – 6.11 .

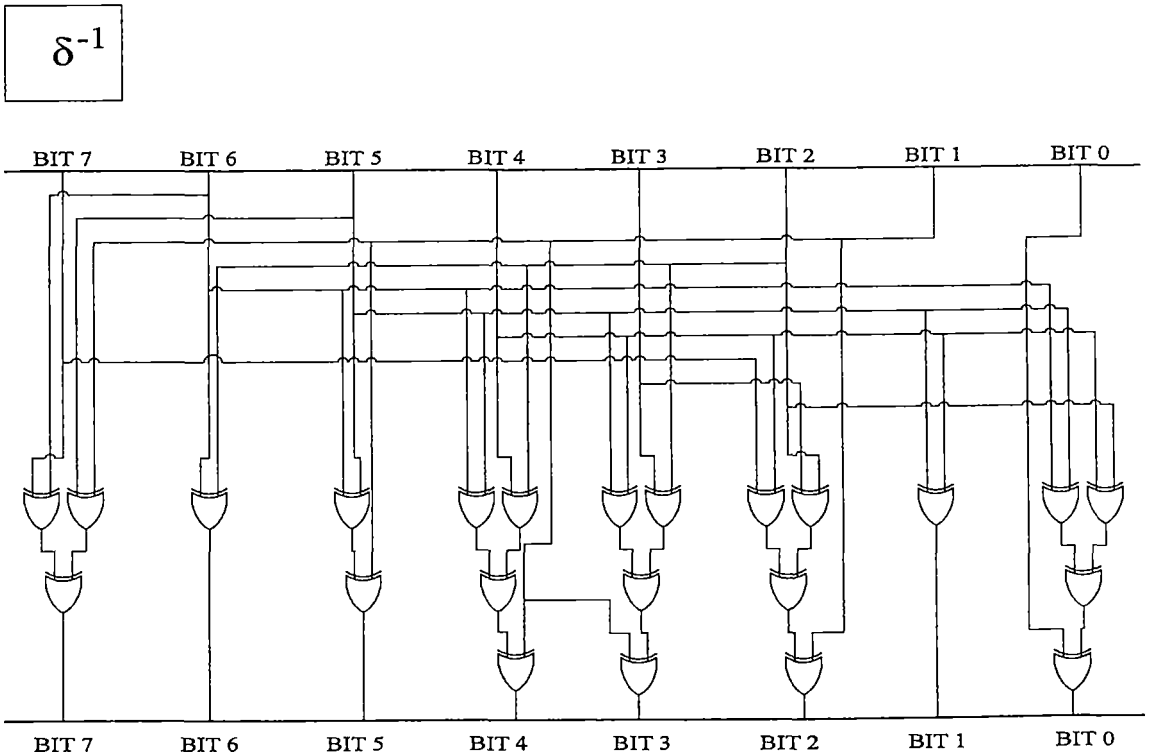


Σχήμα 6.7: *Το κύκλωμα υπολογισμού του αντίστροφου σε $GF(2^8)$.*

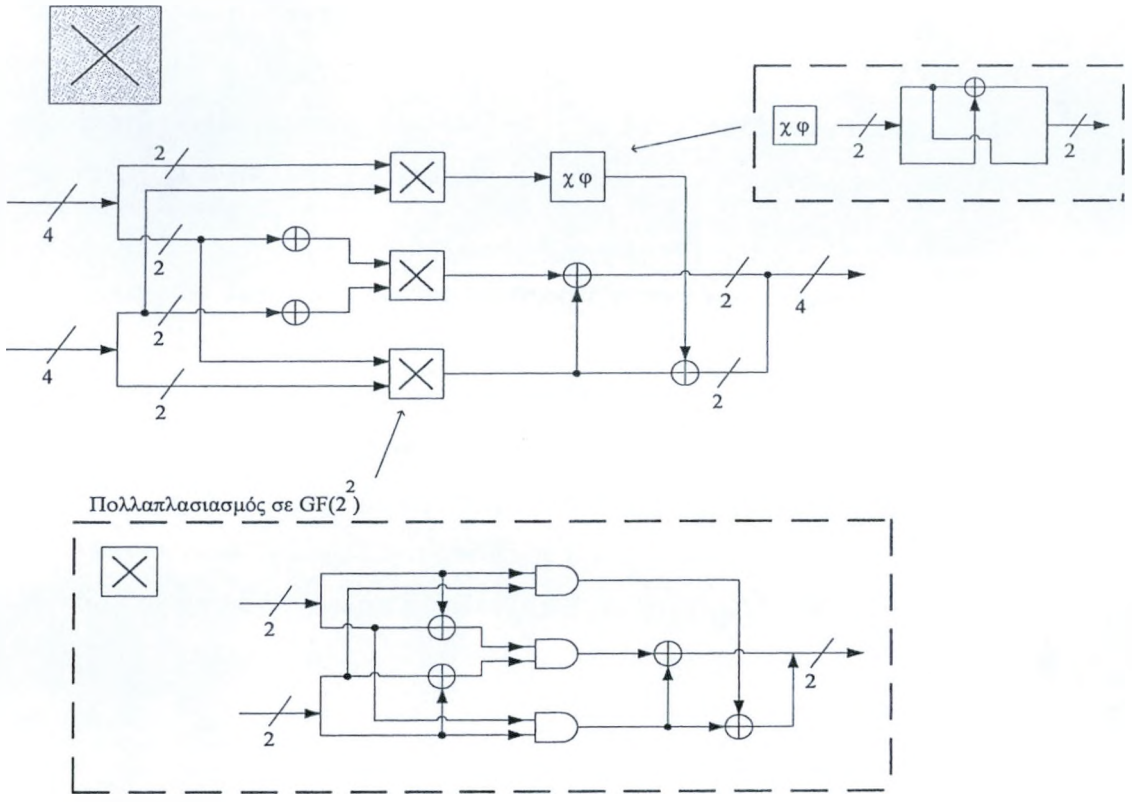
Στη συνέχεια ακολουθούν τα επιμέρους υποκυκλώματα του Σχήματος 6.7



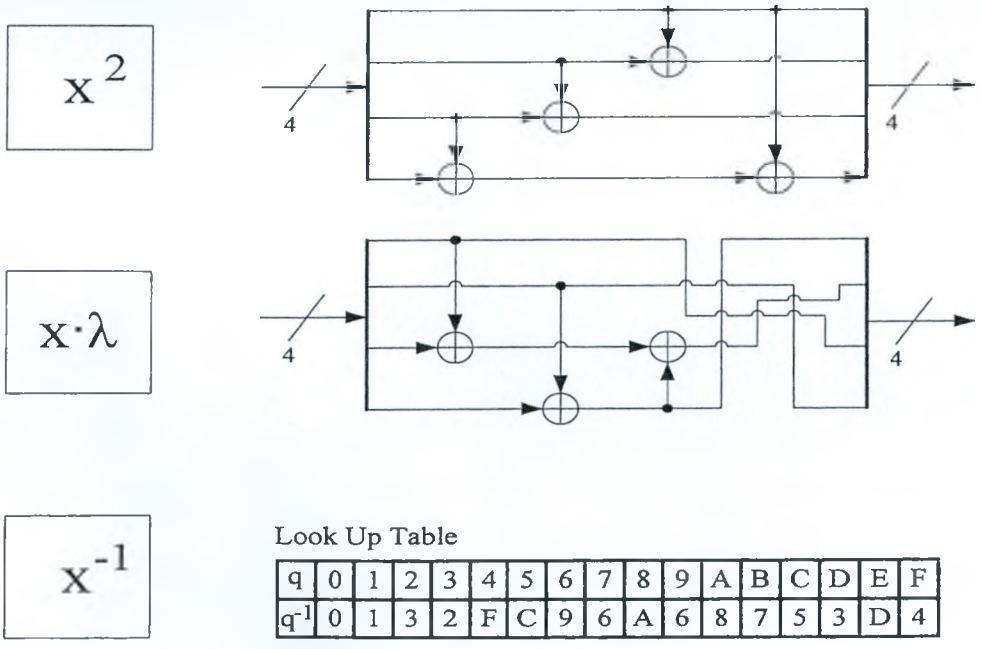
Σχήμα 6.8: *Κύκλωμα ισόμορφης αντικατάστασης σε σύνθετα πεδία.*



Σχήμα 6.9: *Κύκλωμα αντιστροφής ισόμορφης αντικατάστασης σε $GF(2^8)$.*



Σχήμα 6.10 *Κύκλωμα πολλαπλασιασμού σε $GF(2^8)$.*



Look Up Table

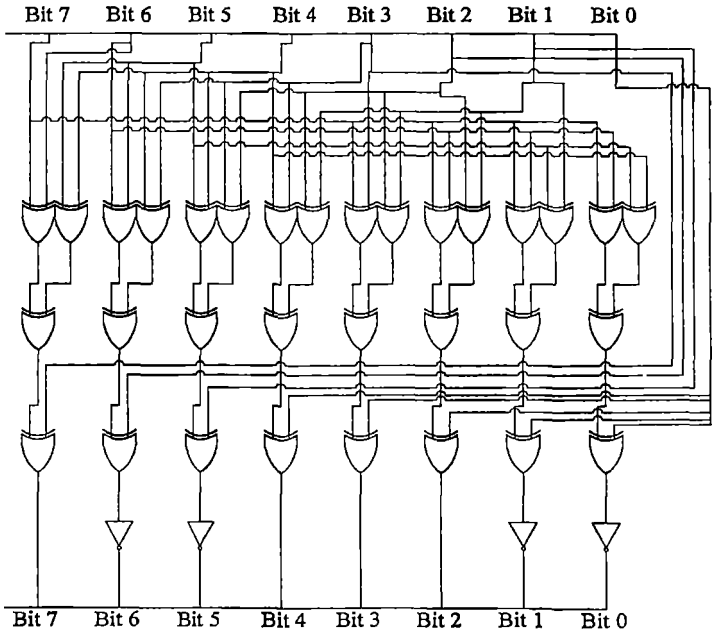
q	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
q^{-1}	0	1	3	2	F	C	9	6	A	6	8	7	5	3	D	4

Σχήμα 6.11: *Κυκλώματα τετραγωνισμού, πολλαπλασιασμού με τη σταθερά λ και αντιστροφής σε $GF(2^4)$.*

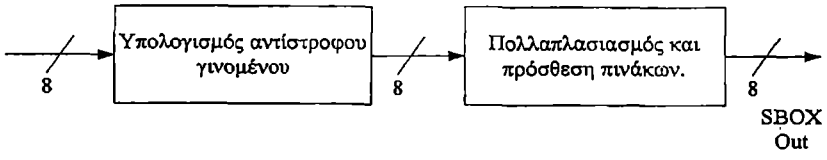
Στο σχήμα 6.11 η αντιστροφή σε GF(2^4) δίνεται από το πίνακα στο κάτω μέρος του σχήματος. Λόγω του μικρού πλήθους αντίστροφων τιμών σε GF(2^4) στα SBOXes του κυκλώματος Fugue 256 Clock Circles 49 δεν χρησιμοποιείται κάποιο συνδυαστικό κύκλωμα για τον υπολογισμό τους αλλά αποθηκεύονται σαν πίνακας χρησιμοποιώντας τη κατανεμημένη μνήμη.

Υπολογίζοντας τον αντίστροφο γινομένου ενός byte μπορεί να υλοποιηθεί ο πολλαπλασιασμός και η πρόσθεση πινάκων του Σχήματος 5.5 . Οι δύο πίνακες του Σχήματος 5.5 αποτελούνται μόνο από τους αριθμούς 0 και 1 κάτι το οποίο συνεπάγεται ότι ο υπολογισμός τους γίνεται μόνο μέσω πράξεων αποκλειστικού-Η. Στο Σχήμα 6.12 (α) φαίνεται το κύκλωμα που υπολογίζει τις παραπάνω πράξεις. Η είσοδος του είναι η έξοδος του κυκλώματος του Σχήματος 6.7 και ή έξοδος του Σχήματος 6.12 αποτελεί την έξοδο του κυκλώματος SBOX. Στο Σχήμα 6.12 (β) φαίνεται η συνολική εικόνα ενός SBOX.

α) Πολλαπλασιασμός και πρόσθεση πινάκων του Σχήματος 5.5.

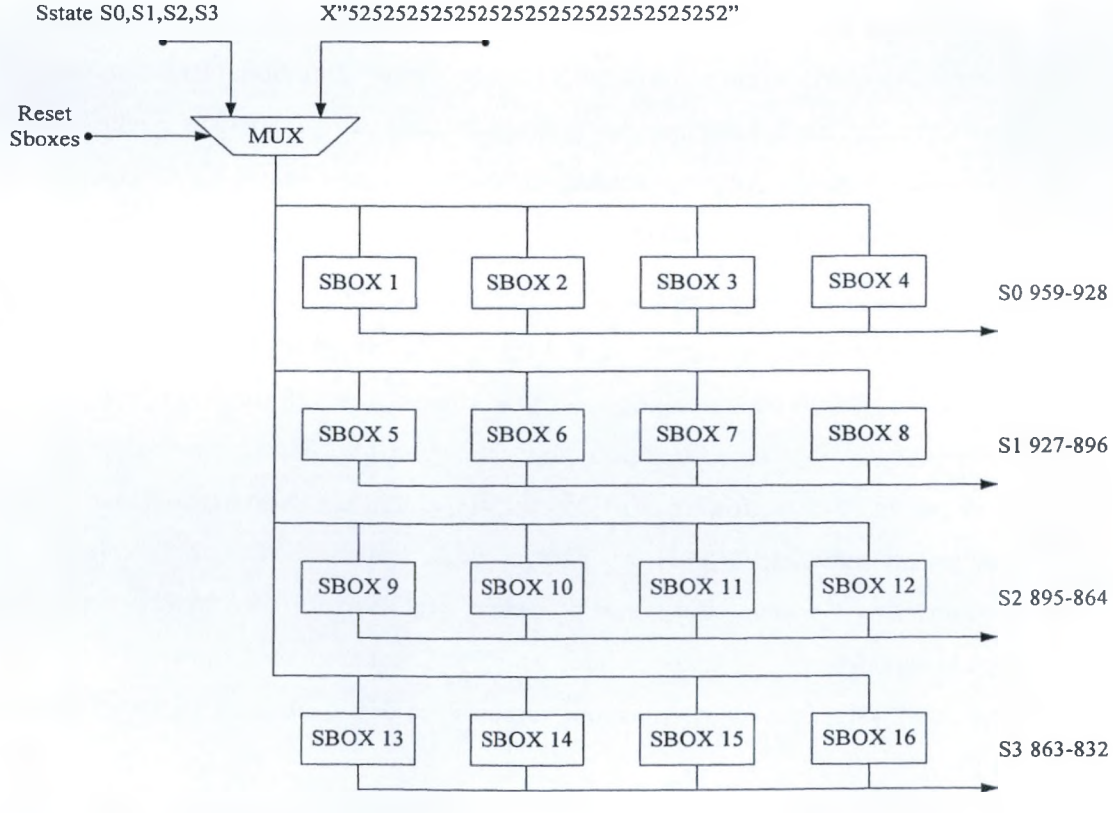


β) Ένα SBOX



Σχήμα 6.12: *α) Η υλοποίηση του Σχήματος 5.5 β) Η συνολική εικόνα ενός SBOX.*

Στο Σχήμα 6.13 φαίνεται το κύκλωμα που είναι υπεύθυνο για τη διαχείριση των 16 SBOXes. Η είσοδος και η έξοδος δεδομένων είναι μήκος 128-bit. Επιπλέον εισέρχεται ένα ακόμα σήμα που ονομάζεται Reset SBOXes ενός bit. Ο λόγος ύπαρξης αυτού του σήματος είναι για να λύσει το πρόβλημα ότι εκ φύσεως το SBOX για είσοδο δεδομένων που βρίσκονται στο λογικό 0 έχει έξοδο διαφορετική από λογικό 0. Κάτι το οποίο προκαλεί πρόβλημα κατά το στάδιο της αρχικοποίηση ή της εξαναγκασμένης εξωτερικής επαναφοράς (Reset) όπου όλα τα δεδομένα στο κύκλωμα πρέπει να βρίσκονται στο λογικό 0. Στο Σχήμα 6.13 το σήμα Reset SBOXes ελέγχει το πολυπλέκτη και όταν είναι επιθυμητή η επαναφορά ή αρχικοποίηση του συστήματος ο πολυπλέκτης δίνει ως είσοδο στα 16 SBOXes τη δεκαεξαδική τιμή ''52525252525252525252525252525252''. Το κάθε SBOX παίρνει τη τιμή ''52'' ως είσοδο κάτι το οποίο αναγκάζει το SBOX να δώσει για έξοδο δεδομένα που βρίσκονται στο λογικό 0.



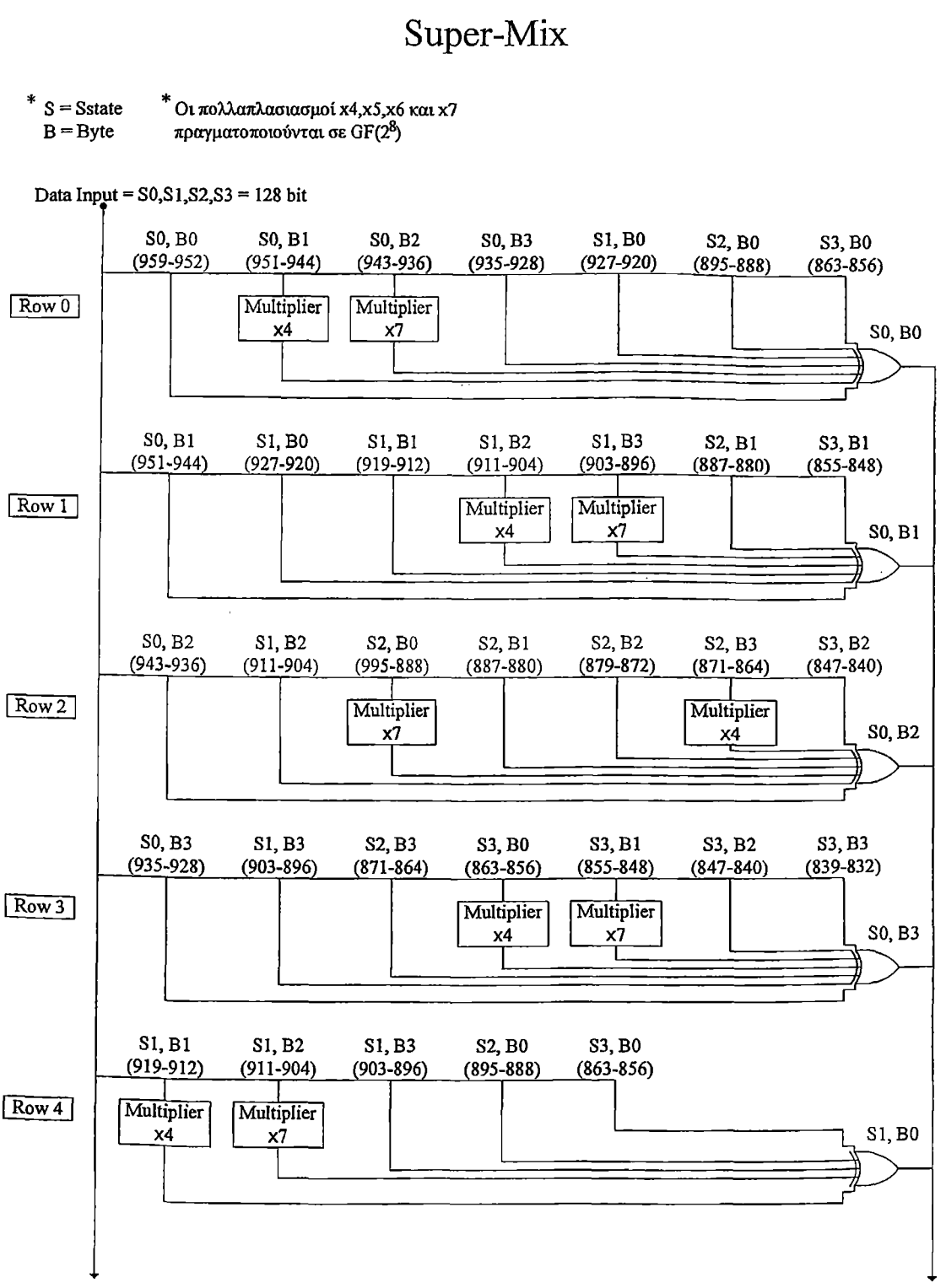
Σχήμα 6.13: *Κύκλωμα διασύνδεσης των 16 SBOXes.*

SMIX Super-Mix

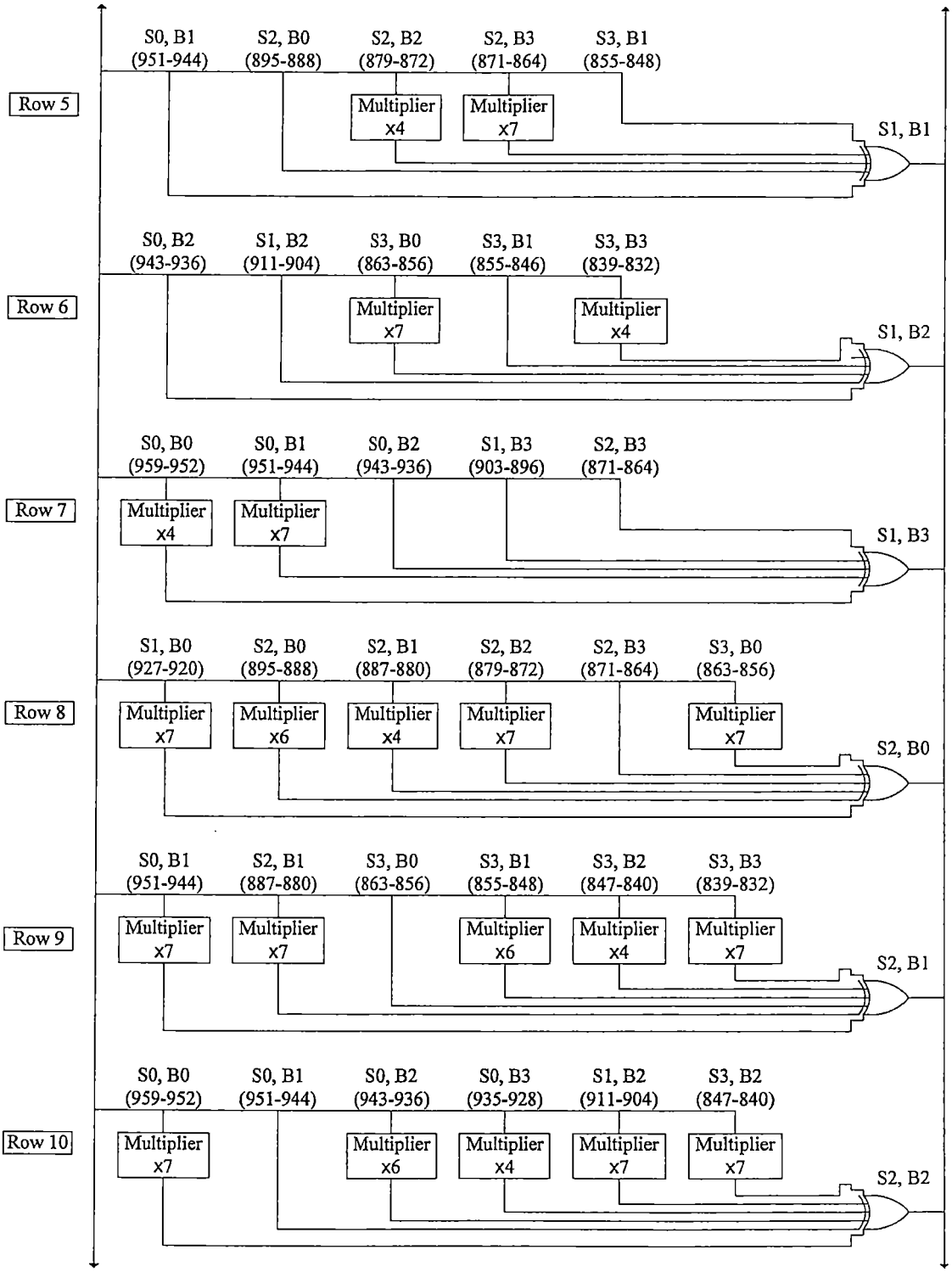
Το δεύτερο υποκύκλωμα του SMIX ονομάζεται Super-Mix και όπως έχει αναφερθεί και στη παράγραφο SMIX του κεφαλαίου 5.2 υπάρχουν δύο τρόποι υλοποίησης του. Στο κύκλωμα Fugue 256 Clock Circles 49 έχει χρησιμοποιηθεί ο δεύτερος τρόπος στον οποίο η έξοδος των SBOXes θεωρείται ως ένας πίνακας 16 γραμμών και 1 στήλης (16x1) και πραγματοποιείται ο αριστερός πολλαπλασιασμός με το πίνακα N 16x16 στοιχείων (Πίνακα 5.3). Κάθε γραμμή του πίνακα 16x1 θεωρείται ως ένα byte της κάθε στήλης του τμήματος Sstate. Στα Σχήματα 6.14, 6.15 και 6.16 δίνεται το κύκλωμα που υλοποιεί το παραπάνω πολλαπλασιασμό. Με S και B συμβολίζονται οι στήλες και τα bytes τους ενώ μέσα στις παρενθέσεις δίνονται οι θέσεις των bit σε σχέση με το τμήμα Sstate.

Βάση του πίνακα N οι πολλαπλασιασμοί που χρειάζονται να υλοποιηθούν είναι οι x4, x5, x6 και x7 σε αριθμητικό πεδίο $GF(2^8)$. Για το λόγο αυτό σχεδιάστηκαν 4 κυκλώματα που υλοποιούν τους παραπάνω πολλαπλασιασμούς τα οποία και δίνονται από τα Σχήματα 6.17 και 6.18. Ο σχεδιασμός τους στηρίχτηκε στη θεωρία των πεπερασμένων πεδίων που περιγράφτηκε στο κεφάλαιο 5.1 . Από τη θεωρία προκύπτει ότι όταν ένας αριθμός ο οποίος είναι μικρότερος από το 128 και πολλαπλασιάζεται με τον αριθμό 2 τότε το αποτέλεσμα ισούται με 2 επί τον αριθμό (2•αριθμός) ενώ στη περίπτωση που είναι μεγαλύτερος από το 128 τότε το αποτέλεσμα ισούται με 2 επί τον αριθμό (2•αριθμός) $\oplus$ 0x1b. Όπου το ένα 0x1b προκύπτει από το πολυώνυμο που χρησιμοποιείται στον αλγόριθμο Fugue για τη πράξη του πολλαπλασιασμού. Από τη παραπάνω ιδιότητα μπορούν να προκύψουν οι πράξεις του πολλαπλασιασμού x4, x5, x6 και x7.

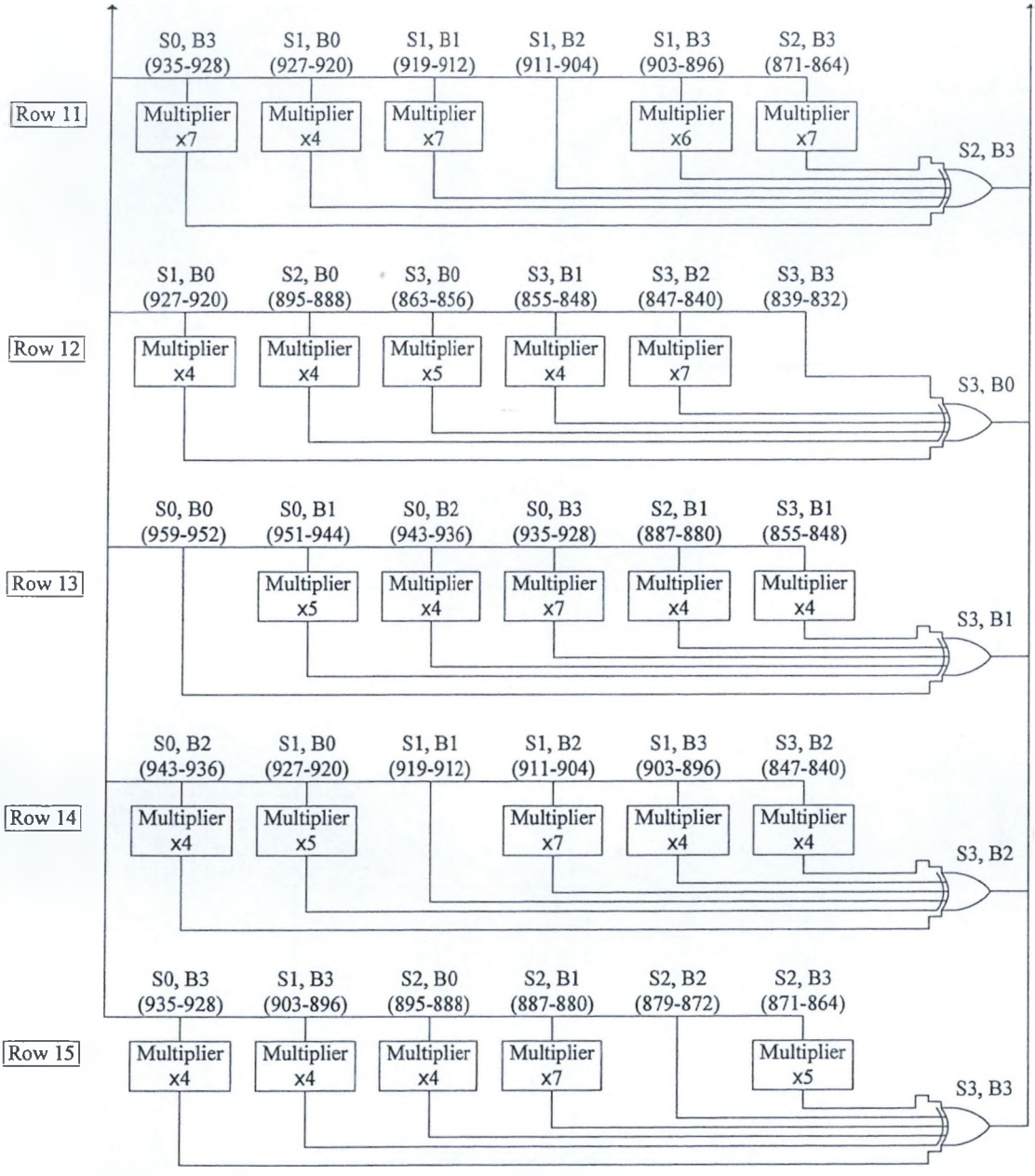
Στη περίπτωση του πολλαπλασιασμού x4 απλά χρησιμοποιείται δύο φορές αυτή η ιδιότητα. Ο πολλαπλασιασμός με τον αριθμό 2 επιτυγχάνεται με τη δεξιά ολίσθηση. Όσων αφορά τον έλεγχο αν ο αριθμός είναι μεγαλύτερος του 128 αυτό που συμβαίνει είναι ότι το πιο σημαντικό ψηφίο του αριθμού οδηγείται στους ακροδέκτες των πυλών αποκλειστικού-Η ούτως ώστε αν είναι στο λογικό 1 θα συμβεί η πράξη αποκλειστικού-Η ενώ αν όχι το επίπεδο των πυλών αποκλειστικού-Η δεν θα έχει καμία επίδραση στον αριθμό. Χρησιμοποιείται το σημαντικότερο ψηφίο καθώς εάν βρίσκεται στο λογικό 1 συνεπάγεται ότι ο αριθμός είναι μεγαλύτερος του 128. Επίσης όπως προκύπτει και από το Σχήμα 6.17 το σημαντικότερο ψηφίο οδηγείται σε συγκεκριμένους ακροδέκτες ούτος ώστε να υλοποιείται το πολυώνυμο 0x1b (00011011).



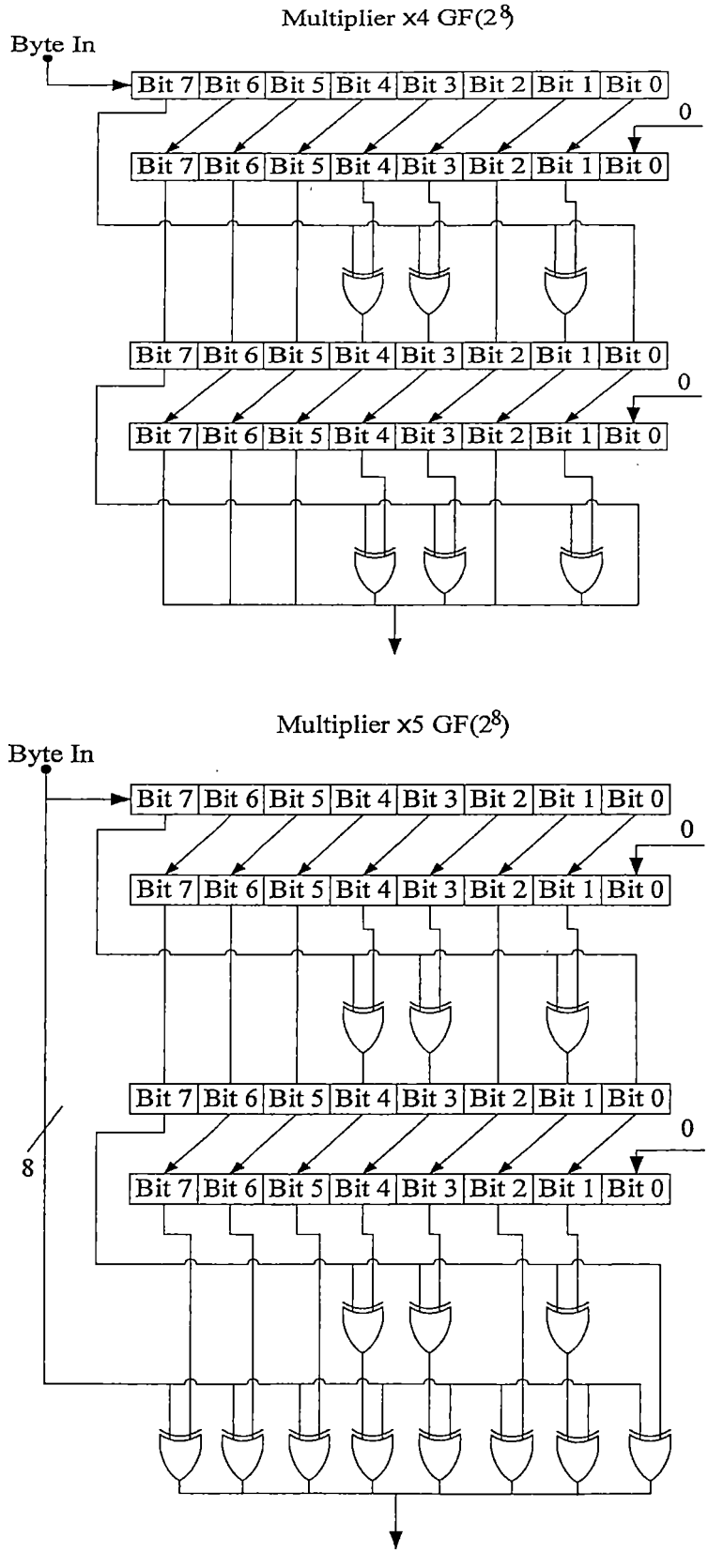
Σχήμα 6.14: *Οι πρώτες 5 γραμμές του πολλαπλασιασμού πινάκων.*



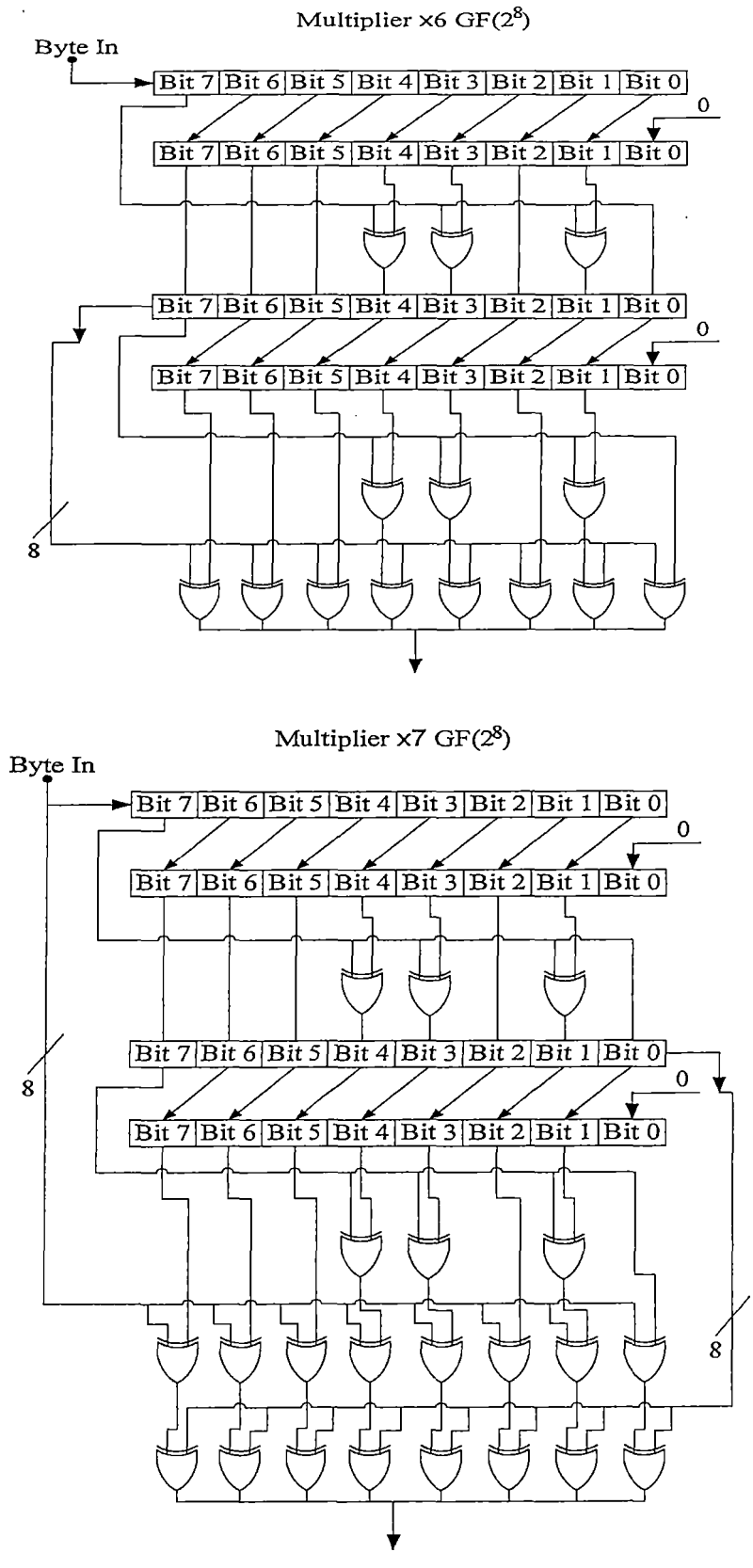
Σχήμα 6.15: *Οι γραμμές 5 έως 10 του πολλαπλασιασμού πινάκων.*



Σχήμα 6.16: *Οι γραμμές 11 έως 15 του πολλαπλασιασμού πινάκων.*



Σχήμα 6.17: *Κυκλώματα πολλαπλασιασμού x4 και x5.*

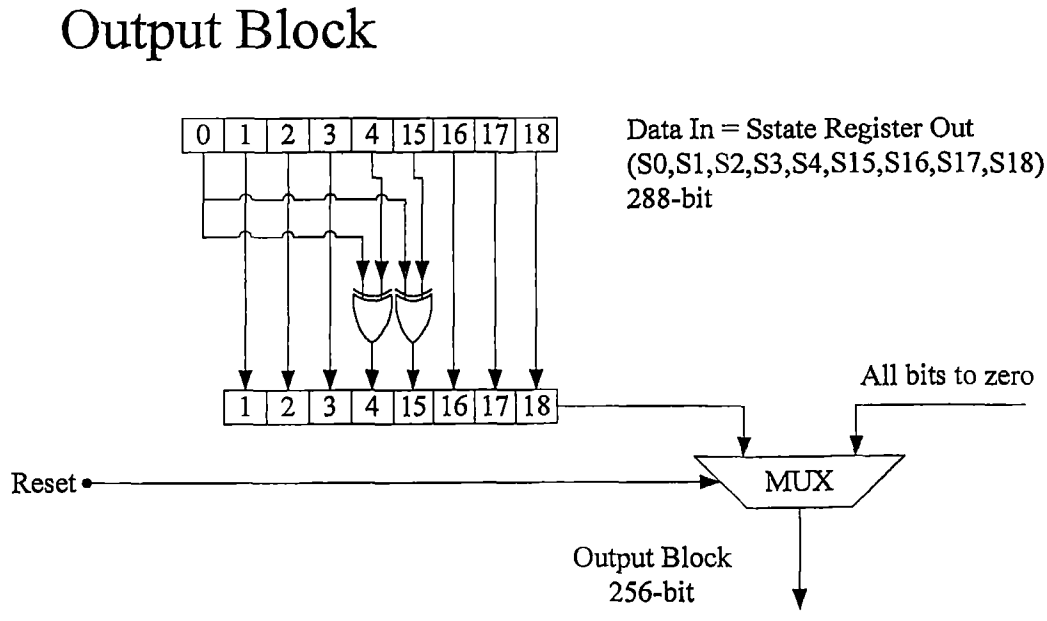


Σχήμα 6.18: *Κυκλώματα πολλαπλασιασμού x6 και x7.*

Για την υλοποίηση του κυκλώματος x5 χρησιμοποιείται το κύκλωμα x4 συν ότι υπάρχει ένα ακόμα επίπεδο πυλών αποκλειστικού-Η για να προστεθεί στο αποτέλεσμα μια ακόμα φορά ο αριθμός που εισέρχεται. Στο κύκλωμα x6 συμβαίνει το ίδιο πράγμα με τη διαφορά ότι δε προστίθεται ξανά ο αριθμός που εισέρχεται αλλά το αποτέλεσμα του πρώτου κυκλώματος πολλαπλασιασμού x2. Τέλος στο κύκλωμα x7 υπάρχει ένα ακόμα επίπεδο πυλών αποκλειστικού-Η και προστίθενται και ο εισερχόμενος αριθμός αλλά και το αποτέλεσμα του πρώτου κυκλώματος αποκλειστικού-Η.

Output Block

Το τελευταίο υποκύκλωμα πριν την έξοδο του κυκλώματος Fugue 256 ονομάζεται Output Block. Ουσιαστικά πρόκειται για το κύκλωμα το οποίο διαλέγει ποιες από τις στήλες του τμήματος δεδομένων Sstate θα καταλήξουν στην έξοδο. Στο Σχήμα 6.19 δίνεται το κύκλωμα Output Block. Η είσοδος του είναι μήκους 288-bit και ουσιαστικά πρόκειται για τις στήλες S0, S1, S2, S3, S4, S15, S16, S17, και S18. Μεταξύ των στηλών S0, S4 και S15 συμβαίνουν δύο πράξεις αποκλειστικού-Η και στη συνέχεια καταλήγουν στην έξοδο του κυκλώματος πλην της στήλης S0. Ο πολυπλέκτης του σχήματος χρησιμεύει στην ασύγχρονη αρχικοποίηση του κυκλώματος.



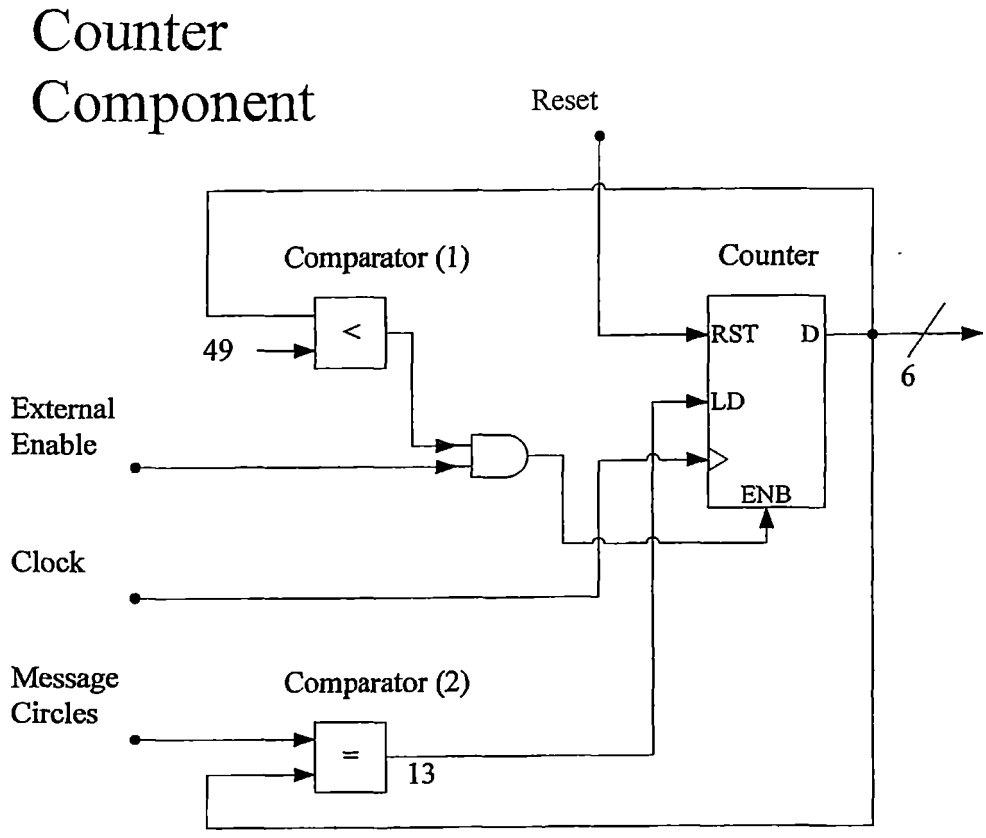
Σχήμα 6.19: *Το κύκλωμα Output Block.*

Controller

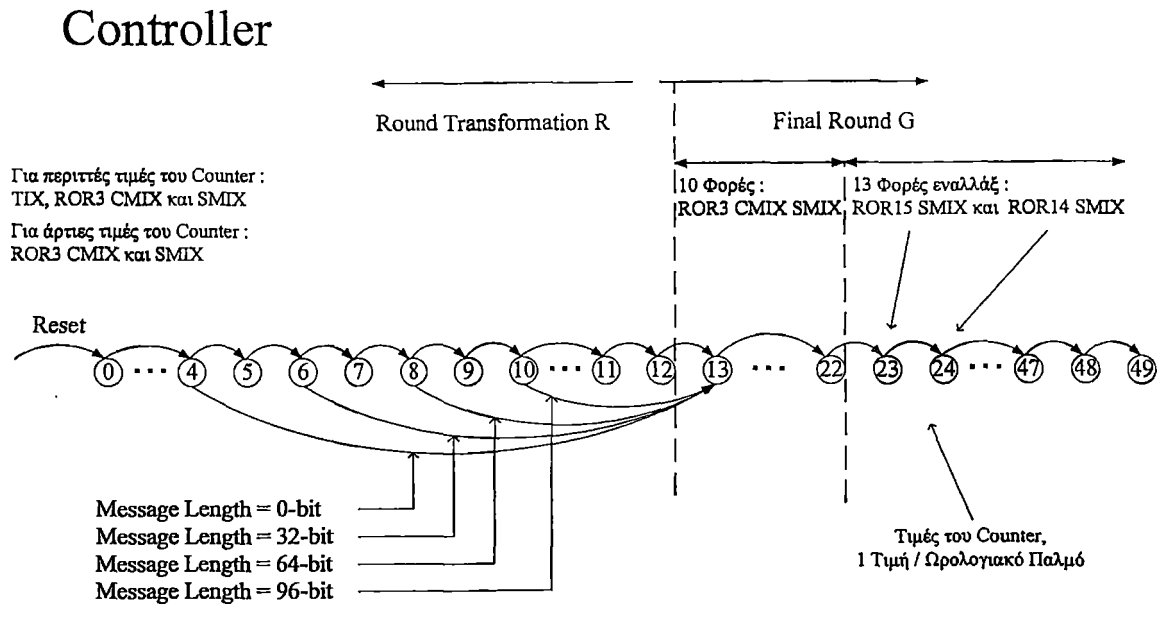
Όπως αναφέρθηκε και στις παραπάνω παραγράφους το κύκλωμα που είναι υπεύθυνο για την ορθή λειτουργία του κυκλώματος, δηλαδή τον συγχρονισμό είναι ο Controller. Αυτό το κύκλωμα μπορεί να θεωρηθεί ως μια μηχανή πεπερασμένων καταστάσεων (Finite State Machine - FSM) η οποία αποτελείται από 49 καταστάσεις στη περίπτωση του κυκλώματος Fugue 256 Clock Circles 49. Ο έλεγχος των επιμέρους κυκλωμάτων γίνεται μέσω των 6 σημάτων ελέγχου του Controller τα οποία μεταβάλλονται κατάλληλα ανάλογα τη λειτουργία που πρέπει να πραγματοποιηθεί σε κάθε κατάσταση.

Για το καθορισμό των 49 καταστάσεων χρησιμοποιείται ένας μετρητής (Counter) ο οποίος και δίνεται από το Σχήμα 6.20 . Το σήμα Message Circles διαβιβάζεται από το κύκλωμα του Padder στο κύκλωμα του Controller και στη συνέχεια στο κύκλωμα του Counter ούτος ώστε αν ισχύει η ισότητα του δεύτερου συγκριτή (Comparator (2)) στον Counter να φορτώνεται η τιμή 13. Η σύγκριση γίνεται μεταξύ της εκάστοτε τιμής του Counter και της τιμής Message Circles έτσι ώστε το σύστημα να καταλαβαίνει αν συμπληρώθηκαν οι επιθυμητοί κύκλοι επεξεργασίας για το συγκεκριμένο μέγεθος μηνύματος. Η τιμή 13 φορτώνεται στον Counter για να προσπεραστούν οι κύκλοι επεξεργασίας που δε χρειάζονται αν το μήνυμα είναι μικρότερο από 96-bit. Έτσι ο Counter συνεχίζει να αυξάνει από τη τιμή 13 και μετά όπου οι καταστάσεις του κυκλώματος είναι κοινές για όλα τα μεγέθη εισερχόμενων μηνυμάτων. Αυτό συμβαίνει γιατί στο κύκλωμα Fugue 256 Clock Circles 49 οι καταστάσεις 0 έως 12 υλοποιούν τη διαδικασία Round Transformation R ενώ οι καταστάσεις 13 έως 49 υλοποιούν τη διαδικασία Final Round G.

Στο Σχήμα 6.21 δίνεται σχηματικά η περιγραφή του Controller. Οι τιμές των καταστάσεων προκύπτουν από τον μετρητή ο οποίος αυξάνει σε κάθε παλμό του ρολογιού.



Σχήμα 6.20: *Το κύκλωμα μετρητή.*



Σχήμα 6.21: *Σχηματική περιγραφή της λειτουργίας του κυκλώματος ελέγχου.*

6.2 Οι παραλλαγές υλοποίησης της βασικής αρχιτεκτονικής της Fugue

Όπως αναφέρθηκε και στο κεφάλαιο 6.1 σχεδιάστηκαν 13 παραλλαγές του βασικού κυκλώματος Fugue που σκοπό έχουν τη μείωση κύκλων επεξεργασίας ή τη μείωση της δυναμικής κατανάλωσης. Αυτά τα κυκλώματα θα αναλυθούν παρακάτω παρουσιάζοντας τις διαφορές από τη βασική αρχιτεκτονική που παρουσιάστηκε στο προηγούμενο κεφάλαιο.

Fugue 256-bit Clock Circles 36 και Fugue 256-bit Clock Circles 25

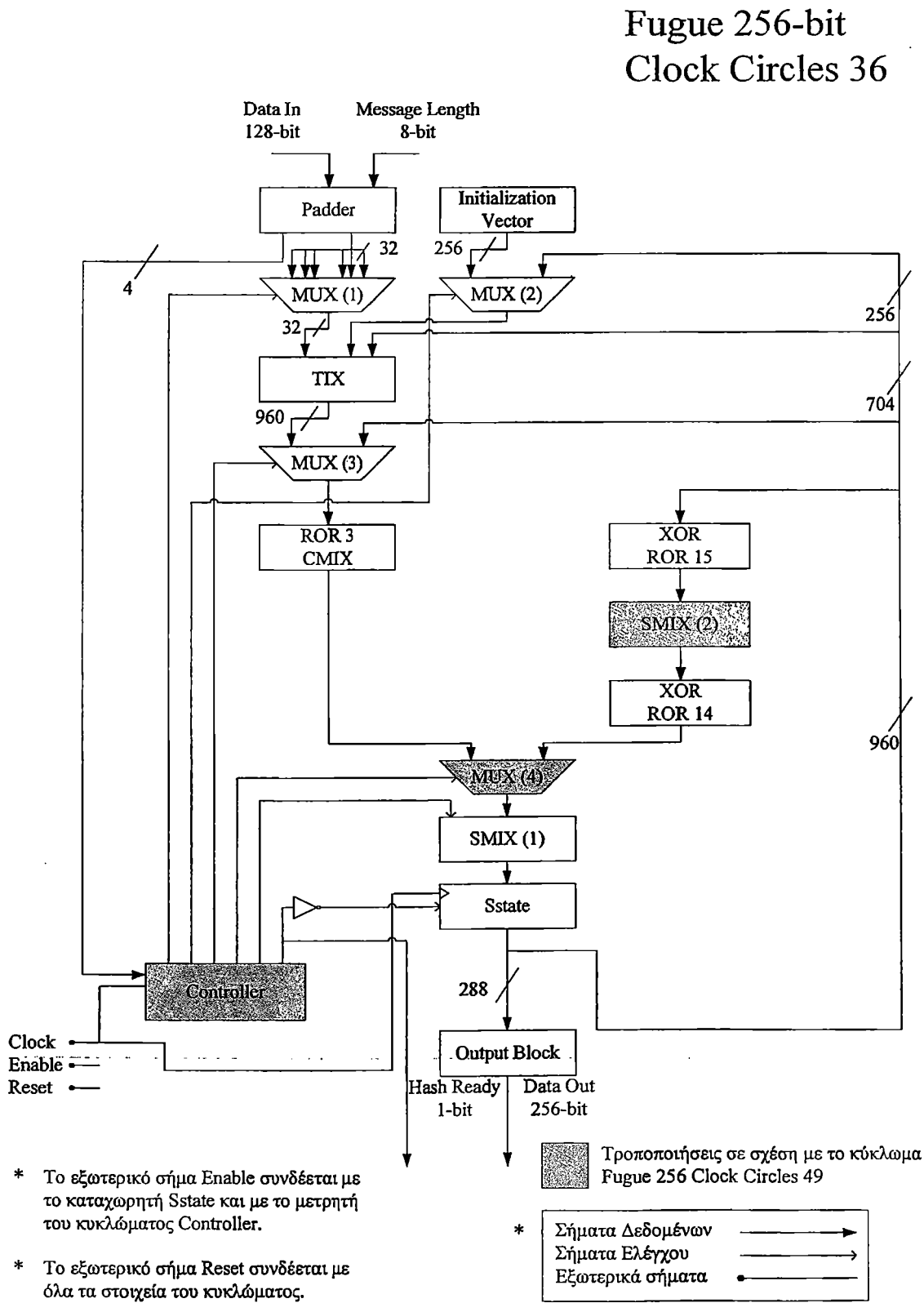
Οι δύο αυτές αρχιτεκτονικές έχουν στόχο τη μείωση των κύκλων επεξεργασίας από 49 που είναι στο αρχικό κύκλωμα σε 36 και 25 αντίστοιχα. Για την επίτευξη αυτού του στόχου χρησιμοποιήθηκαν κάποια από τα επιμέρους κυκλώματα του Σχήματος 6.1 περισσότερες από μια φορές. Η δομή αυτών των κυκλωμάτων δεν έχει αλλάξει σε σχέση με το κύκλωμα Fugue 256-bit Clock Circles 49. Επίσης οι είσοδοι και οι έξοδοι του κυκλώματος παραμένουν οι ίδιες.

Στα Σχήματα 6.22 και 6.23 δίνονται οι αρχιτεκτονικές των δύο κυκλωμάτων. Τα στοιχεία με σκούρο γκρι συμβολίζουν τις διαφορές σε σχέση με το κύκλωμα Fugue 256-bit Clock Circles 49.

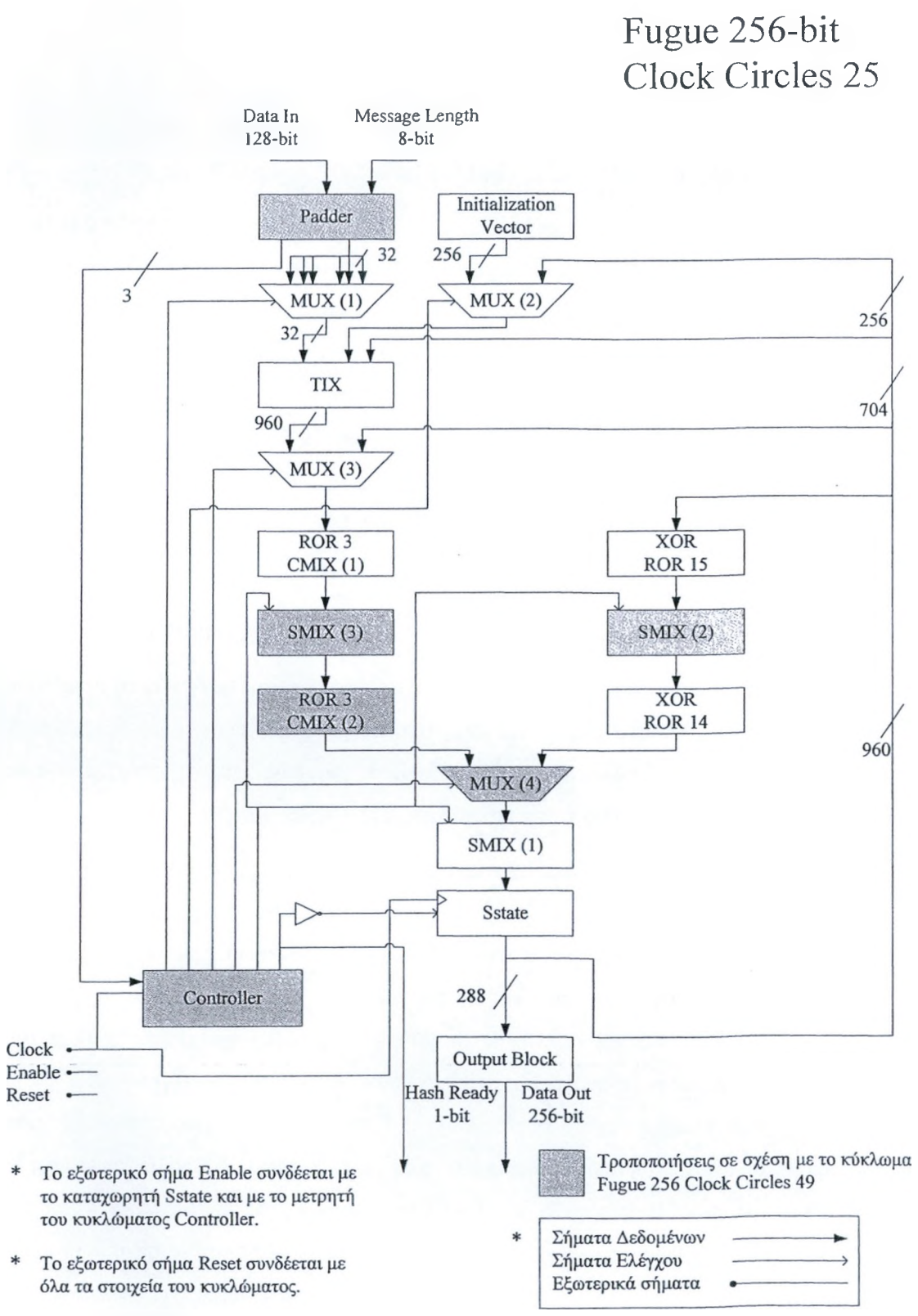
Στο κύκλωμα Fugue 256-bit Clock Circles 36 η βασική διαφορά είναι ότι το κύκλωμα SMIX χρησιμοποιείται δύο φορές ούτως ώστε να είναι εφικτό η μείωση των κύκλων επεξεργασίας του επιπέδου Final Round G από 26 σε 13. Με τη χρήση του επιπλέον κυκλώματος SMIX(2) είναι εφικτό το κύκλωμα XOR ROR 14 σε λογική συνέχεια του κυκλώματος XOR ROR 15 κάτι το οποίο συνεπάγεται την ολοκλήρωση ενός γύρου Final Round G μόνο σε ένα κύκλο ρολογιού.

Οι υπόλοιπες διαφορές είναι ότι ο πολυπλέκτης του 3 σε 1 (MUX (4)) του Σχήματος 6.1 αντικαθίσταται από ένα πολυπλέκτη 2 σε 1. Επίσης του κύκλωμα του Controller πρέπει να τροποποιηθεί σύμφωνα με τους μειωμένους κύκλους επεξεργασίας που προκύπτουν για να μπορεί να ελέγξει τη λειτουργία του κυκλώματος. Η διαφορά του Controller σε σχέση με το Σχήμα 6.21 είναι ότι η δεύτερη φάση του Final Round G ξεκινάει πάλι από το κύκλο 23 αλλά τελειώνει στο κύκλο 36.

Όσων αφορά το κύκλωμα Fugue 256-bit Clock Circles 25 χρησιμοποιούνται δύο επιπλέον κυκλώματα SMIX και ένα ακόμα κύκλωμα ROR 3 CMIX. Η μείωση επιτυγχάνεται στο κομμάτι του Final Round G όπου οι κύκλοι μειώνονται κατά 5 ακόμα σε σχέση με το



Σχήμα 6.22: *Αρχιτεκτονική του κυκλώματος Fugue 256 Clock Circles 36.*



Σχήμα 6.23: *Αρχιτεκτονική του κυκλώματος Fugue 256 Clock Circles 25.*

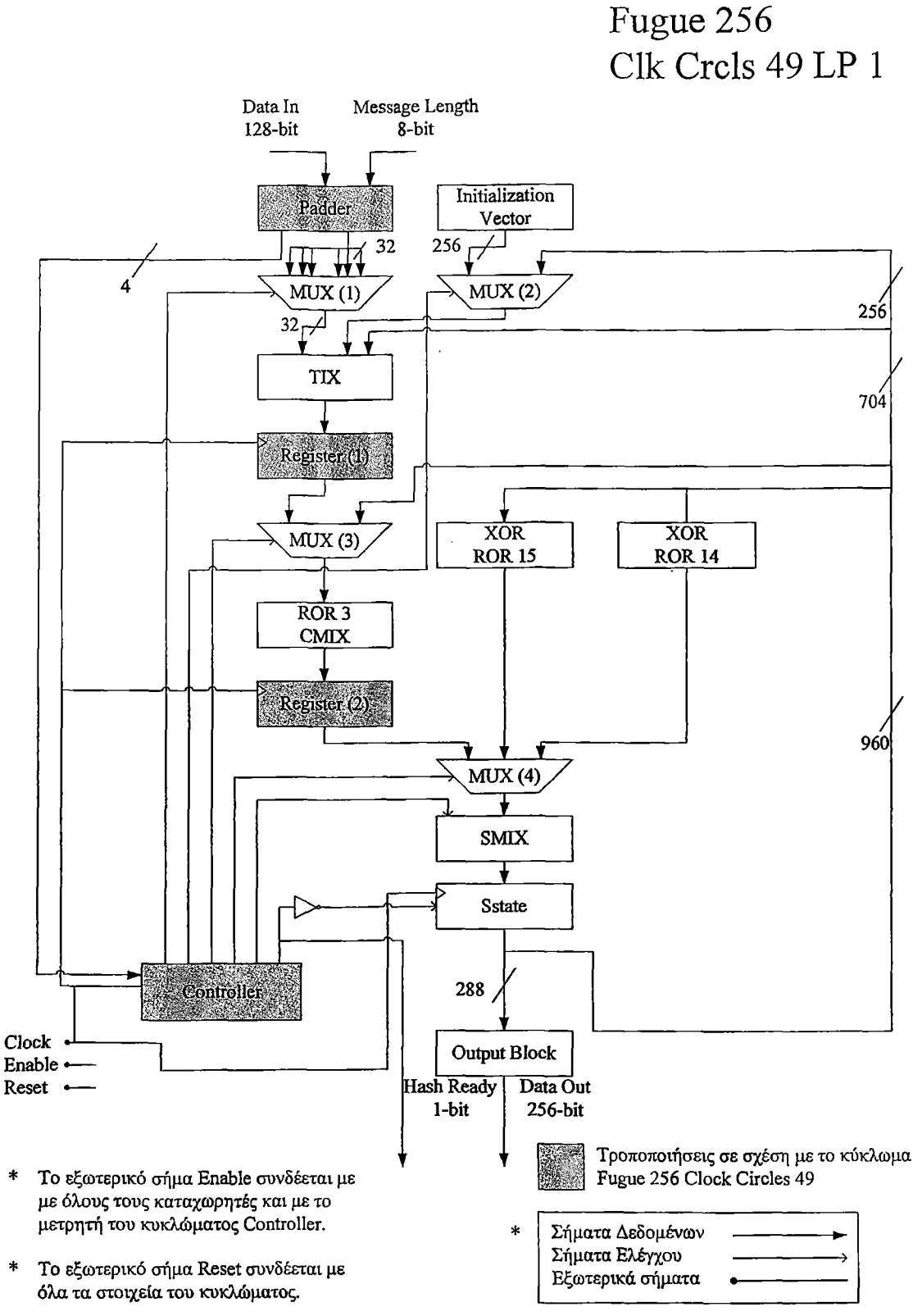
κύκλωμα 256-bit Clock Circles 36 καθώς επίσης και στο κομμάτι του Round Transformation R στο οποίο οι κύκλοι επεξεργασίας μειώνονται από 22 σε 11. Η προσθήκη των κυκλωμάτων SMIX(3) και ROR 3 CMIX(2) δίνει τη δυνατότητα της ολοκλήρωσης ενός γύρου Round Transformation R μόνο σε ένα κύκλο ρολογιού. Επίσης για αυτό το λόγο μειώνεται κατά 5 κύκλους και η πρώτη φάση του Final Round G.

Οι επιμέρους τροποποιήσεις του κυκλώματος Fugue 256-bit Clock Circles 36 σε σχέση με το βασικό κύκλωμα ισχύουν και στο κύκλωμα Fugue 256-bit Clock Circles 25. Εκτός του ότι υπάρχει μια επιπλέον τροποποίηση στο κύκλωμα του Padder. Καθώς το κύκλωμα του Controller όπως έχει αναφερθεί χρησιμοποιεί ένα σήμα (message circles) 4-bit για τη παραμετρική συμπεριφορά του μετρητή του κυκλώματος όσων αφορά τους γύρους του επιπέδου Round Transformation R το οποίο και δέχεται από το κύκλωμα του Padder. Η διαφορά του κυκλώματος Padder σε σχέση με το Σχήμα 6.2 είναι ότι οι προκαθορισμένες τιμές του πολυπλέκτη MUX(2) είναι 2, 3, 4, 5 και 6.

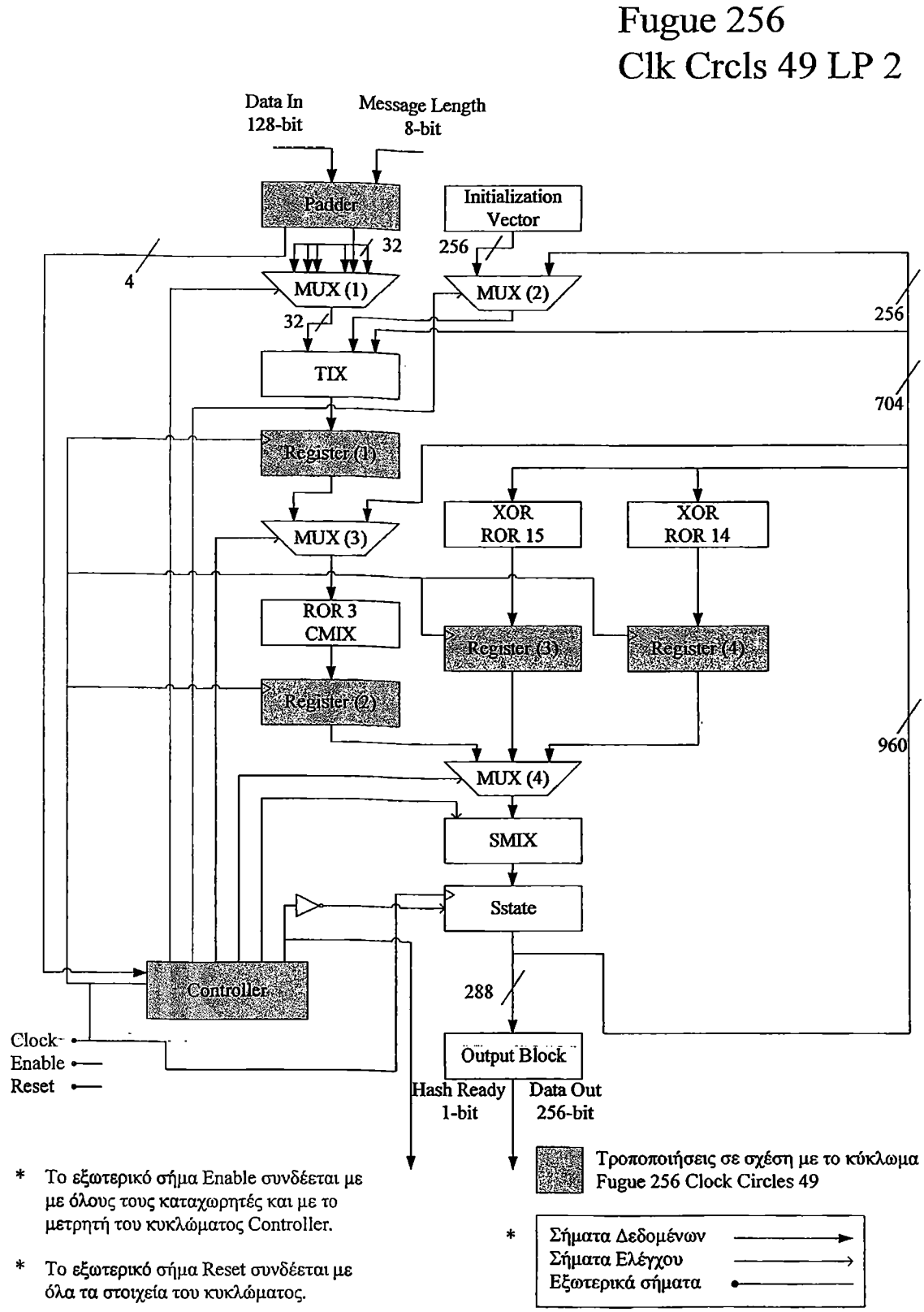
Fugue 256-bit Clk Crcls 49 LP 1 και Fugue 256-bit Clκ Crcls 49 LP 2

Οι παρακάτω παραλλαγές του βασικού κυκλώματος αφορούν τη προσπάθεια μείωσης της δυναμικής κατανάλωσης. Με τα αρχικά LP (Low Power) και τον αριθμό που έπεται στο όνομα τους συμβολίζονται οι αρχιτεκτονικές χαμηλής κατανάλωσης που εφαρμόστηκαν πάνω στο βασικό κύκλωμα Fugue 256-bit Clock Circles 49.

Η τεχνική χαμηλής κατανάλωσης που χρησιμοποιήθηκε σε αυτές τις δύο αρχιτεκτονικές είναι η pipeling η οποία αναφέρθηκε στη παράγραφο Pipeling στο κεφάλαιο 4.3 . Η υλοποίηση της τεχνικής στηρίχτηκε σε δυο παράγοντες. Ο πρώτος είναι να μοιραστεί ισόποσα το χρονικό μονοπάτι της μεγίστης διαδρομής δεδομένων και ο δεύτερος η εύρεση κυκλωμάτων με υψηλή παραγωγή ανεπιθύμητων μεταβολών (glitches). Στην αρχιτεκτονική LP 1 προστέθηκαν δύο καταχωρητές στο επίπεδο Round Transformation R όπου ο καθένας έπεται των κυκλωμάτων TIX και ROR 3 CMIX. Στη συνέχεια στην αρχιτεκτονική LP 2 προστέθηκαν άλλοι δύο καταχωρητές στο επίπεδο Final Round G οι οποίοι έπονται των κυκλωμάτων ROR 15 και ROR 14. Ο λόγος ύπαρξης και της αρχιτεκτονικής LP 2 είναι να μπορεί να παρθεί η απόφαση για τη βέλτιστη χρήση της τεχνικής pipeling στο κύκλωμα Fugue 256 Clock Circles 49 μέσω της σύγκρισης των δεδομένων κατανάλωσης των δύο αρχιτεκτονικών. Στα Σχήματα 6.24 και 6.25 δίνονται αυτές οι δύο υλοποιήσεις.



Σχήμα 6.24: *Αρχιτεκτονική του κυκλώματος Fugue 256 Clk Crcls 49 LP 1.*



Σχήμα 6.25: *Αρχιτεκτονική του κυκλώματος Fugue 256 Clk Crcls 49 LP 2.*

Το μειονέκτημα της χρήσης επιπλέον καταχωρητών είναι η αύξηση των κύκλων επεξεργασίας σε σχέση με το βασικό κύκλωμα. Στη περίπτωση LP 1 υπάρχει αύξηση κατά 28 κύκλους επεξεργασίας ενώ στη περίπτωση LP 2 η αύξηση είναι κατά 54 κύκλους. Συνολικά οι κύκλοι επεξεργασίας που απαιτούνται στις αρχιτεκτονικές LP 1 και LP 2 είναι 77 και 103 κύκλοι αντίστοιχα. Εκτός της πρόσθεσης των καταχωρητών οι υπόλοιπες διαφορές σε σχέση με το κύκλωμα του Σχήματος 6.1 είναι ίδιες με αυτές που αναφέρθηκαν στη παράγραφο Fugue 256-bit Clock Circles 36 και Fugue 256-bit Clock Circles 25. Οι προκαθορισμένες τιμές του σήματος message circle για τις αρχιτεκτονικές LP 1 και LP 2 είναι 10, 15, 20, 25 και 30.

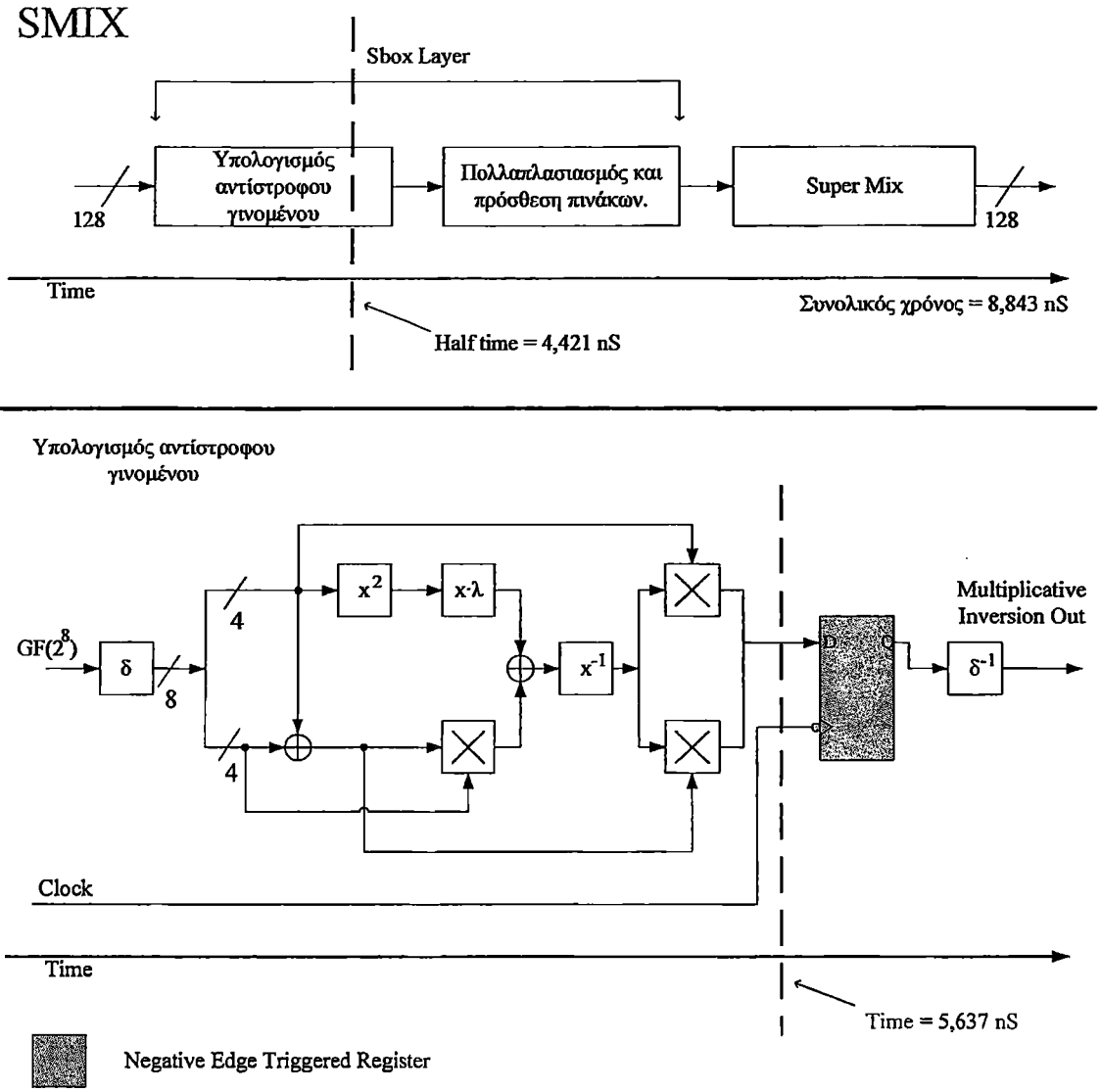
Fugue 256-bit Clk Crcls 49 LP 2 V2

Η υλοποίηση LP 2 V2 είναι ίδια με την υλοποίηση LP 2 με τη μόνη διαφορά τη χρήση της τεχνικής Double Edge Trigger (DET) pipeling η οποία αναφέρθηκε στο κεφάλαιο 5.2 . Η σύγκριση των δεδομένων κατανάλωσης της υλοποίησης LP 2 V2 θα γίνει με την υλοποίηση LP 2. Αναλόγως των αποτελεσμάτων η τεχνική αυτή μπορεί να εφαρμοστεί και στην αρχιτεκτονική LP 1.

Η εφαρμογή της τεχνικής αποφασίστηκε να χρησιμοποιηθεί στο κύκλωμα SMIX το οποίο χρησιμοποιείται σε όλη τη διάρκεια λειτουργίας του κυκλώματος Fugue 256-bit Clock circles καθώς επίσης και για το λόγο του ότι το κύκλωμα SMIX αποτελείται από πολλά υποκυκλώματα συνδυαστικής λογικής. Για το σκοπό αυτό μετρήθηκε η χρονική διαδρομή του κυκλώματος SMIX και ο αρνητικά ακμοπυροδότητος καταχωρητής τοποθετήθηκε στη μέση της χρονικής διαδρομής. Στο Σχήμα 6.26 δίνεται το κύκλωμα του SMIX μετά τη τοποθέτηση του αρνητικά ακμοπυροδότητου καταχωρητή.

Όπως προκύπτει και από το Σχήμα 6.26 συνολικός χρόνος της καθυστέρησης των δεδομένων διαμέσου του SMIX είναι 8,843 nS. Η μέση της χρονικής καθυστέρησης υπολογίζεται στα 4,421 nS και δείχνεται στο πάνω μέρος του Σχήματος 6.26 με τη διακεκομμένη γραμμή. Ο αρνητικά ακμοπυροδότητος καταχωρητής τοποθετείται στα 5,637 nS καθώς η χρονική καθυστέρηση αριστερά των δύο πολλαπλασιαστών είναι αρκετά μικρότερη από τη μέση συνολική χρονική καθυστέρηση.

Το πλεονέκτημα της χρήσης αυτής της τεχνικής είναι ότι δεν αυξάνονται οι κύκλοι επεξεργασίας σε σχέση με την αρχιτεκτονική LP 2 ενώ έχει χρησιμοποιηθεί ένας επιπλέον καταχωρητής για το περιορισμό των glitches.



Σχήμα 6.26: *Η υλοποίηση της τεχνικής DET pipeling στο κύκλωμα SMIX της αρχιτεκτονικής Fugue 256 Clk Crcls LP 2.*

Το μέγεθος του καταχωρητή είναι 8-bit και χρησιμοποιούνται 16 καταχωρητές ένα για το κάθε SBox. Επίσης χρησιμοποιείται ένας ακόμα αρνητικά ακμοπυροδότητος καταχωρητής στα δεδομένα εκείνα που εισέρχονται στο SMIX κύκλωμα αλλά δεν επεξεργάζονται όπως φαίνεται και στο Σχήμα 6.1. Ο λόγος είναι ο συγχρονισμός των δεδομένων στην έξοδο του SMIX κυκλώματος.

Fugue 256-bit Clk Crcls 49 LP 3 και Fugue 256-bit Clk Crcls 49 LP 4

Οι υλοποιήσεις Fugue256 Clk Crcls 49 LP 3 και Fugue256 Clk Crcls 49 LP 4 είναι ίδιες με τις υλοποιήσεις LP 1 και LP 2 αντίστοιχα με τη διαφορά ότι εδώ χρησιμοποιείται η τεχνική του Gated Clock. Το Gated Clock που εφαρμόζεται σε αυτές τις δύο υλοποιήσεις χρησιμοποιεί την ενσωματωμένη λειτουργία enable των flip-flop του FPGA. Η τεχνική αυτή δίνεται από το Σχήμα 4.4 .

Στα Σχήματα 6.27 και 6.28 δίνονται οι αρχιτεκτονικές των υλοποιήσεων LP 3 και LP 4. Τη λειτουργία των καταχωρητών ελέγχει του κύκλωμα του Controller μέσω των σημάτων ENB 1, ENB 2 και ENB 3 στη περίπτωση της αρχιτεκτονικής LP 3 και στη περίπτωση της αρχιτεκτονικής LP 4 μέσω των σημάτων ENB 1, ENB 2, ENB 3, ENB 4 και ENB 5. Οι κύκλοι επεξεργασίας των LP 3 και LP 4 παραμένουν οι ίδιοι με τους κύκλους επεξεργασίας των LP 1 και LP 2.

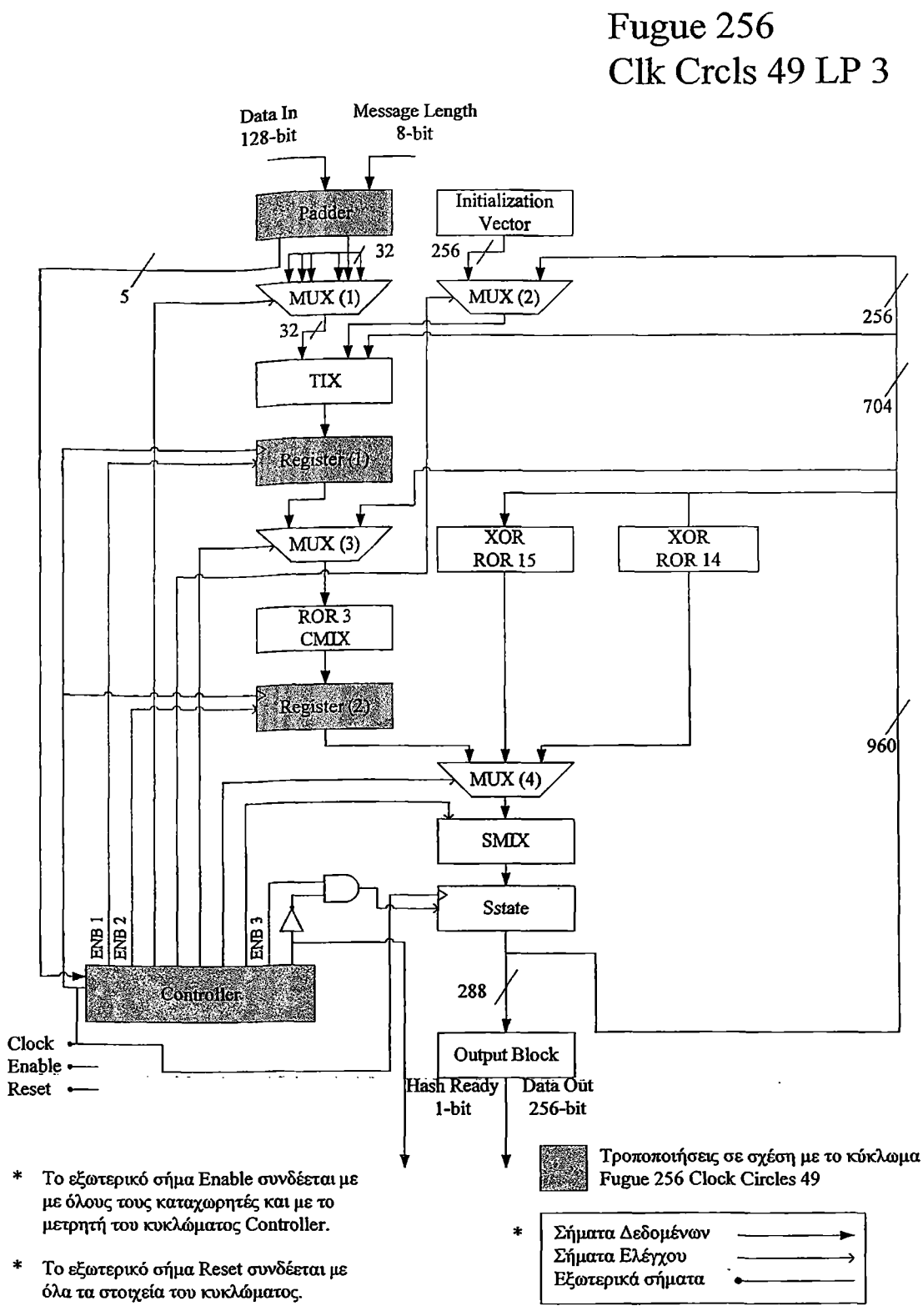
Fugue 256-bit Clk Crcls 49 LP 5 και Fugue 256-bit Clk Crcls 49 LP 6

Τα κυκλώματα Fugue256 Clk Crcls 49 LP 5 και Fugue256 Clk Crcls 49 LP 6 έχουν ίδια αρχιτεκτονική με το κύκλωμα LP 4 μόνο που στο LP 5 χρησιμοποιείται Gating Clock με τη χρήση πύλης AND όπως φαίνεται στο Σχήμα 4.3 ενώ στο LP 6 χρησιμοποιείται η μέθοδος του Feedback MUX του Σχήματος 4.4 . Οι κύκλοι επεξεργασίας είναι 77 και 103 για τα κυκλώματα LP 5 και LP 6.

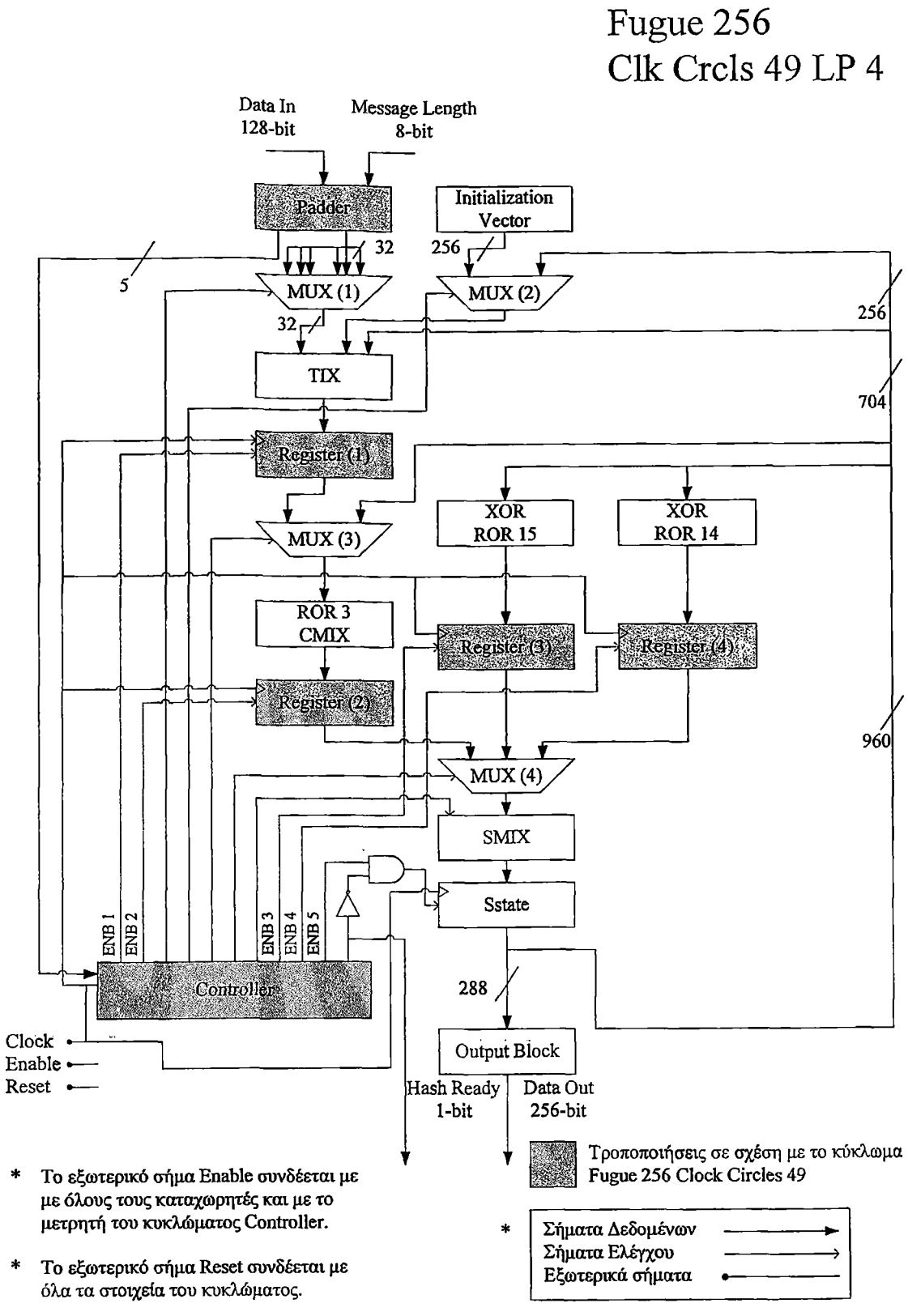
Fugue 256-bit Clk Crcls 49 (LP 7, LP 8, LP 9 και LP 10)

Το κύκλωμα LP 7 έχει ίδια αρχιτεκτονική με το κύκλωμα Fugue 256-bit Clock Circles 49 μόνο που για τη μείωση της δυναμικής κατανάλωσης χρησιμοποιεί την ενσωματωμένη μνήμη RAM που παρέχεται στα FPGAs για την υλοποίηση των 16 SBoxes. Οι κύκλοι επεξεργασίας του LP 7 λόγω της ύπαρξης της μνήμης διπλασιάζονται και γίνονται 98 ενώ οι προκαθορισμένες τιμές του σήματος message circles γίνονται 8, 12, 16, 20 και 24.

Το κύκλωμα LP 8 είναι ο συνδυασμός των κυκλωμάτων LP 4 και LP 7. Δηλαδή για την υλοποίηση των SBoxes χρησιμοποιείται μνήμη RAM καθώς επίσης χρησιμοποιούνται όλες οι τεχνικές χαμηλής κατανάλωσης του κυκλώματος LP 4. Οι κύκλοι επεξεργασίας του LP 8 λόγω του ότι συνδυάζει όλες τις τεχνικές αυξάνονται κατά πολύ και γίνονται 151 ενώ οι προκαθορισμένες τιμές του σήματος message circles γίνονται 14, 21, 28, 35 και 42.



Σχήμα 6.27: *Η αρχιτεκτονική του κυκλώματος Fugue 256 Clk Crcls LP 3.*



Σχήμα 6.28: *Η αρχιτεκτονική του κυκλώματος Fugue 256 Clk Crcls 49 LP 4.*

Το κύκλωμα LP 9 χρησιμοποιεί μνήμη RAM για τα SBoxes και τις τεχνικές χαμηλής κατανάλωσης του κυκλώματος LP 5. Το κύκλωμα LP 10 χρησιμοποιεί και αυτό μνήμη RAM και τις τεχνικές χαμηλής κατανάλωσης του κυκλώματος LP 6. Οι κύκλοι επεξεργασίας του LP 9 ανέρχονται στους 125 ενώ του LP 10 σε 151. Οι προκαθορισμένες τιμές του σήματος message circles για τα κυκλώματα LP 9 και LP 10 είναι ίδιες με αυτές αυτού του LP 8.

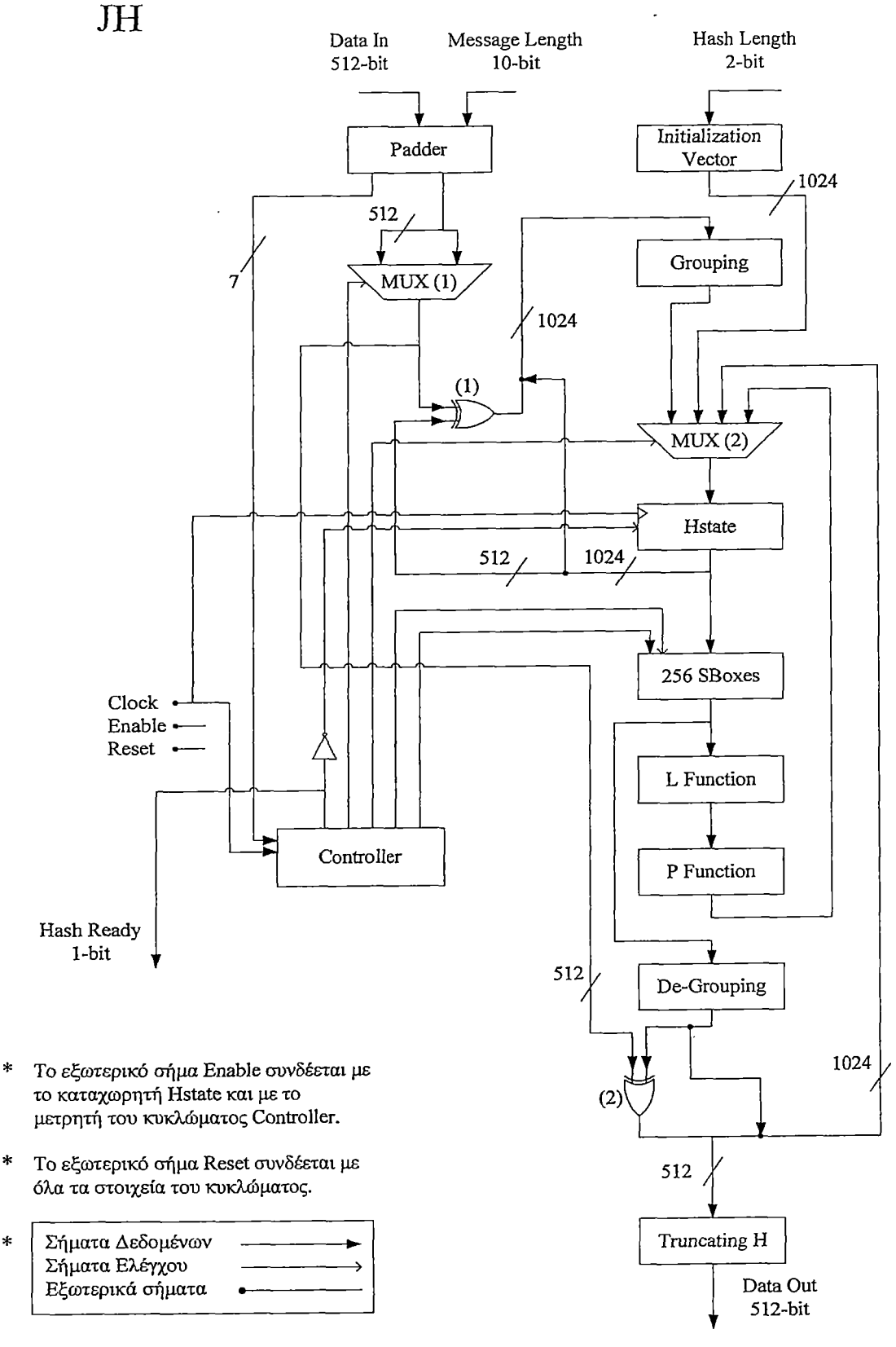
6.3 JH συνάρτηση κατακερματισμού

Στα πλαίσια της υλοποιήσεως του αλγορίθμου JH στη παρούσα εργασία σχεδιάστηκε ένα βασικό κύκλωμα που υλοποιεί τον αλγόριθμο JH. Επίσης σχεδιάστηκαν και άλλες 9 παραλλαγές του βασικού κυκλώματος που στόχο έχουν τη μείωση της κατανάλωσης του κυκλώματος χρησιμοποιώντας τις τεχνικές που παρουσιάστηκαν στο κεφάλαιο 4. Πρώτα θα παρουσιαστεί και θα αναλυθεί το βασικό κύκλωμα και στη συνέχεια θα παρουσιαστούν οι διαφοροποιήσεις του οι οποίες αποτελούν τις 9 παραπάνω παραλλαγές.

JH

Στο σχήμα 6.29 δίνεται το σχεδιάγραμμα του βασικού κυκλώματος. Η είσοδος και η έξοδος του κυκλώματος είναι 512-bit. Στο κύκλωμα υπολογίζεται η σύνοψη του μηνύματος ανάλογα το μέγεθος του εισερχόμενου μηνύματος. Για εισερχόμενο μήνυμα μεγέθους 0-bit απαιτούνται 37 κύκλοι ρολογιού ενώ για μέγεθος από 1 έως 512 bit απαιτούνται 74 κύκλοι ρολογιού. Για αυτή τη παραμετρική συμπεριφορά του κυκλώματος χρειάζεται μια επιπλέον είσοδος στο κύκλωμα, η Message Length, και αποτελείται από 10–bit. Το Message Length φέρει τη πληροφορία για το μήκος του εισερχόμενου μηνύματος. Επίσης στο κύκλωμα εισέρχονται 3 ακόμα σήματα ενός bit τα οποία είναι το ρολόι (Clock), ένα σήμα για την ασύγχρονη αρχικοποίηση του κυκλώματος (Reset) και ένα σήμα για την ασύγχρονη παύση της λειτουργίας του κυκλώματος (Enable).

Το κύκλωμα του Σχήματος 6.29 παρέχει τη δυνατότητα επιλογής του μεγέθους της σύνοψης που δημιουργεί το κύκλωμα. Για την επιλογή εισέρχεται ένα ακόμα σήμα στο κύκλωμα το οποίο ονομάζεται Hash Length και έχει μέγεθος 2-bit.



Σχήμα 6.29: *Η αρχιτεκτονική του κυκλώματος JH.*

Μαζί με την έξοδο των 512-bit υπάρχει και ένα επιπλέον σήμα ενός bit το οποίο οδηγείται σε λογικό 1 μετά την ολοκλήρωση του υπολογισμού της σύνοψης του αρχικού μηνύματος και ονομάζεται Hash Ready.

Όπως αναφέρθηκε στο κεφάλαιο 5.3 ο αλγόριθμος JH διαχειρίζεται ένα τμήμα δεδομένων συνολικού μήκους 1024-bit. Για τη μεταφορά αυτού του τμήματος στο Σχήμα 6.29 χρησιμοποιείται ένα κοινό σύνολο καλωδίων (Bus) πλάτους 1024-bit.

Το πρώτο συνδυαστικό υποκύκλωμα που συναντάται στο Σχήμα 6.29 ονομάζεται Padder και υλοποιεί τη διαδικασία του Padding που περιγράφτηκε στη παράγραφο 5.3 . Έχει δύο εισόδους, μία για το εισερχόμενο μήνυμα μήκους 512-bit και μια για το μέγεθος του μηνύματος μήκους 10-bit. Επίσης έχει δύο εξόδους, μία για τα δεδομένα μετά το Padding μήκους 1024-bit και μια έξοδος 7-bit η οποία περιέχει πληροφορία για τους κύκλους ρολογιού που θα χρειαστούν και κατευθύνεται στο κύκλωμα ελέγχου που ονομάζεται Controller.

Η έξοδος του Padder στη συνέχεια διαχωρίζεται σε 2 ομάδες των 512-bit καθώς βάση του αλγορίθμου το κύκλωμα επεξεργάζεται μηνύματα μήκους 512-bit. Υπεύθυνος για το ποια ομάδα των 512-bit θα επεξεργαστεί είναι ο πολυπλέκτης 2 σε 1 (MUX (1)) που έπεται του Padder.

Στη συνέχεια η έξοδος του πολυπλέκτη 2x1 οδηγείται σε μια πύλη αποκλειστικού-Η (XOR (1)) όπου μαζί με το πρώτο μισό της εξόδου του κύριου καταχωρητή του κυκλώματος υλοποιούν τη διαδικασία που περιγράφηκε στο κεφάλαιο 5.3 καθώς και δόθηκε και από το Σχήμα 5.9 . Η έξοδος της πύλης αποκλειστικού-Η ενώνεται με το δεύτερο μισό της εξόδου του καταχωρητή Hstate και οδηγούνται στο επόμενο κύκλωμα που ονομάζεται Grouping. Το κύκλωμα Grouping πραγματοποιεί τη διαδικασία που περιγράφηκε στη παράγραφο «Grouping – De-Grouping» και από το Σχήμα 5.11 του κεφαλαίου 5 .

Η έξοδος του κυκλώματος Grouping οδηγείται σε ένα πολυπλέκτη 4 σε 1 (MUX (2)). Σε αυτό το πολυπλέκτη οδηγούνται επίσης οι έξοδοι των κυκλωμάτων P Function, η έξοδος της πύλης αποκλειστικού-Η (XOR (2)) και η έξοδος του κυκλώματος Initialization Vector. Το κύκλωμα αυτό είναι υπεύθυνο για τη σωστή επιλογή του πίνακα αρχικοποίηση βάση του σήματος Hash Length.

Ο πολυπλέκτης MUX (2) συνδέεται με ένα καταχωρητή για το συγχρονισμό των δεδομένων ο οποίος ονομάζεται Hstate και είναι μεγέθους 1024-bit. Ο Hstate είναι θετικά ακμοπυροδότητος καταχωρητής και παρέχεται και η δυνατότητα παύσης της λειτουργίας του

από το εξωτερικό σήμα Enable. Επίσης ο καταχωρητής σταματάει τη λειτουργία του όταν ολοκληρωθεί ο υπολογισμός της σύνοψης μέσω ενός σήματος ελέγχου που προέρχεται από κύκλωμα Control και ονομάζεται Hash Ready. Για το λόγο αυτό στην είσοδο του σήματος ελέγχου του καταχωρητή υλοποιείται η πράξη ENABLE REGISTER = External Enable AND $\overline{\text{Hash Ready}}$.

Στη συνέχεια ακολουθεί το κύκλωμα 256 SBoxes το οποίο είναι υπεύθυνο για την αντικατάσταση των δεδομένων με τα δεδομένα των SBoxes. Χρησιμοποιούνται 256 SBoxes καθώς βάση του αλγορίθμου τα 1024-bit χωρίζονται σε τμήματα των 4-bit και χρειάζεται ένα κουτί αντικατάστασης για κάθε τέτοιο τμήμα. Τα κυκλώματα P Function και L Function έπονται του κυκλώματος SBoxes και υλοποιούν τις διαδικασίες που περιγράφηκαν στις παραγράφους P επίπεδο και L επίπεδο του κεφαλαίου 5.3 . Όπως αναφέρθηκε και πριν η έξοδος του κυκλώματος P Function οδηγείται στο πολυπλέκτη MUX (2). Η έξοδος του κυκλώματος SBoxes εκτός του ότι συνδέεται με το κύκλωμα L Function συνδέεται και με το κύκλωμα De-Grouping. Με αυτόν το σχεδιασμό ουσιαστικά πραγματοποιείται η διαδικασία R_d* του Σχήματος 5.10 η οποία αποτελεί τον τριακοστό έκτο γύρο του αλγορίθμου όπου εκτελείται μόνο το επίπεδο αντικατάστασης των SBoxes.

Το δεύτερο μισό της έξοδος του κυκλώματος De-Grouping οδηγείται σε μια πύλη αποκλειστικού-Η (XOR (2)) όπου μαζί με το αρχικό μήνυμα υλοποιούν το δεύτερο στάδιο του Σχήματος 5.9 . Το αποτέλεσμα της πράξης αυτής οδηγείται στο κύκλωμα Truncating H και παράλληλα ενώνεται με το πρώτο μισό της εξόδου του κυκλώματος De-Grouping και στη συνέχεια οδηγούνται στο πολυπλέκτη MUX (2) . Στο πολυπλέκτη MUX (2) οδηγούνται στη περίπτωση που υπάρχει και άλλο μήνυμα προς επεξεργασία ενώ στο κύκλωμα Truncating H οδηγούνται για να δημιουργηθεί η έξοδος του κυκλώματος JH. Το κύκλωμα Truncating H δέχεται το αποτέλεσμα της πύλης αποκλειστικού-Η (XOR (2)) μήκους 512-bit και βάση του σήματος Hash Length αποφασίζει το μέγεθος της σύνοψης του κυκλώματος JH.

Τέλος το κύκλωμα που είναι υπεύθυνο για τη κατάσταση όλων των πολυπλεκτών του Σχήματος 6.29 αλλά και του σήματος Hash Ready όπως και του σήματος αρχικοποίησης των SBOXes ονομάζεται Controller. Έχει 3 εισόδους για τα εξωτερικά σήματα και μια ακόμα είσοδο για το σήμα που περιέχει τη πληροφορία για τους κύκλους επεξεργασίας το οποίο προέρχεται από το κύκλωμα Padder. Επίσης έχει 2 εξόδους για τον έλεγχο των 2 πολυπλεκτών μια έξοδο για την αρχικοποίηση των SBOXes μια έξοδο για το σήμα Hash Ready και μια έξοδο για το σήμα που φέρει την πληροφορία σε ποιο γύρο βρίσκεται ο

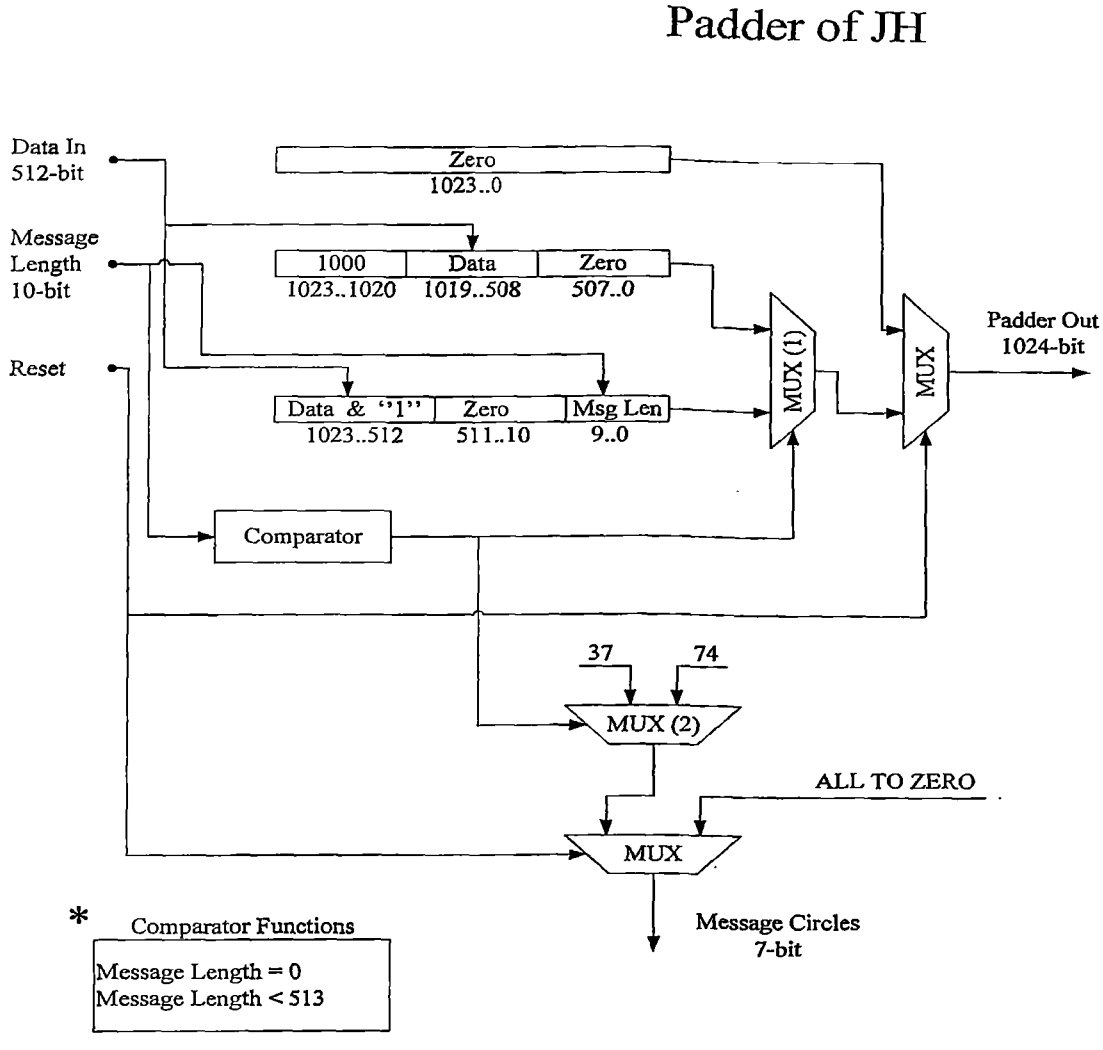
αλγόριθμος. Η πληροφορία αυτή απαιτείται από το κύκλωμα 256 SBoxes για να επιλεγούν τα σωστά τμήματα των κουτιών αντικατάστασης.

Padder

Στο Σχήμα 6.30 δίνεται το κύκλωμα του Padder. Όπως έχει αναφερθεί το κύκλωμα JH είναι παραμετρικό όσων αφορά τους κύκλους ρολογιού που χρειάζονται για την επεξεργασία του μηνύματος. Βάση του αλγορίθμου οι κύκλοι ρολογιού που χρειάζονται εξαρτούνται από το μέγεθος του μηνύματος. Για το σκοπό αυτό λοιπόν στο κύκλωμα Padder γίνονται 2 padding κάθε φορά που εισέρχεται κάποιο μήνυμα ανεξάρτητα από το μέγεθος του. Στη συνέχεια ένας πολυπλέκτης 2 σε 1 (MUX (1)) επιλέγει το σωστό padding για έξοδο βασιζόμενο στο μέγεθος του μηνύματος. Ο πολυπλέκτης στην είσοδο του σήματος επιλογής δέχεται το αποτέλεσμα ενός συγκριτή (Comparator) ο οποίος πραγματοποιεί τις συγκρίσεις που φαίνονται στο κάτω μέρος του Σχήματος 6.30. Βάση των αποτελεσμάτων αυτών των συγκρίσεων ο πολυπλέκτης επιλέγει τη σωστή έξοδο του Padder. Ακριβώς με την ίδια λογική ο δεύτερος πολυπλέκτης 2 σε 1 (MUX (2)) επιλέγει τη σωστή προκαθορισμένη τιμή για το σήμα Message Circle. Το σήμα Message Circle όπως θα αναφερθεί και στη παράγραφο για τον Controller χρησιμοποιείται από τον μετρητή του κυκλώματος για να μπορέσει να σταματήσει η λειτουργία του κυκλώματος αν έχει ολοκληρωθεί η σύνοψη του μηνύματος που απαιτείται. Οι δύο πολυπλέκτες 2 σε 1 (MUX) του κυκλώματος χρησιμεύουν στην ασύγχρονη αρχικοποίηση του κυκλώματος μέσω του εξωτερικού σήματος Reset.

Η έξοδος του padding έχει μήκος 1024-bit και αυτό οφείλεται στο ότι απαιτούνται, βάση της παραγράφου 5.3 του κεφαλαίου 5, 16 byte (128-bit) για τη δυαδική αναπαράσταση του μεγέθους του αρχικού μηνύματος και 384 padded bits όπου το πρώτο bit τίθενται στο λογικό 1 και τα υπόλοιπα στο λογικό 0. Κάτι το οποίο δίνει το συνολικό αποτέλεσμα των 1024-bit εφόσον η είσοδος έχει μήκος 512-bit. Για τη δυαδική αναπαράσταση του αριθμού 512 απαιτούνται 10-bit. Για το λόγο αυτό τα πρώτα 118-bit των 16 byte που απαιτούνται για τη δυαδική αναπαράσταση είναι προκαθορισμένα στο λογικό 0 ενώ τα υπόλοιπα 10 bit παίρνουν τιμές ανάλογα του μεγέθους του μηνύματος.

Στη πρώτη περίπτωση padding όπου η είσοδος και το μέγεθος του κυκλώματος είναι 0 όλα τα bits τίθενται στο λογικό 0 εκτός του πρώτου ενώ στην άλλη περίπτωση στο πρώτο μέρος τοποθετείται το μήνυμα και στη συνέχεια ανάλογα του μεγέθους του μηνύματος τοποθετούνται τα padded bits. Η τοποθέτηση των padded bits γίνεται βάση της προϋπόθεσης



Σχήμα 6.30: *Η αρχιτεκτονική του Padder του κυκλώματος JH.*

ότι το μήνυμα πρέπει συμπληρώνεται με λογικά 0 έτσι ώστε να σχηματίζονται ομάδες πολλαπλάσιες των 512-bit.

Initialization Vector

Όπως έχει αναφερθεί και στη παράγραφο 5.3 «Αρχικοποίηση» του κεφαλαίου 5 ο υπολογισμός των 4 πινάκων αρχικοποίησης δίνεται από τη Σχέση 5.15. Ο υπολογισμός των πινάκων μπορεί να γίνει είτε δυναμικά χρησιμοποιώντας το ίδιο το κύκλωμα JH είτε να προ-υπολογιστούν και να χρησιμοποιηθούν ως προκαθορισμένοι. Στα κυκλώματα της παρούσας εργασίας έχουν υπολογιστεί και οι 4 πίνακες αρχικοποίησης και χρησιμοποιούνται στο

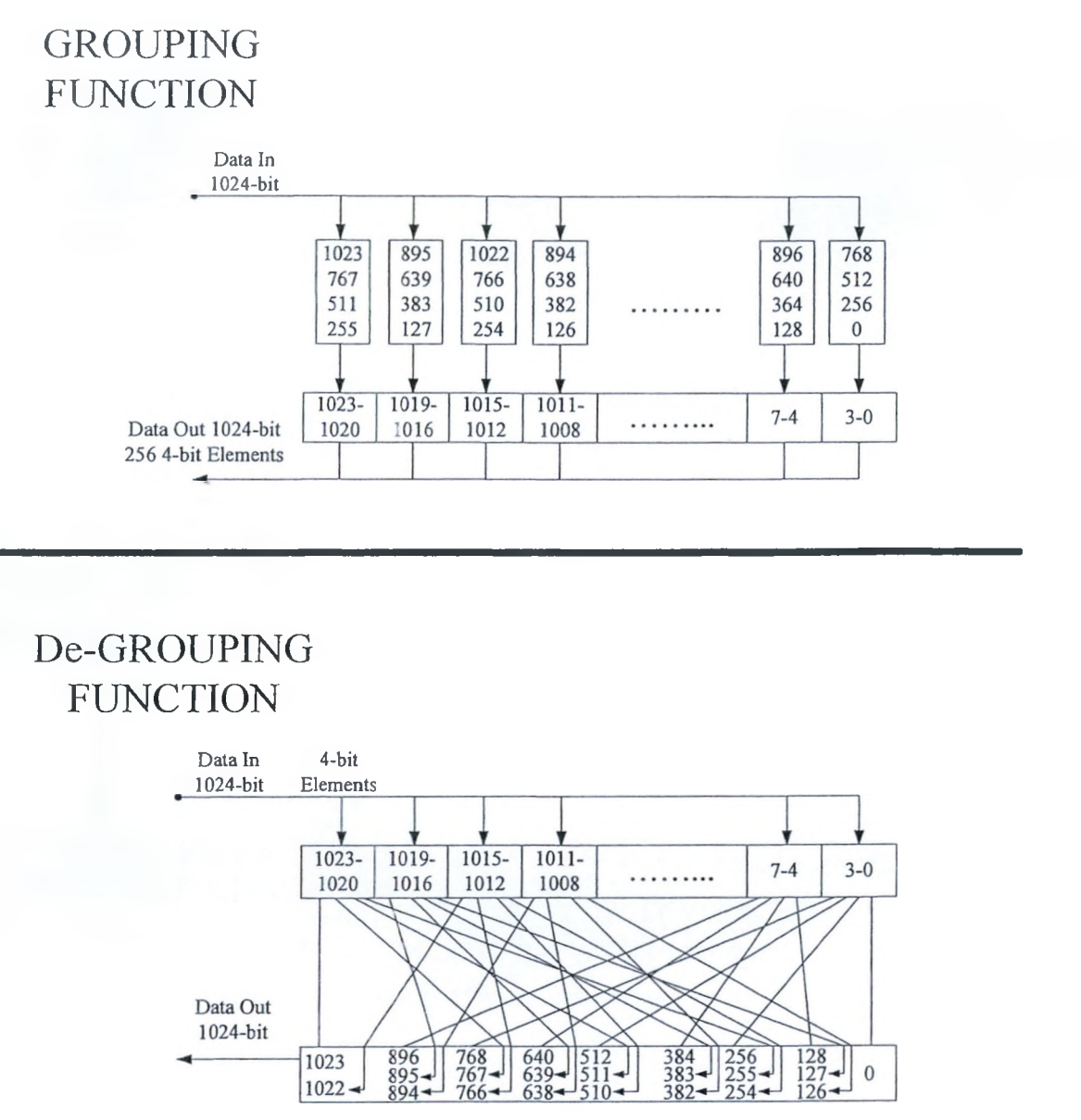
κύκλωμα ως προκαθορισμένες τιμές. Η επιλογή του σωστού πίνακα γίνεται μέσω ενός πολυπλέκτη 4 σε 1 ο οποίος χρησιμοποιεί το σήμα Hash Length για αυτό το σκοπό. Στο κύκλωμα JH οι πίνακες αρχικοποίησης αποθηκεύονται στη κατανεμημένη μνήμη των FPGA. Οι τιμές των 4 πινάκων αρχικοποίησης δίνονται από το Πίνακα 6.1 σε δεκαεξαδική μορφή.

Πίνακας 6.1: *Οι πίνακες αρχικοποίησης του αλγορίθμου JH.*

Hash Length	Initialization Vectors
224-bit	82C270E00BED02308D0C3A9E31CE34B18F0C942FBA46CD871EC4D80AFC7971C4
	61E01ABB69962D7BAF71893DE13D8697D2520460F7C9C094C76349CA3DA5799CF
	D8B551FBDBCEB9F0834BD5BB442F8BFBA515C35B9C7999E55A44E6271CC13B38
	5725793C185F72545366B69005025D23390EBDB27DD1EDFCCBAADE17E603DE9
256-bit	C968B8E2C53A596E427E45EF1D7AE6E56145B7D906711F7A2FC7617806A922017B
	2991C1B91929E2C42B4CE18CC5A2D66220BECA901B5DDFD3B205638EA7AC5F14
	3E8CBA6D313104B0E70054905272714CCE321E075DE5101BA800ECE20251789F577
	2795FD104A5F0B8B63425F5B2381670FA3E5F907F17E28FC064E769AC90
384-bit	079C23AB64AB2D408CB51CE447DEE98D8D9BB1627EC25269BAB62D2B002FFC80
	CBAFBCEF308C173AAD6FA3AA31194031898977423A6F4CE3BF2E732B440DDB7D
	F2C43ECAA63A54E58A37B80AFC4422C5A397C3BC04E9E09137A80453E14860FA71
	31D33A5FD4BEA6DCDA4AF8F43385126EC7F8F4C84958D08B9E94A34695B6A9
512-bit	50AB6058C60942CC4CE7A54CBDB9DC1BAF2E7AFBD1A15E24E5F44EABC4D5C0
	A14CF243660C562073999381EA9A8B3D18CF65D9FCA940B6C79E831273BEFE3B6
	60F9A2F7E0A32D8E017D491558E0B134005B5E4DEC44E5F3F8CBC5AEE98FD1D32
	14081C25E46CE6C41B4B95BCE1BD43DB7F229EC243B680140A33B909333C0303

Grouping και De-Grouping

Η περιγραφή των κυκλωμάτων Grouping και De-Grouping δόθηκε στη παράγραφο 5.3 από τα Σχήματα 5.11 και 5.12. Στο Σήμα 6.31 δίνεται η ίδια λειτουργία των δύο κυκλωμάτων για το τμήμα των 1024-bit Hstate που χρησιμοποιεί το κύκλωμα JH. Στο κύκλωμα Grouping όπως προκύπτει και από το Σχήμα 6.31 τα bit από τις θέσεις 1024, 767, 511 και 255 τοποθετούνται στις θέσεις 1023 έως 1020. Στη συνέχεια τα bit από τις θέσεις 895,639,383 και 121 τοποθετούνται στις θέσεις 1019 έως 1016. Ακολουθώντας το ίδιο πρότυπο μέχρι το τελευταίο bit πραγματοποιείται η διαδικασία του Grouping. Στο κύκλωμα De-Grouping γίνεται ακριβώς η αντίστροφη διαδικασία. Τα bit από τις θέσεις 1023 έως 1020 τοποθετούνται ξανά στις θέσεις 1023, 767, 511 και 255, τα bit από τις θέσεις 1019



Σχήμα 6.31: *Η λειτουργία των κυκλωμάτων Grouping και De-Grouping.*

έως 1016 τοποθετούνται ξανά στις θέσεις 895, 639, 383 και 121. Πάλι αν ακολουθηθεί το ίδιο πρότυπο μέχρι το τέλος θα δημιουργηθεί ξανά η διάταξη του αρχικού τμήματος Hstate.

256 SBoxes

Το κύκλωμα 256 SBoxes υλοποιεί την αντικατάσταση των δεδομένων με τα δεδομένα των κουτιών αντικατάστασης. Η αντικατάσταση αυτή γίνεται χωρίζοντας το τμήμα Hstate που προκύπτει από το κύκλωμα Grouping σε τμήματα των 4-bit. Στο κύκλωμα JH για να

πραγματοποιείται αυτή η αντικατάσταση σε ένα κύκλο ρολογιού υπάρχουν 256 SBoxes του Πίνακα 5.4 για το κάθε τμήμα 4-bit. Βάση του αλγορίθμου που περιγράφηκαν στις παραγράφους «S-box» και «Η συνάρτηση R_d» του κεφαλαίου 5.3 για την επιλογή του ενός SBox από το ζευγάρι των δύο του Πίνακα 5.4 απαιτείται μια σταθερά. Αυτές οι σταθερές μπορούν να υπολογίζονται δυναμικά κατά τη διάρκεια λειτουργίας ή να έχουν προ-υπολογιστεί και να χρησιμοποιούνται ως προκαθορισμένες τιμές. Στο κύκλωμα JH έχουν προ-υπολογιστεί και αποθηκεύονται στη κατανεμημένη μνήμη του που παρέχεται στα FPGA. Στη συνέχεια ένας πολυπλέκτης αποφασίζει ποια σταθερά θα χρησιμοποιηθεί βάση του σήματος ελέγχου που φέρει τη πληροφορία για το γύρο που βρίσκεται η εκτέλεση.

Στο Σχήμα 6.32 δίνεται το κύκλωμα του 256 SBoxes. Στο πάνω μέρους του σχήματος φαίνεται πως διαχειρίζεται το κύκλωμα τα 256 SBoxes μαζί με το κύκλωμα που επιλέγει τη σταθερά που θα χρησιμοποιηθεί. Στο κάτω μέρος δίνονται το κύκλωμα Constant Value Component που επιλέγει τη σταθερά του γύρου και επίσης δίνεται και ένα SBox Component. Στο κύκλωμα SBox Component γίνεται η αντικατάσταση της εισερχόμενης τιμής και με τα δύο SBoxes και στη συνέχεια επιλέγεται η σωστή αντικατάσταση βάση της σταθεράς του κάθε γύρου.

Όπως συνέβαινε και στα SBoxes του κυκλώματος Fugue έτσι και εδώ για την αρχικοποίηση του κυκλώματος 256 SBoxes υπάρχει ένα ξεχωριστό σήμα που ονομάζεται Reset SBoxes και ελέγχεται από το κύκλωμα του Controller. Αυτό το σήμα ελέγχει έναν πολυπλέκτη ο οποίος κατά τη διάρκεια της αρχικοποίησης δίνει ως είσοδο στο κάθε SBox τη τιμή 1 η οποία βάση του Πίνακα 5.4 αναγκάζει τα SBoxes να δώσουν έξοδο το λογικό 0.

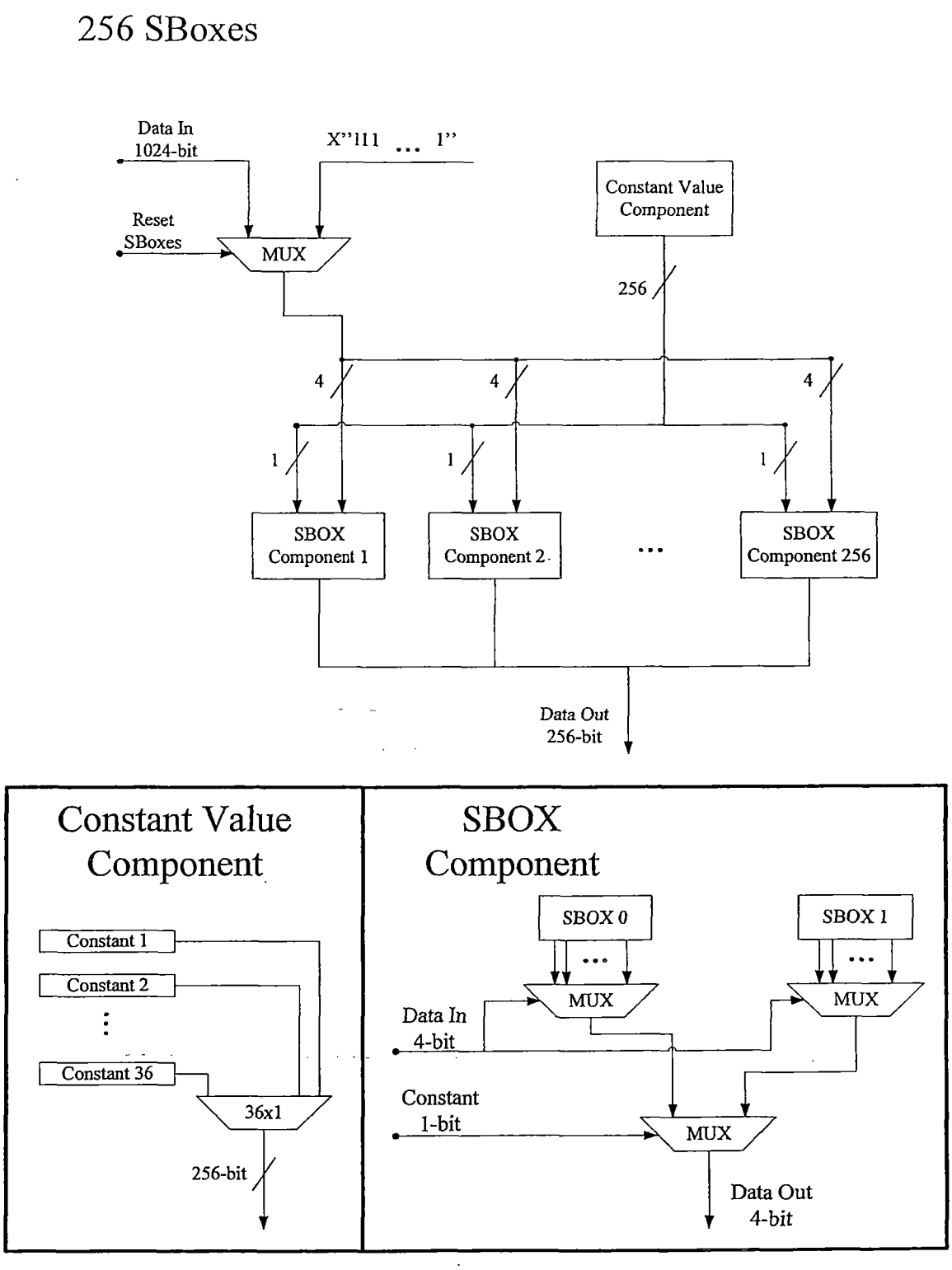
Οι 36 υπολογισμένες σταθερές που χρησιμοποιούνται από το κύκλωμα Constant Value Component δίνονται από το Πίνακα 6.2 σε δεκαεξαδική μορφή.

L Function

Το κύκλωμα L Function υλοποιεί τη Σχέση 5.10 που δόθηκε στη παράγραφο «L Επίπεδο» του κεφαλαίου 5. Εκεί οι 10 πράξεις αποκλειστικού-Η γίνονται μεταξύ του A και B που υποτίθεται ότι έχουν μέγεθος 4-bit. Στο κύκλωμα L Function τα δεδομένα του εισερχόμενου τμήματος Hstate χωρίζονται ανά 8-bit και οδηγούνται σε 128 Linear Transformation κυκλώματα και δίνονται από το Σχήμα 6.33 . Εκεί υλοποιείται η Σχέση 5.10 όπου τα bits 7 έως 4 αναπαριστούν το A της Σχέσης 5.10 ενώ τα υπόλοιπα bits

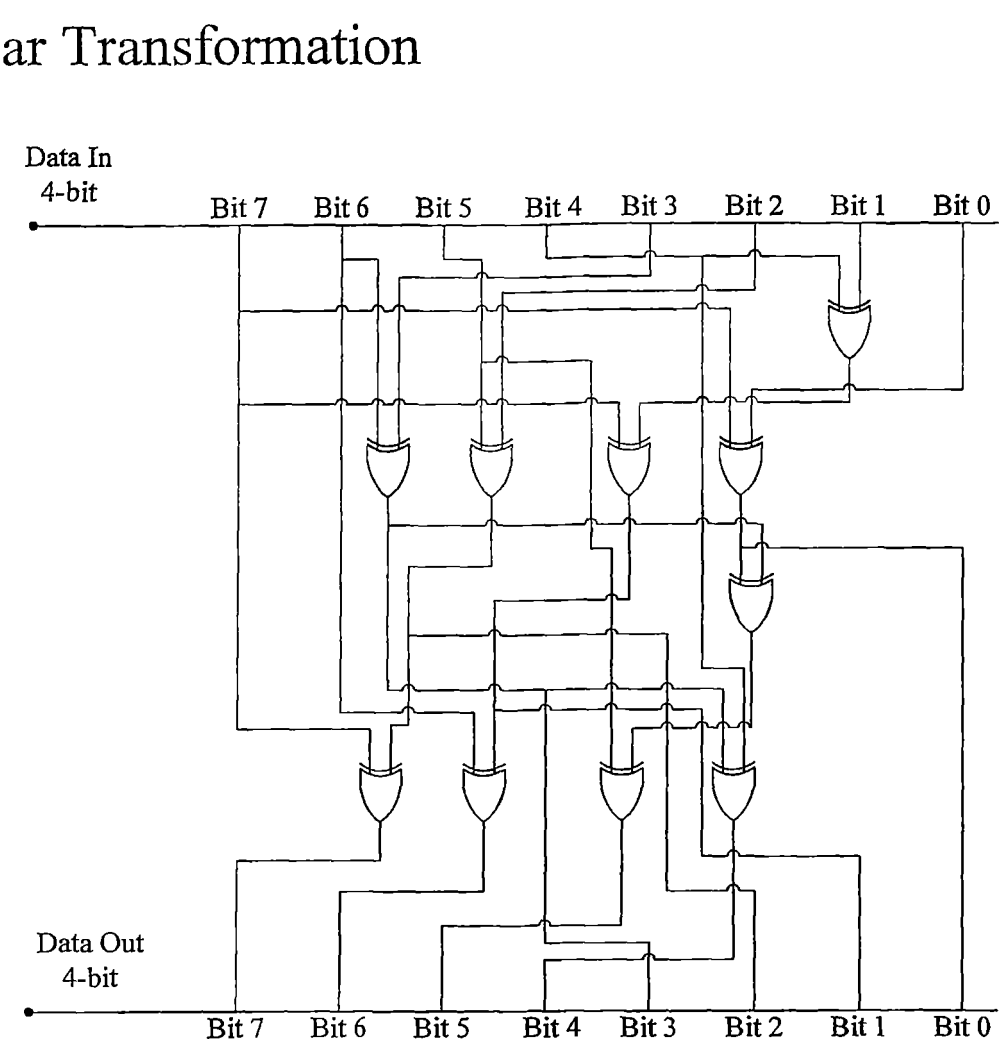
Πίνακας 6.2: *Οι 36 σταθερές σε δεκαεξαδική μορφή που χρησιμοποιούνται από το κύκλωμα Constant Value Component.*

Round	Constant Values
1	6a09e667f3bcc908b2fb1366ea957d3e3adec17512775099da2f590b0667322a
2	bb896bf05955abcd5281828d66e7d99ac4203494f89bf12817deb43288712231
3	1836e76b12d79c55118a1139d2417df52a2021225ff6350063d88e5f1f91631c
4	263085a7000fa9c3317c6ca8ab65f7a7713cf4201060ce886af855a90d6a4eed
5	1cebafd51a156aeb62a11fb3be2e14f60b7e48de85814270fd62e97614d7b441
6	e5564cb574f7e09c75e2e244929e9549279ab224a28e445d57185e7d7a09fdc1
7	5820f0f0d764cff3a5552a5e41a82b9eff6ee0aa615773bb07e8603424c3cf8a
8	b126fb741733c5bfcef6f43a62e8e5706a26656028aa897ec1ea4616ce8fd510
9	dbf0de32bca77254bb4f562581a3bc991cf94f225652c27f14eae958ae6aa616
10	e6113be617f45f3de53cff03919a94c32c927b093ac8f23b47f7189aadb9bc67
11	80d0d26052ca45d593ab5fb3102506390083afb5ffe107dacfcba7dbe601a12b
12	43af1c76126714dfa950c368787c81ae3beecf956c85c962086ae16e40ebb0b4
13	9aee8994d2d74a5cdb7b1ef294eed5c1520724dd8ed58c92d3f0e174b0c32045
14	0b2aa58ceb3bdb9e1eef66b376e0c565d5d8fe7bacb8da866f859ac521f3d571
15	7a1523ef3d970a3a9b0b4d610e02749d37b8d57c1885fe4206a7f338e8356866
16	2c2db8f7876685f2cd9a2e0ddb64c9d5bf13905371fc39e0fa86e1477234a297
17	9df085eb2544ebf62b50686a71e6e828dfed9dbe0b106c9452ceddff3d138990
18	e6e5c42cb2d460c9d6e4791a1681bb2e222e54558eb78d5244e217d1bfcf5058
19	8f1f57e44e126210f00763ff57da208a5093b8ff7947534a4c260a17642f72b2
20	ae4ef4792ea148608cf116cb2bff66e8fc74811266cd641112cd17801ed38b59
21	91a744efbf68b192d0549b608bdb3191fc12a0e83543cec5f882250b244f78e4
22	4b5d27d3368f9c17d4b2a2b216c7e74e7714d2cc03e1e44588cd9936de74357c
23	0ea17cafb8286131bda9e3757b3610aa3f77a6d0575053fc926eea7e237df289
24	848af9f57eb1a616e2c342c8cea528b8a95a5d16d9d87be9bb3784d0c351c32b
25	c0435cc3654fb85dd9335ba91ac3dbde1f85d567d7ad16f9de6e009bca3f95b5
26	927547fe5e5e45e2fe99f1651ea1cbf097dc3a3d40ddd21cee260543c288ec6b
27	c117a3770d3a34469d50dfa7db020300d306a365374fa828c8b780ee1b9d7a34
28	8ff2178ae2dbe5e872fac789a34bc228debf54a882743caad14f3a550fdbe68f
29	abd06c52ed58ff091205d0f627574c8cbc1fe7cf79210f5a2286f6e23a27efa0
30	631f4acb8d3ca4253e301849f157571d3211b6c1045347befb7c77df3c6ca7bd
31	ae88f2342c23344590be2014fab4f179fd4bf7c90db14fa4018fcce689d2127b
32	93b89385546d71379fe41c39bc602e8b7c8b2f78ee914d1f0af0d437a189a8a4
33	1d1e036abeef3f44848cd76ef6baa889fcec56cd7967eb909a464bfc23c72435
34	a8e4ede4c5fe5e88d4fb192e0a0821e935ba145bbfc59c2508282755a5df53a5
35	8e4e37a3b970f079ae9d22a499a714c875760273f74a9398995d32c05027d810
36	61cfa42792f93b9fde36eb163e978709fafa7616ec3c7dad0135806c3d91a21b



Σχήμα 6.32: *Το κύκλωμα 256 SBoxes και τα επιμέρους κυκλώματα του.*

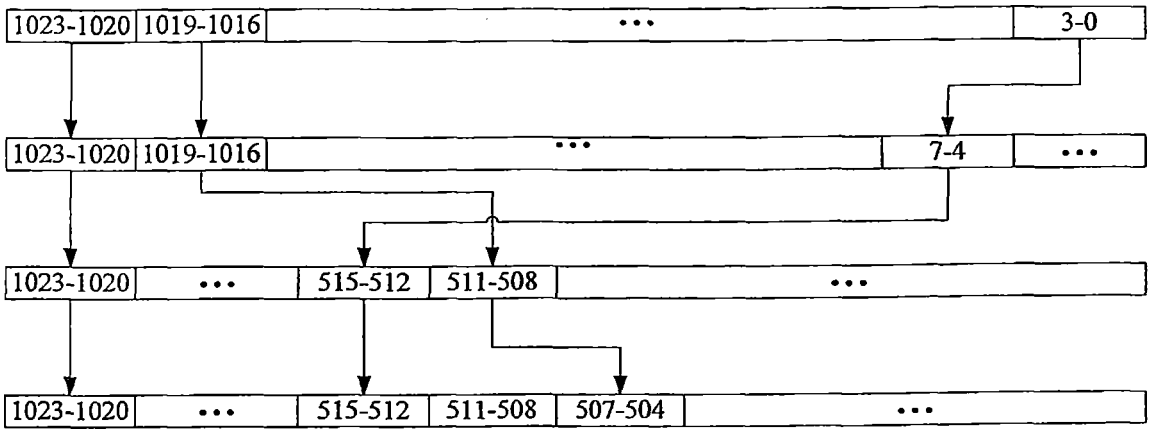
αναπαριστούν το B της Σχέσης 5.10 . Οι έξοδοι των 128 κυκλωμάτων ενώνονται και σχηματίζουν πάλι το τμήμα των 1024 bit Hstate.



Σχήμα 6.33: *Το κύκλωμα Linear Transformation της Σχέσης 5.10 .*

P Function

Στο κύκλωμα P Function συμβαίνει απλή μετατόπιση των bits βάση των αλγορίθμων που περιγράφηκαν στη παράγραφο «P επίπεδο» του κεφαλαίου 5 . Στο Σχήμα 6.34 δίνεται ένα παράδειγμα μετατόπισης των πρώτων 8-bit και των τελευταίων 4-bit. Τα πρώτα 4 bit εξέρχονται στις ίδιες θέσεις με αυτές που εισήλθαν. Η δεύτερη ομάδα 4-bit από τις θέσεις



Σχήμα 6.34: *Παράδειγμα μετατόπισης των πρώτων 8-bit και των 4 τελευταίων bit στο κύκλωμα P Function.*

1019-1016 καταλήγει να εξέρχεται από τις θέσεις 507 έως 504. Και τέλος η τελευταία ομάδα bit καταλήγει να εξέρχεται από τις θέσεις 515 έως 512.

Truncating H

Η έξοδος του κυκλώματος JH δίνεται από το κύκλωμα Truncating H το οποίο ανάλογα του σήματος Hash Length δίνει 4 εξόδους. Η είσοδος του κυκλώματος είναι μήκους 512-bit και πρόκειται για το δεύτερο μισό του τμήματος Hstate. Στη πρώτη έξοδο αν το σήμα Hash Length έχει τη τιμή 0 δίνει τα τελευταία 226-bit της εισόδου του ως έξοδο συμπληρωμένα με 0 για να σχηματιστούν τα 512-bit. Στη περίπτωση όπου το Hash Length έχει τιμή 1 τότε δίνει τα 256-bit τα οποία και αυτά συμπληρώνονται. Όταν το Hash Length έχει τιμή 2 τότε δένει ως έξοδο τα 384-bit τα οποία επίσης συμπληρώνονται και τέλος για τιμή 3 του Hash Length δίνει ως έξοδο απευθείας την είσοδο του. Την παραπάνω λειτουργία αναλαμβάνει να την εκτελέσει ένας πολυπλέκτης.

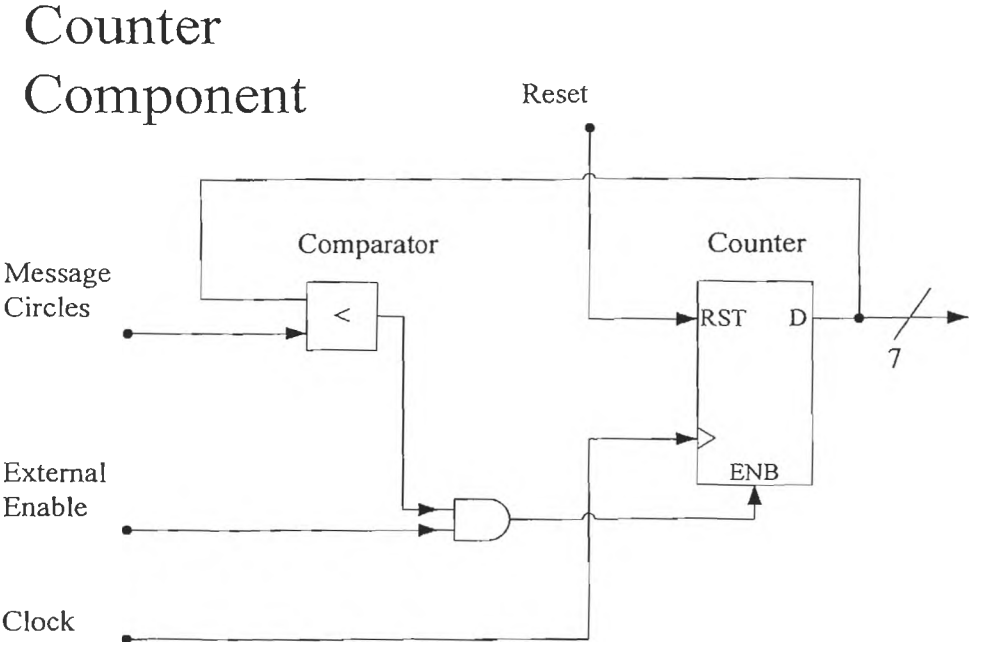
Controller

Όπως συμβαίνει και στο κύκλωμα Fugue το κύκλωμα που είναι υπεύθυνο για τη διαχείριση των 2 πολυπλεκτών, των δύο σημάτων ελέγχου και το γύρου στον οποίο βρίσκεται

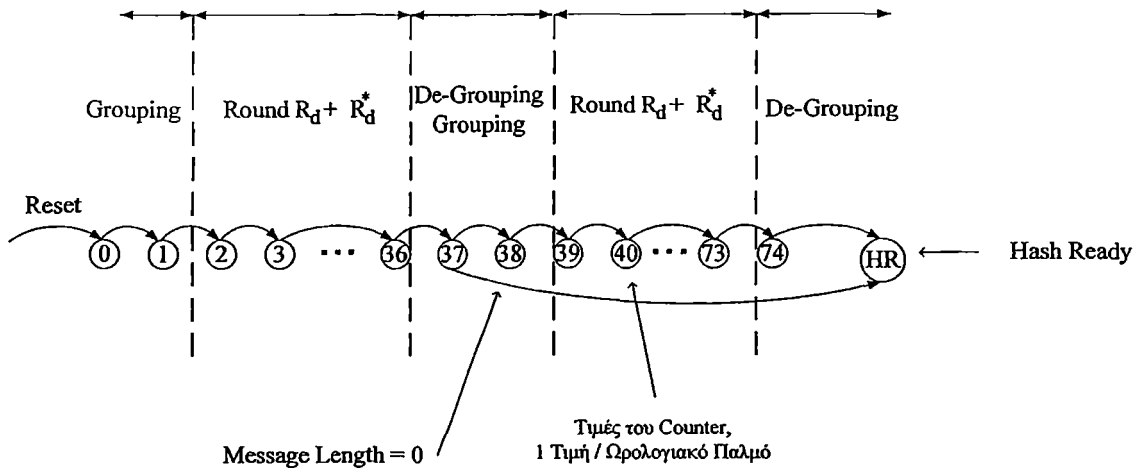
το κύκλωμα JH του Σχήματος 6.26 ονομάζεται Controller. Αυτό το κύκλωμα μπορεί να θεωρηθεί ως μια μηχανή πεπερασμένων καταστάσεων (Finite State Machine - FSM) η οποία αποτελείται από 74 καταστάσεις στη περίπτωση του κυκλώματος JH. Ο έλεγχος των επιμέρους κυκλωμάτων γίνεται μέσω των 5 σημάτων του Controller τα οποία μεταβάλλονται κατάλληλα ανάλογα τη λειτουργία που πρέπει να πραγματοποιηθεί σε κάθε κατάσταση.

Για το καθορισμό των 74 καταστάσεων χρησιμοποιείται ένας μετρητής (Counter) ο οποίος και δίνεται από το Σχήμα 6.35 . Το σήμα Message Circles διαβιβάζεται από το κύκλωμα του Padder στο κύκλωμα του Controller και στη συνέχεια στο κύκλωμα του Counter ούτος ώστε εάν στο κύκλωμα έχει εισαχθεί μήνυμα μήκους 0-bit όπου και απαιτούνται μόνο 37 κύκλοι ο Counter να σταματάει τη μέτρηση του σε αυτόν τον αριθμό αλλιώς να συνεχίζει στον αριθμό 74 που είναι οι κύκλοι που απαιτούνται για οποιοδήποτε άλλο μέγεθος εισερχόμενου μηνύματος.

Στο Σχήμα 6.36 δίνεται σχηματικά η περιγραφή του Controller. Οι τιμές των καταστάσεων προκύπτουν από τον μετρητή ο οποίος αυξάνει σε κάθε παλμό του ρολογιού.



Σχήμα 6.35: *Ο μετρητής του κυκλώματος JH.*



Σχήμα 6.36: *Σχηματική περιγραφή της λειτουργίας του κυκλώματος ελέγχου.*

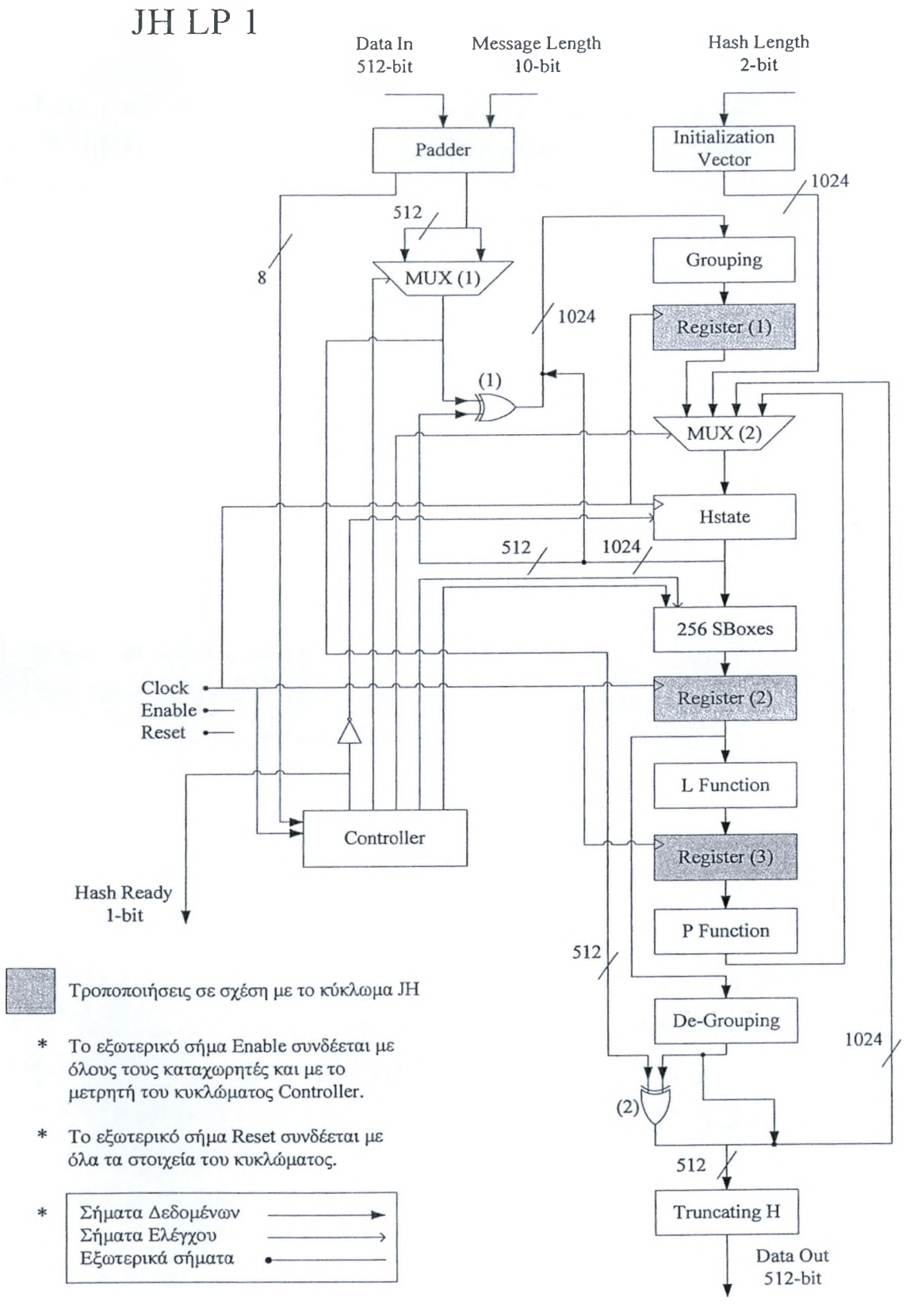
6.4 Οι παραλλαγές υλοποίησης της βασικής αρχιτεκτονικής της JH

Όπως αναφέρθηκε και στο κεφάλαιο 6.3 σχεδιάστηκαν 9 παραλλαγές του βασικού κυκλώματος JH που σκοπό έχουν τη μείωση της δυναμικής κατανάλωσης. Αυτά τα κυκλώματα θα αναλυθούν παρακάτω παρουσιάζοντας τις διαφορές από τη βασική αρχιτεκτονική που παρουσιάστηκε στο προηγούμενο κεφάλαιο.

JH LP 1

Όπως και στις υλοποιήσεις χαμηλής κατανάλωσης του κυκλώματος Fugue έτσι και εδώ η κάθε διαφορετική υλοποίηση του κυκλώματος JH συμβολίζεται με τα αρχικά LP (Low Power) και τον αριθμό που υποδηλώνει τις διάφορες υλοποιήσεις.

Στο κύκλωμα JH LP 1 χρησιμοποιήθηκε η τεχνική pipeling. Η μεθοδολογία που χρησιμοποιήθηκε για την υλοποίηση της είναι η ίδια που ακολουθήθηκε και στο αντίστοιχο κύκλωμα Fugue Clk Crcls 49 LP 1. Στο Σχήμα 6.37 δίνεται η αρχιτεκτονική του κυκλώματος JH LP 1.



Σχήμα 6.37: *Η αρχιτεκτονική του κυκλώματος JH LP 1.*

Στο παραπάνω κύκλωμα σε σχέση με το κύκλωμα του Σχήματος 6.29 έχουν προστεθεί 3 θετικά ακμοπυροδότητοι καταχωρητές. Ο πρώτος καταχωρητής Register (1) έχει τοποθετηθεί μετά το κύκλωμα Grouping, ο δεύτερος καταχωρητής Register (2) έχει τοποθετηθεί μετά το κύκλωμα 256 SBoxes και τέλος ο τρίτος καταχωρητής έχει τοποθετηθεί μετά το κύκλωμα L Function. Στην την αρχιτεκτονική του κυκλώματος JH LP 1 υπάρχουν καταχωρητές σε όλα τα επίπεδα της συναρτήσεως E που περιγράφηκε στη παράγραφο 5.3 «Η συνάρτηση E». Ως αποτέλεσμα τη δυνατότητα ελέγχου όλου του κυκλώματος αλλά και του περιορισμού των ανεπιθύμητων μεταβολών των σημάτων σε όλα τα επίπεδα του αλγορίθμου.

Η χρήση αυτών των 3 επιπλέον καταχωρητών συνεπάγεται την αύξηση των κύκλων επεξεργασίας όπως συνέβαινε και στα αντίστοιχα κυκλώματα χαμηλής κατανάλωσης του αλγορίθμου Fugue. Για την ολοκλήρωση ενός γύρου R_d χρειάζονται δύο επιπλέον κύκλοι ρολογιού καθώς μεσολαβούν δύο επιπλέον καταχωρητές οι οποίοι είναι ο Register (2) και Register (3). Αυτό αυξάνει τους κύκλους από 36 σε 108 για την ολοκλήρωση ενός γύρου E_d . Ο μέγιστος αριθμός εκτέλεσης του γύρου E_d είναι 2 φορές οπότε ο συνολικός αριθμός των κύκλων ρολογιού που απαιτούνται συμπεριλαμβανομένων και των κύκλων που απαιτούνται για την έξοδο του κυκλώματος Grouping είναι 218 από 79 που είναι στο κύκλωμα JH. Οι επιπλέον κύκλοι ρολογιού που απαιτούνται για την έξοδο του κυκλώματος Grouping οφείλονται στην ύπαρξη του καταχωρητή Register (1). Τέλος, καθώς αλλάζουν οι κύκλοι ρολογιού αλλάζουν και οι προκαθορισμένες τιμές του σήματος message circle του Σχήματος 6.30 οι οποίες γίνονται 109 και 218 αντίστοιχα.

JH LP 1 V2

Η υλοποίηση JH LP 1 V2 είναι ίδια με την υλοποίηση JH LP 1 με τη μόνη διαφορά τη χρήση της τεχνικής Double Edge Trigger (DET) pipeling η οποία αναφέρθηκε στο κεφάλαιο 5.2 και χρησιμοποιήθηκε στην υλοποίηση του κυκλώματος Fugue 256 Clk Crcls LP 2 V2.

Η χρήση αυτής της τεχνικής στο κύκλωμα JH LP 1 διαφέρει από τη χρήση που έγινε στο κύκλωμα Fugue 256 Clk Crcls LP 2 V2. Εδώ λόγω της μη ύπαρξης αρκετά συνεχόμενης συνδυαστικής λογικής στα επιμέρους κυκλώματα δεν προστέθηκε ένας επιπλέον αρνητικά ακμοπυροδότητος καταχωρητής αλλά ο καταχωρητής Register (3) μετατράπηκε από θετικά ακμοπυροδότητος σε αρνητικά ακμοπυροδότητος. Ο σκοπός ήταν η μείωση των κύκλων επεξεργασίας που απαιτούνται από το κύκλωμα JH LP 1 V2 σε σχέση με το κύκλωμα JH LP 1 με στόχο την μείωση της δυναμικής κατανάλωσης καθώς θα απαιτούνται λιγότεροι κύκλοι..

Με τη μετατροπή του καταχωρητή Register (3) οι κύκλοι ρολογιού που απαιτούνται από το κύκλωμα JH LP 1 V2 μειώθηκαν κατά 70 σε σχέση με το κύκλωμα JH LP 1 φτάνοντας τους συνολικά 148 κύκλους. Ενώ οι προκαθορισμένες τιμές του σήματος message circle γίνονται 74 και 148. Ουσιαστικά το κύκλωμα JH LP 1 V2 διατηρεί τα πλεονεκτήματα της χρήσης καταχωρητών μειώνοντας όμως του κύκλους ρολογιού που απαιτούνται στο κύκλωμα.

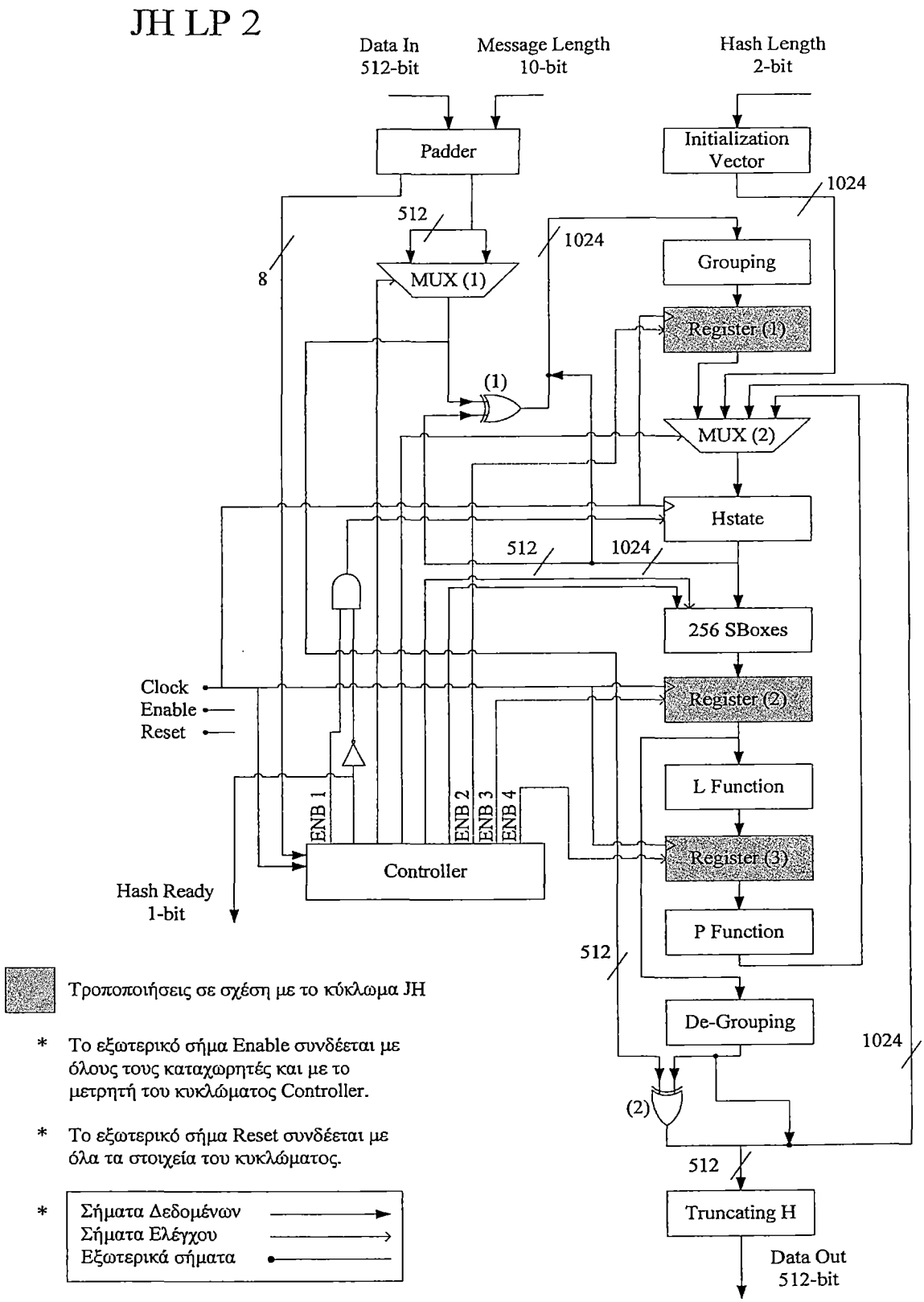
JH LP 2

Το κύκλωμα JH LP 2 είναι ίδιο με την υλοποίηση του κυκλώματος JH LP 1 με τη διαφορά ότι εδώ χρησιμοποιείται η τεχνική του Gated Clock όπως και έγινε και στα αντίστοιχα κυκλώματα του αλγορίθμου Fugue. Το Gated Clock που εφαρμόστηκε στο κύκλωμα JH LP 2 χρησιμοποιεί την ενσωματωμένη λειτουργία Enable των flip-flop του FPGA. Η τεχνική αυτή δίνεται επίσης και στο Σχήμα 4.4 .

Στο Σχήμα 6.38 δίνεται η αρχιτεκτονική του κυκλώματος JH LP 2. Τη λειτουργία των καταχωρητών ελέγχει του κύκλωμα του Controller μέσω των σημάτων ENB 1, ENB 2, ENB 3 και ENB 4. Οι κύκλοι επεξεργασίας που απαιτούνται στο κύκλωμα JH LP 2 είναι ίδιοι με αυτούς του κυκλώματος JH LP 1 συν έναν επιπλέον κύκλο που χρειάζεται κατά το στάδιο της αρχικοποίησης. Ο επιπλέον κύκλος χρειάζεται καθώς κατά τη διάρκεια της αρχικοποίησης στο κύκλωμα JH LP 2 οι καταχωρητές δεν λειτουργούν αμέσως όταν τα εξωτερικά σήματα reset και enable το επιτρέψουν αλλά όταν τα σήματα ελέγχου του Controller φτάσουν στους καταχωρητές. Συνολικά λοιπόν στο κύκλωμα JH LP 2 απαιτούνται 219 κύκλοι ρολογιού ενώ οι προκαθορισμένες τιμές του σήματος message circle γίνονται 110 και 219.

JH LP 3 και JH LP 4

Τα κυκλώματα JH LP 3 και JH LP 4 έχουν την ίδια αρχιτεκτονική με το κύκλωμα JH LP 2 μόνο που στο κύκλωμα JH LP 3 χρησιμοποιείται Gating Clock με τη χρήση πύλης AND όπως φαίνεται στο Σχήμα 4.3 ενώ στο LP 4 χρησιμοποιείται η μέθοδος του Feedback MUX του Σχήματος 4.4 .Οι αντίστοιχοι κύκλοι επεξεργασίας είναι 147 και 219.



Σχήμα 6.38: *Η αρχιτεκτονική του κυκλώματος JH LP 2.*

JH (LP 5, LP 6, LP7 και LP 8)

Το κύκλωμα JH LP 5 έχει ίδια αρχιτεκτονική με το κύκλωμα JH μόνο που για τη μείωση της δυναμικής κατανάλωσης χρησιμοποιεί την ενσωματωμένη μνήμη RAM που παρέχεται στα FPGAs για την υλοποίηση των 256 SBoxes και των πινάκων αρχικοποίησης (Initialization Vector). Οι κύκλοι επεξεργασίας του JH LP 5 λόγω της ύπαρξης της μνήμης στο επίπεδο της συναρτήσεως R_d διπλασιάζονται και γίνονται 72. Συνολικά συμπεριλαμβανομένου και των κύκλων αρχικοποίησης απαιτούνται 146 κύκλοι ρολογιού ενώ οι προκαθορισμένες τιμές του σήματος message circle γίνονται 73 και 146.

Το κύκλωμα JH LP 6 είναι ο συνδυασμός των κυκλωμάτων JH LP 2 και LP 5. Δηλαδή για την υλοποίηση των SBoxes και των πινάκων αρχικοποίησης χρησιμοποιείται μνήμη RAM καθώς επίσης χρησιμοποιούνται όλες οι τεχνικές χαμηλής κατανάλωσης του κυκλώματος LP 2 με τη μόνη διαφορά ότι δε χρησιμοποιείται ο καταχωρητής Register (2) του Σχήματος 6.38 . Ο λόγος που αφαιρείται ο καταχωρητής μετά το κύκλωμα 256 SBoxes είναι η ύπαρξη της μνήμης στο κύκλωμα 256 SBoxes κάτι που εγγυάται τον συγχρονισμό των δεδομένων στην έξοδο του κυκλώματος αλλά και τη μη ύπαρξη τυχαίων μεταβολών. Όπως επίσης το σήμα Enable που οδηγείται στο καταχωρητή Register (2) του Σχήματος 6.38 πλέον οδηγείται στο κύκλωμα 256 SBoxes για τον έλεγχο της μνήμης που χρησιμοποιείται για την υλοποίηση των SBoxes. Οι κύκλοι επεξεργασίας και οι προκαθορισμένες τιμές του σήματος message circle του κυκλώματος JH LP 6 προκύπτουν όσοι και του κυκλώματος JH LP 2.

Τα κυκλώματα JH LP 7 και JH LP 8 είναι ίδια με το κύκλωμα JH LP 6 με τη διαφορά ότι στο κύκλωμα JH LP 7 γίνεται χρήση της μεθόδου Gated Clock με τη χρήση της πύλης AND ενώ στο κύκλωμα JH LP 8 γίνεται χρήση της μεθόδου Gated Clock με τη χρήση του πολυπλέκτη ανάδρασης. Οι κύκλοι επεξεργασίας που προκύπτουν είναι 218 και 219 για τα κυκλώματα JH LP 7 και JH LP 8 ενώ οι προκαθορισμένες τιμές του σήματος message circle είναι 109 και 218 για το κύκλωμα JH LP 7 ενώ για το κύκλωμα JH LP 8 είναι 110 και 219.

ΑΝΑΦΟΡΕΣ

[1] Edwin NC Mui Custom R & D Engineer, Practical Implementation of Rijndael S-Box Using Combinational Logic.

ΚΕΦΑΛΑΙΟ 7

ΜΕΤΡΗΣΕΙΣ – ΣΥΣΓΚΡΙΣΕΙΣ

7.1 Εισαγωγή

Στο παρόν κεφάλαιο θα παρουσιαστούν τα αποτελέσματα των μετρήσεων ως προς τους πόρους υλικού, την απόδοση ως προς το χρόνο εκτέλεσης και τη κατανάλωση ισχύος όλων των αρχιτεκτονικών του κεφαλαίου 6. Επίσης θα παρουσιαστούν και αποτελέσματα εξομοίωσης των αρχιτεκτονικών για τη πιστοποίηση της ορθής λειτουργίας της κάθε αρχιτεκτονικής. Τέλος, θα γίνουν συγκρίσεις μεταξύ των αποτελεσμάτων των άλλων αρχιτεκτονικών.

Για τις διαδικασίες σύνθεσης και υλοποίησης χρησιμοποιήθηκε το λογισμικό Xilinx ISE 12.3 . Για την χρονική εξομοίωση και την εξομοίωση της κατανάλωσης ισχύος των κυκλωμάτων χρησιμοποιήθηκε το λογισμικό Modelsim XE III 6.2c . Τέλος, χρησιμοποιήθηκε το λογισμικό Xilinx 12.3 XPower Analyzer για την μέτρηση της κατανάλωσης ισχύος του εκάστοτε κυκλώματος.

Στα πλαίσια των μετρήσεων τα μεγέθη που θα παρουσιαστούν είναι: ο αριθμός λογικών τμημάτων (Logical Slices ή Slices) που απαιτούνται για τη κάθε αρχιτεκτονική, ο αριθμός των Flip-Flops, ο αριθμός των κύκλων επεξεργασίας, η μικρότερη περίοδος ρολογιού που μπορεί να χρησιμοποιηθεί από τη κάθε αρχιτεκτονική και η διεκπαιρεωτική ικανότητα (Throughput) του κάθε κυκλώματος. Ο υπολογισμός του Throughput δίνεται από τη παρακάτω σχέση:

(7.1)

$$\text{Throughput} = \frac{\#\text{bits δεδομένων εξόδου}}{\text{Clock Circles}} \times \text{Operational Frequency (Hz)}$$

Ο αριθμητής του κλάσματος συμβολίζει το μέγεθος της εξόδου κάθε κυκλώματος και μετριέται σε bit. Ο παρανομαστής συμβολίζει του κύκλους επεξεργασίας που απαιτούνται για τον υπολογισμό της σύνοψης κάθε μηνύματος. Με τον όρο Frequency εννοείται η μέγιστη συχνότητα ρολογιού που επιτυγχάνει το κάθε κύκλωμα. Η διεκπαιρεωτική ικανότητα κάθε

λογικού τμήματος δίνεται από το λόγο του Throughput και τον αριθμό των λογικών τμημάτων που απαιτούνται σε κάθε κύκλωμα.

Επίσης θα παρουσιαστούν η συνολική κατανάλωση ισχύος κάθε κυκλώματος μαζί με τις δύο συνιστώσες της, την δυναμική (Dynamic) και στατική (Static) κατανάλωση.

Το FPGA που χρησιμοποιήθηκε για όλες τις παραπάνω μετρήσεις είναι το Virtex5 XC5VLX330-2FF1760 που παρουσιάστηκε στο κεφάλαιο 3.5 . Για την ορθή σύγκριση των αποτελεσμάτων πρέπει το ολοκληρωμένο κύκλωμα που χρησιμοποιείται να είναι το ίδιο για όλες τις αρχιτεκτονικές. Για αυτό το λόγο χρησιμοποιήθηκε το παραπάνω FPGA το οποίο έχει το κατάλληλο μέγεθος για να δεχθεί όλες τις αρχιτεκτονικές.

Όπως έχει ήδη αναφερθεί, θα παρουσιαστούν αποτελέσματα εξομοίωσης που θα πιστοποιούν την ορθή λειτουργία των κυκλωμάτων. Για το σκοπό αυτό μαζί με τα αποτελέσματα κάθε αρχιτεκτονικής θα δίνεται και ένα σχήμα το οποίο θα περιέχει το διάγραμμα χρονισμού που προέκυψε από την εξομοίωση. Τα δεδομένα εισόδου που χρησιμοποιούνται διαφέρουν σε μέγεθος μηνύματος και προέρχονται από τα επίσημα αρχεία που κατατέθηκαν από τους σχεδιαστές των αλγορίθμων στο διαγωνισμό NIST τα οποία περιέχουν δεδομένα δοκιμών (Test Vectors). Τα δεδομένα των Test Vector που χρησιμοποιήθηκαν στις εξομοιώσεις δίνονται στο Παράρτημα Γ.

Τα σήματα που χρησιμοποιούνται στα διαγράμματα χρονισμού είναι έξι στις υλοποιήσεις του αλγορίθμου Fugue και επτά στις υλοποιήσεις του αλγορίθμου JH τα οποία και χωρίζονται σε σήματα εισόδου δεδομένων, σε σήματα εξόδου δεδομένων και σε σήματα χρονισμού των κυκλωμάτων.

Θεωρητικά ο χρόνος εξομοίωσης που απαιτείται για την ανάλυσης της συμπεριφοράς ενός κυκλώματος όσον αφορά τη κατανάλωση ισχύος πρέπει είναι όσον το δυνατόν μεγαλύτερος. Ο χρόνος αυτός περιορίζεται από την πραγματική χρονική διάρκεια που απαιτείται για την ολοκλήρωση κάθε εξομοίωσης αλλά και από το μέγεθος των παραγόμενων δεδομένων. Στα πλαίσια εύρεσης του μέγιστου χρόνου εξομοίωσης σε σχέση με τους παραπάνω περιορισμούς ο χρόνος που επιλέχτηκε είναι τα 400μS . Ο χρόνος αυτός είναι πολύ μεγαλύτερος από τη περίοδο λειτουργίας των κυκλωμάτων (πάνω από μια τάξη μεγέθους) και είναι ικανοποιητικός χρόνος για την ανάλυση της κατανάλωσης ισχύος καθώς κατά τη διάρκεια αυτή συμβαίνει ένας αρκετά μεγάλος αριθμός μεταβάσεων στα κυκλώματα. Για τη σύγκριση των αποτελεσμάτων όλων των αρχιτεκτονικών ο παραπάνω χρόνος είναι κοινός για όλα τις εξομοιώσεις.

7.2 Αποτελέσματα μετρήσεων των αρχιτεκτονικών Fugue

Fugue 256-bit Clock Circles 49

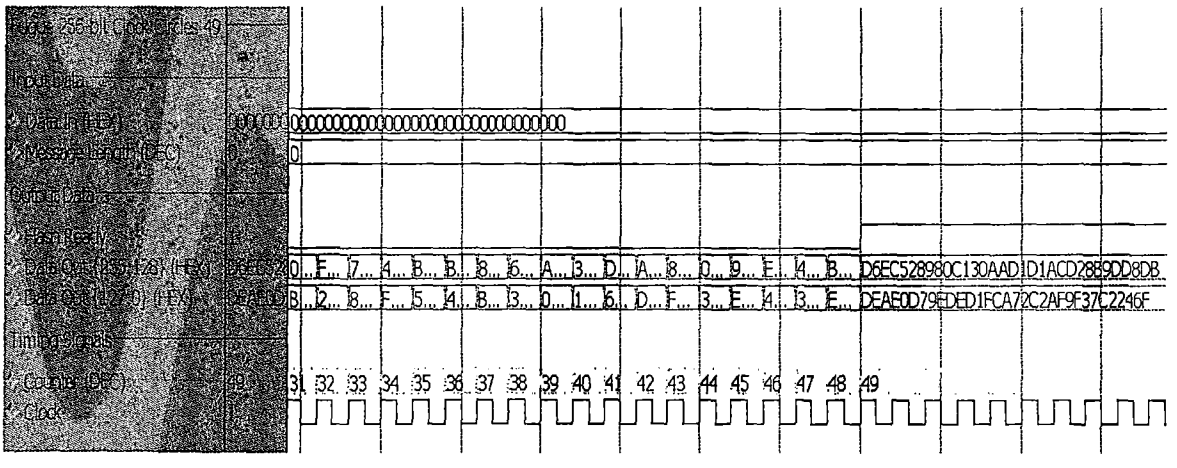
Στο Πίνακα 7.1 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clock Circles 49. το οποίο και αποτελεί τη κύκλωμα αναφορά για τις υπόλοιπες υλοποιήσεις του αλγορίθμου Fugue. Στο Σχήμα 7.1 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 0-bit .

Πίνακας 7.1: *Μετρήσεις του κυκλώματος Fugue 256-bit Clock Circles 49.*

Design Clock Circles 49	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
	1,394	969	49	9.693	514.01	377.579

Power Supply (mW)		
Dynamic	Static	Total
1,182.02	3,264.10	4,446.12

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 10 nS



Σχήμα 7.1: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clock Circles 49.*

Fugue 256-bit Clock Circles 36

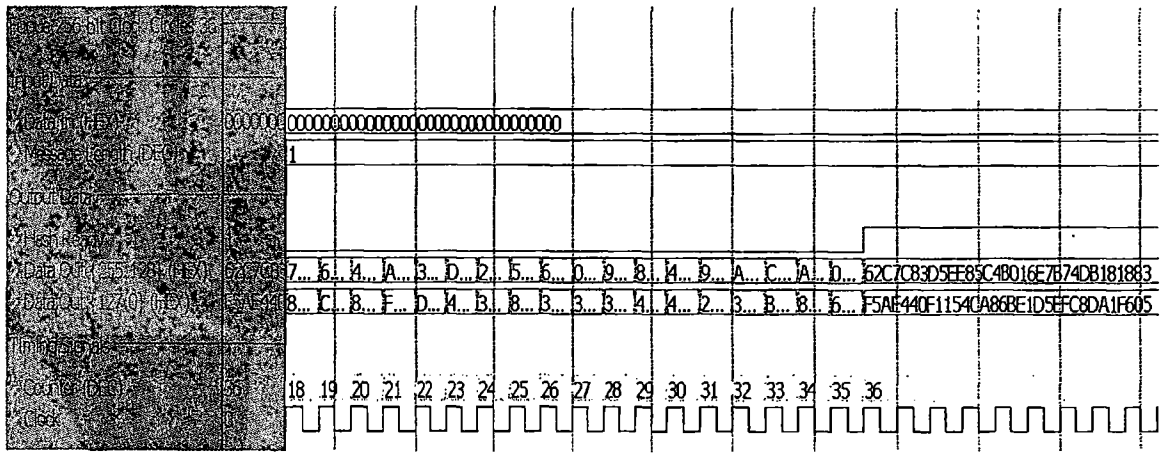
Στο Πίνακα 7.2 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clock Circles 36. Στο Σχήμα 7.2 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 1-bit και εισερχόμενο μήνυμα «0» .

Πίνακας 7.2: *Μετρήσεις του κυκλώματος Fugue 256-bit Clock Circles 36.*

Design / Clock Circles	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
Circles 36	1,894	972	36	14.108	480.692	259.888

Power Supply (mW)		
Dynamic	Static	Total
1,086.61	3,264.10	4,350.71

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 15 nS



Σχήμα 7.2: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clock Circles 36.*

Fugue 256-bit Clock Circles 25

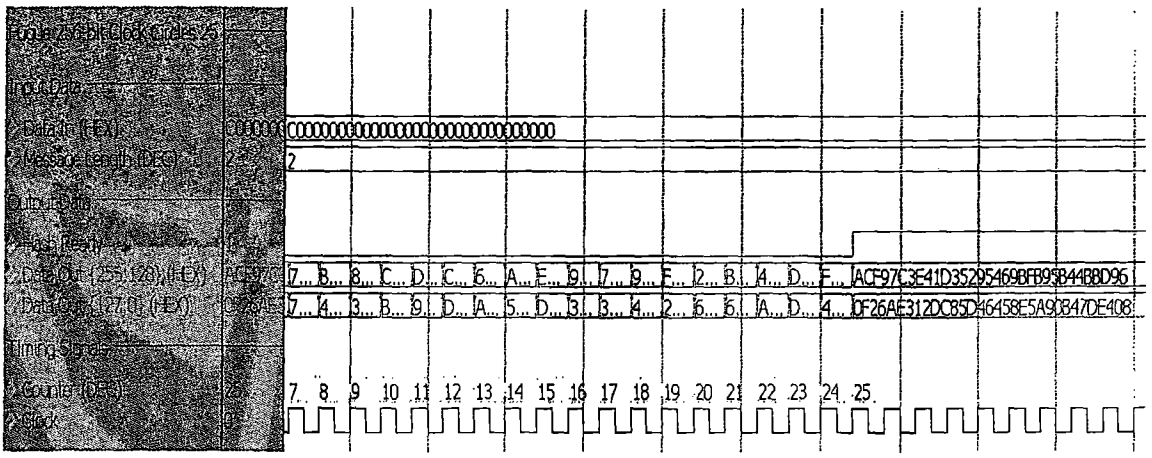
Στο Πίνακα 7.3 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clock Circles 36. Στο Σχήμα 7.3 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 2-bit και εισερχόμενο μήνυμα «0xC0» .

Πίνακας 7.3: *Μετρήσεις του κυκλώματος Fugue 256-bit Clock Circles 25.*

Design Clock Circles 25	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
	2,433	966	25	15.888	614.667	258.7

Power Supply (mW)		
Dynamic	Static	Total
774.68	3,294.10	4,068.78

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 16 nS



Σχήμα 7.3: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clock Circles 25.*

Fugue 256-bit Clk Crcls 49 LP 1

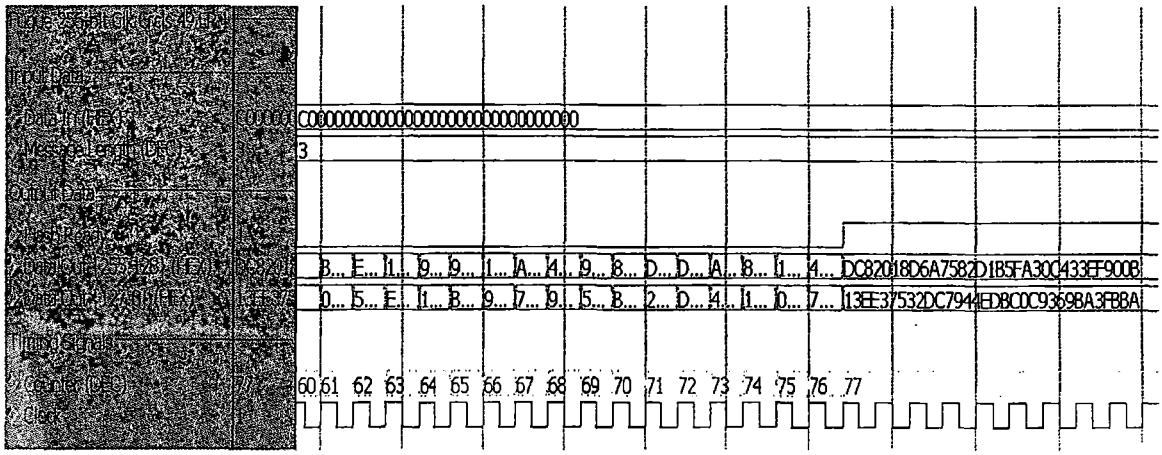
Στο Πίνακα 7.4 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 1. Στο Σχήμα 7.4 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 3-bit και εισερχόμενο μήνυμα «0xC0» .

Πίνακας 7.4: *Μετρήσεις του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 1.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
LP 1	1,686	2,889	77	8.735	362.967	220.449

Power Supply (mW)		
Dynamic	Static	Total
1,108.25	3,264.10	4,372.35

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 10 nS



Σχήμα 7.4: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 1 .*

Fugue 256-bit Clk Crcls 49 LP 2

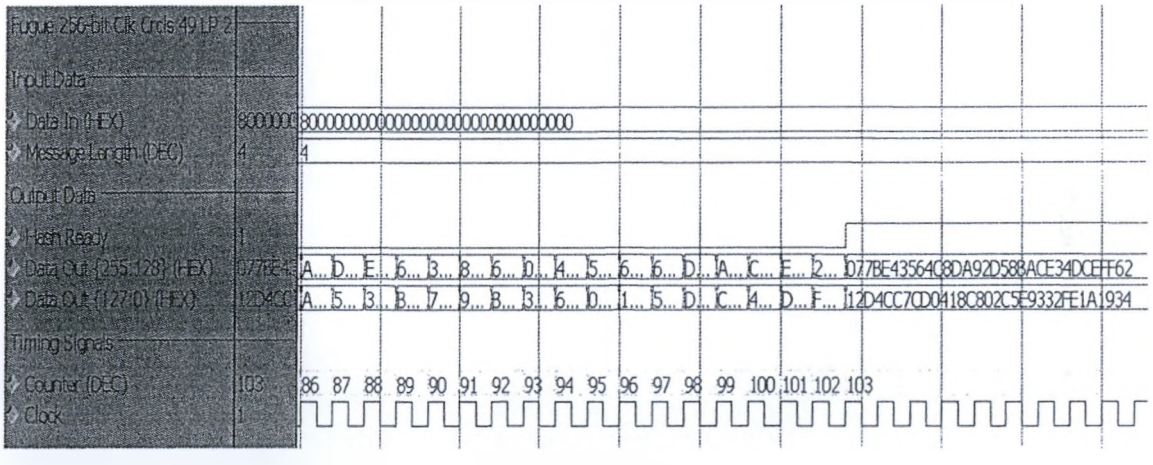
Στο Πίνακα 7.5 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 2. Στο Σχήμα 7.5 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 4-bit και εισερχόμενο μήνυμα «0x80» .

Πίνακας 7.5: *Μετρήσεις του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 2.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
LP 2	1,821	3,369	103	8.843	268.042	150.727

Power Supply (mW)		
Dynamic	Static	Total
1,241.76	3,264.10	4,505.86

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 10 nS



Σχήμα 7.5: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 2.*

Fugue 256-bit Clk Crcls 49 LP 2 V2

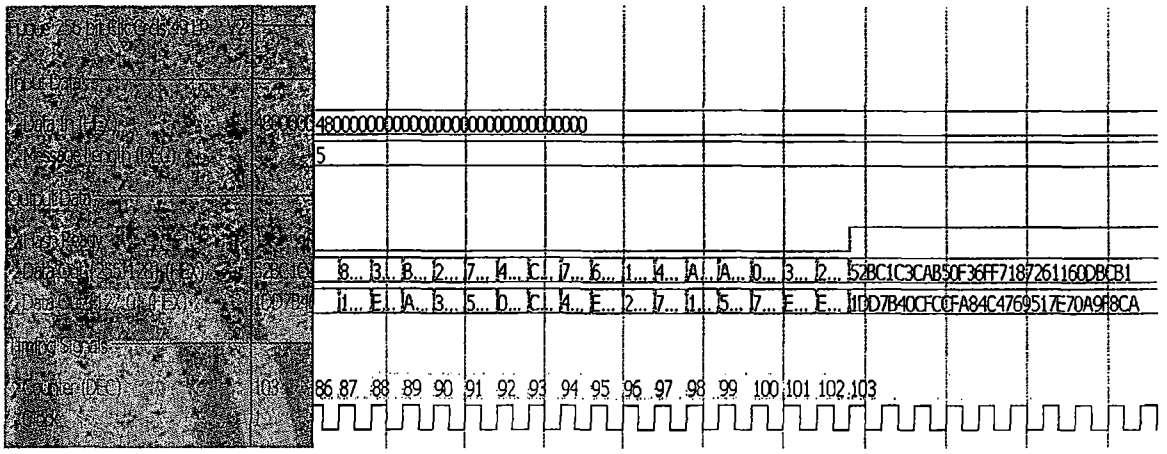
Στο Πίνακα 7.6 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 2 V2. Στο Σχήμα 7.6 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 5-bit και εισερχόμενο μήνυμα «0x48» . Η μέτρηση της κατανάλωσης θα γίνει με διαφορετική περίοδο από τα άλλα κυκλώματα καθώς η περίοδος του κυκλώματος είναι μεγαλύτερη από τη περίοδο του κυκλώματος αναφοράς. Για το λόγο αυτό η χρησιμοποιούμενη περίοδος είναι 13 nS η οποία είναι η πλησιέστερη στη περίοδο του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 2 V2.

Πίνακας 7.6: *Μετρήσεις του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 2 V2.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
LP 2 V2	2,018	4,330	103	12.710	186.492	94.632

Power Supply (mW)		
Dynamic	Static	Total
976.35	3,264.10	4,240.45

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 13 nS



Σχήμα 7.6: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 2 V2.*

Fugue 256-bit Clk Crcls 49 LP 3

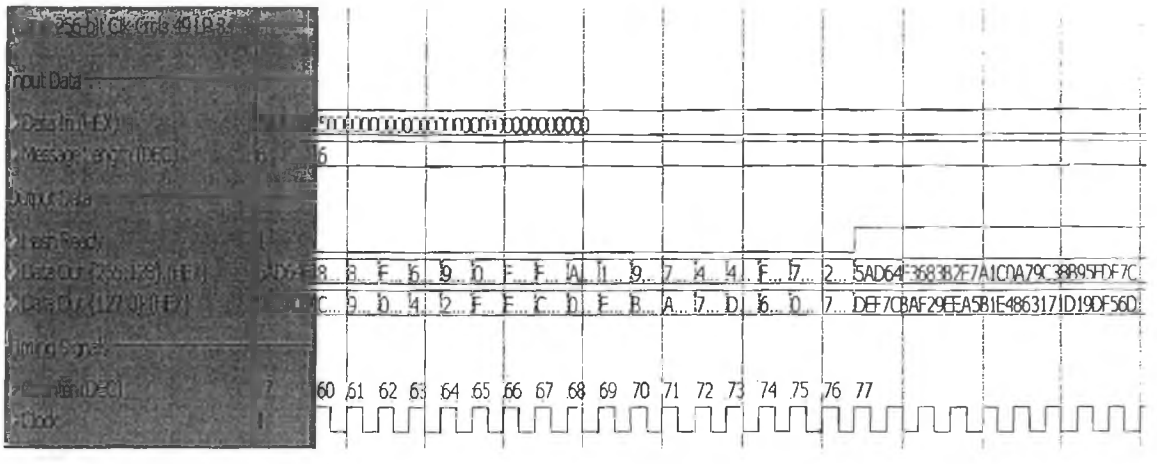
Στο Πίνακα 7.7 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 3. Στο Σχήμα 7.7 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 6-bit και εισερχόμενο μήνυμα «0x50» .

Πίνακας 7.7: *Μετρήσεις του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 3.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
LP 3	1,723	2,889	77	8.759	362,006	215.142

Power Supply (mW)		
Dynamic	Static	Total
879.74	3,264.10	4,143.84

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 10 nS



Σχήμα 7.7: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 3.*

Fugue 256-bit Clk Crcls 49 LP 4

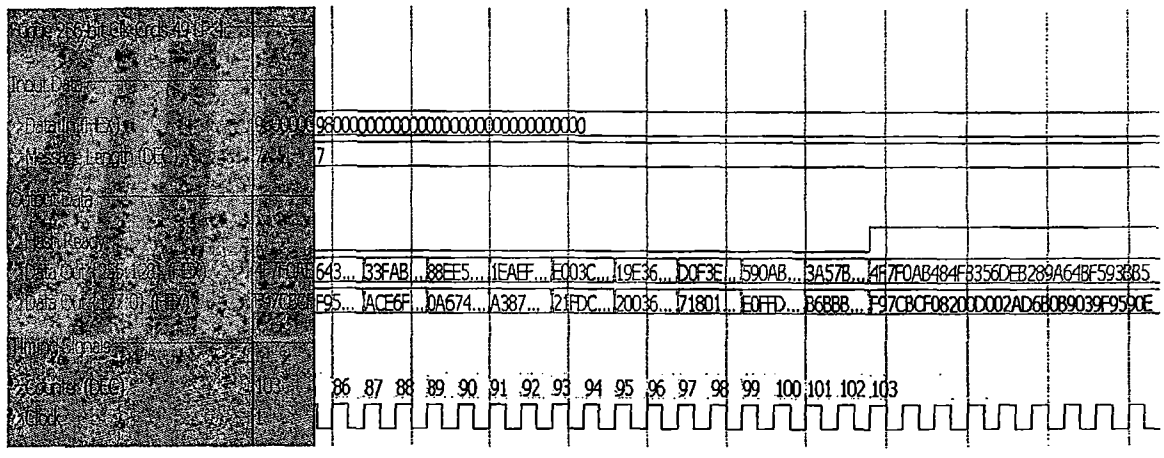
Στο Πίνακα 7.8 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 4. Στο Σχήμα 7.8 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 7-bit και εισερχόμενο μήνυμα «0x98» .

Πίνακας 7.8: *Μετρήσεις του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 4.*

Design LP4	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
	2,374	4,809	103	8.853	267.732	115.483

Power Supply (mW)		
Dynamic	Static	Total
831,47	3,264.10	4,095,57

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 10 nS



Σχήμα 7.8: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 4.*

Fugue 256-bit Clk Crcls 49 LP 5

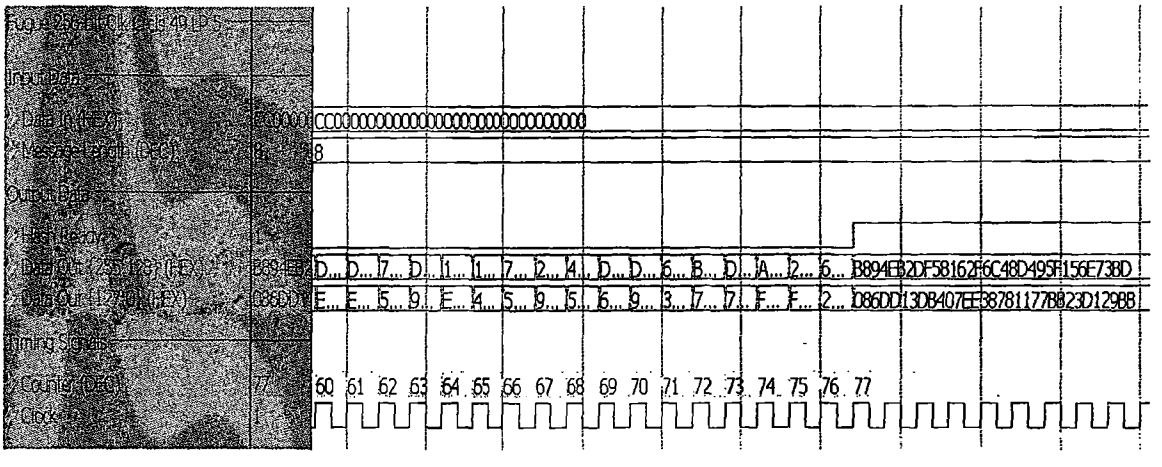
Στο Πίνακα 7.9 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 5. Στο Σχήμα 7.9 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 8-bit και εισερχόμενο μήνυμα «0xCC» .

Πίνακας 7.9: *Μετρήσεις του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 5.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
LP 5	1,997	4,809	77	2.196	1,443.964	740.420

Power Supply (mW)		
Dynamic	Static	Total
868,29	3,264.10	4,132.39

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 10 nS



Σχήμα 7.9: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 5.*

Fugue 256-bit Clk Crcls 49 LP 6

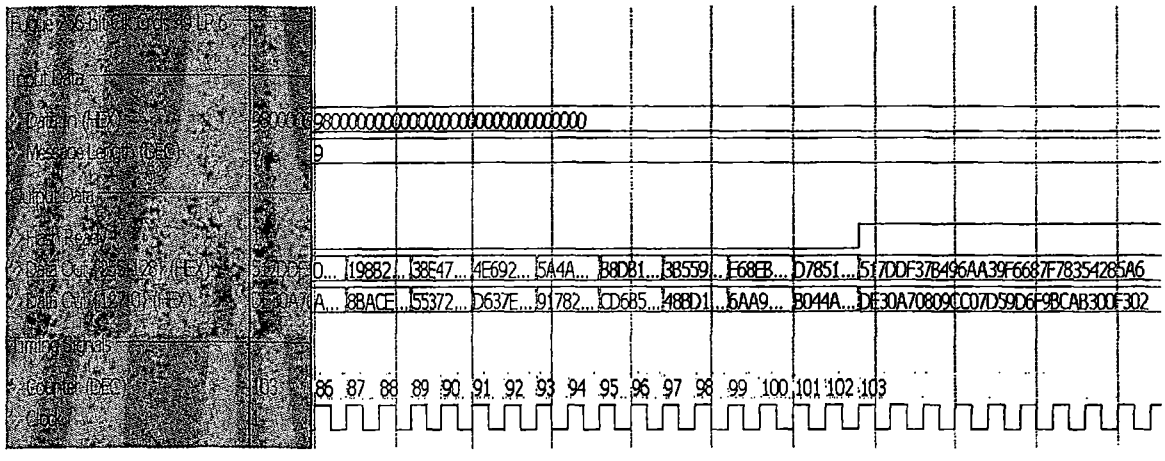
Στο Πίνακα 7.10 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 6. Στο Σχήμα 7.10 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 9-bit και εισερχόμενο μήνυμα «0x9800» .

Πίνακας 7.10: *Μετρήσεις του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 6.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
LP 6	2,593	4,815	103	8.814	268.933	106.204

Power Supply (mW)		
Dynamic	Static	Total
772.84	3,264.10	4,036.94

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 10 nS



Σχήμα 7.10: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 6.*

Fugue 256-bit Clk Crcls 49 LP 7

Στο Πίνακα 7.11 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 7. Οι δύο νέες στήλες που εμφανίζονται στο πίνακα δείχνουν την χρησιμοποιούμενη μνήμη καθώς στις υλοποιήσεις LP 7 έως LP 10 η υλοποίηση των SBoxes γίνεται με τη χρήση της ενσωματωμένης μνήμης. Ως BRAM συμβολίζονται τα τμήματα μνήμης των 18 Kbit το καθένα που παρέχονται στο FPGA. Στο Σχήμα 7.11 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 32-bit και εισερχόμενο μήνυμα «0xC1ECFDFC» .

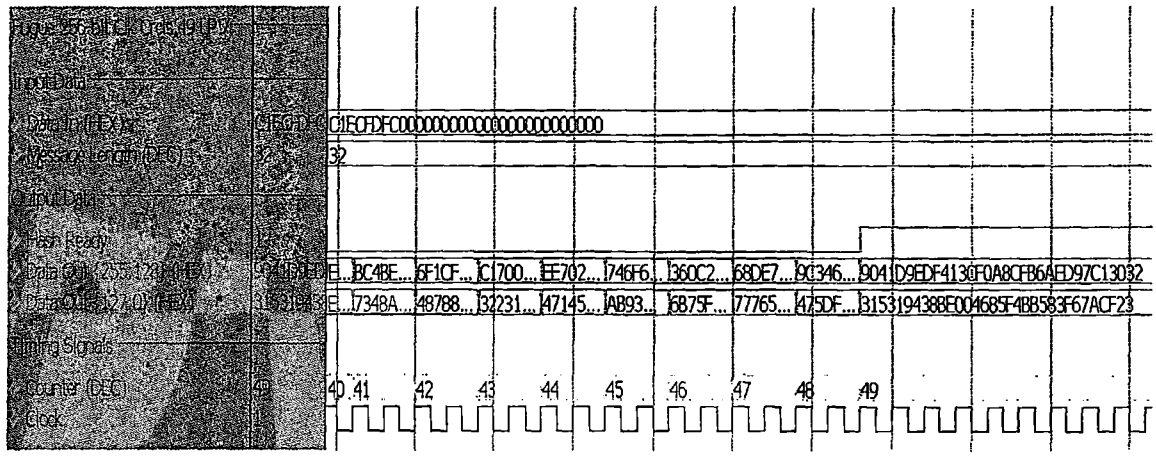
Πίνακας 7.11: *Μετρήσεις του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 7.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
LP 7	1,093	1,794	98	3.606	690.885	647.270

Memory	
BRAM	Memory Used (Kb)
8	144

Power Supply (mW)		
Dynamic	Static	Total
589.49	3,264.10	3,853.59

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 10 nS



Σχήμα 7.11: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 7.*

Fugue 256-bit Clk Crcls 49 LP 8

Στο Πίνακα 7.12 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 8. Στο Σχήμα 7.12 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 64-bit και εισερχόμενο μήνυμα «0x4A4F202484512526» .

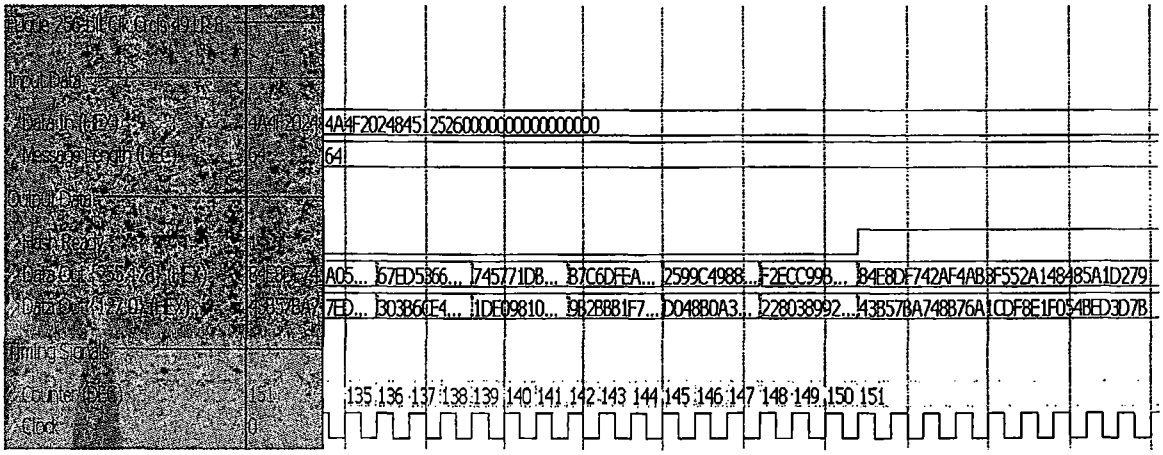
Πίνακας 7.12: *Μετρήσεις του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 8.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
LP 8	1,868	5,640	151	3.862	418.646	229.493

Memory	
BRAM	Memory Used (Kb)
8	144

Power Supply (mW)		
Dynamic	Static	Total
479.73	3,264.10	3,743.83

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 10 nS



Σχήμα 7.12: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 8.*

Fugue 256-bit Clk Crcls 49 LP 9

Στο Πίνακα 7.13 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 9. Στο Σχήμα 7.13 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 96-bit και εισερχόμενο μήνυμα «0x5BE43C90F22902E4FE8ED2D3» .

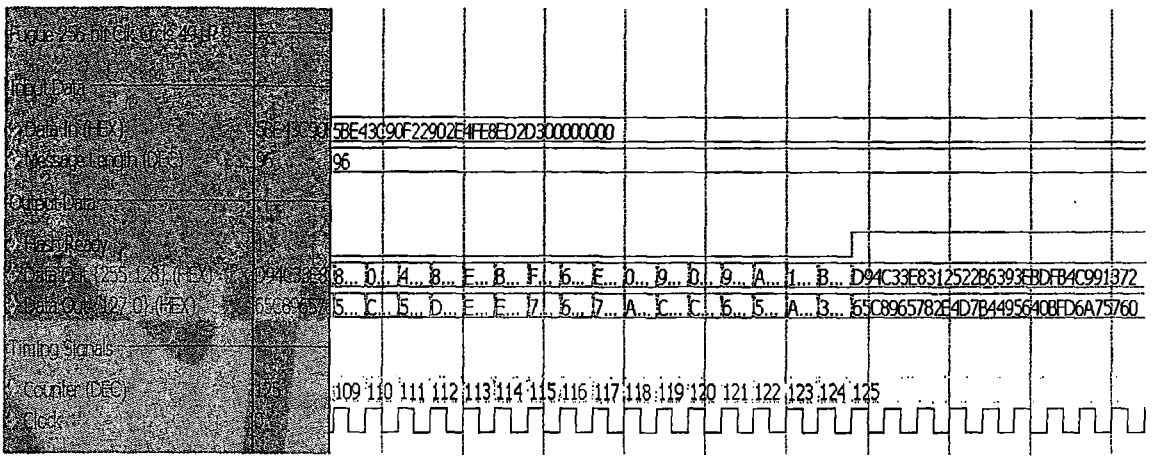
Πίνακας 7.13: *Μετρήσεις του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 9.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
LP 9	2,223	5,639	125	3.399	574.675	264.717

Memory	
BRAM	Memory Used (Kb)
8	144

Power Supply (mW)		
Dynamic	Static	Total
711.35	3,264.10	3,975.46

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 10 nS



Σχήμα 7.13: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 9.*

Fugue 256-bit Clk Crcls 49 LP 10

Στο Πίνακα 7.14 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 10. Στο Σχήμα 7.14 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 128-bit και εισερχόμενο μήνυμα «0x52A608AB21CCDD8A4457A57EDE782176» .

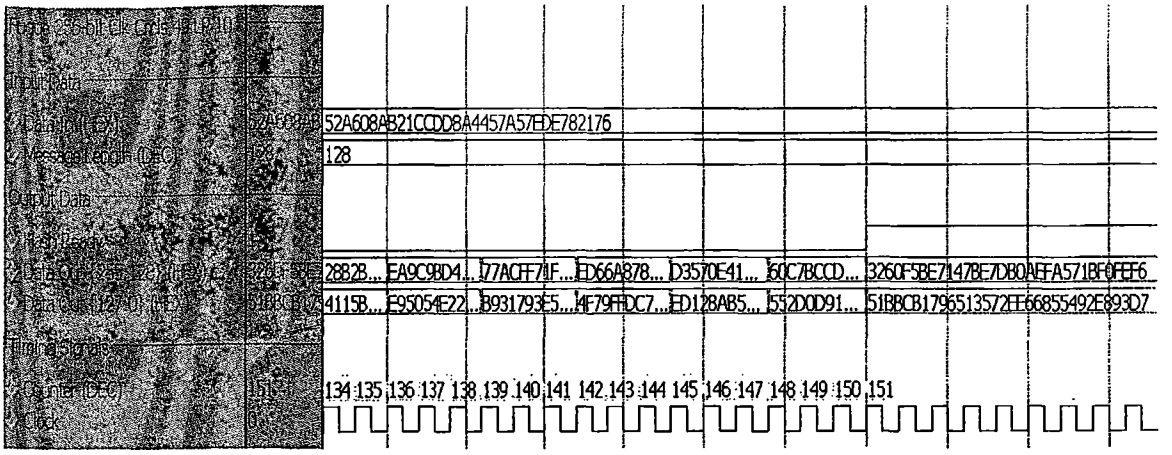
Πίνακας 7.14: *Μετρήσεις του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 10.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
LP 10	2,285	5,646	151	3.823	422.909	189.522

Memory	
BRAM	Memory Used (Kb)
8	144

Power Supply (mW)		
Dynamic	Static	Total
485.76	3,264.10	3,749.86

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 10 nS



Σχήμα 7.14: *Διάγραμμα χρονισμού του κυκλώματος Fugue 256-bit Clk Crcls 49 LP 10.*

7.3 Αποτελέσματα μετρήσεων των αρχιτεκτονικών JH

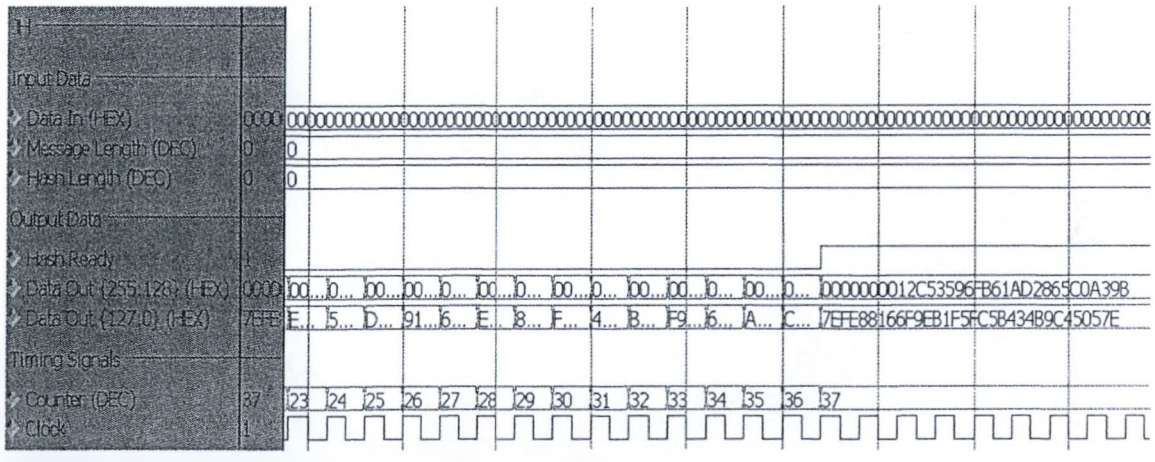
Στο Πίνακα 7.15 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος JH το οποίο και αποτελεί το κύκλωμα αναφοράς των υπόλοιπων κυκλωμάτων του αλγορίθμου JH. Στο Σχήμα 7.15 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 0-bit, εισερχόμενο μήνυμα «0» και μέγεθος σύνοψης 224-bit. Το επιπλέον σήμα Hash Length που υπάρχει στα διαγράμματα χρονισμού καθορίζει το μέγεθος της σύνοψης. Για Hash Length = 0 η σύνοψη έχει μέγεθος 224-bit, για Hash Length = 1 η σύνοψη έχει μέγεθος 256-bit, για Hash Length = 2 η σύνοψη έχει μέγεθος 384-bit και για Hash Length = 3 η σύνοψη έχει μέγεθος 512-bit. Το εύρος του σήματος Data Out θα εμφανίζεται ανάλογα με το μέγεθος της σύνοψης της κάθε εξομοίωσης.

Πίνακας 7.15: *Μετρήσεις του κυκλώματος JH.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
JH	2,251	1,032	74	4.969	1,327.847	604.049

Power Supply (mW)		
Dynamic	Static	Total
8,187.80	3,264.10	11,451.90

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 5 nS



Σχήμα 7.15: *Διάγραμμα χρονισμού του κυκλώματος JH.*

JH LP 1

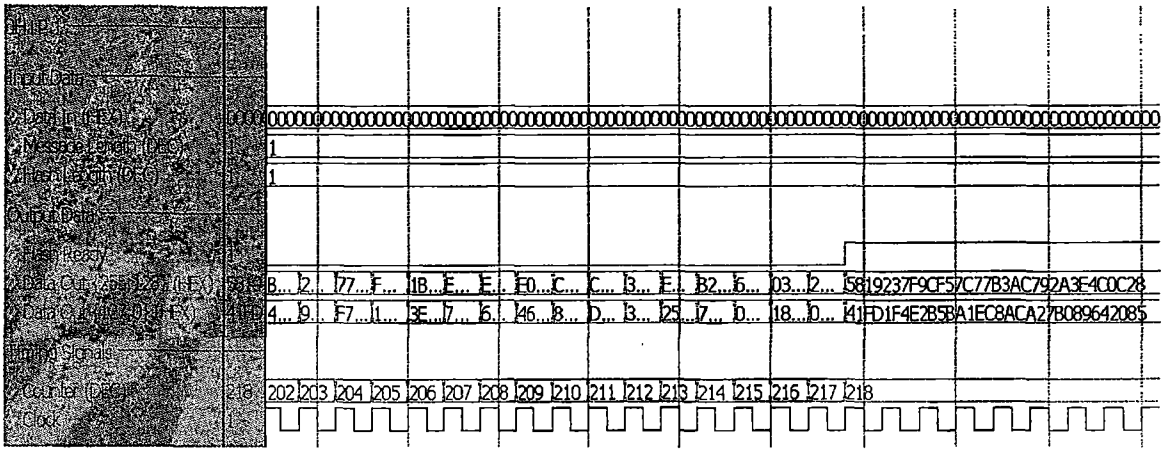
Στο Πίνακα 7.16 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος JH LP 1. Στο Σχήμα 7.16 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 1-bit, εισερχόμενο μήνυμα «0» και μέγεθος σύνοψης 256-bit.

Πίνακας 7.16: *Μετρήσεις του κυκλώματος JH LP 1.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
JH LP 1	2,045	4,106	218	3.879	577.358	289.102

Power Supply (mW)		
Dynamic	Static	Total
7,461.80	3,264.10	10,725.90

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 5 nS



Σχήμα 7.16: *Διάγραμμα χρονισμού του κυκλώματος JH LP 1.*

JH LP 1 V2

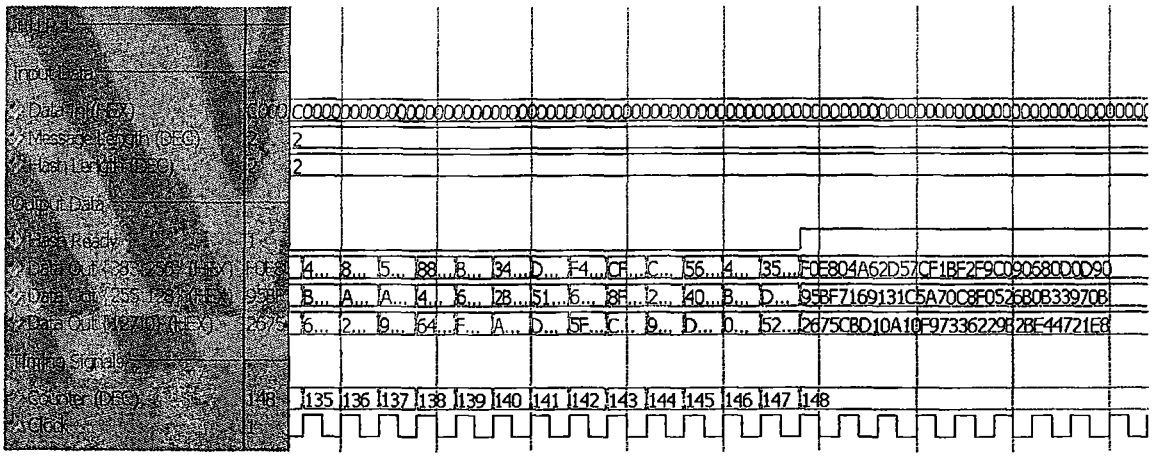
Στο Πίνακα 7.17 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος JH LP 1 V2. Στο Σχήμα 7.17 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 2-bit, εισερχόμενο μήνυμα «0xC0» και μέγεθος σύνοψης 384-bit.

Πίνακας 7.17: *Μετρήσεις του κυκλώματος JH LP 1 V2.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
JH LP 1 V2	2,049	4,104	148	3.626	909.915	454.735

Power Supply (mW)		
Dynamic	Static	Total
7,408.36	3,264.10	10,672.46

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 5 nS



Σχήμα 7.17: *Διάγραμμα χρονισμού του κυκλώματος JH LP 1 V2.*

JH LP 2

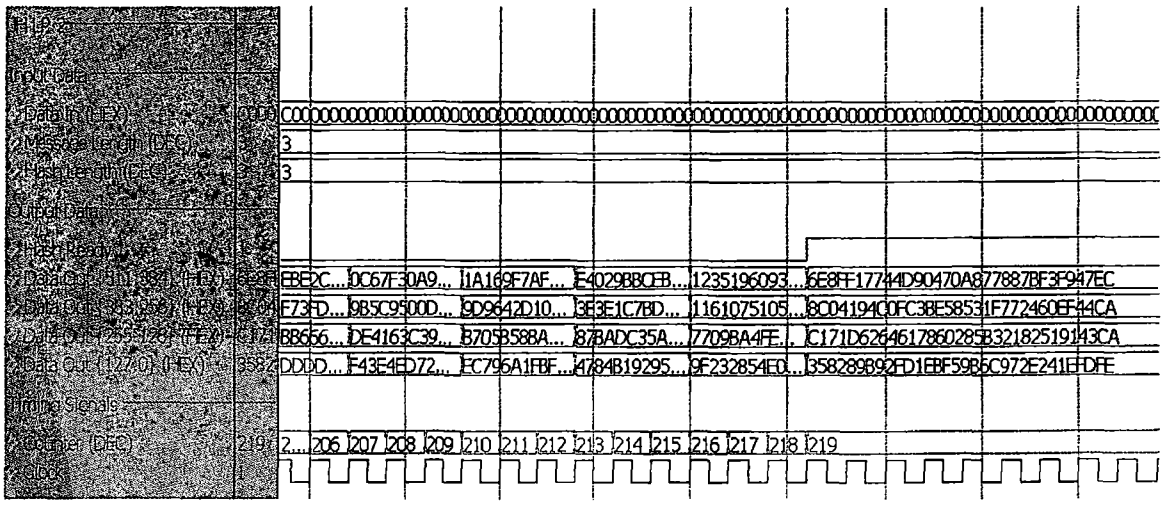
Στο Πίνακα 7.18 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος JH LP 2. Στο Σχήμα 7.18 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 3-bit, εισερχόμενο μήνυμα «0xC0» και μέγεθος σύνοψης 512-bit.

Πίνακας 7.18: *Μετρήσεις του κυκλώματος JH LP 2.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
JH LP 2	2,302	4,104	219	3.795	587.451	261.316

Power Supply (mW)		
Dynamic	Static	Total
4,005.03	3,264.10	7,269.13

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 5 nS



Σχήμα 7.18: *Διάγραμμα χρονισμού του κυκλώματος JH LP 2.*

JH LP 3

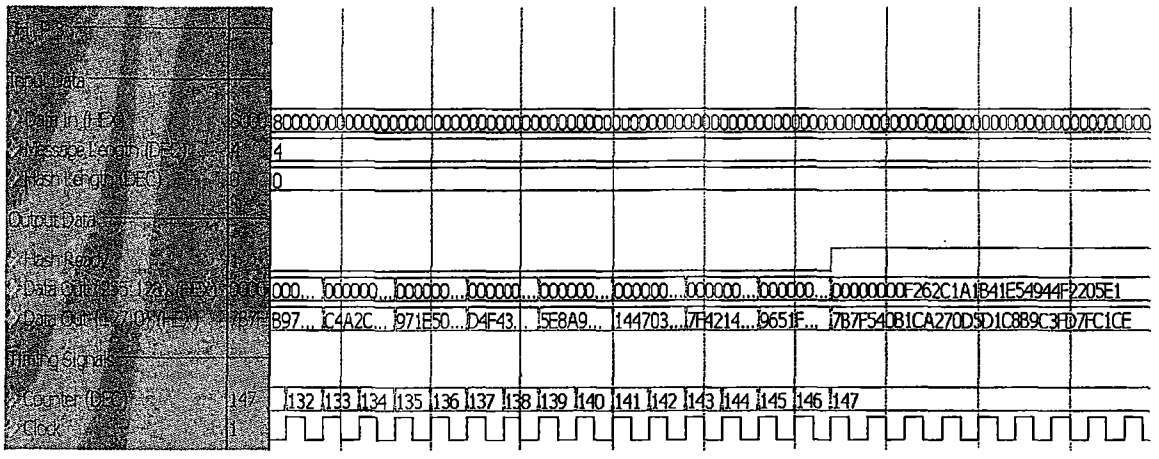
Στο Πίνακα 7.19 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος JH LP 3. Στο Σχήμα 7.19 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 4-bit, εισερχόμενο μήνυμα «0x80» και μέγεθος σύνοψης 224-bit.

Πίνακας 7.19: *Μετρήσεις του κυκλώματος JH LP 3.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
JH LP 3	2,058	4,104	147	2.377	1,397.198	695.204

Power Supply (mW)		
Dynamic	Static	Total
4,096.08	3,264.10	7,360.18

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 5 nS



Σχήμα 7.19: *Διάγραμμα χρονισμού του κυκλώματος JH LP 3.*

JH LP 4

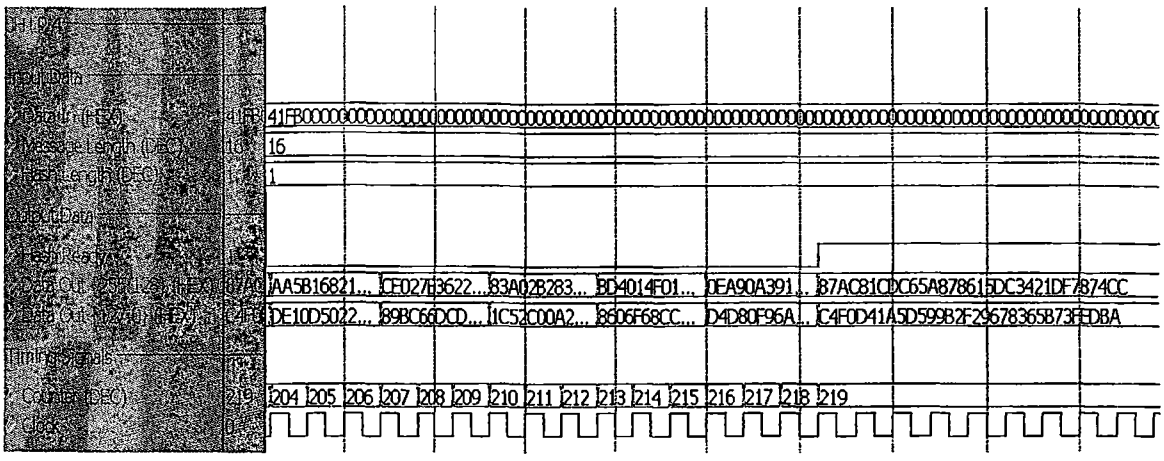
Στο Πίνακα 7.20 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος JH LP 4. Στο Σχήμα 7.20 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 16-bit, εισερχόμενο μήνυμα «0x41FB» και μέγεθος σύνοψης 256-bit.

Πίνακας 7.20: *Μετρήσεις του κυκλώματος JH LP 4.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
JH LP 4	3,247	4,109	219	3.606	618.293	194.989

Power Supply (mW)		
Dynamic	Static	Total
4,420.88	3,264.10	7,684.98

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 5 nS



Σχήμα 7.20: *Διάγραμμα χρονισμού του κυκλώματος JH LP 4.*

JH LP 5

Στο Πίνακα 7.21 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος JH LP 5. Όπως συνέβη και στο Πίνακα 7.11 του κυκλώματος Fugue LP 7 έτσι και εδώ εμφανίζονται οι δύο στήλες που δείχνουν τη χρήση μνήμης από το κύκλωμα JH LP 5. Η τεράστια διαφορά που υπάρχει σε χρήση μνήμης μεταξύ των κυκλωμάτων Fugue και JH οφείλεται στο γεγονός ότι στο σχεδιασμό των κυκλωμάτων JH LP 5 έως JH LP 8 χρησιμοποιείται η ενσωματωμένη μνήμη RAM για την υλοποίηση των SBoxes καθώς και των 4 πινάκων αρχικοποίησης. Στο Σχήμα 7.21 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 32-bit, εισερχόμενο μήνυμα «0xC1ECFDFC» και μέγεθος σύνοψης 384-bit.

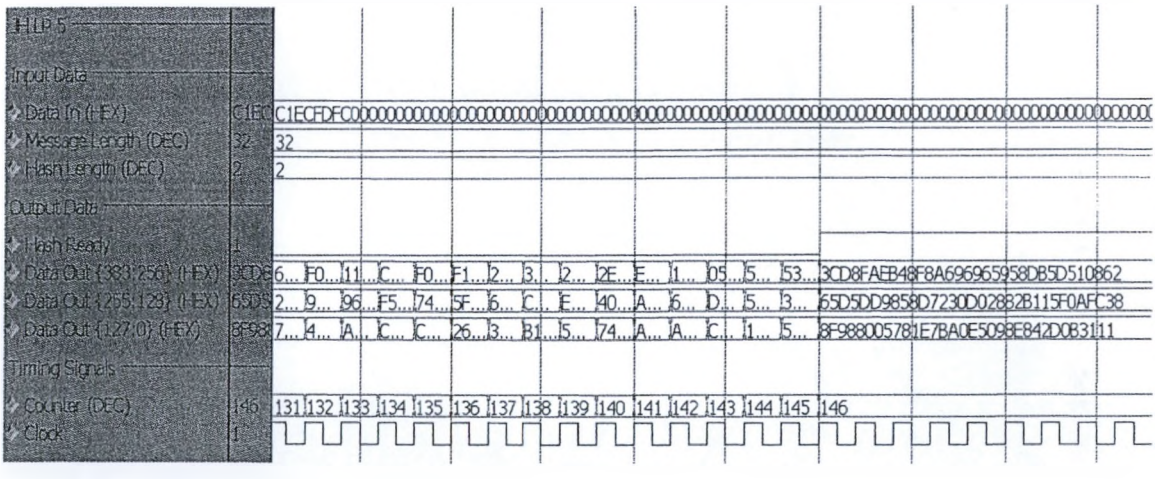
Πίνακας 7.21: *Μετρήσεις του κυκλώματος JH LP 5.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
JH LP 5	2,064	1,032	146	3.704	902.875	447.937

Memory	
BRAM	Memory Used (Kb)
208	3,744

Power Supply (mW)		
Dynamic	Static	Total
8,998.50	3,264.10	12,262.60

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 5 nS



Σχήμα 7.21: *Διάγραμμα χρονισμού του κυκλώματος JH LP 5.*

JH LP 6

Στο Πίνακα 7.22 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος JH LP 6. Στο Σχήμα 7.22 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 64-bit, εισερχόμενο μήνυμα «0x4A4F202484512526» και μέγεθος σύνοψης 512-bit.

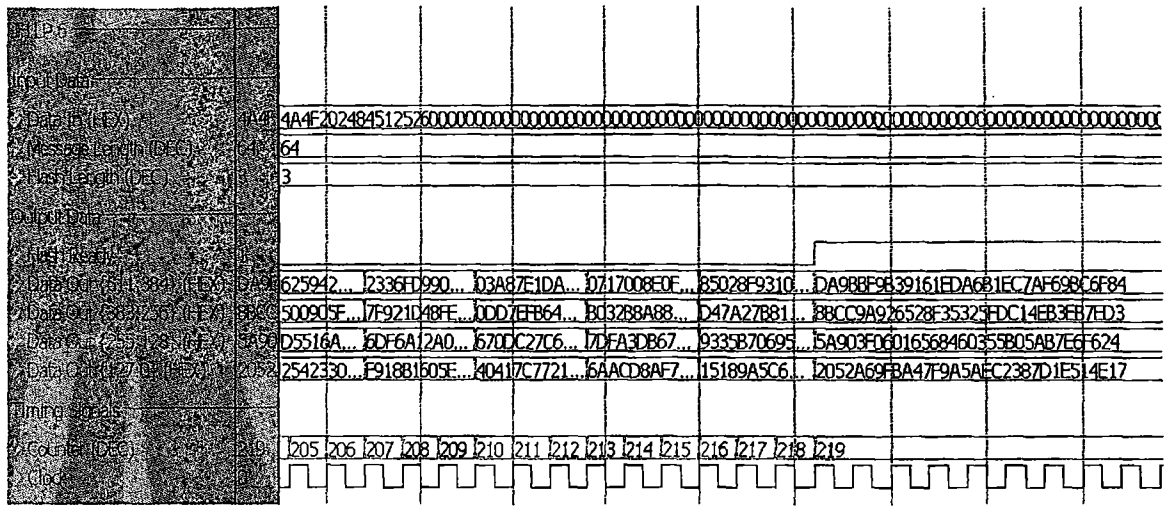
Πίνακας 7.22: *Μετρήσεις του κυκλώματος JH LP 6.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
JH LP 6	2,199	3,085	219	3.455	645.278	300.484

Memory	
BRAM	Memory Used (Kb)
208	3,744

Power Supply (mW)		
Dynamic	Static	Total
4,671.77	3,264.10	7,935.87

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 5 nS



Σχήμα 7.22: *Διάγραμμα χρονισμού του κυκλώματος JH LP 6.*

JH LP 7

Στο Πίνακα 7.23 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος JH LP 7. Στο Σχήμα 7.23 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 96-bit, εισερχόμενο μήνυμα «0x5BE43C90F22902E4FE8ED2D3» και μέγεθος σύνοψης 224-bit.

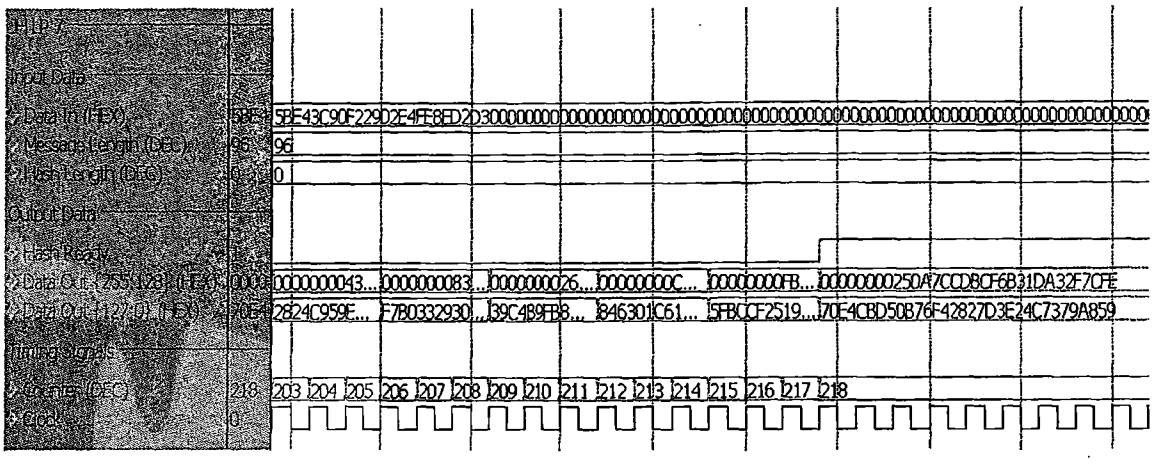
Πίνακας 7.23: *Μετρήσεις του κυκλώματος JH LP 7.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
JH LP 7	2,804	3,087	218	2.979	751.845	274.568

Memory	
BRAM	Memory Used (Kb)
208	3,744

Power Supply (mW)		
Dynamic	Static	Total
4,451.25	3,264.10	7,715.35

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 5 nS



Σχήμα 7.23: *Διάγραμμα χρονισμού του κυκλώματος JH LP 7.*

JH LP 8

Στο Πίνακα 7.24 δίνονται τα αποτελέσματα των μετρήσεων του κυκλώματος JH LP 8. Στο Σχήμα 7.24 δίνεται το διάγραμμα χρονισμού του ίδιου κυκλώματος για είσοδο μεγέθους 128-bit, εισερχόμενο μήνυμα «0x52A608AB21CCDD8A4457A57EDE782176» και μέγεθος σύνοψης 256-bit.

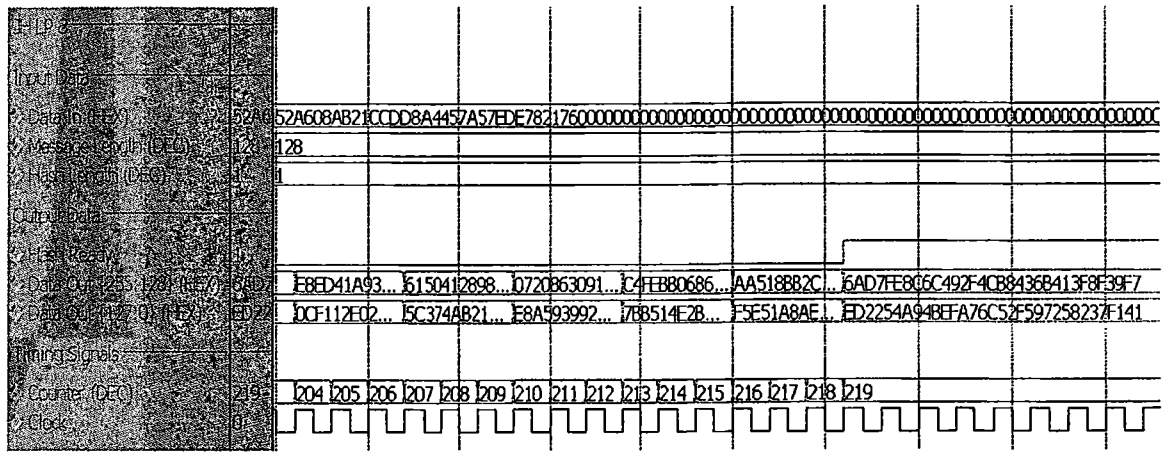
Πίνακας 7.24: *Μετρήσεις του κυκλώματος JH LP 8.*

Design	Slices	Flip-Flop	Clock Circles	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)
JH LP 8	3,195	3,081	219	3.755	593.803	190.314

Memory	
BRAM	Memory Used (Kb)
208	3,744

Power Supply (mW)		
Dynamic	Static	Total
5,157.01	3,264.10	8,421.11

Measurement's parameters for power consumption
Simulation time = 400 μS
Simulation period = 5 nS



Σχήμα 7.24: *Διάγραμμα χρονισμού του κυκλώματος JH LP 8.*

7.4 Συγκρίσεις των κυκλωμάτων χαμηλής κατανάλωσης

Στην ενότητα αυτή θα γίνουν οι συγκρίσεις των αποτελεσμάτων των αρχιτεκτονικών χαμηλής κατανάλωσης με σκοπό την εξαγωγή συμπερασμάτων για τη χρήση κάθε τεχνικής στους αλγορίθμους Fugue και JH. Στην επόμενη ενότητα θα γίνουν οι ανάλογες συγκρίσεις για τα κυκλώματα μειωμένων κύκλων επεξεργασίας. Ο Πίνακας 7.25 συνοψίζει τα βασικότερα στοιχεία των κυκλωμάτων χαμηλής κατανάλωσης.

Κατανάλωση

Στο Σχήμα 7.25 γίνεται η σύγκριση των αποτελεσμάτων των κυκλωμάτων του αλγορίθμου Fugue. Το κύκλωμα αναφοράς Fugue έχει τη δεύτερη υψηλότερη κατανάλωση σε σχέση με τις υπόλοιπες υλοποιήσεις που εφαρμόζουν τεχνικές χαμηλής κατανάλωσης. Την υψηλότερη κατανάλωση τη παρουσιάζει το κύκλωμα Fugue LP 2 του Σχήματος 6.22 . Τα

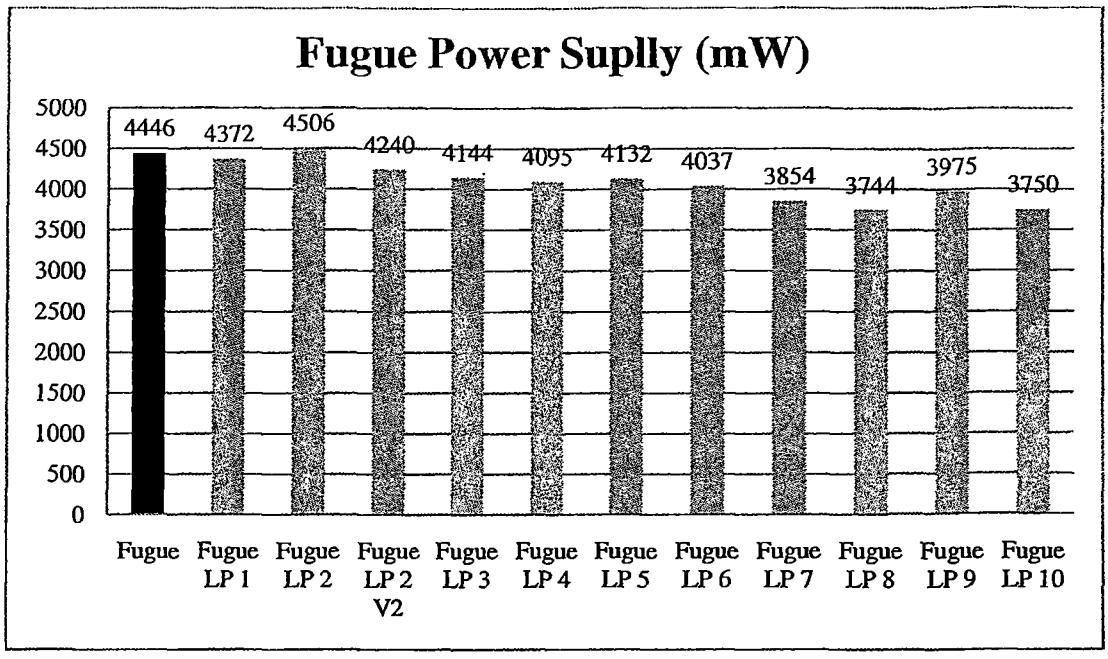
Πίνακας 7.25: *Συνοπτική παρουσίαση αποτελεσμάτων των κυκλωμάτων χαμηλής κατανάλωσης.*

Design	Slices	BRAM 18 Kbit	Clock Circles	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)	Total Power Supply (mW)
Fugue	1,394	-	49	514.01	377.579	4,446.12
Fugue LP 1	1,686	-	77	362.967	220.449	4,372.35
Fugue LP 2	1,821	-	103	268.042	150.727	4,505.86
Fugue LP 2 V2	2,018	-	103	186.492	94.632	4,240.45
Fugue LP 3	1,723	-	77	362.006	215.142	4,143.84
Fugue LP 4	2,374	-	103	267.732	115.483	4,095.57
Fugue LP 5	1,997	-	77	1,443.964	740.420	4,132.39
Fugue LP 6	2,593	-	103	268.933	106.204	4,036.94
Fugue LP 7	1,093	8	98	690.885	647.270	3,853.59
Fugue LP 8	1,868	8	151	418.646	229.493	3,743.83
Fugue LP 9	2,223	8	125	574.645	264.717	3,975.46
Fugue LP 10	2,285	8	151	422.909	189.522	3,749.86
JH	2,251	-	74	1,327.847	604.049	11,451.90
JH LP 1	2,045	-	218	577.358	289.102	10,725.90
JH LP 1 V2	2,049	-	148	909.915	454.735	10,672.46
JH LP 2	2,302	-	219	587.451	261.316	7,269.13
JH LP 3	2,058	-	147	1,397.198	625.204	7,360.18
JH LP 4	3,247	-	219	618.293	194.989	7,684.98
JH LP 5	2,064	208	146	902.875	447.937	12,262.60
JH LP 6	2,199	208	219	615.278	300.484	7,935.87
JH LP 7	2,804	208	218	751.845	254.568	7,735.15
JH LP 8	3,195	208	219	593.803	190.314	8,421.11

Fugue: LP(1,2) = Pipeline (V2=DET), LP(3,4) = Pipeline ENB, LP5 = Pipeline With AND Gate (LP4), LP6 = Pipeline With FBMUX (LP4), LP7 = Only Memory, LP8 = LP7 & LP4, LP9 = LP7 & LP5, LP10 = LP7 & LP6.

JH: LP1 = Pipeline (V2=DET), LP2 = Pipeline ENB, LP3 = Pipeline With AND Gate (LP2), LP4 = Pipeline With FBMUX (LP2), LP5 = Only Memory, LP6 = LP5 & LP2, LP7 = LP5 & LP3, LP8 = LP5 & LP4.

κυκλώματα Fugue LP 1 και Fugue LP 2 χρησιμοποιούν και τα δύο τη μέθοδο pipelining με τη διαφορά ότι στο κύκλωμα Fugue LP 2 υπάρχουν επιπλέον καταχωρητές σε σχέση με το κύκλωμα Fugue LP 1. Όπως αποδεικνύεται όμως βάση των αποτελεσμάτων η χρήση των επιπλέον καταχωρητών δε μείωσε τη κατανάλωση αλλά την αύξησε κατά 59.74 mW και αυτό γιατί το επιπλέον υλικό που προστέθηκε επιβάρυνε το κύκλωμα. Αυτό συνεπάγεται ότι στη περίπτωση χρήσης της μεθόδου απλού pipelining είναι προτιμότερη η αρχιτεκτονική του Σχήματος 6.21 καθώς μειώνεται η κατανάλωση κατά 73.77 mW . Στη περίπτωση όμως που χρησιμοποιηθεί το κύκλωμα Fugue LP 2 σε συνδυασμό με τη τεχνική DET που υλοποιήθηκε από το κύκλωμα Fugue LP 2 V2 τότε επιτυγχάνεται μείωση κατά 205.67 mW. Από τη



Σχήμα 7.25: *Διάγραμμα κατανάλωσης των κυκλωμάτων, με μαύρο χρώμα παρουσιάζεται τα κύκλωμα αναφοράς Fugue.*

σύγκριση των τριών πρώτων κυκλωμάτων χαμηλής κατανάλωσης γίνεται εμφανές ότι η βέλτιστη εφαρμογή pipeling στο κύκλωμα αναφοράς γίνεται από το κύκλωμα Fugue LP 2 V2.

Στα δύο επόμενα κυκλώματα έγινε η χρήση της μεθόδου Gating Clock με το ενσωματωμένο Enable των flip-flop πάνω στα κυκλώματα Fugue LP 1 και Fugue LP 2. Τα κυκλώματα αυτά είναι τα Fugue LP 3 και Fugue LP 4 αντίστοιχα και παρουσιάζουν μια τελείως διαφορετική συμπεριφορά από τα κυκλώματα Fugue LP 1 και Fugue LP 2. Εδώ η μεγαλύτερη μείωση επιτεύχθηκε στο κύκλωμα με τους περισσότερους καταχωρητές το οποίο είναι το κύκλωμα Fugue LP 4. Ο λόγος για αυτή την αλλαγή συμπεριφοράς είναι με τη χρήση του Gating Clock επιτρέπεται ο έλεγχος της διάδοσης δεδομένων στο κύκλωμα οπότε όσο περισσότεροι καταχωρητές υπάρχουν στο κύκλωμα τόσο μεγαλύτερος είναι ο έλεγχος που μπορεί να γίνει στη διάδοση των δεδομένων. Τα κυκλώματα Fugue LP 3 και Fugue LP 4 πέτυχαν μια μείωση κατά 302.28 mW και 350.55 mW αντίστοιχα από το κύκλωμα αναφοράς.

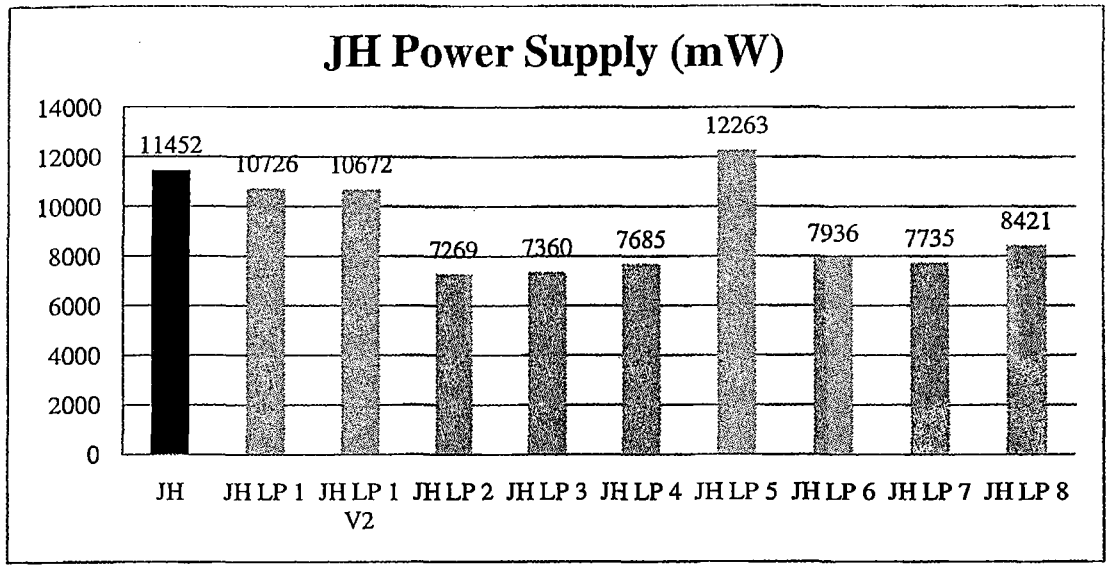
Τα κυκλώματα Fugue LP 5 και Fugue LP 6 κάνουν χρήση της ίδιας μεθόδου με τη διαφορά ότι χρησιμοποιούσαν τη πύλη AND και του πολυπλέκτη ανάδρασης αντίστοιχα πάνω στο κύκλωμα Fugue LP 2. Το κύκλωμα Fugue LP 5 παρουσιάζει της κατανάλωσης σε σχέση με το κύκλωμα Fugue LP 4 κάτι το οποίο επαληθεύει τη μη συνιστώμενη χρήση Gating Clock με πύλη AND στα FPGA. Το κύκλωμα Fugue LP 6 παρουσιάζει σχεδόν την

ίδια (αλλά λίγο καλύτερη) συμπεριφορά με το κύκλωμα Fugue LP 4 κάτι το οποίο δείχνει ότι στη περίπτωση που δεν παρέχεται από το FPGA επιλογή του ενσωματωμένου Enable στους καταχωρητές μπορεί να υλοποιηθεί από τον σχεδιαστή η λειτουργία αυτή με εξίσου καλά αποτελέσματα. Επίσης και στη περίπτωση που παρέχεται ίσως η χρήση του πολυπλέκτη ανάδρασης να προσφέρει μεγαλύτερη μείωση από αυτή που προσφέρει η χρήση του ενσωματωμένου Enable των flip-flop. Στη περίπτωση των κυκλωμάτων του αλγορίθμου Fugue δεν μπορεί να βγει κάποιο συγκεκριμένο συμπέρασμα λόγω της μικρής διαφοράς μεταξύ των αποτελεσμάτων των δύο μεθόδων. Από τη σύγκριση των τριών μεθόδων Gating Clock η βέλτιστη υλοποίηση προκύπτει από το κύκλωμα Fugue LP 6.

Το κύκλωμα Fugue LP 7 πρόκειται ουσιαστικά για το κύκλωμα αναφοράς με τη διαφορά ότι κάνει χρήση της ενσωματωμένης μνήμης RAM του FPGA για την υλοποίηση των SBoxes. Το αποτέλεσμα της μείωσης της κατανάλωσης κατά 592.53 mW από το κύκλωμα αναφοράς συνεπάγεται ότι η χρήση των ενσωματωμένων κυκλωμάτων που παρέχονται αντί των λογικών βαθμίδων αποτελεί την αποτελεσματικότερη μέθοδο μείωσης της κατανάλωσης για το κύκλωμα Fugue 256-bit Clock Circles 49.

Βάση του προηγούμενου συμπεράσματος στα τρία επόμενα κυκλώματα Fugue LP (8, 9 και 10) έγινε η χρήση της προηγούμενης μεθόδου σε συνδυασμό με τη χρήση της μεθόδου Gating Clock πάνω στο κύκλωμα Fugue LP 2. Ουσιαστικά αυτό που έγινε είναι ότι στα κυκλώματα Fugue LP (4,5 και 6) χρησιμοποιήθηκε παράλληλα και η ενσωματωμένη μνήμη RAM του FPGA. Τα κυκλώματα Fugue LP (8,9 και 10) παρουσιάζουν την ίδια συμπεριφορά με τα κυκλώματα Fugue LP (4,5 και 6) με τη διαφορά τη μεγαλύτερη μείωση της κατανάλωσης λόγω της συνεισφοράς της χρήσης ενσωματωμένης μνήμης RAM. Επίσης επαληθεύεται και αυτό που αναφέρθηκε στη προηγούμενη παράγραφο δηλαδή ότι δεν μπορεί να βγει ασφαλές συμπέρασμα για το αν η μέθοδος Gating Clock με Enable είναι αποτελεσματικότερη από τη μέθοδο Gating Clock με πολυπλέκτη ανάδρασης. Καθώς πάλι τα αποτελέσματα των κυκλωμάτων Fugue LP (8 και 10) είναι σχεδόν ίδια με λίγο καλύτερα αυτή τη φορά τα αποτελέσματα του κυκλώματος Fugue LP 8 όπου γίνεται χρήση του ενσωματωμένου Enable. Τελικός η μεγαλύτερη μείωση κατανάλωσης που επιτεύχθηκε στο κύκλωμα αναφοράς είναι 702.29 mW και επιτυγχάνεται από το κύκλωμα Fugue LP 8 με κατανάλωση 3743.83 mW το οποίο κάνει χρήση των μεθόδων Pipeling, Gating Clock με Enable και ενσωματωμένης μνήμης RAM.

Το Σχήμα 7.26 παρουσιάζει την αντίστοιχη σύγκριση των κυκλωμάτων του αλγορίθμου JH.



Σχήμα 7.26: *Διάγραμμα κατανάλωσης των κυκλωμάτων JH, με μαύρο χρώμα παρουσιάζεται το κύκλωμα αναφοράς JH.*

Το κύκλωμα αναφοράς JH έχει και εδώ τη δεύτερη υψηλότερη κατανάλωση με 11,451.9 mW. Τα κυκλώματα JH LP 1 και JH LP 1 V2 κάνουν χρήση της μεθόδου pipeling και DET pipeling και έχουν σχεδόν την ίδια κατανάλωση όπως και ήταν αναμενόμενο καθώς στο κύκλωμα JH LP 2 V2 δεν προστέθηκε επιπλέον καταχωρητής, όπως έγινε στο κύκλωμα Fugue LP 2 V2, απλά ο υπάρχον καταχωρητής Register (3) του Σχήματος 6.34 μετατράπηκε από θετικά ακμοπυροδότητος σε αρνητικά ακμοπυροδότητος με στόχο τη μείωση των κύκλων επεξεργασίας. Η μείωση της κατανάλωσης που επιτυγχάνουν τα κυκλώματα JH LP 1 και JH LP 1 V2 σε σχέση με το κύκλωμα αναφοράς είναι 726 mW και 779.44 mW αντίστοιχα.

Τα κυκλώματα JH LP (2, 3 και 4) κάνουν χρήση της μεθόδου Gating Clock και λόγω του περιορισμού των συνεχόμενων μεταδόσεων δεδομένων στο κύκλωμα επιτυγχάνουν μεγαλύτερη μείωση κατανάλωσης από το κύκλωμα JH LP 1 και JH LP 1 V2. Μια διαφορά με τα αντίστοιχα κυκλώματα του αλγορίθμου Fugue είναι ότι εδώ το κύκλωμα που χρησιμοποιεί Gating Clock με πύλη AND δεν έχει την υψηλότερη κατανάλωση από τα τρία αλλά την έχει το κύκλωμα με τη χρήση του πολυπλέκτη ανάδρασης Έτσι λοιπόν αντίθετα με τα αντίστοιχα

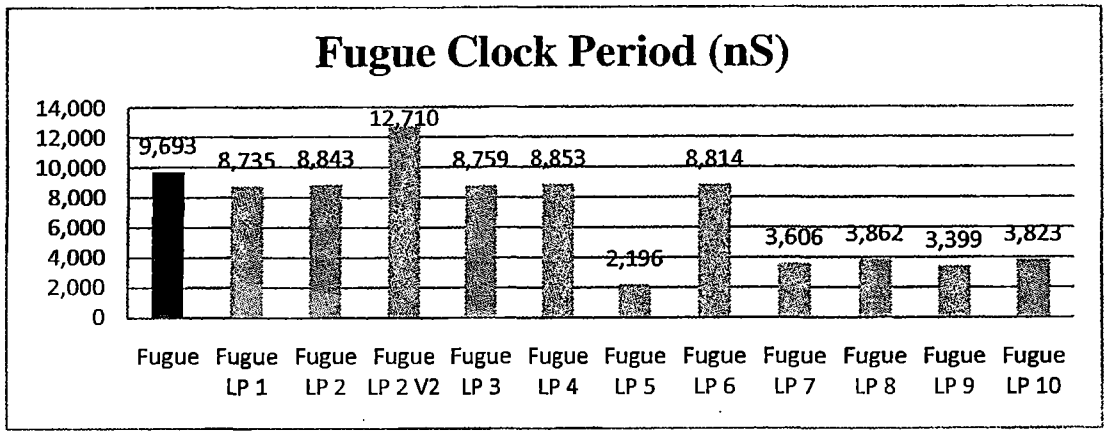
κυκλώματα του αλγορίθμου Fugue εδώ είναι ξεκάθαρο το ότι το Gating Clock με το ενσωματωμένο Enable των flip-flop στο FPGA έχει πολύ μικρότερη κατανάλωση σε σχέση με το Gating Clock με πολυπλέκτη ανάδρασης στη περίπτωση των κυκλωμάτων του αλγορίθμου JH. Όπως φαίνεται και από το Σχήμα 7.25 το κύκλωμα JH LP 2 επιτυγχάνει τη μεγαλύτερη μείωση κατανάλωσης σε σχέση με το κύκλωμα αναφοράς η οποία είναι 4182.77 mW.

Σε αντίθεση με το κύκλωμα Fugue LP 7 που κάνει χρήση μνήμης το κύκλωμα JH LP 5 που κάνει και αυτό χρήση μνήμης έχει εξαιρετικά αυξημένη κατανάλωση. Η κατανάλωση αυτή φτάνει τα 12262.6 mW και αποτελεί τη μεγαλύτερη κατανάλωση που μετρήθηκε από όλα τα κυκλώματα. Ο λόγος της τόσο αυξημένης κατανάλωσης είναι η πολύ μεγάλη χρήση της ενσωματωμένης μνήμης κατά 200 BRAM παραπάνω από το αντίστοιχο κύκλωμα του αλγορίθμου Fugue όπως προκύπτει και από το Πίνακα 7.25 . Ένα συμπέρασμα που προκύπτει από τα παραπάνω είναι ότι η χρήση μνήμης έναντι των λογικών βαθμίδων μειώνει τη κατανάλωση αλλά αυτό εξαρτάται άμεσα από το μέγεθος της.

Τα επόμενα τρία κυκλώματα JH LP (6, 7 και 8) χρησιμοποιούν μνήμη και αυτά όπως επίσης χρησιμοποιούν και τη μέθοδο Gating Clock. Ουσιαστικά πρόκειται για τα κυκλώματα JH LP (2, 3 και 4) με παράλληλη χρήση μνήμης. Για το περιορισμό της υψηλής κατανάλωσης λόγω της μεγάλης χρήσης μνήμης αυτά τα τρία κυκλώματα χρησιμοποιούν τη μέθοδο του Gating Clock και στα τμήματα μνήμης. Έτσι περιορίζεται η υψηλή κατανάλωση που εμφανίζεται στο κύκλωμα JH LP 5 λόγω της μνήμης. Αυτά δεν παρουσιάζουν την ίδια συμπεριφορά με τα κυκλώματα JH LP (2, 3 και 4). Δηλαδή το κύκλωμα που χρησιμοποιεί τη το πολυπλέκτη ανάδρασης για Gating Clock έχει πάλι τη μεγαλύτερη κατανάλωση από τα τρία αλλά το κύκλωμα που χρησιμοποιεί το ενσωματωμένο Enable έχει υψηλότερη κατανάλωση από το κύκλωμα που χρησιμοποιεί τη πύλη AND. Παρόλα αυτά η αύξηση της κατανάλωσης λόγω της μνήμης καθιστά την τριάδα των κυκλωμάτων JH LP (2, 3 και 4) που χρησιμοποιούν Gating Clock χωρίς μνήμη τη πιο αποτελεσματική για τη μείωση της κατανάλωσης.

Throughput και Slices

Δύο επιπλέον πολύ σημαντικά μεγέθη του Πίνακα 7.25 είναι το Throughput και ο αριθμός των Slices που χρησιμοποιούνται. Όπως προκύπτει από τη Σχέση 7.1 για τον υπολογισμό του Throughput χρειάζονται δύο μεγέθη τα οποία είναι η περίοδος του κυκλώματος και ο αριθμός των κύκλων επεξεργασίας, οι συγκρίσεις αυτών των μεγεθών δίνονται από τα Σχήματα 7.27, 7.28 και 7.29, 730 για τα κυκλώματα Fugue και JH αντίστοιχα.

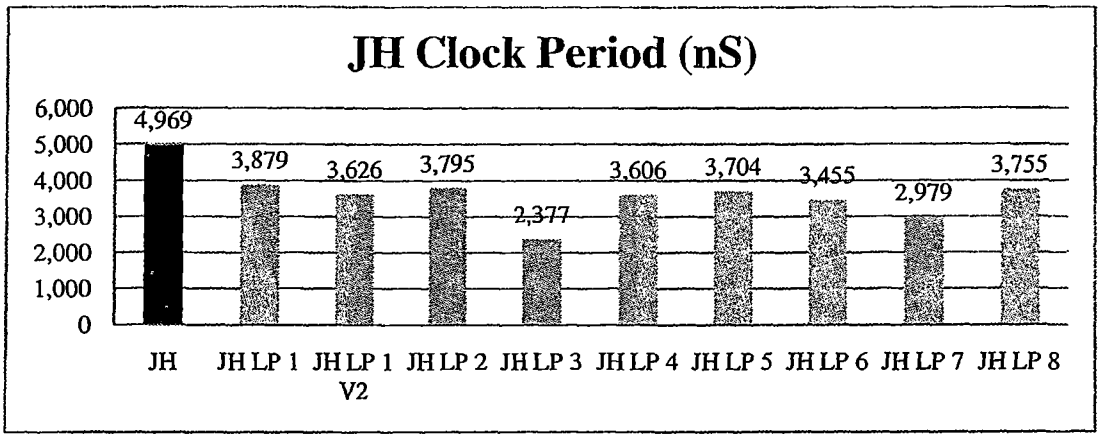


Σχήμα 7.27: *Διάγραμμα περιόδων των κυκλωμάτων Fugue, με μαύρο χρώμα παρουσιάζεται το κύκλωμα αναφοράς.*

Τη μεγαλύτερη περίοδο τη παρουσιάζει το κύκλωμα LP 2 V2 η οποία είναι αυξημένη κατά 3.017 nS σε σχέση με το κύκλωμα αναφοράς. Τα κυκλώματα όπου έγινε χρήση μόνο της μεθόδου pipeling αλλά και χρήση του gating clock υπήρξε μικρή μείωση της περιόδου εξαιρουμένου του κυκλώματος LP 5 όπου έγινε χρήση του gating clock με πύλη AND. Η μικρή καθυστέρηση που προσθέτει η πύλη AND έχει ως αποτέλεσμα τη μικρή περίοδο ρολογιού που παρουσιάζεται στο Σχήμα 7.27 . Επίσης βάση του σχήματος η χρήση ενσωματωμένης μνήμης στο κύκλωμα LP 7 παρουσιάζει μικρή περίοδο και ως αποτέλεσμα τα κυκλώματα LP 8 έως LP 10 που χρησιμοποιούν ενσωματωμένη μνήμη και gating clock παρουσιάζουν μικρές περιόδους. Ανάμεσα σε αυτά τα κυκλώματα τη μικρότερη περίοδο την έχει το κύκλωμα LP 9 το οποίο για την υλοποίηση του gating clock χρησιμοποιεί πάλι πύλη AND.

Στα κυκλώματα του αλγορίθμου JH βάση του Σχήματος 7.28 τη μεγαλύτερη περίοδο τη παρουσιάζει το κύκλωμα αναφοράς. Όλα τα υπόλοιπα κυκλώματα εξαιρουμένων των

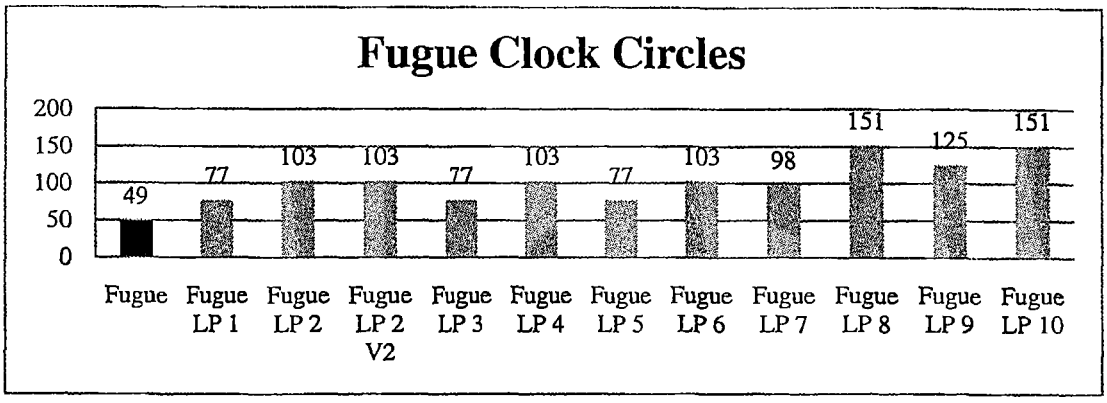
κυκλωμάτων JH LP 3 και LP 7 παρουσιάζουν μικρή μείωση της τάξεως ενός nSecond. Αυτά τα κυκλώματα παρουσιάζουν μικρότερες περιόδους καθώς υλοποιούν τη τεχνική του gating clock με πύλη AND κάτι το οποίο παρατηρήθηκε και στα αντίστοιχα κυκλώματα του αλγορίθμου Fugue.



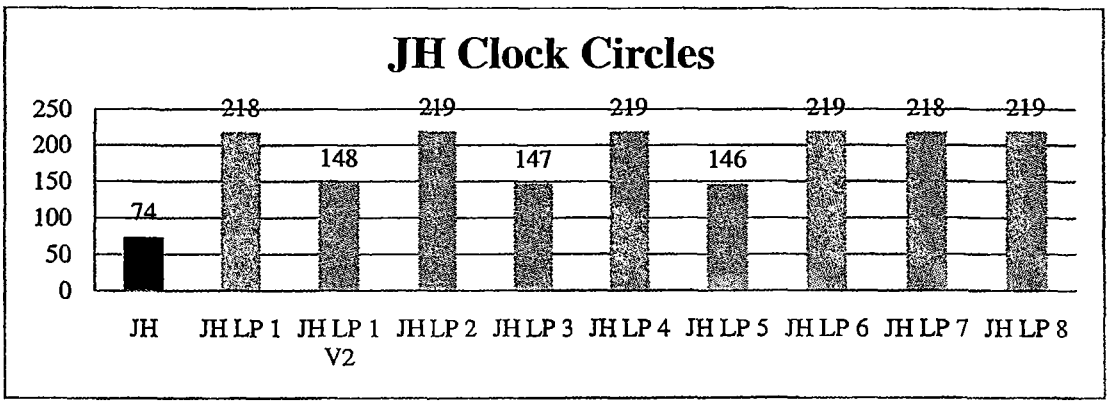
Σχήμα 7.28: *Διάγραμμα περιόδων των κυκλωμάτων JH, με μαύρο χρώμα παρουσιάζεται το κύκλωμα αναφοράς.*

Τα Σχήματα 7.29 και 7.30 δείχνουν τους κύκλους επεξεργασίας που απαιτούνται για κάθε μια από τις αρχιτεκτονικές των κυκλωμάτων των Fugue και JH αντίστοιχα. Και στις δύο περιπτώσεις, όπως είναι λογικό, τα κυκλώματα αναφοράς έχουν τους λιγότερους κύκλους μια και δεν έχει γίνει, καμία επέμβαση με επιπλέον καταχωρητές. Όσο προστίθενται στοιχεία μνήμης τα οποία αλληλεπιδρούν με το ρολόι αυξάνουν οι παλμοί που απαιτούνται για την ολοκλήρωση της λειτουργίας του κάθε κυκλώματος. Επίσης οι κύκλοι εξαρτούνται από τη δομή του κάθε κυκλώματος αλλά και από τη μέθοδο χαμηλής κατανάλωσης που χρησιμοποιήθηκε σε κάθε περίπτωση. Από τις τεχνικές που υλοποιούν pipeling με gating clock ή pipeling με gating clock και χρήση ενσωματωμένης μνήμης, τους λιγότερους κύκλους απαιτούν αυτές που για gating clock χρησιμοποιούν πύλη AND καθώς η μικρή καθυστέρηση της πύλης δίνει τη δυνατότητα σχεδιαστικά τη μείωση των κύκλων. Επίσης από το Σχήμα 7.30 φαίνεται το ένα πλεονέκτημα της τεχνικής DET pipeling καθώς τα κυκλώματα JH LP 1 και JH LP 1 V2 είναι ίδια από άποψη υλικού αλλά χάρη στη χρήση DET pipeling οι κύκλοι επεξεργασίας στο κύκλωμα JH LP 1 V2 μειώθηκαν κατά 70. Στην αντίστοιχη περίπτωση στα κυκλώματα Fugue δε παρατηρείται μείωση καθώς όπως έχει ήδη αναφερθεί εκεί η τεχνική DET pipeling χρησιμοποιήθηκε για τη μείωση της κατανάλωσης.

Από τις αρχιτεκτονικές του αλγορίθμου Fugue τους περισσότερους κύκλους απαιτούν τα κυκλώματα Fugue LP 8 και LP 10 οι οποίοι είναι 151 κύκλοι. Στις αρχιτεκτονικές του αλγορίθμου JH τα κυκλώματα με του περισσότερους κύκλους επεξεργασίας είναι τα JH LP 2, LP 4, LP 6 και LP 8 οι οποίοι είναι 219.



Σχήμα 7.29: *Διάγραμμα των απαιτούμενων κύκλω επεξεργασίας των κυκλωμάτων Fugue, με μαύρο χρώμα παρουσιάζεται το κύκλωμα αναφοράς.*

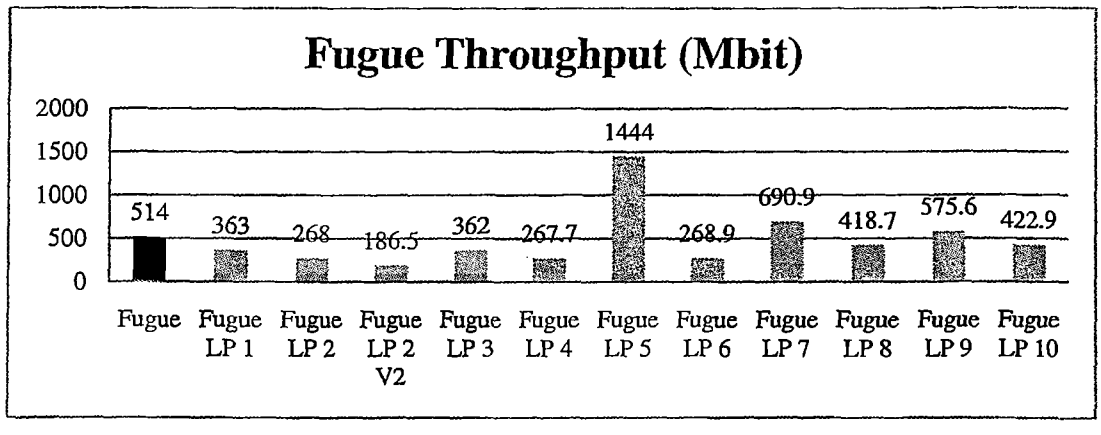


Σχήμα 7.30: *Διάγραμμα των απαιτούμενων κύκλω επεξεργασίας των κυκλωμάτων JH, με μαύρο χρώμα παρουσιάζεται τα κύκλωμα αναφοράς.*

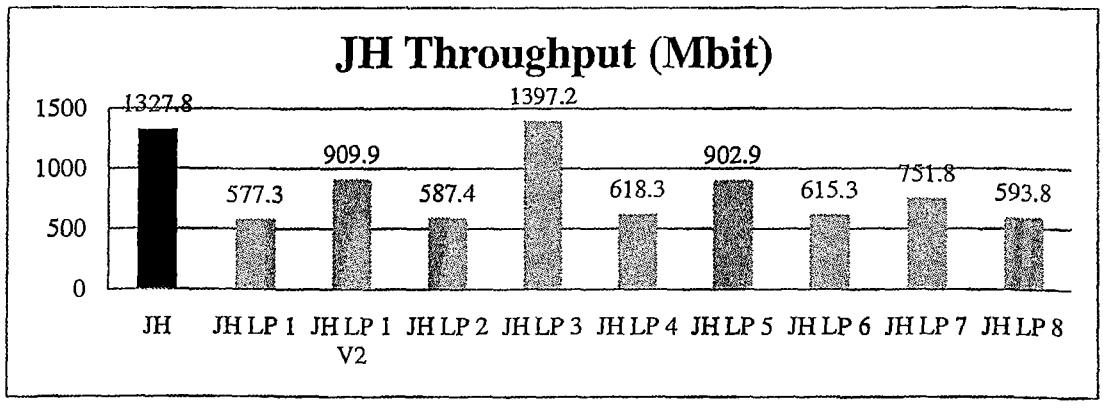
Η διεκπεραιωτική ικανότητα (Throughput) των κυκλωμάτων παρουσιάζεται από τα Σχήματα 7.31 και 7.32 η οποία προκύπτει από τη Σχέση 5.1 και από τα μεγέθη που περιγράφηκαν στις προηγούμενες παραγράφους. Το throughput στα κυκλώματα Fugue κυμαίνεται περίπου στα ίδια επίπεδα με το κύκλωμα αναφοράς με εξαίρεση τα κυκλώματα Fugue LP 2 V2 και Fugue LP 5. Το κύκλωμα Fugue LP 5 παρουσιάζει πολύ υψηλό throughput λόγω της πολύ μεγάλης συχνότητας λειτουργίας και των πολύ λίγων κύκλων

επεξεργασίας ενώ αντιθέτως το κύκλωμα Fugue LP 2 V2 παρουσιάζει πολύ χαμηλό throughput κυρίως λόγω της μικρής συχνότητας λειτουργίας η οποία είναι μικρότερη και από αυτή του κυκλώματος αναφοράς.

Το throughput των κυκλωμάτων JH διακυμαίνεται και αυτό ανάλογα με τη συχνότητα λειτουργίας και κύκλων επεξεργασίας που περιγράφηκαν στις προηγούμενες παραγράφους. Το μεγαλύτερο throughput παρατηρείται στην αρχιτεκτονική JH LP 3 το οποίο όμως δε διαφέρει πολύ από το throughput του κυκλώματος αναφοράς.

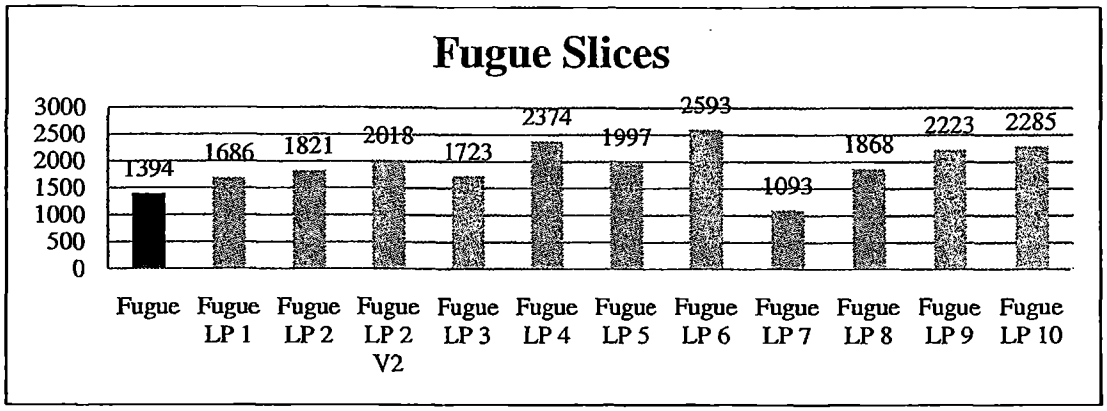


Σχήμα 7.31: *Διάγραμμα του Throughput των κυκλωμάτων Fugue, με μαύρο χρώμα παρουσιάζεται το κύκλωμα αναφοράς.*

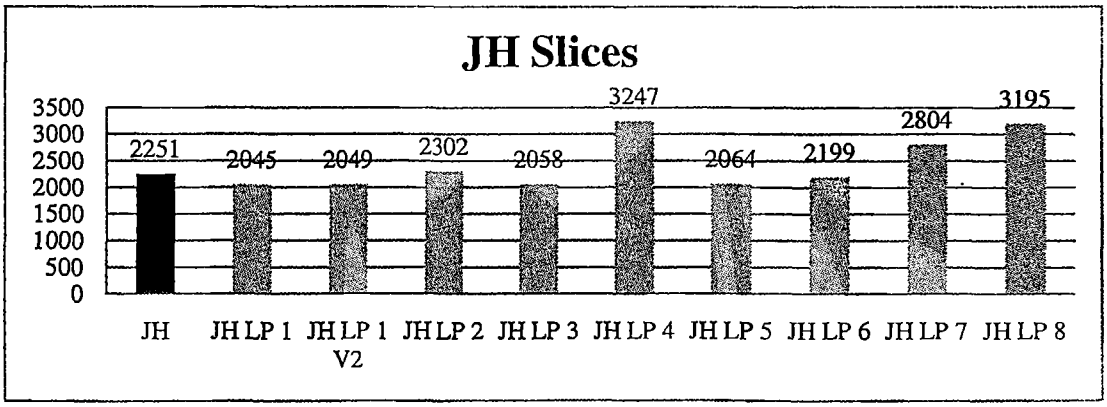


Σχήμα 7.32: *Διάγραμμα του Throughput των κυκλωμάτων JH, με μαύρο χρώμα παρουσιάζεται το κύκλωμα αναφοράς.*

Τα Σχήματα 7.33 και 7.34 παρουσιάζουν τα αποτελέσματα των απαιτούμενων λογικών τμημάτων για τα κυκλώματα Fugue και JH. Στις αρχιτεκτονικές του αλγορίθμου Fugue τα λιγότερα τμήματα απαιτεί το κύκλωμα Fugue LP 7 το οποίο κάνει χρήση της ενσωματωμένης μνήμης ενώ τα περισσότερα το κύκλωμα Fugue LP 6 το οποίο υλοποιεί τη μέθοδο pipeling και gating clock με πολυπλέκτη ανάδρασης.



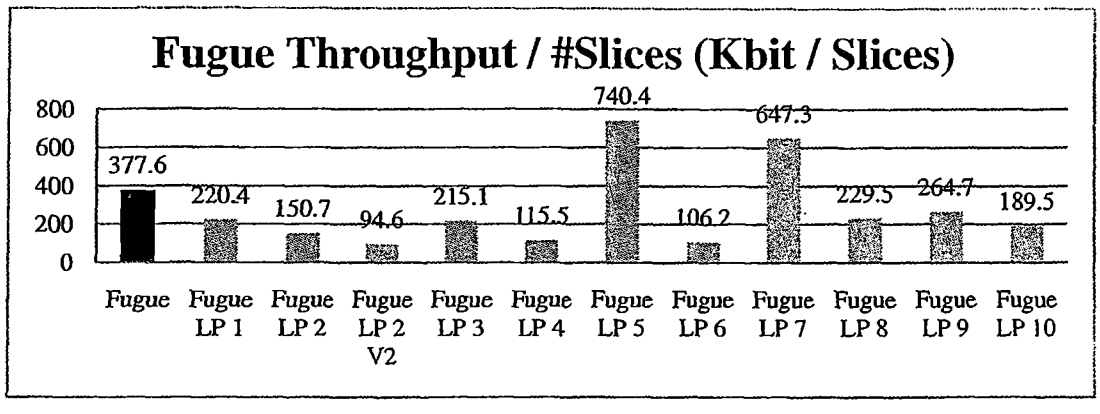
Σχήμα 7.33: *Διάγραμμα των απαιτούμενων Slices των κυκλωμάτων Fugue, με μαύρο χρώμα παρουσιάζεται τα κύκλωμα αναφοράς.*



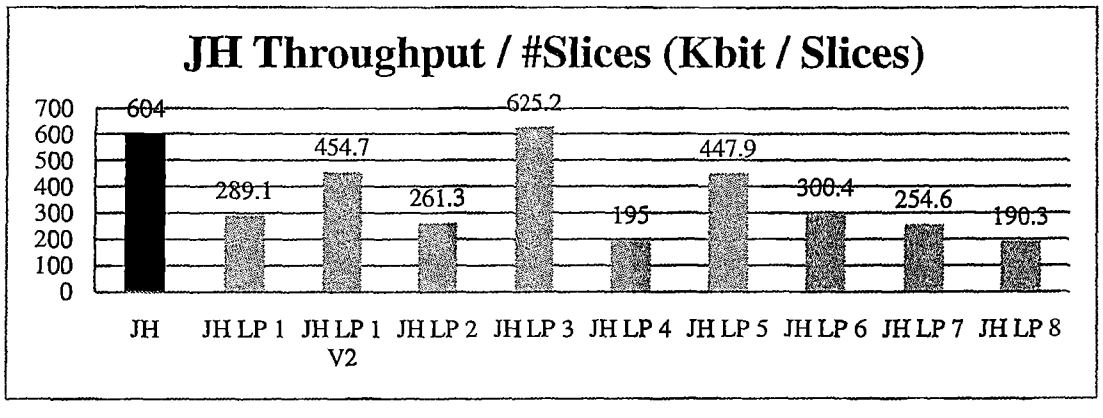
Σχήμα 7.34: *Διάγραμμα των απαιτούμενων Slices των κυκλωμάτων JH, με μαύρο χρώμα παρουσιάζεται το κύκλωμα αναφοράς.*

Όσων αφορά τα κυκλώματα του αλγορίθμου JH τα περισσότερα λογικά τμήματα απαιτεί πάλι το κύκλωμα που υλοποιεί τη μέθοδο pipeling και gating clock με πολυπλέκτη ανάδρασης το οποίο είναι το JH LP 4 ενώ τα λιγότερα παρατηρούνται στο κύκλωμα JH LP 1 το οποίο κάνει χρήση μόνο της μεθόδου pipeling.

Τέλος, ένα ακόμα σημαντικό μέγεθος είναι η απόδοση ανά μονάδα υλικού σε κάθε αρχιτεκτονική. Όσο μεγαλύτερη τιμή έχει αυτό το μέγεθος τόσο καλύτερη είναι η αντίστοιχη σχεδίαση. Το μέγεθος αυτό προκύπτει από το λόγο throughput / slices και δίνεται από τα Σχήματα 7.35 και 7.36 για τα κυκλώματα των αλγορίθμων Fugue και JH.



Σχήμα 7.35: *Διάγραμμα Throughput / Slices των κυκλωμάτων Fugue, με μαύρο χρώμα παρουσιάζεται το κύκλωμα αναφοράς.*



Σχήμα 7.36: *Διάγραμμα Throughput / Slices των κυκλωμάτων JH, με μαύρο χρώμα παρουσιάζεται το κύκλωμα αναφοράς.*

Σύμφωνα με τα σχήματα στη περίπτωση των αρχιτεκτονικών Fugue η βέλτιστη τιμή επιτυγχάνεται με την αρχιτεκτονική Fugue LP 5 λόγω του υψηλού throughput και τη χρήση ελάχιστων λογικών τμημάτων (Slices). Αντιθέτως τη χειρότερη τιμή παρουσιάζει η αρχιτεκτονική Fugue LP 5 V2. Όσων αφορά τα κυκλώματα του αλγορίθμου JH η καλύτερη απόδοση παρατηρείται από το κύκλωμα JH LP 3 η οποία είναι σχετικά κοντά στα επίπεδα του κυκλώματος αναφοράς ενώ η χείριστη περίπτωση παρατηρείται από το κύκλωμα JH LP 8.

7.5 Κυκλώματα μειωμένων κύκλων επεξεργασίας

Τα κυκλώματα Fugue 256-bit Clock Circles 36 και 25 αποτελούν δύο παραλλαγές του κυκλώματος Fugue 256-bit Clock Circles 49 και παρακάτω θα παρουσιαστούν τα αποτελέσματα των μετρήσεων τους. Ο Πίνακας 7.26 συνοψίζει τα σημαντικότερα χαρακτηριστικά των παραπάνω κυκλωμάτων.

Πίνακας 7.26: *Συνοπτική παρουσίαση αποτελεσμάτων των Fugue 256-bit Clock Circles(49, 36 και 25).*

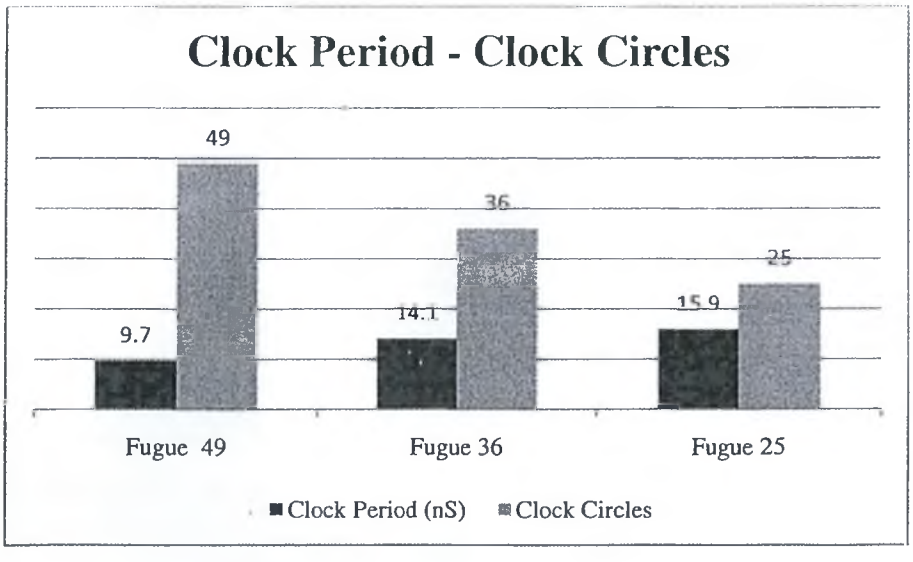
Design	Slices	Clock Period (nS)	Throughput (Mbit)	Throughput / #Slices (Kbit / Slices)	Total Power Supply (mW)
Fugue Clock Circles 49	1,394	9.693	514.01	377.579	4,446.12
Fugue Clock Circles 36	1,894	14.108	480.692	259.888	4,350.71
Fugue Clock Circles 25	2,433	15.888	614.667	258.7	4068.78

Όσων αφορά τα λογικά τμήματα παρατηρείται αύξηση καθώς μειώνονται οι κύκλοι επεξεργασίας η οποία για την επίτευξη της χρησιμοποιείται επιπλέον υλικό όπως παρουσιάστηκε στο κεφάλαιο 6. Επίσης αυτό έχει επίδραση και στη περίοδο του κάθε κυκλώματος καθώς όσο μειώνονται οι κύκλοι επεξεργασίας αυξάνεται και η περίοδος αφού μεταξύ δύο διαδοχικών καταχωρητών μεσολαβούν επιπλέον λογικά κυκλώματα. Το Σχήμα 7.37 δείχνει αυτή τη σχέση μεταξύ κύκλων ρολογιού που απαιτούνται και περιόδου ολοκλήρωσης του κάθε κύκλου.

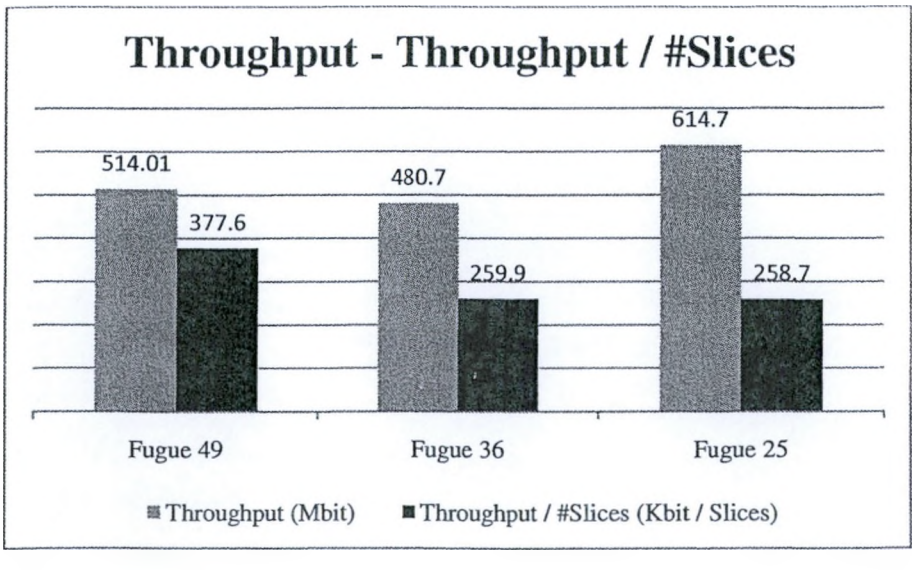
Επίσης ένα συμπέρασμα που προκύπτει από το Πίνακα 7.27 είναι ότι ενώ το Throughput αυξάνει καθώς μειώνονται οι κύκλοι επεξεργασίας η βέλτιστη απόδοση των λογικών τμημάτων παρατηρείται στο κύκλωμα Fugue 256-bit Clock Circles 49 και μειώνεται αντιστρόφως ανάλογα με την αύξηση του Throughput. Το παραπάνω συμπέρασμα φαίνεται και από το Σχήμα 7.38 .

Τέλος, παρατηρείται η μικρότερη κατανάλωση στο κύκλωμα Fugue 256-bit Clock Circles 25 και αυτή μειώνεται ανάλογα με τη μείωση των κύκλων επεξεργασίας. Αυτό οφείλεται στο ότι η διάρκεια λειτουργίας του κυκλώματος μειώνεται ως αποτέλεσμα τις λιγότερες μεταβάσεις σημάτων. Αυτό συνεπάγεται ότι αν υπάρχει η δυνατότητα παροχής των

απαιτούμενων λογικών τμημάτων η χρήση του κυκλώματος Fugue 256-bit Clock Circles 25 είναι προτιμότερη. Επίσης εάν εφαρμοστούν οι τεχνικές χαμηλής κατανάλωσης που εφαρμόστηκαν στο κεφάλαιο 6 θα επιτευχθούν πολύ μεγαλύτερες μειώσεις κατανάλωσης σε συνδυασμό με καλύτερο Throughput από ότι επιτεύχθηκε στις αρχιτεκτονικές χαμηλής κατανάλωσης του κυκλώματος Fugue 256-bit Clock Circles 49.



Σχήμα 7.37: *Κύκλοι ρολογιού που απαιτούνται και η περίοδος του κάθε κύκλου .*



Σχήμα 7.38: *Η σχέση του Throughput με το λόγο Throughput /Λογικά τμήματα .*

ΚΕΦΑΛΑΙΟ 8

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΡΟΤΑΣΕΙΣ ΠΕΡΑΙΤΕΡΩ ΜΕΛΕΤΗΣ

8.1 Συμπεράσματα

Στα πλαίσια της παρούσας εργασίας μελετήθηκαν, αναλύθηκαν και υλοποιήθηκαν δύο συναρτήσεις κατακερματισμού με τη χρήση τεχνικών χαμηλής κατανάλωσης οι οποίες αναλύθηκαν και παρουσιάστηκαν στο κεφάλαιο 4. Επίσης έγινε μια εισαγωγή στα ολοκληρωμένα κυκλώματα και αναλύθηκαν δύο από αυτά τα οποία χρησιμοποιούνται στο εμπόριο.

Ο κάθε αλγόριθμος παρουσιάζει πλεονεκτήματα και μειονεκτήματα από άποψη κατανάλωσης ισχύος, ταχύτητας και απαιτούμενους πόρους υλικού σύμφωνα με το κεφάλαιο 7. Η επιλογή χρήσης κάποιου αλγορίθμου εξαρτάται άμεσα από την εφαρμογή που προορίζεται. Βασικοί παράγοντες αυτής της απόφασης είναι η κατανάλωση ισχύος ή η μεγαλύτερη απόδοση. Οι αρχιτεκτονικές του αλγορίθμου Fugue απαιτούν λιγότερους πόρους για την υλοποίηση τους και μπορούν να επιτύχουν χαμηλά επίπεδα κατανάλωσης σε αντίθεση με τις αρχιτεκτονικές του αλγορίθμου JH. Από την άλλη όπου απαιτούνται υψηλές ταχύτητες οι αρχιτεκτονικές JH είναι καταλληλότερες. Τα παραπάνω συμπεράσματα αντιπροσωπεύουν τις περισσότερες των υλοποιήσεων καθώς υπήρξαν και εξαιρέσεις όπου τα κυκλώματα παρουσίασαν αντίθετα αποτελέσματα από τα επιθυμητά.

8.2 Προτάσεις για περαιτέρω μελέτη

Καθώς έγινε προσπάθεια να παρουσιαστούν και να υλοποιηθούν όσο το δυνατόν περισσότερες τεχνικές χαμηλής κατανάλωσης υπάρχουν περιθώρια βελτιστοποίησης των εκάστοτε αποτελεσμάτων. Είτε αυτά αφορούν την έρευνα ως προς την περαιτέρω μείωση κατανάλωσης είτε τη προσπάθεια αύξησης της ταχύτητας εκτέλεσης των αρχιτεκτονικών είτε τη προσπάθεια μείωσης των απαιτούμενων πόρων.

Όσων αφορά τον αλγόριθμο Fugue παρουσιάστηκαν 3 κυκλώματα τα Fugue 256-bit Clock Circles (49, 36 και 25) τα οποία επιτυγχάνουν διαφορετικές ταχύτητες. Από αυτά τα τρία ως κύκλωμα αναφοράς για την εφαρμογή των τεχνικών χαμηλής κατανάλωσης χρησιμοποιήθηκε το Fugue 256-bit Clock Circles 49. Τα αποτελέσματα του κεφαλαίου 7 έδειξαν όμως ότι αν υπάρχουν διαθέσιμοι πόροι τότε η αρχιτεκτονική Fugue 256-bit Clock Circles 25 είναι ανώτερη καθώς επιτυγχάνει υψηλότερες ταχύτητες και μικρότερη

κατανάλωση. Αυτό συνεπάγεται ότι, πάνω σε αυτό το κύκλωμα ως κύκλωμα εφαρμοστούν επιπλέον τεχνικές χαμηλής κατανάλωσης θα προκύψουν ακόμα μικρότερες καταναλώσεις οι οποίες θα συνδυάζονται και με υψηλές ταχύτητες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Συγγράμματα

1. Πομπόσης Ανδρέας, Παπαδημητρίου Γεώργιος, Ασφάλεια Δικτύων Υπολογιστών. Εκδόσεις: Τζιόλα.
2. Andrew S. Tanenbaum, Δίκτυα Υπολογιστών (Τέταρτη Αμερικάνικη Έκδοση). Εκδόσεις: Κλειδάριθμος
3. James F. Kurose, Keith W. Ross, Δικτύωση Υπολογιστών (Τέταρτη Έκδοση). Εκδόσεις: Γκιούρδα.
4. William Stallings, Επικοινωνίες Υπολογιστών & Δεδομένων (Εκτη Έκδοση). Εκδόσεις: Τζιόλα.
5. Stephen Brown, Zvonko Vranesic, Σχεδιασμός Ψηφιακών Συστημάτων με τη Γλώσσα VHDL. Εκδόσεις: Τζιόλα.
6. Floyd, Digital Fundamentals (Ninth Edition). Εκδόσεις: Pearson.
7. Volnei A. Pedroni, Circuit Design with VHDL. Εκδόσεις: Massachusetts Institute of Technology
8. Maire McLoone, John V. McCanny, System-On-Chip Architectures and Implementations for Private-Key Data Encryption. Εκδόσεις: Springer.

Δημοσιευμένες εργασίες και εγχειρίδια

1. Shai Halevi, William E. Hall, Charanjit S. Jutla, *The Hush Function «Fugue»* . IBM T. J. Watchon Research Center, September 15, 2009.
2. Hongjun Wu, *The Hush Function «JH»* . Institute for Infocomm Research, Singapore September 15, 2009.
3. Paris Kitsos, Nikolas Sklavos, Athanasios N. Skodras, *Low Power FPGA Implementations of 256-bit Luffa Hash Function.* Proceedings of 13th Euromicro Conference on Digital Systems (DSD'10), Lille, France, September 1-3, 2010.
4. P. Kitsos, M. D. Galanis, O Koufopavlou, *Architectures and FPGA Implementations of the 64-bit Misty Block Cipher.* Journal of Circuits, Systems, and Computers Vol. 15, No. 6 (2006) 817–831 © World Scientific Publishing Company.
5. G. Sutter, E. Boemo, *Experiments in Low Power FPGA Design.* School of Engineering, Universidad Autonoma de Madrid, Madrid Spain. 2007.
6. Gustavo Sutter Elias Todorovich, Eduardo Boemo, *Design of Power Aware FPGA-based Systems.* School of Engineering, Universidad Autonoma de Madrid, Madrid Spain.
7. Yan Zhang, Jussi Roivainen, Aarne Mammela, *Clock-Gating in FPGAs: A Novel and Comparative Evaluation.* DSD '06 Proceedings of the 9th EUROMICRO Conference on Digital System Design IEEE Computer Society Washington, DC, USA ©2006.
8. Tomasz S. Czajkowski, Stephen D. Brown, *Using Negative Edge Triggered FFs to Reduce Glitching Power in FPGA Circuits.* Proceeding DAC '07 Proceedings of the 44th annual Design Automation Conference ACM New York, NY, USA ©2007.
9. Mouzam Khan, *Power Optimization in FPGA Designs.* Altera Corporation SNUG San Jose 2006
10. David Kenney, *Energy Efficiency Analysis and Implementation of AES on an FPGA.* University of Waterloo. Waterloo, Ontario, Canada, 2008.
11. Synplicity, *Gated Clock Conversion with Synplicity's Synthesis Products.* Synplicity 2003.
12. Edwin NC Mui Custom R & D Engineer, *Practical Implementation of Rijndael S-Box Using Combinational Logic.* Texco Enterprise Ptd. Ltd.

13. S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J-MSchmidt,A. Szekely, *High-Speed Hardware Implementations of BLAKE, Blue Midnight Wish, CubeHash,ECHO, Fugue, Grostl, Hamsi, JH, Keccak,Lufa, Shabal, SHAvite-3, SIMD, and Skein.* Graz University of Technology, Institute for Applied Information Processing and Communications, In_eldgasse 16a, A{8010 Graz, Austria. Version 2.0, November 11, 2009.
14. DS529 August 19, 2010, Spartan-3A FPGA Family: Data Sheet.
15. DS100 (v5.0) February 6, 2009, Virtex-5 Family Overview: Product Specification.

Ηλεκτρονικές διευθύνσεις

1. http://el.wikipedia.org/wiki/Κρυπτογραφία.
2. http://en.wikipedia.org/wiki/Finite_field_arithmetic.

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

AES	Advanced Encryption Standard
ASIC	Application Specific Integrated Circuit
BICMOS	Bipolar Complementary Metal Oxide Semiconductor
BIT	Binary Digit
BRAM	Block Random Access Memory
CD	Compact Disk
CLB	Configurable Logic Block
CLK	Clock
CMIX	Column Mix
CMOS	Complementary Metal Oxide Semiconductor
CMT	Clock Management Tiles
CPLD	Complex Programmable Logic Device
CRCLS	Circles
DCM	Digital Clock Manager
DEC	Decemical
DES	Data Encryption Standard
DET	Double Edge Trigger
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processing
EEPROM	Electrically Erasable Programmable Read Only Memory
ENB	Enable
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GF	Galois Field
HEX	Hexadecimal
IBM	International Business Machines
IC	Integrated Circuit
IV	Initialization Vector
Kb	Kilo Bit
Kbit	Kilo Bit
LP	Low Power
LSI	Large Scale Integration
LUT	Look Up Table
Mbit	Mega Bit
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MSI	Medium Scale Integration
MUL	Multiplication
MUX	Multiplexer
mW	mili Watt

NIST	National Instute Standards Techonolgy
NMOS	N-type Metal Oxide Semiconductor
nS	nano Second
PAL	Programmable Array Logic
PIC	Programmable Integrated Circuit
PLA	Programmable Logic Array
PLD	Programmable Logic Device
PLL	Phase Locked Loops
PMOS	P-type Metal Oxide Semiconductor
RAM	Random Access Memory
ROR	Rotetion
SBOX	Substitution Box
SMIX	Super Mix
SPLD	Simple Programmable Logic Device
SRAM	Static Random Access Memory
SSI	Small Scale Integration
TTL	Transistor Transistor Logic
ULSI	Ultra Scale Integration
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration
XOR	Exclusive-Or
μS	micro Second

ΠΑΡΑΡΤΗΜΑ Α: Πίνακας Σχημάτων

ΠΑΡΑΡΤΗΜΑ Β: Πίνακας Πινάκων

ΠΑΡΑΡΤΗΜΑ Γ: Αρχεία δεδομένων εισόδων / εξόδων των αλγορίθμων

Παρακάτω δίνονται τα επίσημα αρχεία τα οποία περιέχουν δεδομένα εισόδων και εξόδων των αλγορίθμων Fugue και JH. Τα παρακάτω αρχεία κατατέθηκαν από τους σχεδιαστές των αλγορίθμων στο διαγωνισμό NIST και μπορούν να αναζητηθούν στο διαδικτυακό τόπο του διαγωνισμού. Το μήκος του εισερχόμενου μηνύματος συμβολίζεται με «Len - Length», το μήνυμα προς επεξεργασία συμβολίζεται με «Msg - Message» και με «MD – Message Digest» συμβολίζεται η σύνοψη του μηνύματος. Για τον αλγόριθμο JH όπου υπάρχουν 4 μεγέθη εξόδου δίνονται 4 αρχεία ξεχωριστά ανάλογα το μέγεθος της σύνοψης.

KAT_MCT/ShortMsgKAT_256.txt
Algorithm Name: Fugue
Principal Submitter: Charanjit S. Jutla

Len = 0 Msg = 00
MD = D6EC528980C130AAD1D1ACD28B9DD8DBDEAE0D79EDED1FCA72C2AF9F37C2246F

Len = 1 Msg = 00
MD = 62C7C83D5EE85C4B016E7B74DB181883F5AE440F1154CA86BE1D5EFC8DA1F605

Len = 2 Msg = C0
MD = ACF97C3E41D35295469BFB95B44BBD960F26AE312DC85D46458E5A90B47DE408

Len = 3 Msg = C0
MD = DC82018D6A7582D1B5FA30C433EF900B13EE37532DC7944EDBC0C9369BA3FBBA

Len = 4 Msg = 80
MD = 077BE43564C8DA92D58BACE34DCEFF6212D4CC7CD0418C802C5E9332FE1A1934

Len = 5 Msg = 48
MD = 52BC1C3CAB50F36FF7187261160DBCB11DD7B40CFCCFA84C4769517E70A9F8CA

Len = 6 Msg = 50
MD = 5AD64F3683B2E7A1C0A79C38B95FDF7CDEF7CBAF29EEA5B1E4863171D19DF56D

Len = 7 Msg = 98
MD = 4F7F0AB484F3356DEB289A64BF593BB5F97CBCF08200D002AD6B0B9039F9590E

Len = 8 Msg = CC
MD = B894EB2DF58162F6C48D495F156E73BD086DD13DB407EE38781177BB23D129BB

Len = 9 Msg = 9800
MD = 517DDF37B496AA39F6687F78354285A6DE30A70809CC07D59D6F9BCAB300F302

Len = 32 Msg = C1ECFDFC
MD = 9041D9EDF413CF0A8CFB6AED97C13032315319438BE004685F4BB583F67ACF23

Len = 64 Msg = 4A4F202484512526
MD = 84E8DF742AF4AB3F552A148485A1D27943B57BA748B76A1CDF8E1F054BED3D7B

Len = 96 Msg = 5BE43C90F22902E4FE8ED2D3
MD = D94C33E8312522B6393EBDFB4C99137265C8965782E4D7B4495640BFD6A75760

Len = 128 Msg = 52A608AB21CCDD8A4457A57EDE782176
MD = 3260F5BE7147BE7DB0AEFA571BF0FEF651BBCB1796513572EE66855492E893D7

ShortMsgKAT_224.txt
Algorithm Name: JH
Principal Submitter: Hongjun Wu

Len = 0 Msg = 00
MD = 12C53596FB61AD2865C0A39B7EFE88166F9EB1F5FC5B434B9C45057E

Len = 4 Msg = 80
MD = F262C1A1B41E54944F2205E17B7F540B1CA270D5D1C8B9C3FD7FC1CE

Len = 96 Msg = 5BE43C90F22902E4FE8ED2D3
MD = 250A7CCDBCF6B31DA32F7CFE70E4CBD50B76F42827D3E24C7379A859

ShortMsgKAT_256.txt
Algorithm Name: JH
Principal Submitter: Hongjun Wu

Len = 1 Msg = 00
MD = 5819237F9CF57C77B3AC792A3E4C0C2841FD1F4E2B5BA1EC8ACA27B089642085

Len = 16 Msg = 41FB
MD = 87AC81CDC65A878615DC3421DF7874CCC4F0D41A5D599B2F29678365B73FEDBA

Len = 128 Msg = 52A608AB21CCDD8A4457A57EDE782176
MD = 6AD7FE8C6C492F4CB8436B413F8F39F7ED2254A94BEFA76C52F597258237F141

ShortMsgKAT_384.txt
Algorithm Name: JH
Principal Submitter: Hongjun Wu

Len = 2 Msg = C0
MD =
F0E804A62D57CF1BF2F9C090680D0D9095BF7169131C5A70C8F0526B0B33970B2675CBD10A10F97336
229B2BE44721E8

Len = 32 Msg = C1ECFDFC
MD =
3CD8FAEB48F8A696965958DB5D51086265D5DD9858D7230D02882B115F0AFC388F988005781E7BA0E5
098E842D0B3111

ShortMsgKAT_512.txt
Algorithm Name: JH
Principal Submitter: Hongjun Wu

Len = 3 Msg = C0
MD =
6E8FF17744D90470A877887BF3F947EC8C04194C0FC3BE58531F772460EF44CAC171D6264617860285B3
2182519143CA358289B92FD1EBF59B6C972E241EFDFE

Len = 64 Msg = 4A4F202484512526
MD =

DA9BBF9B39161EDA6B1EC7AF69BC6F848BCC9A926528F35325FDC14EB3EB7ED35A903F0601656846

0355B05AB7E6F6242052A69FBA47F9A5AEC2387D1E514E17

ΠΑΡΑΡΤΗΜΑ Δ: Κώδικας περιγραφής κυκλωμάτων VHDL

Παρακάτω δίνεται ο κώδικας VHDL των αρχιτεκτονικών που περιγράφηκαν στο Κεφάλαιο 6. Αρχικά δίνεται ολόκληρος ο κώδικας του κάθε κυκλώματος αναφοράς και στη συνέχεια δίνεται ο τροποποιημένος κώδικας ο οποίος υλοποιεί την εκάστοτε παραλλαγή της αρχικής αρχιτεκτονικής. Όπου στο Κεφάλαιο 6 δίνεται σχήμα που περιγράφει την αρχιτεκτονική θα συσχετίζεται με το κώδικα VHDL.

Αρχιτεκτονική: Fugue 256-bit Clock Circles 49

Κώδικας του κυκλώματος διασύνδεσης των επιμέρους κυκλωμάτων (Σχήμα 6.1).

```
entity Fugue is
port(   dataIn :in std_logic_vector(127 downto 0);
        messageLength :in std_logic_vector(7 downto 0);
        clk,reset,enable : in std_logic;
        hashReady : out std_logic;
        FugueOut : out std_logic_vector(255 downto 0));
end Fugue;

architecture Behavioral of Fugue is
        Component Padder
        port(       reset :in std_logic;
                    dataIn :in std_logic_vector(127 downto 0);
                    messageLength : in std_logic_vector(7 downto 0);
                    messagesCircles : out std_logic_vector(3 downto 0);
                    paddedDataOut : out std_logic_vector(191 downto 0)
                    );
        end component;
        Component IV
        port(
            reset : in std_logic;
                        IVOut : out std_logic_vector(255 downto 0)
          );
        end component;
        Component T_I_X
        port(
                    reset : in std_logic;
                    messageI : in std_logic_vector(31 downto 0);
                    SstateIn : in std_logic_vector(959 downto 256);
                    mux2x1IVOut : in std_logic_vector(255 downto 0);
                    SstateOut : out std_logic_vector(959 downto 0)
                   );
        end component;
        Component ROR3CMIX
        port(
                    reset : in std_logic;
                    SstateIn : in std_logic_vector(959 downto 0);
                    SstateOut.: out std_logic_vector(959 downto 0)
                    );
        end component;
        Component SMIX
```

```
port(
        reset,resetSboxes : in std_logic;
        SstateIn : in std_logic_vector(959 downto 0);
        SstateOut : out std_logic_vector(959 downto 0)
      );
end component;
Component Controller
port(
        clk,reset,enable : in std_logic;
        messagesCircles : in std_logic_vector(3 downto 0);
        resetSboxes,hashReady : out std_logic;
        selectionMUX2x1_IV,selectionMUX2x1 : out std_logic;
        selectionMUX3x1 : out std_logic_vector(1 downto 0);
        selectionMUX6x1 : out std_logic_vector(2 downto 0)
      );
end component;
Component ROR15
port(
        reset : in std_logic;
        SstateIn : in std_logic_vector(959 downto 0);
        SstateOut : out std_logic_vector(959 downto 0)
        );
end component;
Component ROR14
port(
   reset : in std_logic;
        SstateIn : in std_logic_vector(959 downto 0);
        SstateOut : out std_logic_vector(959 downto 0)
      );
end component;

Component OutPutBlock
port(
        reset : in std_logic;
        S0In,S1In,S2In,S3In,S4In,S15In,S16In,S17In,S18In : in std_logic_vector(31 downto 0);
   dataOut : out std_logic_vector(255 downto 0)
        );
end component;
Component SstateRegister
port(
        clk,reset,enable,hashReady : in std_logic;
        SstateIn : in std_logic_vector(959 downto 0);
        SstateOut : out std_logic_vector(959 downto 0)
      );
end component;
Component MUX2x1_IV
port(
        selection : in std_logic;
        Input1,Input2 : in std_logic_vector(255 downto 0);
        Output : out std_logic_vector(255 downto 0)
   );
end component;
Component MUX2x1
port(
        selection : in std_logic;
        Input1,Input2 : in std_logic_vector(959 downto 0);
        Output : out std_logic_vector(959 downto 0)
   );
end component;
Component MUX3x1
```

```
        port(
                selection : in std_logic_vector(1 downto 0);
                Input1,Input2,Input3 : in std_logic_vector(959 downto 0);
                Output : out std_logic_vector(959 downto 0)
          );
        end component;
        Component MUX6x1
        port(
                selection : in std_logic_vector(2 downto 0);
                Input1,Input2,Input3,Input4,Input5,Input6 : in std_logic_vector(31 downto 0);
                Output : out std_logic_vector(31 downto 0)
           );
        end component;

-- S I G N A L S for padder component
SIGNAL padderOut : std_logic_vector(191 downto 0);
SIGNAL messagesCircles :  std_logic_vector(3 downto 0);

-- S I G N A L S for IV component
 SIGNAL IVOut : std_logic_vector(255 downto 0);

-- S I G N A L S for T_I_X component
 SIGNAL messageWord : std_logic_vector(31 downto 0);
 SIGNAL TIX_Out : std_logic_vector(959 downto 0);

-- S I G N A L S for ROR3CMIX component
SIGNAL ROR3CMIXOut : std_logic_vector(959 downto 0);

-- S I G N A L S for ROR15 component
SIGNAL ROR15Out : std_logic_vector(959 downto 0);

-- S I G N A L S for ROR14 component
SIGNAL ROR14Out : std_logic_vector(959 downto 0);

-- S I G N A L S for SMIX component
SIGNAL SMIXOut :  std_logic_vector(959 downto 0);

-- S I G N A L S for OutPutBlock component

-- S I G N A L S for SstateRegister component
SIGNAL SstateOut :  std_logic_vector(959 downto 0);

-- S I G N A L S for Cotroller component
  SIGNAL resetSboxes,hashReadySig : std_logic;

-- S I G N A L S for MUX components
  SIGNAL mux2x1IVOut : std_logic_vector(255 downto 0);
  SIGNAL mux2x1Out : std_logic_vector(959 downto 0);
  SIGNAL mux3x1Out : std_logic_vector(959 downto 0);
  SIGNAL selectionMUX2x1_IV,selectionMUX2x1 : std_logic;
  SIGNAL selectionMUX3x1 : std_logic_vector(1 downto 0);
  SIGNAL selectionMUX6x1 : std_logic_vector(2 downto 0);
begin
------------------------------PORT MAPS----------------------------------------------------------------------------------
Padder_comp : Padder port map(reset,dataIn,messageLength,messagesCircles,padderOut);

IV_comp : IV port map(reset,IVOut);

T_I_X_comp : T_I_X port map(reset,messageWord,SstateOut(959 downto 256), mux2x1IVOut, TIX_Out);
```

```
ROR3CMIX_comp : ROR3CMIX port map(reset,mux2x1Out,ROR3CMIXOut);

ROR14_comp : ROR14 port map(reset,SstateOut,ROR14Out);

ROR15_comp : ROR15 port map(reset,SstateOut,ROR15Out);

SMIX_comp : SMIX port map(reset,resetSboxes,mux3x1Out,SMIXOut);

SstateRegister_comp : SstateRegister port map(clk,reset,enable,hashReadySig,SMIXOut,SstateOut);

--                                                  Sstate          S0                      S1
OutputBlock_comp : OutPutBlock port map(reset,SstateOut(959 downto 928),SstateOut(927 downto 896),
--              S2                          S3                      S4
 SstateOut(895 downto 864), SstateOut(863 downto 832),SstateOut(831 downto 800),
--              S15                         S16                     S17                     S18
SstateOut(479 downto 448),SstateOut(447 downto 416),SstateOut(415 downto 384),SstateOut(383 downto
352),FugueOut);

Controller_comp : Controller port map(clk,reset,enable,messagesCircles, resetSboxes,hashReadySig,
selectionMUX2x1_IV,selectionMUX2x1,selectionMUX3x1,selectionMUX6x1);

MUX2x1_IV_comp : MUX2x1_IV port map(selectionMUX2x1_IV,SstateOut(255 downto 0), IVOut,
mux2x1IVOut);

MUX6x1_comp : MUX6x1 port map(selectionMUX6x1,padderOut(191 downto 160),padderOut(159 downto
128),padderOut(127 downto 96),padderOut(95 downto 64),padderOut(63 downto 32),padderOut(31 downto 0),
messageWord);

MUX2x1_comp : MUX2x1 port map(selectionMUX2x1,TIX_Out,SstateOut,mux2x1Out);
MUX3x1_comp : MUX3x1 port map(selectionMUX3x1,ROR3CMIXOut,ROR15Out,ROR14Out,mux3x1Out);
-----------------------------------------END PORT MAPS-----------------------------------------------------

hashReady <= hashReadySig;

end Behavioral;
```

Κώδικας του κυκλώματος Padder (Σχήμα 6.2).

```
entity Padder is
port( reset :in std_logic;
      dataIn :in std_logic_vector(127 downto 0);
      messageLength : in std_logic_vector(7 downto 0);
      messagesCircles : out std_logic_vector(3 downto 0);
      paddedDataOut : out std_logic_vector(191 downto 0)
      );
end Padder;

architecture Behavioral of Padder is
begin

Padding: process(reset,dataIn,messageLength)
        begin
                if(reset = '0')then
----------START PADDING-----------------------------------------------------------------------------------------

                        if(messageLength = X"00")then
                                paddedDataOut <= dataIn & X"0000000000000000";
                                messagesCircles <= "0100";
```

```
                elsif(messageLength < 33)then
                        paddedDataOut(191 downto 160) <= dataIn(127 downto 96);
                        paddedDataOut(159 downto 104) <= (others => '0');
                        paddedDataOut(103 downto 96) <= messageLength;
                        paddedDataOut(95 downto 0) <= (others => '0');
                        messagesCircles <= "0110";

                elsif(messageLength > 32 and messageLength < 65)then
                        paddedDataOut(191 downto 128) <= dataIn(127 downto 64);
                        paddedDataOut(127 downto 72) <= (others => '0');
                        paddedDataOut(71 downto 64) <= messageLength;
                        paddedDataOut(63 downto 0) <= (others => '0');
                        messagesCircles <= "1000";

                elsif(messageLength > 64 and messageLength < 97)then
                        paddedDataOut(191 downto 96) <= dataIn(127 downto 32);
                        paddedDataOut(95 downto 40) <= (others => '0');
                        paddedDataOut(39 downto 32) <= messageLength;
                        paddedDataOut(31 downto 0) <= (others => '0');
                        messagesCircles <= "1010";

                elsif(messageLength > 96 and messageLength < 129)then
                        paddedDataOut(191 downto 64) <= dataIn(127 downto 0);
                        paddedDataOut(63 downto 8) <= (others => '0');
                        paddedDataOut(7 downto 0) <= messageLength;
                        messagesCircles <= "1100";
                else
                        paddedDataOut <= (others => '0');
                        messagesCircles <= "0000";
                end if;
----------END PADDING----------
        else
--                      <-              192*0                   ->
                paddedDataOut <= (others => '0');
                messagesCircles <= "0100";
        end if;
       end process;
end Behavioral;
```

Κώδικας του κυκλώματος TIX (Σχήμα 6.3).

```
entity T_I_X is
port
 ( reset : in std_logic;
  messageI : in std_logic_vector(31 downto 0);
  SstateIn : in std_logic_vector(959 downto 256);
  mux2x1IVOut : in std_logic_vector(255 downto 0);
  SstateOut : out std_logic_vector(959 downto 0));
end T_I_X;

architecture Behavioral of T_I_X is
SIGNAL sigXOR1,sigXOR2,sigXOR3 : std_logic_vector(31 downto 0);
SIGNAL signal4 : std_logic_vector(959 downto 0);
begin

sigXOR1 <= SstateIn(639 downto 608) xor SstateIn(959 downto 928); -- sigXOR1 <= S10 xor S0;
sigXOR2 <= SStateIn(703 downto 672) xor messageI;               -- sigXOR2 <= S8 xor messageI;
sigXOR3 <= SstateIn(927 downto 896) xor mux2x1IVOut(191 downto 160); -- sigXOR3 <= S1 xor S24;
```

```
signal4 <= messageI & sigXOR3 & SstateIn(895 downto 704) & sigXOR2 & SstateIn(671 downto 640) &
sigXOR1 & SstateIn(607 downto 256) & mux2x1IVOut(255 downto 0);

        TIX : process(reset,SstateIn,mux2x1IVOut,signal4)
                begin
                        if(reset = '0')then
                          SstateOut <= signal4;
                        else
                          SstateOut <= (others => '0');
                        end if;
                end process;
end Behavioral;
```

Κώδικας του κυκλώματος ROR 3 CMIX (Σχήμα 6.4).

```
entity ROR3CMIX is
port( reset : in std_logic;
      SstateIn : in std_logic_vector(959 downto 0);
      SstateOut : out std_logic_vector(959 downto 0));
end ROR3CMIX;

architecture Behavioral of ROR3CMIX is
SIGNAL sigROR3,signal8 : std_logic_vector(959 downto 0);
SIGNAL sigXOR1,sigXOR2,sigXOR3,sigXOR4,sigXOR5,sigXOR6 : std_logic_vector(31 downto 0);

begin
sigROR3 <= SstateIn(95 downto 0) & SstateIn(959 downto 96); --sigROR3 <= ROR3 Sstate;

sigXOR1 <= sigROR3(959 downto 928) xor sigROR3(831 downto 800); --sigXOR1 <= S0 xor S4;
sigXOR2 <= sigROR3(927 downto 896) xor sigROR3(799 downto 768); --sigXOR2 <= S1 xor S5;
sigXOR3 <= sigROR3(895 downto 864) xor sigROR3(767 downto 736); --sigXOR3 <= S2 xor S6;
sigXOR4 <= sigROR3(479 downto 448) xor sigROR3(831 downto 800); --sigXOR4 <= S15 xor S4;
sigXOR5 <= sigROR3(447 downto 416) xor sigROR3(799 downto 768); --sigXOR5 <= S16 xor S5;
sigXOR6 <= sigROR3(415 downto 384) xor sigROR3(767 downto 736); --sigXOR6 <= S17 xor S6;

signal8 <= sigXOR1 & sigXOR2 & sigXOR3 & sigROR3(863 downto 480) & sigXOR4 & sigXOR5 &
sigXOR6 & sigROR3(383 downto 0);

ROR3CMIX : process(reset,signal8)
        begin
          if(reset = '0')then
                  SstateOut <= signal8;
          else
                  SstateOut <= (others => '0');
          end if;
        end process;
end Behavioral;
```

Κώδικας του κυκλώματος XOR ROR 15 (Σχήμα 6.5).

```
entity ROR15 is
port( reset : in std_logic;
      SstateIn : in std_logic_vector(959 downto 0);
      SstateOut : out std_logic_vector(959 downto 0));
end ROR15;
```

```
architecture Behavioral of ROR15 is
SIGNAL sigXOR1,sigXOR2 : std_logic_vector(31 downto 0);
SIGNAL signal3 : std_logic_vector(959 downto 0);
begin
sigXOR1 <= SstateIn(831 downto 800) xor SstateIn(959 downto 928); --sigXOR1 <= S4 xor S0;
sigXOR2 <= SstateIn(479 downto 448) xor SstateIn(959 downto 928); --sigXOR1 <= S15 xor S0;

signal3 <= SstateIn(959 downto 832) & sigXOR1 & SstateIn(799 downto 480) & sigXOR2 & SstateIn(447
downto 0);

ROR15 : process(reset,SstateIn,sigXOR1,sigXOR2,signal3)
    begin
        if(reset = '0')then
          SstateOut <= signal3(479 downto 0) & signal3(959 downto 480); --SstateOut <= ROR15 signal3;
        else
          SstateOut <= (others => '0');
        end if;
    end process;
end Behavioral;
```

Κώδικας του κυκλώματος XOR ROR 14 (Σχήμα 6.6).

```
entity ROR14 is
port( reset : in std_logic;
    SstateIn : in std_logic_vector(959 downto 0);
    SstateOut : out std_logic_vector(959 downto 0));
end ROR14;

architecture Behavioral of ROR14 is
SIGNAL sigXOR1,sigXOR2 : std_logic_vector(31 downto 0);
SIGNAL signal3 : std_logic_vector(959 downto 0);
begin

sigXOR1 <= SstateIn(831 downto 800) xor SstateIn(959 downto 928); --sigXOR1 <= S4 xor S0;
sigXOR2 <= SstateIn(447 downto 416) xor SstateIn(959 downto 928); --sigXOR1 <= S16 xor S0;

signal3 <= SstateIn(959 downto 832) & sigXOR1 & SstateIn(799 downto 448) & sigXOR2 & SstateIn(415
downto 0);

ROR14 : process(reset,SstateIn,sigXOR1,sigXOR2,signal3)
    begin
        if(reset = '0')then
          SstateOut <= signal3(447 downto 0) & signal3(959 downto 448); --SstateOut <= ROR14 signal3;
        else
          SstateOut <= (others => '0');
        end if;
    end process;
end Behavioral;
```

Κώδικας του κυκλώματος διασύνδεσης των επιμέρους κυκλωμάτων ενός SBox.

```
entity SBOXv2 is
port( trsfmtnByteIn : in std_logic_vector(7 downto 0);
    trsfmtnByteOut : out std_logic_vector(7 downto 0));
end SBOXv2;
```

```
architecture Behavioral of SBOXv2 is

        component isomorphicMapping
        port(isoMapByteIn : in std_logic_vector(7 downto 0);
             isoMapByteOut : out std_logic_vector(7 downto 0));
        end component;

        component isomorphicMappingINVERSE
        port(isoMapByteIn : in std_logic_vector(7 downto 0);
             isoMapByteOut : out std_logic_vector(7 downto 0));
        end component;

        component sqrt_AndMultConstant_L
        port(bit4In : in std_logic_vector(3 downto 0);
             bit4Out : out std_logic_vector(3 downto 0));
        end component;

        component inversionGF16
        port(bit4In : in std_logic_vector(3 downto 0);
             bit4Out : out std_logic_vector(3 downto 0));
        end component;

        component mul4x4GF
        port(mulA,mulB : in std_logic_vector(3 downto 0);
             mulOut : out std_logic_vector(3 downto 0));
        end component;

        component affineTrans
        port(byteIn : in std_logic_vector(7 downto 0);
             byteOut : out std_logic_vector(7 downto 0));
  end component;

signal isoMapOut,sig,isoMapINVOut : std_logic_vector(7 downto 0);
signal sigXor1Out,sigXor2Out,sqrt_multOut,mulComp1Out,invGF16Out,mulComp2Out,mulComp3Out :
std_logic_vector(3 downto 0);
begin
----------PORT MAPS----------
isoMapping_Comp : isomorphicMapping port map(trsfmtnByteIn,isoMapOut);

sqrt_AndMultConstant_L_Comp : sqrt_AndMultConstant_L port map(isoMapOut(7 downto 4),sqrt_multOut);

mul4x4GF_Comp1 : mul4x4GF port map(sigXor1Out,isoMapOut(3 downto 0),mulComp1Out);

inversionGF16_Comp : inversionGF16 port map(sigXor2Out,invGF16Out);

mul4x4GF_Comp2 : mul4x4GF port map(isoMapOut(7 downto 4),invGF16Out,mulComp2Out);
mul4x4GF_Comp3 : mul4x4GF port map(sigXor1Out,invGF16Out,mulComp3Out);

isoMappingInverse_Comp : isomorphicMappingINVERSE port map(sig,isoMapINVOut);

affineTrans_Comp : affineTrans port map(isoMapINVOut,trsfmtnByteOut);
----------END PORT MAPS----------

sigXor1Out <= isoMapOut(7 downto 4) xor isoMapOut(3 downto 0);
sigXor2Out <= sqrt_multOut xor mulComp1Out;

sig <= mulComp2Out & mulComp3Out;
end Behavioral;
```

Κώδικας του κυκλώματος ισόμορφης αντικατάστασης σε σύνθετα πεδία.(Σχήμα 6.8).

```
entity isomorphicMapping is
port(isoMapByteIn : in std_logic_vector(7 downto 0);
     isoMapByteOut : out std_logic_vector(7 downto 0));
end isomorphicMapping;

architecture Behavioral of isomorphicMapping is
SIGNAL sig6_a,sig6_b,sig6_c,sig6_d : std_logic;
SIGNAL sig5_a,sig5_b : std_logic;
SIGNAL sig4_a,sig4_b,sig4_c : std_logic;
SIGNAL sig3_a,sig3_b : std_logic;
SIGNAL sig2_a,sig2_b,sig2_c : std_logic;
SIGNAL sig1_a : std_logic;
SIGNAL sig_a : std_logic;
begin
isoMapByteOut(7) <= isoMapByteIn(7) xor isoMapByteIn(5);

sig6_a <= isoMapByteIn(7) xor isoMapByteIn(6);
sig6_b <= isoMapByteIn(4) xor isoMapByteIn(3);
sig6_c <= isoMapByteIn(2) xor isoMapByteIn(1);
sig6_d <= sig6_a xor sig6_b;
isoMapByteOut(6) <= sig6_d xor sig6_c;

sig5_a <= isoMapByteIn(7) xor isoMapByteIn(5);
sig5_b <= isoMapByteIn(3) xor isoMapByteIn(2);
isoMapByteOut(5) <= sig5_a xor sig5_b;

sig4_a <= isoMapByteIn(7) xor isoMapByteIn(5);
sig4_b <= isoMapByteIn(3) xor isoMapByteIn(2);
sig4_c <= sig4_a xor sig4_b;
isoMapByteOut(4) <= sig4_c xor isoMapByteIn(1);

sig3_a <= isoMapByteIn(7) xor isoMapByteIn(6);
sig3_b <= isoMapByteIn(2) xor isoMapByteIn(1);
isoMapByteOut(3) <= sig3_a xor sig3_b;

sig2_a <= isoMapByteIn(7) xor isoMapByteIn(4);
sig2_b <= isoMapByteIn(3) xor isoMapByteIn(2);
sig2_c <= sig2_a xor sig2_b;
isoMapByteOut(2) <= sig2_c xor isoMapByteIn(1);

sig1_a <= isoMapByteIn(6) xor isoMapByteIn(4);
isoMapByteOut(1) <= sig1_a xor isoMapByteIn(1);

sig_a <= isoMapByteIn(6) xor isoMapByteIn(1);
isoMapByteOut(0) <= sig_a xor isoMapByteIn(0);

end Behavioral;
```

Κώδικας του κυκλώματος αντιστροφής ισόμορφης αντικατάστασης σε $GF(2^8)$ (Σχήμα 6.9).

```
entity isomorphicMappingINVERSE is
port(isoMapByteIn : in std_logic_vector(7 downto 0);
     isoMapByteOut : out std_logic_vector(7 downto 0));
end isomorphicMappingINVERSE;
```

```
architecture Behavioral of isomorphicMappingINVERSE is
SIGNAL sig7_a,sig7_b : std_logic;
SIGNAL sig5_a : std_logic;
SIGNAL sig4_a,sig4_b,sig4_c : std_logic;
SIGNAL sig3_a,sig3_b,sig3_c : std_logic;
SIGNAL sig2_a,sig2_b,sig2_c : std_logic;
SIGNAL sig_a,sig_b,sig_c : std_logic;
begin

sig7_a <= isoMapByteIn(7) xor isoMapByteIn(6);
sig7_b <= isoMapByteIn(5) xor isoMapByteIn(1);
isoMapByteOut(7) <= sig7_a xor sig7_b;

isoMapByteOut(6) <= isoMapByteIn(6) xor isoMapByteIn(2);

sig5_a <= isoMapByteIn(6) xor isoMapByteIn(5);
isoMapByteOut(5) <= sig5_a xor isoMapByteIn(1);

sig4_a <= isoMapByteIn(6) xor isoMapByteIn(5);
sig4_b <= isoMapByteIn(4) xor isoMapByteIn(2);
sig4_c <= sig4_a xor sig4_b;
isoMapByteOut(4) <= sig4_c xor isoMapByteIn(1);

sig3_a <= isoMapByteIn(5) xor isoMapByteIn(4);
sig3_b <= isoMapByteIn(3) xor isoMapByteIn(2);
sig3_c <= sig3_a xor sig3_b;
isoMapByteOut(3) <= sig3_c xor isoMapByteIn(1);

sig2_a <= isoMapByteIn(7) xor isoMapByteIn(4);
sig2_b <= isoMapByteIn(3) xor isoMapByteIn(2);
sig2_c <= sig2_a xor sig2_b;
isoMapByteOut(2) <= sig2_c xor isoMapByteIn(1);

isoMapByteOut(1) <= isoMapByteIn(5) xor isoMapByteIn(4);

sig_a <= isoMapByteIn(6) xor isoMapByteIn(5);
sig_b <= isoMapByteIn(4) xor isoMapByteIn(2);
sig_c <= sig_a xor sig_b;
isoMapByteOut(0) <= sig_c xor isoMapByteIn(0);

end Behavioral;
```

Κώδικας κυκλώματος πολλαπλασιασμού 4x4 σε GF(2^8) (Σχήμα 6.10).

```
entity mul4x4GF is
port(mulA,mulB : in std_logic_vector(3 downto 0);
     mulOut : out std_logic_vector(3 downto 0));
end mul4x4GF;

architecture Behavioral of mul4x4GF is
        component mul2x2GF
        Port ( mulQ,mulW : in  STD_LOGIC_VECTOR (1 downto 0);
          bit2Out : out  STD_LOGIC_VECTOR (1 downto 0));
         end component;

SIGNAL sigAComp2In,sigBComp2In,comp1Out,comp2Out,comp3Out,sig,sig2,sig3 : std_logic_vector(1
downto 0);
begin
```

```
mul2x2_Comp1 : mul2x2GF port map(mulA(3 downto 2),mulB(3 downto 2),comp1Out);
mul2x2_Comp2 : mul2x2GF port map(sigAComp2In,sigBComp2In,comp2Out);
mul2x2_Comp3 : mul2x2GF port map(mulA(1 downto 0),mulB(1 downto 0),comp3Out);

sigAComp2In <= mulA(3 downto 2) xor mulA(1 downto 0);
sigBComp2In <= mulB(3 downto 2) xor mulB(1 downto 0);

sig = comp1Out x 2     in GF2
sig(1) <= comp1Out(1) xor comp1Out(0); sig(0) <= comp1Out(1);

sig2 <= comp2Out xor comp3Out;
sig3 <= comp3Out xor sig;
mulOut <= sig2 & sig3;

end Behavioral;
```

Κώδικας κυκλώματος πολλαπλασιασμού 2x2 σε GF(2^8) (Σχήμα 6.10).

```
entity mul2x2GF is
   Port (mulQ,mulW : in  STD_LOGIC_VECTOR (1 downto 0);
         bit2Out : out  STD_LOGIC_VECTOR (1 downto 0));
end mul2x2GF;

architecture Behavioral of mul2x2GF is
SIGNAL sig : std_logic_vector(4 downto 0);
SIGNAL sigXOR : std_logic;
begin

sig(4) <= mulQ(1) and mulW(1);
sig(3) <= mulQ(0) and mulW(1);
sig(2) <= mulQ(1) and mulW(0);
sig(1) <= mulQ(1) and mulW(1);
sig(0) <= mulQ(0) and mulW(0);

sigXOR <= sig(4) xor sig(3);
bit2Out(1) <= sigXOR xor sig(2);
bit2Out(0) <= sig(1) xor sig(0);

end Behavioral;
```

Κώδικας κυκλώματος τετραγωνισμού και πολλαπλασιασμού με τη σταθερά λ (Σχήμα 6.11).

```
entity sqrt_AndMultConstant_L is
port(bit4In : in std_logic_vector(3 downto 0);
     bit4Out : out std_logic_vector(3 downto 0));
end sqrt_AndMultConstant_L;

architecture Behavioral of sqrt_AndMultConstant_L is
SIGNAL sqrt : std_logic_vector(2 downto 0);
SIGNAL sigXORsqrt,sigXORMult,sigXORMult2 : std_logic;
begin

sqrt(2) <= bit4In(3) xor bit4In(2);
sqrt(1) <= bit4In(2) xor bit4In(1);
```

```
sigXORsqrt <= bit4In(3) xor bit4In(1);
sqrt(0) <= sigXORsqrt xor bit4In(0);

bit4Out(3) <= sqrt(2) xor sqrt(0);
sigXORMult <= bit4In(3) xor sqrt(2);
sigXORMult2 <= sqrt(1) xor sqrt(0);
bit4Out(2) <=  sigXORMult xor sigXORMult2;
bit4Out(1) <= bit4In(3);
bit4Out(0) <= sqrt(2);

end Behavioral;
```

Κώδικας κυκλώματος αντιστροφής σε GF(2^4) (Σχήμα 6.11).

```
entity inversionGF16 is
port(bit4In : in std_logic_vector(3 downto 0);
     bit4Out : out std_logic_vector(3 downto 0));
end inversionGF16;

architecture Behavioral of inversionGF16 is
begin

bit4Out <= X"0" when bit4In = X"0" else
           X"1" when bit4In = X"1" else
           X"3" when bit4In = X"2" else
           X"2" when bit4In = X"3" else
           X"F" when bit4In = X"4" else
           X"C" when bit4In = X"5" else
           X"9" when bit4In = X"6" else
           X"B" when bit4In = X"7" else
           X"A" when bit4In = X"8" else
           X"6" when bit4In = X"9" else
           X"8" when bit4In = X"A" else
           X"7" when bit4In = X"B" else
           X"5" when bit4In = X"C" else
           X"E" when bit4In = X"D" else
           X"D" when bit4In = X"E" else
           X"4" when bit4In = X"F" else
           X"0";

end Behavioral;
```

Κώδικας υλοποίησης του Σχήματος 5.5 (Σχήμα 6.12).

```
entity affineTrans is
port(byteIn : in std_logic_vector(7 downto 0);
     byteOut : out std_logic_vector(7 downto 0));
end affineTrans;

architecture Behavioral of affineTrans is

SIGNAL sig7_a,sig7_b,sig7_c,sig7_d : std_logic;
SIGNAL sig6_a,sig6_b,sig6_c,sig6_d : std_logic;
SIGNAL sig5_a,sig5_b,sig5_c,sig5_d : std_logic;
SIGNAL sig4_a,sig4_b,sig4_c,sig4_d : std_logic;
SIGNAL sig3_a,sig3_b,sig3_c,sig3_d : std_logic;
SIGNAL sig2_a,sig2_b,sig2_c,sig2_d : std_logic;
SIGNAL sig1_a,sig1_b,sig1_c,sig1_d : std_logic;
```

```
SIGNAL sig_a,sig_b,sig_c,sig_d : std_logic;
begin

sig7_a <= byteIn(7) xor byteIn(6);
sig7_b <= byteIn(5) xor byteIn(4);
sig7_c <= sig7_a xor sig7_b;
sig7_d <= sig7_c xor byteIn(3);

sig6_a <= byteIn(6) xor byteIn(5);
sig6_b <= byteIn(4) xor byteIn(3);
sig6_c <= sig6_a xor sig6_b ;
sig6_d <= sig6_c xor byteIn(2);

sig5_a <= byteIn(5) xor byteIn(4);
sig5_b <= byteIn(3) xor byteIn(2);
sig5_c <= sig5_a xor sig5_b;
sig5_d <= sig5_c xor byteIn(1);

sig4_a <= byteIn(4) xor byteIn(3);
sig4_b <= byteIn(2) xor byteIn(1);
sig4_c <= sig4_a xor sig4_b;
sig4_d <= sig4_c xor byteIn(0);

sig3_a <= byteIn(7) xor byteIn(3);
sig3_b <= byteIn(2) xor byteIn(1);
sig3_c <= sig3_a xor sig3_b;
sig3_d <= sig3_c xor byteIn(0);

sig2_a <= byteIn(7) xor byteIn(6);
sig2_b <= byteIn(2) xor byteIn(1);
sig2_c <= sig2_a xor sig2_b;
sig2_d <= sig2_c xor byteIn(0);

sig1_a <= byteIn(7) xor byteIn(6);
sig1_b <= byteIn(5) xor byteIn(1);
sig1_c <= sig1_a xor sig1_b;
sig1_d <= sig1_c xor byteIn(0);
sig_a <= byteIn(7) xor byteIn(6);
sig_b <= byteIn(5) xor byteIn(4);
sig_c <= sig_a xor sig_b;
sig_d <= sig_c xor byteIn(0);

byteOut(7) <= sig7_d
byteOut(6) <= sig6_d xor '1';
byteOut(5) <= sig5_d xor '1';
byteOut(4) <= sig4_d
byteOut(3) <= sig3_d
byteOut(2) <= sig2_d
byteOut(1) <= sig1_d xor '1';
byteOut(0) <= sig_d xor '1';

end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος διασύνδεσης των 16 SBoxes (Σχήμα 6.13).

```
entity trsfmtnSBOX is
port(resetSboxes : in std_logic;
     SstateIn : in std_logic_vector(959 downto 832);
```

```
    SstateOut :  out std_logic_vector(959 downto 832));
end trsfmtnSBOX;

architecture Behavioral of trsfmtnSBOX is
        Component SBOXv2
        port(trsfmtnByteIn : in std_logic_vector(7 downto 0);
             trsfmtnByteOut : out std_logic_vector(7 downto 0));
        end component;

SIGNAL initSstateForSboxes : std_logic_vector(959 downto 832);
begin
sbox_comp1 : SBOXv2 port map(initSstateForSboxes(959 downto 952),SstateOut(959 downto 952));
sbox_comp2 : SBOXv2 port map(initSstateForSboxes(951 downto 944),SstateOut(951 downto 944));
sbox_comp3 : SBOXv2 port map(initSstateForSboxes(943 downto 936),SstateOut(943 downto 936));
sbox_comp4 : SBOXv2 port map(initSstateForSboxes(935 downto 928),SstateOut(935 downto 928));

sbox_comp5 : SBOXv2 port map(initSstateForSboxes(927 downto 920),SstateOut(927 downto 920));
sbox_comp6 : SBOXv2 port map(initSstateForSboxes(919 downto 912),SstateOut(919 downto 912));
sbox_comp7 : SBOXv2 port map(initSstateForSboxes(911 downto 904),SstateOut(911 downto 904));
sbox_comp8 : SBOXv2 port map(initSstateForSboxes(903 downto 896),SstateOut(903 downto 896));

sbox_comp9 : SBOXv2 port map(initSstateForSboxes(895 downto 888),SstateOut(895 downto 888));
sbox_comp10 : SBOXv2 port map(initSstateForSboxes(887 downto 880),SstateOut(887 downto 880));
sbox_comp11 : SBOXv2 port map(initSstateForSboxes(879 downto 872),SstateOut(879 downto 872));
sbox_comp12 : SBOXv2 port map(initSstateForSboxes(871 downto 864),SstateOut(871 downto 864));

sbox_comp13 : SBOXv2 port map(initSstateForSboxes(863 downto 856),SstateOut(863 downto 856));
sbox_comp14 : SBOXv2 port map(initSstateForSboxes(855 downto 848),SstateOut(855 downto 848));
sbox_comp15 : SBOXv2 port map(initSstateForSboxes(847 downto 840),SstateOut(847 downto 840));
sbox_comp16 : SBOXv2 port map(initSstateForSboxes(839 downto 832),SstateOut(839 downto 832));

--Initialization     for Sboxes with value 0x52. Because   0x52 --> SBOX = 0x00,  0x00 --> SBOX = 0x63.
initSstateForSboxes <= SstateIn when (resetSboxes = '0') else X"52525252525252525252525252525252";

end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος Super-Mix (Σχήμα 6.14).

```
entity multArray is
port(SstateIn : in std_logic_vector(959 downto 832);
     SstateOut :  out std_logic_vector(959 downto 832));
end multArray;

architecture Behavioral of multArray is
        Component mul4
                port(numIn : in std_logic_vector(7 downto 0);
                     numOut : out std_logic_vector(7 downto 0));
        end component;

        Component mul5
                port(numIn : in std_logic_vector(7 downto 0);
                     numOut : out std_logic_vector(7 downto 0));
        end component;

        Component mul6
                port(numIn : in std_logic_vector(7 downto 0);
                     numOut : out std_logic_vector(7 downto 0));
        end component;
```

```
        Component mul7
                port(numIn : in std_logic_vector(7 downto 0);
                     numOut : out std_logic_vector(7 downto 0));
        end component;

SIGNAL mulOutRow0_x4,mulOutRow0_x7,mulOutRow1_x4,mulOutRow1_x7,mulOutRow2_x4,
mulOutRow2_x7,mulOutRow3_x4,mulOutRow3_x7 : std_logic_vector(7 downto 0);

SIGNAL mulOutRow4_x4,mulOutRow4_x7,mulOutRow5_x4,mulOutRow5_x7,mulOutRow6_x4,
mulOutRow6_x7,mulOutRow7_x4,mulOutRow7_x7 : std_logic_vector(7 downto 0);

SIGNAL mulOutRow8_x7,mulOutRow8_x7_b,mulOutRow8_x7_c,mulOutRow8_x6,mulOutRow8_x4 :
std_logic_vector(7 downto 0);

SIGNAL mulOutRow9_x7,mulOutRow9_x7_b,mulOutRow9_x7_c,mulOutRow9_x6,mulOutRow9_x4 :
std_logic_vector(7 downto 0);

SIGNAL mulOutRow10_x7,mulOutRow10_x7_b,mulOutRow10_x7_c,mulOutRow10_x6,mulOutRow10_x4 :
std_logic_vector(7 downto 0);

SIGNAL mulOutRow11_x7,mulOutRow11_x7_b,mulOutRow11_x7_c,mulOutRow11_x6,mulOutRow11_x4 :
std_logic_vector(7 downto 0);

SIGNAL mulOutRow12_x4,mulOutRow12_x4_b,mulOutRow12_x4_c,mulOutRow12_x7,mulOutRow12_x5 :
std_logic_vector(7 downto 0);

SIGNAL mulOutRow13_x4,mulOutRow13_x4_b,mulOutRow13_x4_c,mulOutRow13_x7,mulOutRow13_x5 :
std_logic_vector(7 downto 0);

SIGNAL mulOutRow14_x4,mulOutRow14_x4_b,mulOutRow14_x4_c,mulOutRow14_x7,mulOutRow14_x5 :
std_logic_vector(7 downto 0);

SIGNAL mulOutRow15_x4,mulOutRow15_x4_b,mulOutRow15_x4_c,mulOutRow15_x7,mulOutRow15_x5 :
std_logic_vector(7 downto 0);


SIGNAL xorX0_a,xorX0_b,xorX0_c,xorX0_d,xorX0_e : std_logic_vector(7 downto 0);
SIGNAL xorX1_a,xorX1_b,xorX1_c,xorX1_d,xorX1_e : std_logic_vector(7 downto 0);
SIGNAL xorX2_a,xorX2_b,xorX2_c,xorX2_d,xorX2_e : std_logic_vector(7 downto 0);
SIGNAL xorX3_a,xorX3_b,xorX3_c,xorX3_d,xorX3_e : std_logic_vector(7 downto 0);

SIGNAL xorX4_a,xorX4_b,xorX4_c : std_logic_vector(7 downto 0);
SIGNAL xorX5_a,xorX5_b,xorX5_c : std_logic_vector(7 downto 0);
SIGNAL xorX6_a,xorX6_b,xorX6_c : std_logic_vector(7 downto 0);
SIGNAL xorX7_a,xorX7_b,xorX7_c : std_logic_vector(7 downto 0);

SIGNAL xorX8_a,xorX8_b,xorX8_c,xorX8_d : std_logic_vector(7 downto 0);
SIGNAL xorX9_a,xorX9_b,xorX9_c,xorX9_d : std_logic_vector(7 downto 0);
SIGNAL xorX10_a,xorX10_b,xorX10_c,xorX10_d : std_logic_vector(7 downto 0);
SIGNAL xorX11_a,xorX11_b,xorX11_c,xorX11_d : std_logic_vector(7 downto 0);

SIGNAL xorX12_a,xorX12_b,xorX12_c,xorX12_d : std_logic_vector(7 downto 0);
SIGNAL xorX13_a,xorX13_b,xorX13_c,xorX13_d : std_logic_vector(7 downto 0);
SIGNAL xorX14_a,xorX14_b,xorX14_c,xorX14_d : std_logic_vector(7 downto 0);
SIGNAL xorX15_a,xorX15_b,xorX15_c,xorX15_d : std_logic_vector(7 downto 0);

begin
-------------------------PORT MAPS WITH MULTS-----------------------------------------------------------------------
--Sstate 0 Multiplication
        mul4Comp_row0_S0 : mul4 port map(SstateIn(951 downto 944),mulOutRow0_x4);
```

```
mul7Comp_row0_S0 : mul7 port map(SstateIn(943 downto 936),mulOutRow0_x7);

mul4Comp_row1_S0 : mul4 port map(SstateIn(911 downto 904),mulOutRow1_x4);
mul7Comp_row1_S0 : mul7 port map(SstateIn(903 downto 896),mulOutRow1_x7);

mul7Comp_row2_S0 : mul7 port map(SstateIn(895 downto 888),mulOutRow2_x7);
mul4Comp_row2_S0 : mul4 port map(SstateIn(871 downto 864),mulOutRow2_x4);

mul4Comp_row3_S0 : mul4 port map(SstateIn(863 downto 856),mulOutRow3_x4);
mul7Comp_row3_S0 : mul7 port map(SstateIn(855 downto 848),mulOutRow3_x7);

--Sstate 1 Multiplication----------
mul4Comp_row0_S1 : mul4 port map(SstateIn(919 downto 912),mulOutRow4_x4);
mul7Comp_row0_S1 : mul7 port map(SstateIn(911 downto 904),mulOutRow4_x7);

mul4Comp_row1_S1 : mul4 port map(SstateIn(879 downto 872),mulOutRow5_x4);
mul7Comp_row1_S1 : mul7 port map(SstateIn(871 downto 864),mulOutRow5_x7);

mul7Comp_row2_S1 : mul7 port map(SstateIn(863 downto 856),mulOutRow6_x7);
mul4Comp_row2_S1 : mul4 port map(SstateIn(839 downto 832),mulOutRow6_x4);

mul4Comp_row3_S1 : mul4 port map(SstateIn(959 downto 952),mulOutRow7_x4);
mul7Comp_row3_S1 : mul7 port map(SstateIn(951 downto 944),mulOutRow7_x7);

--Sstate 2 Multiplication----------
mul7Comp_row8_S2 : mul7 port map(SstateIn(927 downto 920),mulOutRow8_x7);
mul6Comp_row8_S2 : mul6 port map(SstateIn(895 downto 888),mulOutRow8_x6);
mul4Comp_row8_S2 : mul4 port map(SstateIn(887 downto 880),mulOutRow8_x4);
mul7Comp_b_row8_S2 : mul7 port map(SstateIn(879 downto 872),mulOutRow8_x7_b);
mul7Comp_c_row8_S2 : mul7 port map(SstateIn(863 downto 856),mulOutRow8_x7_c);

mul7Comp_row9_S2 : mul7 port map(SstateIn(951 downto 944),mulOutRow9_x7);
mul7Comp_b_row9_S2 : mul7 port map(SstateIn(887 downto 880),mulOutRow9_x7_b);
mul6Comp_row9_S2 : mul6 port map(SstateIn(855 downto 848),mulOutRow9_x6);
mul4Comp_row9_S2 : mul4 port map(SstateIn(847 downto 840),mulOutRow9_x4);
mul7Comp_c_row9_S2 : mul7 port map(SstateIn(839 downto 832),mulOutRow9_x7_c);

mul7Comp_row10_S2 : mul7 port map(SstateIn(959 downto 952),mulOutRow10_x7);
mul6Comp_row10_S2 : mul6 port map(SstateIn(943 downto 936),mulOutRow10_x6);
mul4Comp_row10_S2 : mul4 port map(SstateIn(935 downto 928),mulOutRow10_x4);
mul7Comp_b_row10_S2 : mul7 port map(SstateIn(911 downto 904),mulOutRow10_x7_b);
mul7Comp_c_row10_S2 : mul7 port map(SstateIn(847 downto 840),mulOutRow10_x7_c);

mul7Comp_row11_S2 : mul7 port map(SstateIn(935 downto 928),mulOutRow11_x7);
mul4Comp_row11_S2 : mul4 port map(SstateIn(927 downto 920),mulOutRow11_x4);
mul7Comp_b_row11_S2 : mul7 port map(SstateIn(919 downto 912),mulOutRow11_x7_b);
mul6Comp_row11_S2 : mul6 port map(SstateIn(903 downto 896),mulOutRow11_x6);
mul7Comp_c_row11_S2 : mul7 port map(SstateIn(871 downto 864),mulOutRow11_x7_c);

--Sstate 3 Multiplication----------
mul4Comp_row12_S3 : mul4 port map(SstateIn(927 downto 920),mulOutRow12_x4);
mul4Comp_b_row12_S3 : mul4 port map(SstateIn(895 downto 888),mulOutRow12_x4_b);
mul5Comp_row12_S3 : mul5 port map(SstateIn(863 downto 856),mulOutRow12_x5);
mul4Comp_c_row12_S3 : mul4 port map(SstateIn(855 downto 848),mulOutRow12_x4_c);
mul7Comp_row12_S3 : mul7 port map(SstateIn(847 downto 840),mulOutRow12_x7);

mul5Comp_row13_S3 : mul5 port map(SstateIn(951 downto 944),mulOutRow13_x5);
mul4Comp_row13_S3 : mul4 port map(SstateIn(943 downto 936),mulOutRow13_x4);
mul7Comp_row13_S3 : mul7 port map(SstateIn(935 downto 928),mulOutRow13_x7);
mul4Comp_b_row13_S3 : mul4 port map(SstateIn(887 downto 880),mulOutRow13_x4_b);
```

```
        mul4Comp_c_row13_S3 : mul4 port map(SstateIn(855 downto 848),mulOutRow13_x4_c);

        mul4Comp_row14_S3 : mul4 port map(SstateIn(943 downto 936),mulOutRow14_x4);
        mul7Comp_row14_S3 : mul7 port map(SstateIn(927 downto 920),mulOutRow14_x7);
        mul5Comp_row14_S3 : mul5 port map(SstateIn(911 downto 904),mulOutRow14_x5);
        mul4Comp_b_row14_S3 : mul4 port map(SstateIn(903 downto 896),mulOutRow14_x4_b);
        mul4Comp_c_row14_S3 : mul4 port map(SstateIn(847 downto 840),mulOutRow14_x4_c);

        mul4Comp_row15_S3 : mul4 port map(SstateIn(935 downto 928),mulOutRow15_x4);
        mul4Comp_b_row15_S3 : mul4 port map(SstateIn(903 downto 896),mulOutRow15_x4_b);
        mul4Comp_c_row15_S3 : mul4 port map(SstateIn(895 downto 888),mulOutRow15_x4_c);
        mul7Comp_row15_S3 : mul7 port map(SstateIn(887 downto 880),mulOutRow15_x7);
        mul5Comp_row15_S3 : mul5 port map(SstateIn(871 downto 864),mulOutRow15_x5);
----------END PORT MAPS----------


--Sstate(959 downto 832) Multiplication with N matrix

--Sstate 0 Bytes 0,1,2 and 3 --> X0,X1,X2,X3
------Sstate 0 Byte 0 --> X0----------
----------
                xorX0_a <= SstateIn(959 downto 952) xor mulOutRow0_x4;
                xorX0_b <= SstateIn(935 downto 928) xor mulOutRow0_x7;
                xorX0_c <= SstateIn(927 downto 920) xor SstateIn(895 downto 888);
                xorX0_d <= SstateIn(863 downto 856) xor xorX0_c;
                xorX0_e <=          xorX0_a xor xorX0_b;
        SstateOut(959 downto 952) <= xorX0_d xor xorX0_e;
------Sstate 0 Byte 1 --> X1----------
----------
    xorX1_a <= SstateIn(951 downto 944) xor SstateIn(927 downto 920);
    xorX1_b <= SstateIn(919 downto 912) xor mulOutRow1_x4;
                xorX1_c <= SstateIn(887 downto 880) xor mulOutRow1_x7;
                xorX1_d <= SstateIn(855 downto 848) xor xorX1_c;
                xorX1_e <=          xorX1_a xor xorX1_b;
        SstateOut(951 downto 944) <= xorX1_d xor xorX1_e;
------Sstate 0 Byte 2 --> X2----------

        xorX2_a <= SstateIn(943 downto 936) xor SstateIn(911 downto 904);
                xorX2_b <= SstateIn(887 downto 880) xor mulOutRow2_x7;
                xorX2_c <= SstateIn(879 downto 872) xor mulOutRow2_x4;
                xorX2_d <= SstateIn(847 downto 840) xor xorX2_c;
                xorX2_e <=          xorX2_a xor xorX2_b;
        SstateOut(943 downto 936) <= xorX2_d xor xorX2_e;
------Sstate 0 Byte 3 --> X3----------

        xorX3_a <= SstateIn(935 downto 928) xor SstateIn(903 downto 896);
                xorX3_b <= SstateIn(871 downto 864) xor mulOutRow3_x4;
                xorX3_c <= SstateIn(847 downto 840) xor mulOutRow3_x7;
                xorX3_d <= SstateIn(839 downto 832) xor xorX3_c;
                xorX3_e <=          xorX3_a xor xorX3_b;
        SstateOut(935 downto 928) <= xorX3_d xor xorX3_e;



--Sstate 1 Bytes 0,1,2 and 3 --> X4,X5,X6,X7
------Sstate 1 Byte 0 --> X4----------
    xorX4_a <=      mulOutRow4_x4 xor mulOutRow4_x7;
    xorX4_b <= SstateIn(903 downto 896) xor SstateIn(895 downto 888);
    xorX4_c <=          xorX4_a xor xorX4_b;
        SstateOut(927 downto 920) <= SstateIn(863 downto 856) xor xorX4_c;
```

```
------Sstate 1 Byte 1 --> X5----------------------------------------------------------------------------------------------------

                    xorX5_a <= SstateIn(951 downto 944) xor SstateIn(895 downto 888);
                    xorX5_b <=          mulOutRow5_x4 xor mulOutRow5_x7;
                    xorX5_c <=              xorX5_a xor xorX5_b;
            SstateOut(919 downto 912) <=xorX5_c  xor SstateIn(855 downto 848);
------Sstate 1 Byte 2 --> X6----------------------------------------------------------------------------------------------------
    xorX6_a <= SstateIn(943 downto 936) xor SstateIn(911 downto 904);
                    xorX6_b <= SstateIn(855 downto 848) xor mulOutRow6_x7;
                    xorX6_c <=              xorX6_a xor xorX6_b;
            SstateOut(911 downto 904) <= xorX6_c xor mulOutRow6_x4;
------Sstate 1 Byte 3 --> X7----------------------------------------------------------------------------------------------------
    xorX7_a <=          mulOutRow7_x4 xor mulOutRow7_x7;
    xorX7_b <= SstateIn(943 downto 936) xor SstateIn(903 downto 896);
    xorX7_c <=                      xorX7_a xor xorX7_b;
   SstateOut(903 downto 896) <= xorX7_c xor SstateIn(871 downto 864);



--Sstate 2 Bytes 0,1,2 and 3 --> X8,X9,X10,X11
------Sstate 2 Byte 0 --> X8----------------------------------------------------------------------------------------------------
    xorX8_a <= mulOutRow8_x7 xor mulOutRow8_x6;
                    xorX8_b <= mulOutRow8_x4 xor mulOutRow8_x7_b;
                    xorX8_c <= mulOutRow8_x7_c xor SstateIn(871 downto 864);
                    xorX8_d <=      xorX8_a xor xorX8_b;
   SstateOut(895 downto 888) <= xorX8_c xor xorX8_d;
------Sstate 2 Byte 1 --> X9----------------------------------------------------------------------------------------------------
    xorX9_a <= mulOutRow9_x7 xor mulOutRow9_x7_b;
    xorX9_b <= mulOutRow9_x6 xor SstateIn(863 downto 856);
    xorX9_c <= mulOutRow9_x4 xor mulOutRow9_x7_c;
    xorX9_d <=      xorX9_a xor xorX9_b;
   SstateOut(887 downto 880) <= xorX9_c xor xorX9_d;
------Sstate 2 Byte 2 --> X10----------------------------------------------------------------------------------------------------
    xorX10_a <= mulOutRow10_x7 xor SstateIn(951 downto 944);
                    xorX10_b <= mulOutRow10_x6 xor mulOutRow10_x4;
                    xorX10_c <= mulOutRow10_x7_b xor mulOutRow10_x7_c;
                    xorX10_d <=      xorX10_a xor xorX10_b;
   SstateOut(879 downto 872) <= xorX10_c xor xorX10_d;
------Sstate 2 Byte 3 --> X11----------------------------------------------------------------------------------------------------
                    xorX11_a <= mulOutRow11_x7 xor mulOutRow11_x4;
                    xorX11_b <= mulOutRow11_x7_b xor SstateIn(911 downto 904);
                    xorX11_c <= mulOutRow11_x6 xor mulOutRow11_x7_c;
                    xorX11_d <=     xorX11_a xor xorX11_b;
            SstateOut(871 downto 864) <= xorX11_c xor xorX11_d;



--Sstate 3 Bytes 0,1,2 and 3 --> X12,X13,X14,X15
------Sstate 3 Byte 0 --> X12----------------------------------------------------------------------------------------------------
    xorX12_a <= mulOutRow12_x4 xor mulOutRow12_x4_b;
                    xorX12_b <= mulOutRow12_x5 xor mulOutRow12_x4_c;
                    xorX12_c <= mulOutRow12_x7 xor SstateIn(839 downto 832);
    xorX12_d <=     xorX12_a xor xorX12_b;
            SstateOut(863 downto 856) <= xorX12_c xor xorX12_d;
------Sstate 3 Byte 1 --> X13----------------------------------------------------------------------------------------------------
    xorX13_a <= mulOutRow13_x5 xor SstateIn(959 downto 952);
    xorX13_b <= mulOutRow13_x4 xor mulOutRow13_x7;
    xorX13_c <= mulOutRow13_x4_b xor mulOutRow13_x4_c;
    xorX13_d <=         xorX13_a xor xorX13_b;
            SstateOut(855 downto 848) <= xorX13_c xor xorX13_d;
```

```
------Sstate 3 Byte 2 --> X14----------
        xorX14_a <= mulOutRow14_x4 xor mulOutRow14_x7;
        xorX14_b <= mulOutRow14_x5 xor SstateIn(919 downto 912);
        xorX14_c <= mulOutRow14_x4_b xor mulOutRow14_x4_c;
        xorX14_d <=      xorX14_a xor xorX14_b;
        SstateOut(847 downto 840) <= xorX14_c xor xorX14_d;
------Sstate 3 Byte 3 --> X15----------

        xorX15_a <= mulOutRow15_x4 xor mulOutRow15_x4_b;
        xorX15_b <= mulOutRow15_x4_c xor mulOutRow15_x7;
        xorX15_c <= mulOutRow15_x5 xor SstateIn(879 downto 872);
        xorX15_d <=     xorX15_a xor xorX15_b;
        SstateOut(839 downto 832) <=      xorX15_c xor xorX15_d;

end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος πολλαπλασιασμού x4 σε $GF(2^8)$ (Σχήμα 6.15.α).

```
entity mul4 is
        port(numIn : in std_logic_vector(7 downto 0);
             numOut : out std_logic_vector(7 downto 0));
end mul4;

architecture Behavioral of mul4 is
        SIGNAL sigShift,sigShift2,sigXOR :  std_logic_vector(7 downto 0);
begin

-- Multiple x2 and xored with Polynomial x^4+x^3+x+1(1b) from Polynomial 283
        sigShift <= numIn(6 downto 0)&'0';
  sigXOR <=  sigShift xor "000"&numIn(7)&numIn(7)&'0'&numIn(7)&numIn(7);

-- Multiple x2 and xored with Polynomial x^4+x^3+x+1(1b) from Polynomial 283
        sigShift2 <= sigXOR(6 downto 0)&'0';
  numOut <=  sigShift2 xor "000"&sigXOR(7)&sigXOR(7)&'0'&sigXOR(7)&sigXOR(7);

end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος πολλαπλασιασμού x5 σε $GF(2^8)$ (Σχήμα 6.15.α).

```
entity mul5 is
        port(numIn : in std_logic_vector(7 downto 0);
             numOut : out std_logic_vector(7 downto 0));
end mul5;

architecture Behavioral of mul5 is
        SIGNAL sigShift,sigShift2,sigXOR,sigXOR2 :  std_logic_vector(7 downto 0);
begin

-- Multiple x2 and xored with Polynomial x^4+x^3+x+1(1b) from Polynomial 283
        sigShift <= numIn(6 downto 0)&'0';
  sigXOR <=  sigShift xor "000"&numIn(7)&numIn(7)&'0'&numIn(7)&numIn(7);

-- Multiple x2 and xored with Polynomial x^4+x^3+x+1(1b) from Polynomial 283 and xored with numIn
        sigShift2 <= sigXOR(6 downto 0)&'0';
  sigXOR2 <=  sigShift2 xor "000"&sigXOR(7)&sigXOR(7)&'0'&sigXOR(7)&sigXOR(7);
```

```
        numOut <= sigXOR2 xor numIn;

end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος πολλαπλασιασμού x6 σε $GF(2^8)$ (Σχήμα 6.15.β).

```
entity mul6 is
        port(numIn : in std_logic_vector(7 downto 0);
             numOut : out std_logic_vector(7 downto 0));
end mul6;

architecture Behavioral of mul6 is
        SIGNAL sigShift,sigShift2,sigXOR,sigXOR2 :  std_logic_vector(7 downto 0);
begin

-- Multiple x2 and xored with Polynomial x^4+x^3+x+1(1b) from Polynomial 283
        sigShift <= numIn(6 downto 0)&'0';
  sigXOR <=  sigShift xor "000"&numIn(7)&numIn(7)&'0'&numIn(7)&numIn(7);

-- Multiple x2 and xored with Polynomial x^4+x^3+x+1(1b) from Polynomial 283 and xored with sigXOR
  sigShift2 <= sigXOR(6 downto 0)&'0';
  sigXOR2 <=  sigShift2 xor "000"&sigXOR(7)&sigXOR(7)&'0'&sigXOR(7)&sigXOR(7);
        numOut <= sigXOR2 xor sigXOR;

end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος πολλαπλασιασμού x7 σε $GF(2^8)$ (Σχήμα 6.15.β).

```
entity mul7 is
        port(numIn : in std_logic_vector(7 downto 0);
             numOut : out std_logic_vector(7 downto 0));
end mul7;

architecture Behavioral of mul7 is
        SIGNAL sigShift,sigShift2,sigXOR,sigXOR2,sigXOR3 :  std_logic_vector(7 downto 0);
begin

-- Multiple x2 and xored with Polynomial x^4+x^3+x+1(1b) from Polynomial 283
        sigShift <= numIn(6 downto 0)&'0';
  sigXOR <=  sigShift xor "000"&numIn(7)&numIn(7)&'0'&numIn(7)&numIn(7);

-- Multiple x2 and xored with Polynomial x^4+x^3+x+1(1b) from Polynomial 283 and xored with numIn and
xored with sigXOR
        sigShift2 <= sigXOR(6 downto 0)&'0';
  sigXOR2 <=  sigShift2 xor "000"&sigXOR(7)&sigXOR(7)&'0'&sigXOR(7)&sigXOR(7);
        sigXOR3 <= sigXOR2 xor numIn;
        numOut <= sigXOR3 xor sigXOR;

end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος διασύνδεσης των Sboxes με το Super-Mix επίπεδο.

```
entity SMIX is
        port(reset,resetSboxes : in std_logic;
             SstateIn : in std_logic_vector(959 downto 0);
             SstateOut :  out std_logic_vector(959 downto 0));
end SMIX;

architecture Behavioral of SMIX is
        Component trsfmtnSBOX
        port(resetSboxes : in std_logic;
             SstateIn : in std_logic_vector(959 downto 832);
             SstateOut :  out std_logic_vector(959 downto 832));
        end component;

        Component multArray
        port(SstateIn : in std_logic_vector(959 downto 832);
             SstateOut :  out std_logic_vector(959 downto 832));
        end component;

SIGNAL SBoxOut : std_logic_vector(959 downto 832);
begin

trsfmtnSBOX_Comp : trsfmtnSBOX port map(resetSboxes,SstateIn(959 downto 832),SBoxOut(959 downto
832));

mulArray_comp : multArray port map(SBoxOut(959 downto 832),SstateOut(959 downto 832));

smix : process(reset,SstateIn)
         begin
                if(reset = '0')then
                        SstateOut(831 downto 0) <= SstateIn(831 downto 0);
                else
                        SstateOut(831 downto 0) <= others => '0');
                end if;
         end process;
end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος Output Block (Σχήμα 6.16).

```
entity OutPutBlock is
        port(reset : in std_logic;
             S0In,S1In,S2In,S3In,S4In,S15In,S16In,S17In,S18In : in std_logic_vector(31 downto 0);
             dataOut : out std_logic_vector(255 downto 0) );
end OutPutBlock;

architecture Behavioral of OutPutBlock is
SIGNAL sigXOR1,sigXOR2  : std_logic_vector(31 downto 0);
SIGNAL signal3  : std_logic_vector(255 downto 0);
begin

        sigXOR1 <= S4In xor S0In;                sigXOR2 <= S15In xor S0In;
        signal3 <= S1In & S2In & S3In & sigXOR1 & sigXOR2 & S16In & S17In & S18In;

        OutPutBlock : process(reset,signal3)
                begin
                        if(reset = '0')then
                          dataOut <= signal3;
                        else
                          dataOut <= (others => '0');
                        end if;
```

```
                end process;
end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος Counter (Σχήμα 6.17).

```
entity counterComp is
        port(clk,reset,enable : in std_logic;
             messagesCircles : in std_logic_vector(3 downto 0);
             counterOut : out integer range 0 to 49;
end counterComp;

architecture Behavioral of counterComp is
SIGNAL counter : integer range 0 to 49;
begin

counterOut <= counter;
counterProcess : process(clk,reset,enable,messagesCircles,counter)
                begin
                if(reset = '1')then
                        counter <= 0;
                else
                        if(counter < 49 and enable = '1')then
                                if(clk' event and clk = '1')then
                                        if(counter = "00" & messagesCircles)then

                                                counter <= 13;
                                        else

                                                counter <= counter + 1;
                                        end if;
                                end if;
                        end if;
                end if;
                end process;
end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος Controller (Σχήμα 6.18).

```
entity Controller is
        port(clk,reset,enable : in std_logic;
             messagesCircles : in std_logic_vector(3 downto 0);
             resetSboxes,hashReady : out std_logic;
             selectionMUX2x1_IV,selectionMUX2x1 : out std_logic;
             selectionMUX3x1 : out std_logic_vector(1 downto 0);
             selectionMUX6x1 : out std_logic_vector(2 downto 0));
end Controller;

architecture Behavioral of Controller is
        Component counterComp
        port(clk,reset,enable : in std_logic;
             messagesCircles : in std_logic_vector(3 downto 0);
             counterOut : out integer range 0 to 49);
        end component;

SIGNAL counter : integer range 0 to 49;
begin
counter_Comp : counterComp port map(clk,reset,enable,messagesCircles,counter);
```

```
resetSboxes <= '1' when (counter = 0) else '0';
hashReady <= '1' when (counter = 49) else '0';

Controller : process(reset,counter)
 begin
        if(reset = '0')then
        case counter is
                when 0 => selectionMUX6x1 <= "000";
                        selectionMUX2x1 <= '0';
                        selectionMUX2x1_IV <= '0';selectionMUX3x1 <= "00";

-----------------------------------STARTS ROUND R 1 WORD-----------------------------------------------------
                when 1 => selectionMUX6x1 <= "001";
                        selectionMUX2x1 <= '0';
                        selectionMUX2x1_IV <= '1';selectionMUX3x1 <= "00";

                when 2 => selectionMUX2x1 <= '1';
                        selectionMUX6x1 <= "001";
                        selectionMUX2x1_IV <= '0';selectionMUX3x1 <= "00";
-----------------------------------END ROUND R 1 WORD-------------------------------------------------------
                when 3 => selectionMUX6x1 <= "010";
                        selectionMUX2x1 <= '0';
                        selectionMUX2x1_IV <= '0';selectionMUX3x1 <= "00";

                when 4 => selectionMUX2x1 <= '1';
                        selectionMUX6x1 <= "010";
                        selectionMUX2x1_IV <= '0';selectionMUX3x1 <= "00";
-----------------------------------END ROUND R 2 WORD-------------------------------------------------------
                when 5 => selectionMUX6x1 <= "011";
                        selectionMUX2x1 <= '0';
                        selectionMUX2x1_IV <= '0';selectionMUX3x1 <= "00";

                when 6 => selectionMUX2x1 <= '1';
                        selectionMUX6x1 <= "011";
                        selectionMUX2x1_IV <= '0';selectionMUX3x1 <= "00";
-----------------------------------END ROUND R 3 WORD-------------------------------------------------------
                when 7 => selectionMUX6x1 <= "100";
                        selectionMUX2x1 <= '0';
                        selectionMUX2x1_IV <= '0';selectionMUX3x1 <= "00";

                when 8 => selectionMUX2x1 <= '1';
                        selectionMUX6x1 <= "100";
                        selectionMUX2x1_IV <= '0';selectionMUX3x1 <= "00";
-----------------------------------END ROUND R 4 WORD-------------------------------------------------------
                when 9 => selectionMUX6x1 <= "101";
                        selectionMUX2x1 <= '0';
                        selectionMUX2x1_IV <= '0';selectionMUX3x1 <= "00";

                when 10 => selectionMUX2x1 <= '1';
                        selectionMUX6x1 <= "101";
                        selectionMUX2x1_IV <= '0';selectionMUX3x1 <= "00";
-----------------------------------END ROUND R 5 WORD-------------------------------------------------------
                when 11 => selectionMUX6x1 <= "110";
                        selectionMUX2x1 <= '0';
                        selectionMUX2x1_IV <= '0';selectionMUX3x1 <= "00";

                when 12 => selectionMUX2x1 <= '1';
                        selectionMUX6x1 <= "110";
                        selectionMUX2x1_IV <= '0';selectionMUX3x1 <= "00";
-----------------------------------END ROUND R 6 WORD-------------------------------------------------------
```

```
--
**********************************************************************************************
**************************
----------------------------ROUND 10 TIMES AFTER 10 CIRCLES-----------------------------------------------
----
--
**********************************************************************************************
**************************

-------------------------------------STARTS ROUND G-----------------------------------------------
                when 23 => selectionMUX3x1 <= "01";
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 24 => selectionMUX3x1 <= "10";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 25 => selectionMUX3x1 <= "01";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 26 => selectionMUX3x1 <= "10";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 27 => selectionMUX3x1 <= "01";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 28 => selectionMUX3x1 <= "10";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 29 => selectionMUX3x1 <= "01";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 30 => selectionMUX3x1 <= "10";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 31 => selectionMUX3x1 <= "01";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 32 => selectionMUX3x1 <= "10";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 33 => selectionMUX3x1 <= "01";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 34 => selectionMUX3x1 <= "10";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 35 => selectionMUX3x1 <= "01";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 36 => selectionMUX3x1 <= "10";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 37 => selectionMUX3x1 <= "01";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 38 => selectionMUX3x1 <= "10";
                           selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';
-------------------------------- 8  CIRCLES-----------------------------------------------
                when 39 => selectionMUX3x1 <= "01";
```

```
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 40 => selectionMUX3x1 <= "10";
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 41 => selectionMUX3x1 <= "01";
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 42 => selectionMUX3x1 <= "10";
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 43 => selectionMUX3x1 <= "01";
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 44 => selectionMUX3x1 <= "10";
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 45 => selectionMUX3x1 <= "01";
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 46 => selectionMUX3x1 <= "10";
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 47 => selectionMUX3x1 <= "01";
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 48 => selectionMUX3x1 <= "10";
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';

                when 49 => selectionMUX3x1 <= "10";
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";selectionMUX2x1 <= '1';
----------------------------END ROUND G ----------------------------------------------------------

                when others  => selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";
                    selectionMUX2x1 <= '1';selectionMUX3x1 <= "00";

                end case;
                else
                    selectionMUX2x1_IV <= '0';selectionMUX6x1 <= "000";
                    selectionMUX2x1 <= '0';selectionMUX3x1 <= "00";
                end if;
        end process;
```

Κώδικας υλοποίησης του πολυπλέκτη MUX (1) του Σχήματος 6.1 .

```
entity MUX6x1 is
        port(selection : in std_logic_vector(2 downto 0);
            Input1,Input2,Input3,Input4,Input5,Input6 : in std_logic_vector(31 downto 0);
            Output : out std_logic_vector(31 downto 0));
end MUX6x1;
architecture Behavioral of MUX6x1 is
begin
        MUX6x1 : process(selection,Input1,Input2,Input3,Input4,Input5,Input6)
            begin
                case selection is
                    when "001" => Output <= Input1;
                    when "010" => Output <= Input2;
                    when "011" => Output <= Input3;
```

```
                    when "100" => Output <= Input4;
                    when "101" => Output <= Input5;
                    when "110" => Output <= Input6;
                    when others => Output <= (others => '0');
                end case;
            end process;
end Behavioral;
```

Κώδικας υλοποίησης του πολυπλέκτη MUX (2) του Σχήματος 6.1 .

```
entity MUX2x1_IV is
  Port (selection : in STD_LOGIC;
        Input1 : in  STD_LOGIC_VECTOR (255 downto 0);
        Input2 : in  STD_LOGIC_VECTOR (255 downto 0);
        Output : out  STD_LOGIC_VECTOR (255 downto 0));
end MUX2x1_IV;
architecture Behavioral of MUX2x1_IV is
begin

MUX2x1_IV : process(selection,Input1,Input2)
        begin
                case selection is
                        when '0' => Output <= Input1;
                        when '1' => Output <= Input2;
                        when others => Output <= (others => '0');
                end case;
        end process;
end Behavioral;
```

Κώδικας υλοποίησης του πολυπλέκτη MUX (3) του Σχήματος 6.1 .

```
entity MUX2x1 is
        port(selection : in std_logic;
             Input1,Input2 : in std_logic_vector(959 downto 0);
             Output : out std_logic_vector(959 downto 0));
end MUX2x1;
architecture Behavioral of MUX2x1 is
begin
MUX3x1 : process(selection,Input1,Input2)
        begin
                case selection is
                        when '0' => Output <= Input1;
                        when '1' => Output <= Input2;
                        when others => Output <= (others => '0');
                end case;
        end process;
end Behavioral;
```

Κώδικας υλοποίησης του πολυπλέκτη MUX (4) του Σχήματος 6.1 .

```
entity MUX3x1 is
        port(selection : in std_logic_vector(1 downto 0);
             Input1,Input2,Input3 : in std_logic_vector(959 downto 0);
             Output : out std_logic_vector(959 downto 0));
```

```
end MUX3x1;
architecture Behavioral of MUX3x1 is
begin
MUX3x1 : process(selection,Input1,Input2,Input3)
        begin
                case selection is
                        when "00" => Output <= Input1;
                        when "01" => Output <= Input2;
                        when "10" => Output <= Input3;
                        when others => Output <= (others => '0');
                end case;
        end process;
end Behavioral;
```

Κώδικας υλοποίησης του καταχωρητή Sstate του Σχήματος 6.1 .

```
entity SstateRegister is
        port(clk,reset,enable,hashReady : in std_logic;
             SstateIn : in std_logic_vector(959 downto 0);
             SstateOut : out std_logic_vector(959 downto 0));
end SstateRegister;
architecture Behavioral of SstateRegister is
begin
SstateReg : process(reset,clk,enable,hashReady)
            begin
                if(reset = '1')then
                        SstateOut <= (others => '0');
                else
                        if(enable = '1' and hashReady = '0')then
                                if(clk' event and clk = '1')then
                                        SstateOut <= SstateIn;
                                end if;
                        end if;
                end if;
             end process;
end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος Initialization Vector του Σχήματος 6.1 .

```
entity IV is
        port(reset : in std_logic;
             IVOut : out std_logic_vector(255 downto 0));
end IV;
architecture Behavioral of IV is
begin
IVOut <= X"E952BDDE6671135FE0D4F668D2B0B594F96C621DFBF929DE9149E89934F8C248" when
(reset = '0') else (others => '0');
end Behavioral;
```

Αρχιτεκτονική: JH

Κώδικας του κυκλώματος διασύνδεσης των επιμέρους κυκλωμάτων (Σχήμα 6.26).

```
entity JH is
        port(dataIn : in std_logic_vector(511 downto 0);
             reset,enable,clk : in std_logic;
             hashLength : in std_logic_vector(1 downto 0);
             messageLength : in std_logic_vector(9 downto 0);
             hashReady : out std_logic;
             JH_Out : out std_logic_vector(511 downto 0) );
end JH;

architecture Behavioral of JH is
  Component IV
        port(reset : in std_logic;
             hashLength : in std_logic_vector(1 downto 0);
             IVOut : out std_logic_vector(1023 downto 0));
        end component;

        Component Padder
        port(reset :in std_logic;
             dataIn :in std_logic_vector(511 downto 0);
             messageLength : in std_logic_vector(9 downto 0);
             computingCrcls : out std_logic_vector(6 downto 0);
             paddedDataOut : out std_logic_vector(1023 downto 0));
        end component;

        Component GroupingF
        port(grpngIn : in std_logic_vector(1023 downto 0);
             grpngOut : out std_logic_vector(1023 downto 0));
        end component;

        Component trnsfrmtnSBOX
        port(resetSboxes : in std_logic;
             dataIn : in std_logic_vector(1023 downto 0);
             round  : in integer range 0 to 36;
             dataOut : out std_logic_vector(1023 downto 0));
        end component;

        Component mapping_L_Blocks
        port(dataIn : in std_logic_vector(1023 downto 0);
             dataOut : out std_logic_vector(1023 downto 0) );
        end component;

        Component function_P
        port(dataIn : in std_Logic_vector(1023 downto 0);
             dataOut : out std_Logic_vector(1023 downto 0) );
        end component;

        Component DeGroupingF
        port(DegrpngIn : in std_logic_vector(1023 downto 0);
             DegrpngOut : out std_logic_vector(1023 downto 0) );
        end component;

        Component truncating_H
        port(hashLength : in std_logic_vector(1 downto 0);
             dataIn : in std_logic_vector(511 downto 0);
             dataOut : out std_logic_vector(511 downto 0) );
        end component;
```

```
        Component Controller
        port(clk,reset,enable : in std_logic;
             computingCrcls : in std_logic_vector(6 downto 0);
             resetSboxes,hashReady : out std_logic;
             selectionMUX2x1 : out std_logic;
             selectionMUX4x1 : out std_logic_vector(1 downto 0);
             constantRound : out integer range 0 to 36 );
        end component;

        Component HstateRegister
        port(reset,enable,hashReady,clk : in std_logic;
             dataIn : in std_logic_vector(1023 downto 0);
             dataOut : out std_logic_vector(1023 downto 0));
        end component;

        Component MUX2x1
        port(selection : in std_logic;
             Input1,Input2 : in std_logic_vector(511 downto 0);
             Output : out std_logic_vector(511 downto 0));
        end component;

        Component MUX4x1
        port(selection : in std_logic_vector(1 downto 0);
             Input1,Input2,Input3,Input4 : in std_logic_vector(1023 downto 0);
             Output : out std_logic_vector(1023 downto 0));
        end component;

-- S I G N A L S for IV component
SIGNAL IVOut : std_logic_vector(1023 downto 0);

-- S I G N A L S for Padder component
  SIGNAL computingCrcls : std_logic_vector(6 downto 0);
SIGNAL padderOut : std_logic_vector(1023 downto 0);

-- S I G N A L S for Grouping and DeGrouping component
SIGNAL grpngOut,DegrpngOut : std_logic_vector(1023 downto 0);

-- S I G N A L S for SBOXes
SIGNAL resetSboxes : std_logic;
SIGNAL constantRound  : integer range 0 to 36;
SIGNAL sboxOut : std_logic_vector(1023 downto 0);

-- S I G N A L S for L function
SIGNAL funL_out : std_logic_vector(1023 downto 0);

-- S I G N A L S for P function
SIGNAL funP_out : std_logic_vector(1023 downto 0);

-- S I G N A L S for Controller component
SIGNAL hashReadySig : std_logic;

-- S I G N A L S for HstateRegister component
SIGNAL HstateOut : std_logic_vector(1023 downto 0);

-- S I G N A L S for MUX components
SIGNAL selectionMUX2x1 : std_logic;
SIGNAL selectionMUX4x1 : std_logic_vector(1 downto 0);
SIGNAL MUX4x1Out : std_logic_vector(1023 downto 0);
SIGNAL Mword : std_logic_vector(511 downto 0);
```

```
-- S I G N A L S XOR
SIGNAL sigXOR1,sigXOR2 : std_logic_vector(1023 downto 0);
SIGNAL sigXOR3,sigXOR4 : std_logic_vector(511 downto 0);
begin

IV_Comp : IV port map(reset,hashLength,IVOut);

Padder_Comp : Padder port map(reset,dataIn,messageLength,computingCrcls,padderOut);

GroupingF_Comp : GroupingF port map(sigXOR1,grpngOut);

trnsfrmtnSBOX_Comp : trnsfrmtnSBOX port map(resetSboxes,HstateOut,constantRound,sboxOut);

mapping_L_Blocks_Comp : mapping_L_Blocks port map(sboxOut,funL_out);

function_P_Comp : function_P port map(funL_out,funP_out);

DeGroupingF_Comp : DeGroupingF port map(sboxOut,DegrpngOut);

truncating_H_Comp : truncating_H port map(hashLength,sigXOR2(511 downto 0),JH_Out);

Controller_Comp : Controller port map(clk,reset,enable,computingCrcls,resetSboxes,hashReadySig,
selectionMUX2x1, selectionMUX4x1,constantRound);

HstateReg_Comp : HstateRegister port map(reset,enable,hashReadySig,clk,MUX4x1Out,HstateOut);

MUX2x1_Comp : MUX2x1 port map(selectionMUX2x1,padderOut(1023 downto 512),padderOut(511 downto
0),Mword);

MUX4x1_Comp : MUX4x1 port map(selectionMUX4x1,IVOut,grpngOut,funP_out,sigXOR2,MUX4x1Out);

sigXOR3 <= MWord xor HstateOut(1023 downto 512);
sigXOR1 <= sigXOR3 & HstateOut(511 downto 0);

sigXOR4 <= MWord xor DegrpngOut(511 downto 0);
sigXOR2 <= DegrpngOut(1023 downto 512) & sigXOR4;

hashReady <= hashReadySig;

end Behavioral;
```

Κώδικας του κυκλώματος Padder (Σχήμα 6.27).

```
entity Padder is
        port(reset :in std_logic;
             dataIn :in std_logic_vector(511 downto 0);
             messageLength : in std_logic_vector(9 downto 0);
             computingCrcls : out std_logic_vector(6 downto 0);
             paddedDataOut : out std_logic_vector(1023 downto 0)
                 );
end Padder;
architecture Behavioral of Padder is
begin
Padding : process(reset,dataIn,messageLength)
         begin
         if(reset = '0')then
----------START PADDING----------
                 if(messageLength = "00000000000")then
                         paddedDataOut(1023) <= '1';
```

```
                paddedDataOut(1022 downto 0) <= (others => '0');
                computingCrcls <= "0100101";
        elsif(messageLength < 513)then
                paddedDataOut(1023 downto 512) <= dataIn;
                paddedDataOut(511 downto 10) <= (others => '0');
                paddedDataOut(1023 - conv_Integer(messageLength)) <= '1';
                paddedDataOut(9 downto 0) <= messageLength;
                computingCrcls <= "1001010";
        else
                computingCrcls <= "0000000";
                paddedDataOut <= (others => '0');
        end if;
----------END PADDING----------
    else
        computingCrcls <= "0100101";
        paddedDataOut <= (others => '0');
    end if;
     end process;
end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος Initialization Vector (Σχήματος 6.26).

```
entity IV is
    port(reset : in std_logic;
         hashLength : in std_logic_vector(1 downto 0);
         IVOut : out std_logic_vector(1023 downto 0));
end IV;
architecture Behavioral of IV is
begin
 IV : process(reset,hashLength)
    begin
    if(reset = '0')then
        case hashLength is
        when "00" => IVOut <= X"82C270E00BED02308D0C3A9E31CE34B18F0C942FBA46CD
871EC4D80AFC7971C461E01ABB69962D7BAF71893DE13D8697D2520460F7C9C094C76349CA3DA5799
CFD8B551FBDBCEB9F0834BD5BB442F8BFBA515C35B9C7999E55A44E6271CC13B385725793C185F725
45366B69005025D23390EBDB27DD1EDFCCBAADE17E603DE9";

        when "01" => IVOut <= X"C968B8E2C53A596E427E45EF1D7AE6E56145B7D906711F7
A2FC7617806A922017B2991C1B91929E2C42B4CE18CC5A2D66220BECA901B5DDFD3B205638EA7AC5
F143E8CBA6D313104B0E70054905272714CCE321E075DE5101BA800ECE20251789F5772795FD104A5F0
B8B63425F5B2381670FA3E5F907F17E28FC064E769AC90";

        when "10" => IVOut <= X"079C23AB64AB2D408CB51CE447DEE98D8D9BB1627EC25
269BAB62D2B002FFC80CBAFBCEF308C173AAD6FA3AA31194031898977423A6F4CE3BF2E732B440D
DB7DF2C43ECAA63A54E58A37B80AFC4422C5A397C3BC04E9E09137A80453E14860FA7131D33A5FD4
BEA6DCDA4AF8F43385126EC7F8F4C84958D08B9E94A34695B6A9";

        when "11" => IVOut <= X"50AB6058C60942CC4CE7A54CBDB9DC1BAF2E7AFBD1A1
5E24E5F44EABC4D5C0A14CF243660C562073999381EA9A8B3D18CF65D9FCA940B6C79E831273BEFE3
B660F9A2F7E0A32D8E017D491558E0B134005B5E4DEC44E5F3F8CBC5AEE98FD1D3214081C25E46CE6
C41B4B95BCE1BD43DB7F229EC243B680140A33B909333C0303";

        when others => IVOut <= (others => '0');
        end case;
    else
        IVOut <= (others => '0');
    end if;
```

```
        end process;
end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος Grouping (Σχήμα 6.28).

```
entity GroupingF is
        port(grpngIn : in std_logic_vector(1023 downto 0);
             grpngOut : out std_logic_vector(1023 downto 0));
end GroupingF;
architecture Behavioral of GroupingF is
begin

 grpngOut(1023 downto 1020) <= grpngIn(1023) & grpngIn(767) & grpngIn(511) & grpngIn(255); -- 0 Elmnt
 grpngOut(1019 downto 1016) <= grpngIn(895) & grpngIn(639) & grpngIn(383) & grpngIn(127); -- 1 Elmnt
 grpngOut(1015 downto 1012) <= grpngIn(1022) & grpngIn(766) & grpngIn(510) & grpngIn(254); -- 2 Elmnt
 grpngOut(1011 downto 1008) <= grpngIn(894) & grpngIn(638) & grpngIn(382) & grpngIn(126); -- 3 Elmnt
 grpngOut(1007 downto 1004) <= grpngIn(1021) & grpngIn(765) & grpngIn(509) & grpngIn(253); -- 4 Elmnt
 grpngOut(1003 downto 1000) <= grpngIn(893) & grpngIn(637) & grpngIn(381) & grpngIn(125); -- 5 Elmnt
 grpngOut(999 downto 996) <= grpngIn(1020) & grpngIn(764) & grpngIn(508) & grpngIn(252); -- 6 Elmnt
 grpngOut(995 downto 992) <= grpngIn(892) & grpngIn(636) & grpngIn(380) & grpngIn(124); -- 7 Elmnt
 grpngOut(991 downto 988) <= grpngIn(1019) & grpngIn(763) & grpngIn(507) & grpngIn(251); -- 8 Elmnt
 grpngOut(987 downto 984) <= grpngIn(891) & grpngIn(635) & grpngIn(379) & grpngIn(123); -- 9 Elmnt
 grpngOut(983 downto 980) <= grpngIn(1018) & grpngIn(762) & grpngIn(506) & grpngIn(250); -- 10 Elmnt
 grpngOut(979 downto 976) <= grpngIn(890) & grpngIn(634) & grpngIn(378) & grpngIn(122); -- 11 Elmnt
 grpngOut(975 downto 972) <= grpngIn(1017) & grpngIn(761) & grpngIn(505) & grpngIn(249); -- 12 Elmnt
 grpngOut(971 downto 968) <= grpngIn(889) & grpngIn(633) & grpngIn(377) & grpngIn(121); -- 13 Elmnt
 grpngOut(967 downto 964) <= grpngIn(1016) & grpngIn(760) & grpngIn(504) & grpngIn(248); -- 14 Elmnt
 grpngOut(963 downto 960) <= grpngIn(888) & grpngIn(632) & grpngIn(376) & grpngIn(120); -- 15 Elmnt
 grpngOut(959 downto 956) <= grpngIn(1015) & grpngIn(759) & grpngIn(503) & grpngIn(247); -- 16 Elmnt
 grpngOut(955 downto 952) <= grpngIn(887) & grpngIn(631) & grpngIn(375) & grpngIn(119); -- 17 Elmnt
 grpngOut(951 downto 948) <= grpngIn(1014) & grpngIn(758) & grpngIn(502) & grpngIn(246); -- 18 Elmnt
 grpngOut(947 downto 944) <= grpngIn(886) & grpngIn(630) & grpngIn(374) & grpngIn(118); -- 19 Elmnt
 grpngOut(943 downto 940) <= grpngIn(1013) & grpngIn(757) & grpngIn(501) & grpngIn(245); -- 20 Elmnt
 grpngOut(939 downto 936) <= grpngIn(885) & grpngIn(629) & grpngIn(373) & grpngIn(117); -- 21 Elmnt
 grpngOut(935 downto 932) <= grpngIn(1012) & grpngIn(756) & grpngIn(500) & grpngIn(244); -- 22 Elmnt
 grpngOut(931 downto 928) <= grpngIn(884) & grpngIn(628) & grpngIn(372) & grpngIn(116); -- 23 Elmnt
 grpngOut(927 downto 924) <= grpngIn(1011) & grpngIn(755) & grpngIn(499) & grpngIn(243); -- 24 Elmnt
 grpngOut(923 downto 920) <= grpngIn(883) & grpngIn(627) & grpngIn(371) & grpngIn(115); -- 25 Elmnt
 grpngOut(919 downto 916) <= grpngIn(1010) & grpngIn(754) & grpngIn(498) & grpngIn(242); -- 26 Elmnt
 grpngOut(915 downto 912) <= grpngIn(882) & grpngIn(626) & grpngIn(370) & grpngIn(114); -- 27 Elmnt
 grpngOut(911 downto 908) <= grpngIn(1009) & grpngIn(753) & grpngIn(497) & grpngIn(241); -- 28 Elmnt
 grpngOut(907 downto 904) <= grpngIn(881) & grpngIn(625) & grpngIn(369) & grpngIn(113); -- 29 Elmnt
 grpngOut(903 downto 900) <= grpngIn(1008) & grpngIn(752) & grpngIn(496) & grpngIn(240); -- 30 Elmnt
 grpngOut(899 downto 896) <= grpngIn(880) & grpngIn(624) & grpngIn(368) & grpngIn(112); -- 31 Elmnt
 grpngOut(895 downto 892) <= grpngIn(1007) & grpngIn(751) & grpngIn(495) & grpngIn(239); -- 32 Elmnt
 grpngOut(891 downto 888) <= grpngIn(879) & grpngIn(623) & grpngIn(367) & grpngIn(111); -- 33 Elmnt
 grpngOut(887 downto 884) <= grpngIn(1006) & grpngIn(750) & grpngIn(494) & grpngIn(238); -- 34 Elmnt
 grpngOut(883 downto 880) <= grpngIn(878) & grpngIn(622) & grpngIn(366) & grpngIn(110); -- 35 Elmnt
 grpngOut(879 downto 876) <= grpngIn(1005) & grpngIn(749) & grpngIn(493) & grpngIn(237); -- 36 Elmnt
 grpngOut(875 downto 872) <= grpngIn(877) & grpngIn(621) & grpngIn(365) & grpngIn(109); -- 37 Elmnt
 grpngOut(871 downto 868) <= grpngIn(1004) & grpngIn(748) & grpngIn(492) & grpngIn(236); -- 38 Elmnt
 grpngOut(867 downto 864) <= grpngIn(876) & grpngIn(620) & grpngIn(364) & grpngIn(108); -- 39 Elmnt
 grpngOut(863 downto 860) <= grpngIn(1003) & grpngIn(747) & grpngIn(491) & grpngIn(235); -- 40 Elmnt
 grpngOut(859 downto 856) <= grpngIn(875) & grpngIn(619) & grpngIn(363) & grpngIn(107); -- 41 Elmnt
 grpngOut(855 downto 852) <= grpngIn(1002) & grpngIn(746) & grpngIn(490) & grpngIn(234); -- 42 Elmnt
 grpngOut(851 downto 848) <= grpngIn(874) & grpngIn(618) & grpngIn(362) & grpngIn(106); -- 43 Elmnt
 grpngOut(847 downto 844) <= grpngIn(1001) & grpngIn(745) & grpngIn(489) & grpngIn(233); -- 44 Elmnt
 grpngOut(843 downto 840) <= grpngIn(873) & grpngIn(617) & grpngIn(361) & grpngIn(105); -- 45 Elmnt
```

```
grpngOut(839 downto 836) <= grpngIn(1000) & grpngIn(744) & grpngIn(488) & grpngIn(232); -- 46 Elmnt
grpngOut(835 downto 832) <= grpngIn(872) & grpngIn(616) & grpngIn(360) & grpngIn(104); -- 47 Elmnt
grpngOut(831 downto 828) <= grpngIn(999) & grpngIn(743) & grpngIn(487) & grpngIn(231); -- 48 Elmnt
grpngOut(827 downto 824) <= grpngIn(871) & grpngIn(615) & grpngIn(359) & grpngIn(103); -- 49 Elmnt
grpngOut(823 downto 820) <= grpngIn(998) & grpngIn(742) & grpngIn(486) & grpngIn(230); -- 50 Elmnt
grpngOut(819 downto 816) <= grpngIn(870) & grpngIn(614) & grpngIn(358) & grpngIn(102); -- 51 Elmnt
grpngOut(815 downto 812) <= grpngIn(997) & grpngIn(741) & grpngIn(485) & grpngIn(229); -- 52 Elmnt
grpngOut(811 downto 808) <= grpngIn(869) & grpngIn(613) & grpngIn(357) & grpngIn(101); -- 53 Elmnt
grpngOut(807 downto 804) <= grpngIn(996) & grpngIn(740) & grpngIn(484) & grpngIn(228); -- 54 Elmnt
grpngOut(803 downto 800) <= grpngIn(868) & grpngIn(612) & grpngIn(356) & grpngIn(100); -- 55 Elmnt
grpngOut(799 downto 796) <= grpngIn(995) & grpngIn(739) & grpngIn(483) & grpngIn(227); -- 56 Elmnt
grpngOut(795 downto 792) <= grpngIn(867) & grpngIn(611) & grpngIn(355) & grpngIn(99); -- 57 Elmnt
grpngOut(791 downto 788) <= grpngIn(994) & grpngIn(738) & grpngIn(482) & grpngIn(226); -- 58 Elmnt
grpngOut(787 downto 784) <= grpngIn(866) & grpngIn(610) & grpngIn(354) & grpngIn(98); -- 59 Elmnt
grpngOut(783 downto 780) <= grpngIn(993) & grpngIn(737) & grpngIn(481) & grpngIn(225); -- 60 Elmnt
grpngOut(779 downto 776) <= grpngIn(865) & grpngIn(609) & grpngIn(353) & grpngIn(97); -- 61 Elmnt
grpngOut(775 downto 772) <= grpngIn(992) & grpngIn(736) & grpngIn(480) & grpngIn(224); -- 62 Elmnt
grpngOut(771 downto 768) <= grpngIn(864) & grpngIn(608) & grpngIn(352) & grpngIn(96); -- 63 Elmnt
grpngOut(767 downto 764) <= grpngIn(991) & grpngIn(735) & grpngIn(479) & grpngIn(223); -- 64 Elmnt
grpngOut(763 downto 760) <= grpngIn(863) & grpngIn(607) & grpngIn(351) & grpngIn(95); -- 65 Elmnt
grpngOut(759 downto 756) <= grpngIn(990) & grpngIn(734) & grpngIn(478) & grpngIn(222); -- 66 Elmnt
grpngOut(755 downto 752) <= grpngIn(862) & grpngIn(606) & grpngIn(350) & grpngIn(94); -- 67 Elmnt
grpngOut(751 downto 748) <= grpngIn(989) & grpngIn(733) & grpngIn(477) & grpngIn(221); -- 68 Elmnt
grpngOut(747 downto 744) <= grpngIn(861) & grpngIn(605) & grpngIn(349) & grpngIn(93); -- 69 Elmnt
grpngOut(743 downto 740) <= grpngIn(988) & grpngIn(732) & grpngIn(476) & grpngIn(220); -- 70 Elmnt
grpngOut(739 downto 736) <= grpngIn(860) & grpngIn(604) & grpngIn(348) & grpngIn(92); -- 71 Elmnt
grpngOut(735 downto 732) <= grpngIn(987) & grpngIn(731) & grpngIn(475) & grpngIn(219); -- 72 Elmnt
grpngOut(731 downto 728) <= grpngIn(859) & grpngIn(603) & grpngIn(347) & grpngIn(91); -- 73 Elmnt
grpngOut(727 downto 724) <= grpngIn(986) & grpngIn(730) & grpngIn(474) & grpngIn(218); -- 74 Elmnt
grpngOut(723 downto 720) <= grpngIn(858) & grpngIn(602) & grpngIn(346) & grpngIn(90); -- 75 Elmnt
grpngOut(719 downto 716) <= grpngIn(985) & grpngIn(729) & grpngIn(473) & grpngIn(217); -- 76 Elmnt
grpngOut(715 downto 712) <= grpngIn(857) & grpngIn(601) & grpngIn(345) & grpngIn(89); -- 77 Elmnt
grpngOut(711 downto 708) <= grpngIn(984) & grpngIn(728) & grpngIn(472) & grpngIn(216); -- 78 Elmnt
grpngOut(707 downto 704) <= grpngIn(856) & grpngIn(600) & grpngIn(344) & grpngIn(88); -- 79 Elmnt
grpngOut(703 downto 700) <= grpngIn(983) & grpngIn(727) & grpngIn(471) & grpngIn(215); -- 80 Elmnt
grpngOut(699 downto 696) <= grpngIn(855) & grpngIn(599) & grpngIn(343) & grpngIn(87); -- 81 Elmnt
grpngOut(695 downto 692) <= grpngIn(982) & grpngIn(726) & grpngIn(470) & grpngIn(214); -- 82 Elmnt
grpngOut(691 downto 688) <= grpngIn(854) & grpngIn(598) & grpngIn(342) & grpngIn(86); -- 83 Elmnt
grpngOut(687 downto 684) <= grpngIn(981) & grpngIn(725) & grpngIn(469) & grpngIn(213); -- 84 Elmnt
grpngOut(683 downto 680) <= grpngIn(853) & grpngIn(597) & grpngIn(341) & grpngIn(85); -- 85 Elmnt
grpngOut(679 downto 676) <= grpngIn(980) & grpngIn(724) & grpngIn(468) & grpngIn(212); -- 86 Elmnt
grpngOut(675 downto 672) <= grpngIn(852) & grpngIn(596) & grpngIn(340) & grpngIn(84); -- 87 Elmnt
grpngOut(671 downto 668) <= grpngIn(979) & grpngIn(723) & grpngIn(467) & grpngIn(211); -- 88 Elmnt
grpngOut(667 downto 664) <= grpngIn(851) & grpngIn(595) & grpngIn(339) & grpngIn(83); -- 89 Elmnt
grpngOut(663 downto 660) <= grpngIn(978) & grpngIn(722) & grpngIn(466) & grpngIn(210); -- 90 Elmnt
grpngOut(659 downto 656) <= grpngIn(850) & grpngIn(594) & grpngIn(338) & grpngIn(82); -- 91 Elmnt
grpngOut(655 downto 652) <= grpngIn(977) & grpngIn(721) & grpngIn(465) & grpngIn(209); -- 92 Elmnt
grpngOut(651 downto 648) <= grpngIn(849) & grpngIn(593) & grpngIn(337) & grpngIn(81); -- 93 Elmnt
grpngOut(647 downto 644) <= grpngIn(976) & grpngIn(720) & grpngIn(464) & grpngIn(208); -- 94 Elmnt
grpngOut(643 downto 640) <= grpngIn(848) & grpngIn(592) & grpngIn(336) & grpngIn(80); -- 95 Elmnt
grpngOut(639 downto 636) <= grpngIn(975) & grpngIn(719) & grpngIn(463) & grpngIn(207); -- 96 Elmnt
grpngOut(635 downto 632) <= grpngIn(847) & grpngIn(591) & grpngIn(335) & grpngIn(79); -- 97 Elmnt
grpngOut(631 downto 628) <= grpngIn(974) & grpngIn(718) & grpngIn(462) & grpngIn(206); -- 98 Elmnt
grpngOut(627 downto 624) <= grpngIn(846) & grpngIn(590) & grpngIn(334) & grpngIn(78); -- 99 Elmnt
grpngOut(623 downto 620) <= grpngIn(973) & grpngIn(717) & grpngIn(461) & grpngIn(205); -- 100 Elmnt
grpngOut(619 downto 616) <= grpngIn(845) & grpngIn(589) & grpngIn(333) & grpngIn(77); -- 101 Elmnt
grpngOut(615 downto 612) <= grpngIn(972) & grpngIn(716) & grpngIn(460) & grpngIn(204); -- 102 Elmnt
grpngOut(611 downto 608) <= grpngIn(844) & grpngIn(588) & grpngIn(332) & grpngIn(76); -- 103 Elmnt
grpngOut(607 downto 604) <= grpngIn(971) & grpngIn(715) & grpngIn(459) & grpngIn(203); -- 104 Elmnt
grpngOut(603 downto 600) <= grpngIn(843) & grpngIn(587) & grpngIn(331) & grpngIn(75); -- 105 Elmnt
```

```
grpngOut(599 downto 596) <= grpngIn(970) & grpngIn(714) & grpngIn(458) & grpngIn(202); -- 106 Elmnt
grpngOut(595 downto 592) <= grpngIn(842) & grpngIn(586) & grpngIn(330) & grpngIn(74); -- 107 Elmnt
grpngOut(591 downto 588) <= grpngIn(969) & grpngIn(713) & grpngIn(457) & grpngIn(201); -- 108 Elmnt
grpngOut(587 downto 584) <= grpngIn(841) & grpngIn(585) & grpngIn(329) & grpngIn(73); -- 109 Elmnt
grpngOut(583 downto 580) <= grpngIn(968) & grpngIn(712) & grpngIn(456) & grpngIn(200); -- 110 Elmnt
grpngOut(579 downto 576) <= grpngIn(840) & grpngIn(584) & grpngIn(328) & grpngIn(72); -- 111 Elmnt
grpngOut(575 downto 572) <= grpngIn(967) & grpngIn(711) & grpngIn(455) & grpngIn(199); -- 112 Elmnt
grpngOut(571 downto 568) <= grpngIn(839) & grpngIn(583) & grpngIn(327) & grpngIn(71); -- 113 Elmnt
grpngOut(567 downto 564) <= grpngIn(966) & grpngIn(710) & grpngIn(454) & grpngIn(198); -- 114 Elmnt
grpngOut(563 downto 560) <= grpngIn(838) & grpngIn(582) & grpngIn(326) & grpngIn(70); -- 115 Elmnt
grpngOut(559 downto 556) <= grpngIn(965) & grpngIn(709) & grpngIn(453) & grpngIn(197); -- 116 Elmnt
grpngOut(555 downto 552) <= grpngIn(837) & grpngIn(581) & grpngIn(325) & grpngIn(69); -- 117 Elmnt
grpngOut(551 downto 548) <= grpngIn(964) & grpngIn(708) & grpngIn(452) & grpngIn(196); -- 118 Elmnt
grpngOut(547 downto 544) <= grpngIn(836) & grpngIn(580) & grpngIn(324) & grpngIn(68); -- 119 Elmnt
grpngOut(543 downto 540) <= grpngIn(963) & grpngIn(707) & grpngIn(451) & grpngIn(195); -- 120 Elmnt
grpngOut(539 downto 536) <= grpngIn(835) & grpngIn(579) & grpngIn(323) & grpngIn(67); -- 121 Elmnt
grpngOut(535 downto 532) <= grpngIn(962) & grpngIn(706) & grpngIn(450) & grpngIn(194); -- 122 Elmnt
grpngOut(531 downto 528) <= grpngIn(834) & grpngIn(578) & grpngIn(322) & grpngIn(66); -- 123 Elmnt
grpngOut(527 downto 524) <= grpngIn(961) & grpngIn(705) & grpngIn(449) & grpngIn(193); -- 124 Elmnt
grpngOut(523 downto 520) <= grpngIn(833) & grpngIn(577) & grpngIn(321) & grpngIn(65); -- 125 Elmnt
grpngOut(519 downto 516) <= grpngIn(960) & grpngIn(704) & grpngIn(448) & grpngIn(192); -- 126 Elmnt
grpngOut(515 downto 512) <= grpngIn(832) & grpngIn(576) & grpngIn(320) & grpngIn(64); -- 127 Elmnt
grpngOut(511 downto 508) <= grpngIn(959) & grpngIn(703) & grpngIn(447) & grpngIn(191); -- 128 Elmnt
grpngOut(507 downto 504) <= grpngIn(831) & grpngIn(575) & grpngIn(319) & grpngIn(63); -- 129 Elmnt
grpngOut(503 downto 500) <= grpngIn(958) & grpngIn(702) & grpngIn(446) & grpngIn(190); -- 130 Elmnt
grpngOut(499 downto 496) <= grpngIn(830) & grpngIn(574) & grpngIn(318) & grpngIn(62); -- 131 Elmnt
grpngOut(495 downto 492) <= grpngIn(957) & grpngIn(701) & grpngIn(445) & grpngIn(189); -- 132 Elmnt
grpngOut(491 downto 488) <= grpngIn(829) & grpngIn(573) & grpngIn(317) & grpngIn(61); -- 133 Elmnt
grpngOut(487 downto 484) <= grpngIn(956) & grpngIn(700) & grpngIn(444) & grpngIn(188); -- 134 Elmnt
grpngOut(483 downto 480) <= grpngIn(828) & grpngIn(572) & grpngIn(316) & grpngIn(60); -- 135 Elmnt
grpngOut(479 downto 476) <= grpngIn(955) & grpngIn(699) & grpngIn(443) & grpngIn(187); -- 136 Elmnt
grpngOut(475 downto 472) <= grpngIn(827) & grpngIn(571) & grpngIn(315) & grpngIn(59); -- 137 Elmnt
grpngOut(471 downto 468) <= grpngIn(954) & grpngIn(698) & grpngIn(442) & grpngIn(186); -- 138 Elmnt
grpngOut(467 downto 464) <= grpngIn(826) & grpngIn(570) & grpngIn(314) & grpngIn(58); -- 139 Elmnt
grpngOut(463 downto 460) <= grpngIn(953) & grpngIn(697) & grpngIn(441) & grpngIn(185); -- 140 Elmnt
grpngOut(459 downto 456) <= grpngIn(825) & grpngIn(569) & grpngIn(313) & grpngIn(57); -- 141 Elmnt
grpngOut(455 downto 452) <= grpngIn(952) & grpngIn(696) & grpngIn(440) & grpngIn(184); -- 142 Elmnt
grpngOut(451 downto 448) <= grpngIn(824) & grpngIn(568) & grpngIn(312) & grpngIn(56); -- 143 Elmnt
grpngOut(447 downto 444) <= grpngIn(951) & grpngIn(695) & grpngIn(439) & grpngIn(183); -- 144 Elmnt
grpngOut(443 downto 440) <= grpngIn(823) & grpngIn(567) & grpngIn(311) & grpngIn(55); -- 145 Elmnt
grpngOut(439 downto 436) <= grpngIn(950) & grpngIn(694) & grpngIn(438) & grpngIn(182); -- 146 Elmnt
grpngOut(435 downto 432) <= grpngIn(822) & grpngIn(566) & grpngIn(310) & grpngIn(54); -- 147 Elmnt
grpngOut(431 downto 428) <= grpngIn(949) & grpngIn(693) & grpngIn(437) & grpngIn(181); -- 148 Elmnt
grpngOut(427 downto 424) <= grpngIn(821) & grpngIn(565) & grpngIn(309) & grpngIn(53); -- 149 Elmnt
grpngOut(423 downto 420) <= grpngIn(948) & grpngIn(692) & grpngIn(436) & grpngIn(180); -- 150 Elmnt
grpngOut(419 downto 416) <= grpngIn(820) & grpngIn(564) & grpngIn(308) & grpngIn(52); -- 151 Elmnt
grpngOut(415 downto 412) <= grpngIn(947) & grpngIn(691) & grpngIn(435) & grpngIn(179); -- 152 Elmnt
grpngOut(411 downto 408) <= grpngIn(819) & grpngIn(563) & grpngIn(307) & grpngIn(51); -- 153 Elmnt
grpngOut(407 downto 404) <= grpngIn(946) & grpngIn(690) & grpngIn(434) & grpngIn(178); -- 154 Elmnt
grpngOut(403 downto 400) <= grpngIn(818) & grpngIn(562) & grpngIn(306) & grpngIn(50); -- 155 Elmnt
grpngOut(399 downto 396) <= grpngIn(945) & grpngIn(689) & grpngIn(433) & grpngIn(177); -- 156 Elmnt
grpngOut(395 downto 392) <= grpngIn(817) & grpngIn(561) & grpngIn(305) & grpngIn(49); -- 157 Elmnt
grpngOut(391 downto 388) <= grpngIn(944) & grpngIn(688) & grpngIn(432) & grpngIn(176); -- 158 Elmnt
grpngOut(387 downto 384) <= grpngIn(816) & grpngIn(560) & grpngIn(304) & grpngIn(48); -- 159 Elmnt
grpngOut(383 downto 380) <= grpngIn(943) & grpngIn(687) & grpngIn(431) & grpngIn(175); -- 160 Elmnt
grpngOut(379 downto 376) <= grpngIn(815) & grpngIn(559) & grpngIn(303) & grpngIn(47); -- 161 Elmnt
grpngOut(375 downto 372) <= grpngIn(942) & grpngIn(686) & grpngIn(430) & grpngIn(174); -- 162 Elmnt
grpngOut(371 downto 368) <= grpngIn(814) & grpngIn(558) & grpngIn(302) & grpngIn(46); -- 163 Elmnt
grpngOut(367 downto 364) <= grpngIn(941) & grpngIn(685) & grpngIn(429) & grpngIn(173); -- 164 Elmnt
grpngOut(363 downto 360) <= grpngIn(813) & grpngIn(557) & grpngIn(301) & grpngIn(45); -- 165 Elmnt
```

```
grpngOut(359 downto 356) <= grpngIn(940) & grpngIn(684) & grpngIn(428) & grpngIn(172); -- 166 Elmnt
grpngOut(355 downto 352) <= grpngIn(812) & grpngIn(556) & grpngIn(300) & grpngIn(44); -- 167 Elmnt
grpngOut(351 downto 348) <= grpngIn(939) & grpngIn(683) & grpngIn(427) & grpngIn(171); -- 168 Elmnt
grpngOut(347 downto 344) <= grpngIn(811) & grpngIn(555) & grpngIn(299) & grpngIn(43); -- 169 Elmnt
grpngOut(343 downto 340) <= grpngIn(938) & grpngIn(682) & grpngIn(426) & grpngIn(170); -- 170 Elmnt
grpngOut(339 downto 336) <= grpngIn(810) & grpngIn(554) & grpngIn(298) & grpngIn(42); -- 171 Elmnt
grpngOut(335 downto 332) <= grpngIn(937) & grpngIn(681) & grpngIn(425) & grpngIn(169); -- 172 Elmnt
grpngOut(331 downto 328) <= grpngIn(809) & grpngIn(553) & grpngIn(297) & grpngIn(41); -- 173 Elmnt
grpngOut(327 downto 324) <= grpngIn(936) & grpngIn(680) & grpngIn(424) & grpngIn(168); -- 174 Elmnt
grpngOut(323 downto 320) <= grpngIn(808) & grpngIn(552) & grpngIn(296) & grpngIn(40); -- 175 Elmnt
grpngOut(319 downto 316) <= grpngIn(935) & grpngIn(679) & grpngIn(423) & grpngIn(167); -- 176 Elmnt
grpngOut(315 downto 312) <= grpngIn(807) & grpngIn(551) & grpngIn(295) & grpngIn(39); -- 177 Elmnt
grpngOut(311 downto 308) <= grpngIn(934) & grpngIn(678) & grpngIn(422) & grpngIn(166); -- 178 Elmnt
grpngOut(307 downto 304) <= grpngIn(806) & grpngIn(550) & grpngIn(294) & grpngIn(38); -- 179 Elmnt
grpngOut(303 downto 300) <= grpngIn(933) & grpngIn(677) & grpngIn(421) & grpngIn(165); -- 180 Elmnt
grpngOut(299 downto 296) <= grpngIn(805) & grpngIn(549) & grpngIn(293) & grpngIn(37); -- 181 Elmnt
grpngOut(295 downto 292) <= grpngIn(932) & grpngIn(676) & grpngIn(420) & grpngIn(164); -- 182 Elmnt
grpngOut(291 downto 288) <= grpngIn(804) & grpngIn(548) & grpngIn(292) & grpngIn(36); -- 183 Elmnt
grpngOut(287 downto 284) <= grpngIn(931) & grpngIn(675) & grpngIn(419) & grpngIn(163); -- 184 Elmnt
grpngOut(283 downto 280) <= grpngIn(803) & grpngIn(547) & grpngIn(291) & grpngIn(35); -- 185 Elmnt
grpngOut(279 downto 276) <= grpngIn(930) & grpngIn(674) & grpngIn(418) & grpngIn(162); -- 186 Elmnt
grpngOut(275 downto 272) <= grpngIn(802) & grpngIn(546) & grpngIn(290) & grpngIn(34); -- 187 Elmnt
grpngOut(271 downto 268) <= grpngIn(929) & grpngIn(673) & grpngIn(417) & grpngIn(161); -- 188 Elmnt
grpngOut(267 downto 264) <= grpngIn(801) & grpngIn(545) & grpngIn(289) & grpngIn(33); -- 189 Elmnt
grpngOut(263 downto 260) <= grpngIn(928) & grpngIn(672) & grpngIn(416) & grpngIn(160); -- 190 Elmnt
grpngOut(259 downto 256) <= grpngIn(800) & grpngIn(544) & grpngIn(288) & grpngIn(32); -- 191 Elmnt
grpngOut(255 downto 252) <= grpngIn(927) & grpngIn(671) & grpngIn(415) & grpngIn(159); -- 192 Elmnt
grpngOut(251 downto 248) <= grpngIn(799) & grpngIn(543) & grpngIn(287) & grpngIn(31); -- 193 Elmnt
grpngOut(247 downto 244) <= grpngIn(926) & grpngIn(670) & grpngIn(414) & grpngIn(158); -- 194 Elmnt
grpngOut(243 downto 240) <= grpngIn(798) & grpngIn(542) & grpngIn(286) & grpngIn(30); -- 195 Elmnt
grpngOut(239 downto 236) <= grpngIn(925) & grpngIn(669) & grpngIn(413) & grpngIn(157); -- 196 Elmnt
grpngOut(235 downto 232) <= grpngIn(797) & grpngIn(541) & grpngIn(285) & grpngIn(29); -- 197 Elmnt
grpngOut(231 downto 228) <= grpngIn(924) & grpngIn(668) & grpngIn(412) & grpngIn(156); -- 198 Elmnt
grpngOut(227 downto 224) <= grpngIn(796) & grpngIn(540) & grpngIn(284) & grpngIn(28); -- 199 Elmnt
grpngOut(223 downto 220) <= grpngIn(923) & grpngIn(667) & grpngIn(411) & grpngIn(155); -- 200 Elmnt
grpngOut(219 downto 216) <= grpngIn(795) & grpngIn(539) & grpngIn(283) & grpngIn(27); -- 201 Elmnt
grpngOut(215 downto 212) <= grpngIn(922) & grpngIn(666) & grpngIn(410) & grpngIn(154); -- 202 Elmnt
grpngOut(211 downto 208) <= grpngIn(794) & grpngIn(538) & grpngIn(282) & grpngIn(26); -- 203 Elmnt
grpngOut(207 downto 204) <= grpngIn(921) & grpngIn(665) & grpngIn(409) & grpngIn(153); -- 204 Elmnt
grpngOut(203 downto 200) <= grpngIn(793) & grpngIn(537) & grpngIn(281) & grpngIn(25); -- 205 Elmnt
grpngOut(199 downto 196) <= grpngIn(920) & grpngIn(664) & grpngIn(408) & grpngIn(152); -- 206 Elmnt
grpngOut(195 downto 192) <= grpngIn(792) & grpngIn(536) & grpngIn(280) & grpngIn(24); -- 207 Elmnt
grpngOut(191 downto 188) <= grpngIn(919) & grpngIn(663) & grpngIn(407) & grpngIn(151); -- 208 Elmnt
grpngOut(187 downto 184) <= grpngIn(791) & grpngIn(535) & grpngIn(279) & grpngIn(23); -- 209 Elmnt
grpngOut(183 downto 180) <= grpngIn(918) & grpngIn(662) & grpngIn(406) & grpngIn(150); -- 210 Elmnt
grpngOut(179 downto 176) <= grpngIn(790) & grpngIn(534) & grpngIn(278) & grpngIn(22); -- 211 Elmnt
grpngOut(175 downto 172) <= grpngIn(917) & grpngIn(661) & grpngIn(405) & grpngIn(149); -- 212 Elmnt
grpngOut(171 downto 168) <= grpngIn(789) & grpngIn(533) & grpngIn(277) & grpngIn(21); -- 213 Elmnt
grpngOut(167 downto 164) <= grpngIn(916) & grpngIn(660) & grpngIn(404) & grpngIn(148); -- 214 Elmnt
grpngOut(163 downto 160) <= grpngIn(788) & grpngIn(532) & grpngIn(276) & grpngIn(20); -- 215 Elmnt
grpngOut(159 downto 156) <= grpngIn(915) & grpngIn(659) & grpngIn(403) & grpngIn(147); -- 216 Elmnt
grpngOut(155 downto 152) <= grpngIn(787) & grpngIn(531) & grpngIn(275) & grpngIn(19); -- 217 Elmnt
grpngOut(151 downto 148) <= grpngIn(914) & grpngIn(658) & grpngIn(402) & grpngIn(146); -- 218 Elmnt
grpngOut(147 downto 144) <= grpngIn(786) & grpngIn(530) & grpngIn(274) & grpngIn(18); -- 219 Elmnt
grpngOut(143 downto 140) <= grpngIn(913) & grpngIn(657) & grpngIn(401) & grpngIn(145); -- 220 Elmnt
grpngOut(139 downto 136) <= grpngIn(785) & grpngIn(529) & grpngIn(273) & grpngIn(17); -- 221 Elmnt
grpngOut(135 downto 132) <= grpngIn(912) & grpngIn(656) & grpngIn(400) & grpngIn(144); -- 222 Elmnt
grpngOut(131 downto 128) <= grpngIn(784) & grpngIn(528) & grpngIn(272) & grpngIn(16); -- 223 Elmnt
grpngOut(127 downto 124) <= grpngIn(911) & grpngIn(655) & grpngIn(399) & grpngIn(143); -- 224 Elmnt
grpngOut(123 downto 120) <= grpngIn(783) & grpngIn(527) & grpngIn(271) & grpngIn(15); -- 225 Elmnt
```

```
grpngOut(119 downto 116) <= grpngIn(910) & grpngIn(654) & grpngIn(398) & grpngIn(142); -- 226 Elmnt
grpngOut(115 downto 112) <= grpngIn(782) & grpngIn(526) & grpngIn(270) & grpngIn(14); -- 227 Elmnt
grpngOut(111 downto 108) <= grpngIn(909) & grpngIn(653) & grpngIn(397) & grpngIn(141); -- 228 Elmnt
grpngOut(107 downto 104) <= grpngIn(781) & grpngIn(525) & grpngIn(269) & grpngIn(13); -- 229 Elmnt
grpngOut(103 downto 100) <= grpngIn(908) & grpngIn(652) & grpngIn(396) & grpngIn(140); -- 230 Elmnt
grpngOut(99 downto 96) <= grpngIn(780) & grpngIn(524) & grpngIn(268) & grpngIn(12); -- 231 Elmnt
grpngOut(95 downto 92) <= grpngIn(907) & grpngIn(651) & grpngIn(395) & grpngIn(139); -- 232 Elmnt
grpngOut(91 downto 88) <= grpngIn(779) & grpngIn(523) & grpngIn(267) & grpngIn(11); -- 233 Elmnt
grpngOut(87 downto 84) <= grpngIn(906) & grpngIn(650) & grpngIn(394) & grpngIn(138); -- 234 Elmnt
grpngOut(83 downto 80) <= grpngIn(778) & grpngIn(522) & grpngIn(266) & grpngIn(10); -- 235 Elmnt
grpngOut(79 downto 76) <= grpngIn(905) & grpngIn(649) & grpngIn(393) & grpngIn(137); -- 236 Elmnt
grpngOut(75 downto 72) <= grpngIn(777) & grpngIn(521) & grpngIn(265) & grpngIn(9); -- 237 Elmnt
grpngOut(71 downto 68) <= grpngIn(904) & grpngIn(648) & grpngIn(392) & grpngIn(136); -- 238 Elmnt
grpngOut(67 downto 64) <= grpngIn(776) & grpngIn(520) & grpngIn(264) & grpngIn(8); -- 239 Elmnt
grpngOut(63 downto 60) <= grpngIn(903) & grpngIn(647) & grpngIn(391) & grpngIn(135); -- 240 Elmnt
grpngOut(59 downto 56) <= grpngIn(775) & grpngIn(519) & grpngIn(263) & grpngIn(7); -- 241 Elmnt
grpngOut(55 downto 52) <= grpngIn(902) & grpngIn(646) & grpngIn(390) & grpngIn(134); -- 242 Elmnt
grpngOut(51 downto 48) <= grpngIn(774) & grpngIn(518) & grpngIn(262) & grpngIn(6); -- 243 Elmnt
grpngOut(47 downto 44) <= grpngIn(901) & grpngIn(645) & grpngIn(389) & grpngIn(133); -- 244 Elmnt
grpngOut(43 downto 40) <= grpngIn(773) & grpngIn(517) & grpngIn(261) & grpngIn(5); -- 245 Elmnt
grpngOut(39 downto 36) <= grpngIn(900) & grpngIn(644) & grpngIn(388) & grpngIn(132); -- 246 Elmnt
grpngOut(35 downto 32) <= grpngIn(772) & grpngIn(516) & grpngIn(260) & grpngIn(4); -- 247 Elmnt
grpngOut(31 downto 28) <= grpngIn(899) & grpngIn(643) & grpngIn(387) & grpngIn(131); -- 248 Elmnt
grpngOut(27 downto 24) <= grpngIn(771) & grpngIn(515) & grpngIn(259) & grpngIn(3); -- 249 Elmnt
grpngOut(23 downto 20) <= grpngIn(898) & grpngIn(642) & grpngIn(386) & grpngIn(130); -- 250 Elmnt
grpngOut(19 downto 16) <= grpngIn(770) & grpngIn(514) & grpngIn(258) & grpngIn(2); -- 251 Elmnt
grpngOut(15 downto 12) <= grpngIn(897) & grpngIn(641) & grpngIn(385) & grpngIn(129); -- 252 Elmnt
grpngOut(11 downto 8) <= grpngIn(769) & grpngIn(513) & grpngIn(257) & grpngIn(1); -- 253 Elmnt
grpngOut(7 downto 4) <= grpngIn(896) & grpngIn(640) & grpngIn(384) & grpngIn(128); -- 254 Elmnt
grpngOut(3 downto 0) <= grpngIn(768) & grpngIn(512) & grpngIn(256) & grpngIn(0); -- 255 Elmnt

end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος De-Grouping (Σχήμα 6.28).

```
entity DeGroupingF is
        port(DegrpngIn : in std_logic_vector(1023 downto 0);
             DegrpngOut : out std_logic_vector(1023 downto 0));
end DeGroupingF;
architecture Behavioral of DeGroupingF is

begin
        DegrpngOut(1023) <= DegrpngIn(1023); -- word --> 0
        DegrpngOut(767) <= DegrpngIn(1022);
        DegrpngOut(511) <= DegrpngIn(1021);
        DegrpngOut(255) <= DegrpngIn(1020);
        DegrpngOut(1022) <= DegrpngIn(1015); -- word --> 1
        DegrpngOut(766) <= DegrpngIn(1014);
        DegrpngOut(510) <= DegrpngIn(1013);
        DegrpngOut(254) <= DegrpngIn(1012);
        DegrpngOut(1021) <= DegrpngIn(1007); -- word --> 2
        DegrpngOut(765) <= DegrpngIn(1006);
        DegrpngOut(509) <= DegrpngIn(1005);
        DegrpngOut(253) <= DegrpngIn(1004);
        DegrpngOut(1020) <= DegrpngIn(999); -- word --> 3
        DegrpngOut(764) <= DegrpngIn(998);
        DegrpngOut(508) <= DegrpngIn(997);
        DegrpngOut(252) <= DegrpngIn(996);
        DegrpngOut(1019) <= DegrpngIn(991); -- word --> 4
```

```
DegrpngOut(763) <= DegrpngIn(990);
DegrpngOut(507) <= DegrpngIn(989);
DegrpngOut(251) <= DegrpngIn(988);
DegrpngOut(1018) <= DegrpngIn(983); -- word --> 5
DegrpngOut(762) <= DegrpngIn(982);
DegrpngOut(506) <= DegrpngIn(981);
DegrpngOut(250) <= DegrpngIn(980);
DegrpngOut(1017) <= DegrpngIn(975); -- word --> 6
DegrpngOut(761) <= DegrpngIn(974);
DegrpngOut(505) <= DegrpngIn(973);
DegrpngOut(249) <= DegrpngIn(972);
DegrpngOut(1016) <= DegrpngIn(967); -- word --> 7
DegrpngOut(760) <= DegrpngIn(966);
DegrpngOut(504) <= DegrpngIn(965);
DegrpngOut(248) <= DegrpngIn(964);
DegrpngOut(1015) <= DegrpngIn(959); -- word --> 8
DegrpngOut(759) <= DegrpngIn(958);
DegrpngOut(503) <= DegrpngIn(957);
DegrpngOut(247) <= DegrpngIn(956);
DegrpngOut(1014) <= DegrpngIn(951); -- word --> 9
DegrpngOut(758) <= DegrpngIn(950);
DegrpngOut(502) <= DegrpngIn(949);
DegrpngOut(246) <= DegrpngIn(948);
DegrpngOut(1013) <= DegrpngIn(943); -- word --> 10
DegrpngOut(757) <= DegrpngIn(942);
DegrpngOut(501) <= DegrpngIn(941);
DegrpngOut(245) <= DegrpngIn(940);
DegrpngOut(1012) <= DegrpngIn(935); -- word --> 11
DegrpngOut(756) <= DegrpngIn(934);
DegrpngOut(500) <= DegrpngIn(933);
DegrpngOut(244) <= DegrpngIn(932);
DegrpngOut(1011) <= DegrpngIn(927); -- word --> 12
DegrpngOut(755) <= DegrpngIn(926);
DegrpngOut(499) <= DegrpngIn(925);
DegrpngOut(243) <= DegrpngIn(924);
DegrpngOut(1010) <= DegrpngIn(919); -- word --> 13
DegrpngOut(754) <= DegrpngIn(918);
DegrpngOut(498) <= DegrpngIn(917);
DegrpngOut(242) <= DegrpngIn(916);
DegrpngOut(1009) <= DegrpngIn(911); -- word --> 14
DegrpngOut(753) <= DegrpngIn(910);
DegrpngOut(497) <= DegrpngIn(909);
DegrpngOut(241) <= DegrpngIn(908);
DegrpngOut(1008) <= DegrpngIn(903); -- word --> 15
DegrpngOut(752) <= DegrpngIn(902);
DegrpngOut(496) <= DegrpngIn(901);
DegrpngOut(240) <= DegrpngIn(900);
DegrpngOut(1007) <= DegrpngIn(895); -- word --> 16
DegrpngOut(751) <= DegrpngIn(894);
DegrpngOut(495) <= DegrpngIn(893);
DegrpngOut(239) <= DegrpngIn(892);
DegrpngOut(1006) <= DegrpngIn(887); -- word --> 17
DegrpngOut(750) <= DegrpngIn(886);
DegrpngOut(494) <= DegrpngIn(885);
DegrpngOut(238) <= DegrpngIn(884);
DegrpngOut(1005) <= DegrpngIn(879); -- word --> 18
DegrpngOut(749) <= DegrpngIn(878);
DegrpngOut(493) <= DegrpngIn(877);
DegrpngOut(237) <= DegrpngIn(876);
DegrpngOut(1004) <= DegrpngIn(871); -- word --> 19
```

```
DegrpngOut(748) <= DegrpngIn(870);
DegrpngOut(492) <= DegrpngIn(869);
DegrpngOut(236) <= DegrpngIn(868);
DegrpngOut(1003) <= DegrpngIn(863); -- word --> 20
DegrpngOut(747) <= DegrpngIn(862);
DegrpngOut(491) <= DegrpngIn(861);
DegrpngOut(235) <= DegrpngIn(860);
DegrpngOut(1002) <= DegrpngIn(855); -- word --> 21
DegrpngOut(746) <= DegrpngIn(854);
DegrpngOut(490) <= DegrpngIn(853);
DegrpngOut(234) <= DegrpngIn(852);
DegrpngOut(1001) <= DegrpngIn(847); -- word --> 22
DegrpngOut(745) <= DegrpngIn(846);
DegrpngOut(489) <= DegrpngIn(845);
DegrpngOut(233) <= DegrpngIn(844);
DegrpngOut(1000) <= DegrpngIn(839); -- word --> 23
DegrpngOut(744) <= DegrpngIn(838);
DegrpngOut(488) <= DegrpngIn(837);
DegrpngOut(232) <= DegrpngIn(836);
DegrpngOut(999) <= DegrpngIn(831); -- word --> 24
DegrpngOut(743) <= DegrpngIn(830);
DegrpngOut(487) <= DegrpngIn(829);
DegrpngOut(231) <= DegrpngIn(828);
DegrpngOut(998) <= DegrpngIn(823); -- word --> 25
DegrpngOut(742) <= DegrpngIn(822);
DegrpngOut(486) <= DegrpngIn(821);
DegrpngOut(230) <= DegrpngIn(820);
DegrpngOut(997) <= DegrpngIn(815); -- word --> 26
DegrpngOut(741) <= DegrpngIn(814);
DegrpngOut(485) <= DegrpngIn(813);
DegrpngOut(229) <= DegrpngIn(812);
DegrpngOut(996) <= DegrpngIn(807); -- word --> 27
DegrpngOut(740) <= DegrpngIn(806);
DegrpngOut(484) <= DegrpngIn(805);
DegrpngOut(228) <= DegrpngIn(804);
DegrpngOut(995) <= DegrpngIn(799); -- word --> 28
DegrpngOut(739) <= DegrpngIn(798);
DegrpngOut(483) <= DegrpngIn(797);
DegrpngOut(227) <= DegrpngIn(796);
DegrpngOut(994) <= DegrpngIn(791); -- word --> 29
DegrpngOut(738) <= DegrpngIn(790);
DegrpngOut(482) <= DegrpngIn(789);
DegrpngOut(226) <= DegrpngIn(788);
DegrpngOut(993) <= DegrpngIn(783); -- word --> 30
DegrpngOut(737) <= DegrpngIn(782);
DegrpngOut(481) <= DegrpngIn(781);
DegrpngOut(225) <= DegrpngIn(780);
DegrpngOut(992) <= DegrpngIn(775); -- word --> 31
DegrpngOut(736) <= DegrpngIn(774);
DegrpngOut(480) <= DegrpngIn(773);
DegrpngOut(224) <= DegrpngIn(772);
DegrpngOut(991) <= DegrpngIn(767); -- word --> 32
DegrpngOut(735) <= DegrpngIn(766);
DegrpngOut(479) <= DegrpngIn(765);
DegrpngOut(223) <= DegrpngIn(764);
DegrpngOut(990) <= DegrpngIn(759); -- word --> 33
DegrpngOut(734) <= DegrpngIn(758);
DegrpngOut(478) <= DegrpngIn(757);
DegrpngOut(222) <= DegrpngIn(756);
DegrpngOut(989) <= DegrpngIn(751); -- word --> 34
```

```
DegrpngOut(733) <= DegrpngIn(750);
DegrpngOut(477) <= DegrpngIn(749);
DegrpngOut(221) <= DegrpngIn(748);
DegrpngOut(988) <= DegrpngIn(743); -- word --> 35
DegrpngOut(732) <= DegrpngIn(742);
DegrpngOut(476) <= DegrpngIn(741);
DegrpngOut(220) <= DegrpngIn(740);
DegrpngOut(987) <= DegrpngIn(735); -- word --> 36
DegrpngOut(731) <= DegrpngIn(734);
DegrpngOut(475) <= DegrpngIn(733);
DegrpngOut(219) <= DegrpngIn(732);
DegrpngOut(986) <= DegrpngIn(727); -- word --> 37
DegrpngOut(730) <= DegrpngIn(726);
DegrpngOut(474) <= DegrpngIn(725);
DegrpngOut(218) <= DegrpngIn(724);
DegrpngOut(985) <= DegrpngIn(719); -- word --> 38
DegrpngOut(729) <= DegrpngIn(718);
DegrpngOut(473) <= DegrpngIn(717);
DegrpngOut(217) <= DegrpngIn(716);
DegrpngOut(984) <= DegrpngIn(711); -- word --> 39
DegrpngOut(728) <= DegrpngIn(710);
DegrpngOut(472) <= DegrpngIn(709);
DegrpngOut(216) <= DegrpngIn(708);
DegrpngOut(983) <= DegrpngIn(703); -- word --> 40
DegrpngOut(727) <= DegrpngIn(702);
DegrpngOut(471) <= DegrpngIn(701);
DegrpngOut(215) <= DegrpngIn(700);
DegrpngOut(982) <= DegrpngIn(695); -- word --> 41
DegrpngOut(726) <= DegrpngIn(694);
DegrpngOut(470) <= DegrpngIn(693);
DegrpngOut(214) <= DegrpngIn(692);
DegrpngOut(981) <= DegrpngIn(687); -- word --> 42
DegrpngOut(725) <= DegrpngIn(686);
DegrpngOut(469) <= DegrpngIn(685);
DegrpngOut(213) <= DegrpngIn(684);
DegrpngOut(980) <= DegrpngIn(679); -- word --> 43
DegrpngOut(724) <= DegrpngIn(678);
DegrpngOut(468) <= DegrpngIn(677);
DegrpngOut(212) <= DegrpngIn(676);
DegrpngOut(979) <= DegrpngIn(671); -- word --> 44
DegrpngOut(723) <= DegrpngIn(670);
DegrpngOut(467) <= DegrpngIn(669);
DegrpngOut(211) <= DegrpngIn(668);
DegrpngOut(978) <= DegrpngIn(663); -- word --> 45
DegrpngOut(722) <= DegrpngIn(662);
DegrpngOut(466) <= DegrpngIn(661);
DegrpngOut(210) <= DegrpngIn(660);
DegrpngOut(977) <= DegrpngIn(655); -- word --> 46
DegrpngOut(721) <= DegrpngIn(654);
DegrpngOut(465) <= DegrpngIn(653);
DegrpngOut(209) <= DegrpngIn(652);
DegrpngOut(976) <= DegrpngIn(647); -- word --> 47
DegrpngOut(720) <= DegrpngIn(646);
DegrpngOut(464) <= DegrpngIn(645);
DegrpngOut(208) <= DegrpngIn(644);
DegrpngOut(975) <= DegrpngIn(639); -- word --> 48
DegrpngOut(719) <= DegrpngIn(638);
DegrpngOut(463) <= DegrpngIn(637);
DegrpngOut(207) <= DegrpngIn(636);
DegrpngOut(974) <= DegrpngIn(631); -- word --> 49
```

```
DegrpngOut(718) <= DegrpngIn(630);
DegrpngOut(462) <= DegrpngIn(629);
DegrpngOut(206) <= DegrpngIn(628);
DegrpngOut(973) <= DegrpngIn(623); -- word --> 50
DegrpngOut(717) <= DegrpngIn(622);
DegrpngOut(461) <= DegrpngIn(621);
DegrpngOut(205) <= DegrpngIn(620);
DegrpngOut(972) <= DegrpngIn(615); -- word --> 51
DegrpngOut(716) <= DegrpngIn(614);
DegrpngOut(460) <= DegrpngIn(613);
DegrpngOut(204) <= DegrpngIn(612);
DegrpngOut(971) <= DegrpngIn(607); -- word --> 52
DegrpngOut(715) <= DegrpngIn(606);
DegrpngOut(459) <= DegrpngIn(605);
DegrpngOut(203) <= DegrpngIn(604);
DegrpngOut(970) <= DegrpngIn(599); -- word --> 53
DegrpngOut(714) <= DegrpngIn(598);
DegrpngOut(458) <= DegrpngIn(597);
DegrpngOut(202) <= DegrpngIn(596);
DegrpngOut(969) <= DegrpngIn(591); -- word --> 54
DegrpngOut(713) <= DegrpngIn(590);
DegrpngOut(457) <= DegrpngIn(589);
DegrpngOut(201) <= DegrpngIn(588);
DegrpngOut(968) <= DegrpngIn(583); -- word --> 55
DegrpngOut(712) <= DegrpngIn(582);
DegrpngOut(456) <= DegrpngIn(581);
DegrpngOut(200) <= DegrpngIn(580);
DegrpngOut(967) <= DegrpngIn(575); -- word --> 56
DegrpngOut(711) <= DegrpngIn(574);
DegrpngOut(455) <= DegrpngIn(573);
DegrpngOut(199) <= DegrpngIn(572);
DegrpngOut(966) <= DegrpngIn(567); -- word --> 57
DegrpngOut(710) <= DegrpngIn(566);
DegrpngOut(454) <= DegrpngIn(565);
DegrpngOut(198) <= DegrpngIn(564);
DegrpngOut(965) <= DegrpngIn(559); -- word --> 58
DegrpngOut(709) <= DegrpngIn(558);
DegrpngOut(453) <= DegrpngIn(557);
DegrpngOut(197) <= DegrpngIn(556);
DegrpngOut(964) <= DegrpngIn(551); -- word --> 59
DegrpngOut(708) <= DegrpngIn(550);
DegrpngOut(452) <= DegrpngIn(549);
DegrpngOut(196) <= DegrpngIn(548);
DegrpngOut(963) <= DegrpngIn(543); -- word --> 60
DegrpngOut(707) <= DegrpngIn(542);
DegrpngOut(451) <= DegrpngIn(541);
DegrpngOut(195) <= DegrpngIn(540);
DegrpngOut(962) <= DegrpngIn(535); -- word --> 61
DegrpngOut(706) <= DegrpngIn(534);
DegrpngOut(450) <= DegrpngIn(533);
DegrpngOut(194) <= DegrpngIn(532);
DegrpngOut(961) <= DegrpngIn(527); -- word --> 62
DegrpngOut(705) <= DegrpngIn(526);
DegrpngOut(449) <= DegrpngIn(525);
DegrpngOut(193) <= DegrpngIn(524);
DegrpngOut(960) <= DegrpngIn(519); -- word --> 63
DegrpngOut(704) <= DegrpngIn(518);
DegrpngOut(448) <= DegrpngIn(517);
DegrpngOut(192) <= DegrpngIn(516);
DegrpngOut(959) <= DegrpngIn(511); -- word --> 64
```

```
DegrpngOut(703) <= DegrpngIn(510);
DegrpngOut(447) <= DegrpngIn(509);
DegrpngOut(191) <= DegrpngIn(508);
DegrpngOut(958) <= DegrpngIn(503); -- word --> 65
DegrpngOut(702) <= DegrpngIn(502);
DegrpngOut(446) <= DegrpngIn(501);
DegrpngOut(190) <= DegrpngIn(500);
DegrpngOut(957) <= DegrpngIn(495); -- word --> 66
DegrpngOut(701) <= DegrpngIn(494);
DegrpngOut(445) <= DegrpngIn(493);
DegrpngOut(189) <= DegrpngIn(492);
DegrpngOut(956) <= DegrpngIn(487); -- word --> 67
DegrpngOut(700) <= DegrpngIn(486);
DegrpngOut(444) <= DegrpngIn(485);
DegrpngOut(188) <= DegrpngIn(484);
DegrpngOut(955) <= DegrpngIn(479); -- word --> 68
DegrpngOut(699) <= DegrpngIn(478);
DegrpngOut(443) <= DegrpngIn(477);
DegrpngOut(187) <= DegrpngIn(476);
DegrpngOut(954) <= DegrpngIn(471); -- word --> 69
DegrpngOut(698) <= DegrpngIn(470);
DegrpngOut(442) <= DegrpngIn(469);
DegrpngOut(186) <= DegrpngIn(468);
DegrpngOut(953) <= DegrpngIn(463); -- word --> 70
DegrpngOut(697) <= DegrpngIn(462);
DegrpngOut(441) <= DegrpngIn(461);
DegrpngOut(185) <= DegrpngIn(460);
DegrpngOut(952) <= DegrpngIn(455); -- word --> 71
DegrpngOut(696) <= DegrpngIn(454);
DegrpngOut(440) <= DegrpngIn(453);
DegrpngOut(184) <= DegrpngIn(452);
DegrpngOut(951) <= DegrpngIn(447); -- word --> 72
DegrpngOut(695) <= DegrpngIn(446);
DegrpngOut(439) <= DegrpngIn(445);
DegrpngOut(183) <= DegrpngIn(444);
DegrpngOut(950) <= DegrpngIn(439); -- word --> 73
DegrpngOut(694) <= DegrpngIn(438);
DegrpngOut(438) <= DegrpngIn(437);
DegrpngOut(182) <= DegrpngIn(436);
DegrpngOut(949) <= DegrpngIn(431); -- word --> 74
DegrpngOut(693) <= DegrpngIn(430);
DegrpngOut(437) <= DegrpngIn(429);
DegrpngOut(181) <= DegrpngIn(428);
DegrpngOut(948) <= DegrpngIn(423); -- word --> 75
DegrpngOut(692) <= DegrpngIn(422);
DegrpngOut(436) <= DegrpngIn(421);
DegrpngOut(180) <= DegrpngIn(420);
DegrpngOut(947) <= DegrpngIn(415); -- word --> 76
DegrpngOut(691) <= DegrpngIn(414);
DegrpngOut(435) <= DegrpngIn(413);
DegrpngOut(179) <= DegrpngIn(412);
DegrpngOut(946) <= DegrpngIn(407); -- word --> 77
DegrpngOut(690) <= DegrpngIn(406);
DegrpngOut(434) <= DegrpngIn(405);
DegrpngOut(178) <= DegrpngIn(404);
DegrpngOut(945) <= DegrpngIn(399); -- word --> 78
DegrpngOut(689) <= DegrpngIn(398);
DegrpngOut(433) <= DegrpngIn(397);
DegrpngOut(177) <= DegrpngIn(396);
DegrpngOut(944) <= DegrpngIn(391); -- word --> 79
```

```
DegrpngOut(688) <= DegrpngIn(390);
DegrpngOut(432) <= DegrpngIn(389);
DegrpngOut(176) <= DegrpngIn(388);
DegrpngOut(943) <= DegrpngIn(383); -- word --> 80
DegrpngOut(687) <= DegrpngIn(382);
DegrpngOut(431) <= DegrpngIn(381);
DegrpngOut(175) <= DegrpngIn(380);
DegrpngOut(942) <= DegrpngIn(375); -- word --> 81
DegrpngOut(686) <= DegrpngIn(374);
DegrpngOut(430) <= DegrpngIn(373);
DegrpngOut(174) <= DegrpngIn(372);
DegrpngOut(941) <= DegrpngIn(367); -- word --> 82
DegrpngOut(685) <= DegrpngIn(366);
DegrpngOut(429) <= DegrpngIn(365);
DegrpngOut(173) <= DegrpngIn(364);
DegrpngOut(940) <= DegrpngIn(359); -- word --> 83
DegrpngOut(684) <= DegrpngIn(358);
DegrpngOut(428) <= DegrpngIn(357);
DegrpngOut(172) <= DegrpngIn(356);
DegrpngOut(939) <= DegrpngIn(351); -- word --> 84
DegrpngOut(683) <= DegrpngIn(350);
DegrpngOut(427) <= DegrpngIn(349);
DegrpngOut(171) <= DegrpngIn(348);
DegrpngOut(938) <= DegrpngIn(343); -- word --> 85
DegrpngOut(682) <= DegrpngIn(342);
DegrpngOut(426) <= DegrpngIn(341);
DegrpngOut(170) <= DegrpngIn(340);
DegrpngOut(937) <= DegrpngIn(335); -- word --> 86
DegrpngOut(681) <= DegrpngIn(334);
DegrpngOut(425) <= DegrpngIn(333);
DegrpngOut(169) <= DegrpngIn(332);
DegrpngOut(936) <= DegrpngIn(327); -- word --> 87
DegrpngOut(680) <= DegrpngIn(326);
DegrpngOut(424) <= DegrpngIn(325);
DegrpngOut(168) <= DegrpngIn(324);
DegrpngOut(935) <= DegrpngIn(319); -- word --> 88
DegrpngOut(679) <= DegrpngIn(318);
DegrpngOut(423) <= DegrpngIn(317);
DegrpngOut(167) <= DegrpngIn(316);
DegrpngOut(934) <= DegrpngIn(311); -- word --> 89
DegrpngOut(678) <= DegrpngIn(310);
DegrpngOut(422) <= DegrpngIn(309);
DegrpngOut(166) <= DegrpngIn(308);
DegrpngOut(933) <= DegrpngIn(303); -- word --> 90
DegrpngOut(677) <= DegrpngIn(302);
DegrpngOut(421) <= DegrpngIn(301);
DegrpngOut(165) <= DegrpngIn(300);
DegrpngOut(932) <= DegrpngIn(295); -- word --> 91
DegrpngOut(676) <= DegrpngIn(294);
DegrpngOut(420) <= DegrpngIn(293);
DegrpngOut(164) <= DegrpngIn(292);
DegrpngOut(931) <= DegrpngIn(287); -- word --> 92
DegrpngOut(675) <= DegrpngIn(286);
DegrpngOut(419) <= DegrpngIn(285);
DegrpngOut(163) <= DegrpngIn(284);
DegrpngOut(930) <= DegrpngIn(279); -- word --> 93
DegrpngOut(674) <= DegrpngIn(278);
DegrpngOut(418) <= DegrpngIn(277);
DegrpngOut(162) <= DegrpngIn(276);
DegrpngOut(929) <= DegrpngIn(271); -- word --> 94
```

```
DegrpngOut(673) <= DegrpngIn(270);
DegrpngOut(417) <= DegrpngIn(269);
DegrpngOut(161) <= DegrpngIn(268);
DegrpngOut(928) <= DegrpngIn(263); -- word --> 95
DegrpngOut(672) <= DegrpngIn(262);
DegrpngOut(416) <= DegrpngIn(261);
DegrpngOut(160) <= DegrpngIn(260);
DegrpngOut(927) <= DegrpngIn(255); -- word --> 96
DegrpngOut(671) <= DegrpngIn(254);
DegrpngOut(415) <= DegrpngIn(253);
DegrpngOut(159) <= DegrpngIn(252);
DegrpngOut(926) <= DegrpngIn(247); -- word --> 97
DegrpngOut(670) <= DegrpngIn(246);
DegrpngOut(414) <= DegrpngIn(245);
DegrpngOut(158) <= DegrpngIn(244);
DegrpngOut(925) <= DegrpngIn(239); -- word --> 98
DegrpngOut(669) <= DegrpngIn(238);
DegrpngOut(413) <= DegrpngIn(237);
DegrpngOut(157) <= DegrpngIn(236);
DegrpngOut(924) <= DegrpngIn(231); -- word --> 99
DegrpngOut(668) <= DegrpngIn(230);
DegrpngOut(412) <= DegrpngIn(229);
DegrpngOut(156) <= DegrpngIn(228);
DegrpngOut(923) <= DegrpngIn(223); -- word --> 100
DegrpngOut(667) <= DegrpngIn(222);
DegrpngOut(411) <= DegrpngIn(221);
DegrpngOut(155) <= DegrpngIn(220);
DegrpngOut(922) <= DegrpngIn(215); -- word --> 101
DegrpngOut(666) <= DegrpngIn(214);
DegrpngOut(410) <= DegrpngIn(213);
DegrpngOut(154) <= DegrpngIn(212);
DegrpngOut(921) <= DegrpngIn(207); -- word --> 102
DegrpngOut(665) <= DegrpngIn(206);
DegrpngOut(409) <= DegrpngIn(205);
DegrpngOut(153) <= DegrpngIn(204);
DegrpngOut(920) <= DegrpngIn(199); -- word --> 103
DegrpngOut(664) <= DegrpngIn(198);
DegrpngOut(408) <= DegrpngIn(197);
DegrpngOut(152) <= DegrpngIn(196);
DegrpngOut(919) <= DegrpngIn(191); -- word --> 104
DegrpngOut(663) <= DegrpngIn(190);
DegrpngOut(407) <= DegrpngIn(189);
DegrpngOut(151) <= DegrpngIn(188);
DegrpngOut(918) <= DegrpngIn(183); -- word --> 105
DegrpngOut(662) <= DegrpngIn(182);
DegrpngOut(406) <= DegrpngIn(181);
DegrpngOut(150) <= DegrpngIn(180);
DegrpngOut(917) <= DegrpngIn(175); -- word --> 106
DegrpngOut(661) <= DegrpngIn(174);
DegrpngOut(405) <= DegrpngIn(173);
DegrpngOut(149) <= DegrpngIn(172);
DegrpngOut(916) <= DegrpngIn(167); -- word --> 107
DegrpngOut(660) <= DegrpngIn(166);
DegrpngOut(404) <= DegrpngIn(165);
DegrpngOut(148) <= DegrpngIn(164);
DegrpngOut(915) <= DegrpngIn(159); -- word --> 108
DegrpngOut(659) <= DegrpngIn(158);
DegrpngOut(403) <= DegrpngIn(157);
DegrpngOut(147) <= DegrpngIn(156);
DegrpngOut(914) <= DegrpngIn(151); -- word --> 109
```

```
DegrpngOut(658) <= DegrpngIn(150);
DegrpngOut(402) <= DegrpngIn(149);
DegrpngOut(146) <= DegrpngIn(148);
DegrpngOut(913) <= DegrpngIn(143); -- word --> 110
DegrpngOut(657) <= DegrpngIn(142);
DegrpngOut(401) <= DegrpngIn(141);
DegrpngOut(145) <= DegrpngIn(140);
DegrpngOut(912) <= DegrpngIn(135); -- word --> 111
DegrpngOut(656) <= DegrpngIn(134);
DegrpngOut(400) <= DegrpngIn(133);
DegrpngOut(144) <= DegrpngIn(132);
DegrpngOut(911) <= DegrpngIn(127); -- word --> 112
DegrpngOut(655) <= DegrpngIn(126);
DegrpngOut(399) <= DegrpngIn(125);
DegrpngOut(143) <= DegrpngIn(124);
DegrpngOut(910) <= DegrpngIn(119); -- word --> 113
DegrpngOut(654) <= DegrpngIn(118);
DegrpngOut(398) <= DegrpngIn(117);
DegrpngOut(142) <= DegrpngIn(116);
DegrpngOut(909) <= DegrpngIn(111); -- word --> 114
DegrpngOut(653) <= DegrpngIn(110);
DegrpngOut(397) <= DegrpngIn(109);
DegrpngOut(141) <= DegrpngIn(108);
DegrpngOut(908) <= DegrpngIn(103); -- word --> 115
DegrpngOut(652) <= DegrpngIn(102);
DegrpngOut(396) <= DegrpngIn(101);
DegrpngOut(140) <= DegrpngIn(100);
DegrpngOut(907) <= DegrpngIn(95); -- word --> 116
DegrpngOut(651) <= DegrpngIn(94);
DegrpngOut(395) <= DegrpngIn(93);
DegrpngOut(139) <= DegrpngIn(92);
DegrpngOut(906) <= DegrpngIn(87); -- word --> 117
DegrpngOut(650) <= DegrpngIn(86);
DegrpngOut(394) <= DegrpngIn(85);
DegrpngOut(138) <= DegrpngIn(84);
DegrpngOut(905) <= DegrpngIn(79); -- word --> 118
DegrpngOut(649) <= DegrpngIn(78);
DegrpngOut(393) <= DegrpngIn(77);
DegrpngOut(137) <= DegrpngIn(76);
DegrpngOut(904) <= DegrpngIn(71); -- word --> 119
DegrpngOut(648) <= DegrpngIn(70);
DegrpngOut(392) <= DegrpngIn(69);
DegrpngOut(136) <= DegrpngIn(68);
DegrpngOut(903) <= DegrpngIn(63); -- word --> 120
DegrpngOut(647) <= DegrpngIn(62);
DegrpngOut(391) <= DegrpngIn(61);
DegrpngOut(135) <= DegrpngIn(60);
DegrpngOut(902) <= DegrpngIn(55); -- word --> 121
DegrpngOut(646) <= DegrpngIn(54);
DegrpngOut(390) <= DegrpngIn(53);
DegrpngOut(134) <= DegrpngIn(52);
DegrpngOut(901) <= DegrpngIn(47); -- word --> 122
DegrpngOut(645) <= DegrpngIn(46);
DegrpngOut(389) <= DegrpngIn(45);
DegrpngOut(133) <= DegrpngIn(44);
DegrpngOut(900) <= DegrpngIn(39); -- word --> 123
DegrpngOut(644) <= DegrpngIn(38);
DegrpngOut(388) <= DegrpngIn(37);
DegrpngOut(132) <= DegrpngIn(36);
DegrpngOut(899) <= DegrpngIn(31); -- word --> 124
```

```
DegrpngOut(643) <= DegrpngIn(30);
DegrpngOut(387) <= DegrpngIn(29);
DegrpngOut(131) <= DegrpngIn(28);
DegrpngOut(898) <= DegrpngIn(23); -- word --> 125
DegrpngOut(642) <= DegrpngIn(22);
DegrpngOut(386) <= DegrpngIn(21);
DegrpngOut(130) <= DegrpngIn(20);
DegrpngOut(897) <= DegrpngIn(15); -- word --> 126
DegrpngOut(641) <= DegrpngIn(14);
DegrpngOut(385) <= DegrpngIn(13);
DegrpngOut(129) <= DegrpngIn(12);
DegrpngOut(896) <= DegrpngIn(7); -- word --> 127
DegrpngOut(640) <= DegrpngIn(6);
DegrpngOut(384) <= DegrpngIn(5);
DegrpngOut(128) <= DegrpngIn(4);
DegrpngOut(895) <= DegrpngIn(1019); -- word --> 128
DegrpngOut(639) <= DegrpngIn(1018);
DegrpngOut(383) <= DegrpngIn(1017);
DegrpngOut(127) <= DegrpngIn(1016);
DegrpngOut(894) <= DegrpngIn(1011); -- word --> 129
DegrpngOut(638) <= DegrpngIn(1010);
DegrpngOut(382) <= DegrpngIn(1009);
DegrpngOut(126) <= DegrpngIn(1008);
DegrpngOut(893) <= DegrpngIn(1003); -- word --> 130
DegrpngOut(637) <= DegrpngIn(1002);
DegrpngOut(381) <= DegrpngIn(1001);
DegrpngOut(125) <= DegrpngIn(1000);
DegrpngOut(892) <= DegrpngIn(995); -- word --> 131
DegrpngOut(636) <= DegrpngIn(994);
DegrpngOut(380) <= DegrpngIn(993);
DegrpngOut(124) <= DegrpngIn(992);
DegrpngOut(891) <= DegrpngIn(987); -- word --> 132
DegrpngOut(635) <= DegrpngIn(986);
DegrpngOut(379) <= DegrpngIn(985);
DegrpngOut(123) <= DegrpngIn(984);
DegrpngOut(890) <= DegrpngIn(979); -- word --> 133
DegrpngOut(634) <= DegrpngIn(978);
DegrpngOut(378) <= DegrpngIn(977);
DegrpngOut(122) <= DegrpngIn(976);
DegrpngOut(889) <= DegrpngIn(971); -- word --> 134
DegrpngOut(633) <= DegrpngIn(970);
DegrpngOut(377) <= DegrpngIn(969);
DegrpngOut(121) <= DegrpngIn(968);
DegrpngOut(888) <= DegrpngIn(963); -- word --> 135
DegrpngOut(632) <= DegrpngIn(962);
DegrpngOut(376) <= DegrpngIn(961);
DegrpngOut(120) <= DegrpngIn(960);
DegrpngOut(887) <= DegrpngIn(955); -- word --> 136
DegrpngOut(631) <= DegrpngIn(954);
DegrpngOut(375) <= DegrpngIn(953);
DegrpngOut(119) <= DegrpngIn(952);
DegrpngOut(886) <= DegrpngIn(947); -- word --> 137
DegrpngOut(630) <= DegrpngIn(946);
DegrpngOut(374) <= DegrpngIn(945);
DegrpngOut(118) <= DegrpngIn(944);
DegrpngOut(885) <= DegrpngIn(939); -- word --> 138
DegrpngOut(629) <= DegrpngIn(938);
DegrpngOut(373) <= DegrpngIn(937);
DegrpngOut(117) <= DegrpngIn(936);
DegrpngOut(884) <= DegrpngIn(931); -- word --> 139
```

```
DegrpngOut(628) <= DegrpngIn(930);
DegrpngOut(372) <= DegrpngIn(929);
DegrpngOut(116) <= DegrpngIn(928);
DegrpngOut(883) <= DegrpngIn(923); -- word --> 140
DegrpngOut(627) <= DegrpngIn(922);
DegrpngOut(371) <= DegrpngIn(921);
DegrpngOut(115) <= DegrpngIn(920);
DegrpngOut(882) <= DegrpngIn(915); -- word --> 141
DegrpngOut(626) <= DegrpngIn(914);
DegrpngOut(370) <= DegrpngIn(913);
DegrpngOut(114) <= DegrpngIn(912);
DegrpngOut(881) <= DegrpngIn(907); -- word --> 142
DegrpngOut(625) <= DegrpngIn(906);
DegrpngOut(369) <= DegrpngIn(905);
DegrpngOut(113) <= DegrpngIn(904);
DegrpngOut(880) <= DegrpngIn(899); -- word --> 143
DegrpngOut(624) <= DegrpngIn(898);
DegrpngOut(368) <= DegrpngIn(897);
DegrpngOut(112) <= DegrpngIn(896);
DegrpngOut(879) <= DegrpngIn(891); -- word --> 144
DegrpngOut(623) <= DegrpngIn(890);
DegrpngOut(367) <= DegrpngIn(889);
DegrpngOut(111) <= DegrpngIn(888);
DegrpngOut(878) <= DegrpngIn(883); -- word --> 145
DegrpngOut(622) <= DegrpngIn(882);
DegrpngOut(366) <= DegrpngIn(881);
DegrpngOut(110) <= DegrpngIn(880);
DegrpngOut(877) <= DegrpngIn(875); -- word --> 146
DegrpngOut(621) <= DegrpngIn(874);
DegrpngOut(365) <= DegrpngIn(873);
DegrpngOut(109) <= DegrpngIn(872);
DegrpngOut(876) <= DegrpngIn(867); -- word --> 147
DegrpngOut(620) <= DegrpngIn(866);
DegrpngOut(364) <= DegrpngIn(865);
DegrpngOut(108) <= DegrpngIn(864);
DegrpngOut(875) <= DegrpngIn(859); -- word --> 148
DegrpngOut(619) <= DegrpngIn(858);
DegrpngOut(363) <= DegrpngIn(857);
DegrpngOut(107) <= DegrpngIn(856);
DegrpngOut(874) <= DegrpngIn(851); -- word --> 149
DegrpngOut(618) <= DegrpngIn(850);
DegrpngOut(362) <= DegrpngIn(849);
DegrpngOut(106) <= DegrpngIn(848);
DegrpngOut(873) <= DegrpngIn(843); -- word --> 150
DegrpngOut(617) <= DegrpngIn(842);
DegrpngOut(361) <= DegrpngIn(841);
DegrpngOut(105) <= DegrpngIn(840);
DegrpngOut(872) <= DegrpngIn(835); -- word --> 151
DegrpngOut(616) <= DegrpngIn(834);
DegrpngOut(360) <= DegrpngIn(833);
DegrpngOut(104) <= DegrpngIn(832);
DegrpngOut(871) <= DegrpngIn(827); -- word --> 152
DegrpngOut(615) <= DegrpngIn(826);
DegrpngOut(359) <= DegrpngIn(825);
DegrpngOut(103) <= DegrpngIn(824);
DegrpngOut(870) <= DegrpngIn(819); -- word --> 153
DegrpngOut(614) <= DegrpngIn(818);
DegrpngOut(358) <= DegrpngIn(817);
DegrpngOut(102) <= DegrpngIn(816);
DegrpngOut(869) <= DegrpngIn(811); -- word --> 154
```

```
DegrpngOut(613) <= DegrpngIn(810);
DegrpngOut(357) <= DegrpngIn(809);
DegrpngOut(101) <= DegrpngIn(808);
DegrpngOut(868) <= DegrpngIn(803); -- word --> 155
DegrpngOut(612) <= DegrpngIn(802);
DegrpngOut(356) <= DegrpngIn(801);
DegrpngOut(100) <= DegrpngIn(800);
DegrpngOut(867) <= DegrpngIn(795); -- word --> 156
DegrpngOut(611) <= DegrpngIn(794);
DegrpngOut(355) <= DegrpngIn(793);
DegrpngOut(99) <= DegrpngIn(792);
DegrpngOut(866) <= DegrpngIn(787); -- word --> 157
DegrpngOut(610) <= DegrpngIn(786);
DegrpngOut(354) <= DegrpngIn(785);
DegrpngOut(98) <= DegrpngIn(784);
DegrpngOut(865) <= DegrpngIn(779); -- word --> 158
DegrpngOut(609) <= DegrpngIn(778);
DegrpngOut(353) <= DegrpngIn(777);
DegrpngOut(97) <= DegrpngIn(776);
DegrpngOut(864) <= DegrpngIn(771); -- word --> 159
DegrpngOut(608) <= DegrpngIn(770);
DegrpngOut(352) <= DegrpngIn(769);
DegrpngOut(96) <= DegrpngIn(768);
DegrpngOut(863) <= DegrpngIn(763); -- word --> 160
DegrpngOut(607) <= DegrpngIn(762);
DegrpngOut(351) <= DegrpngIn(761);
DegrpngOut(95) <= DegrpngIn(760);
DegrpngOut(862) <= DegrpngIn(755); -- word --> 161
DegrpngOut(606) <= DegrpngIn(754);
DegrpngOut(350) <= DegrpngIn(753);
DegrpngOut(94) <= DegrpngIn(752);
DegrpngOut(861) <= DegrpngIn(747); -- word --> 162
DegrpngOut(605) <= DegrpngIn(746);
DegrpngOut(349) <= DegrpngIn(745);
DegrpngOut(93) <= DegrpngIn(744);
DegrpngOut(860) <= DegrpngIn(739); -- word --> 163
DegrpngOut(604) <= DegrpngIn(738);
DegrpngOut(348) <= DegrpngIn(737);
DegrpngOut(92) <= DegrpngIn(736);
DegrpngOut(859) <= DegrpngIn(731); -- word --> 164
DegrpngOut(603) <= DegrpngIn(730);
DegrpngOut(347) <= DegrpngIn(729);
DegrpngOut(91) <= DegrpngIn(728);
DegrpngOut(858) <= DegrpngIn(723); -- word --> 165
DegrpngOut(602) <= DegrpngIn(722);
DegrpngOut(346) <= DegrpngIn(721);
DegrpngOut(90) <= DegrpngIn(720);
DegrpngOut(857) <= DegrpngIn(715); -- word --> 166
DegrpngOut(601) <= DegrpngIn(714);
DegrpngOut(345) <= DegrpngIn(713);
DegrpngOut(89) <= DegrpngIn(712);
DegrpngOut(856) <= DegrpngIn(707); -- word --> 167
DegrpngOut(600) <= DegrpngIn(706);
DegrpngOut(344) <= DegrpngIn(705);
DegrpngOut(88) <= DegrpngIn(704);
DegrpngOut(855) <= DegrpngIn(699); -- word --> 168
DegrpngOut(599) <= DegrpngIn(698);
DegrpngOut(343) <= DegrpngIn(697);
DegrpngOut(87) <= DegrpngIn(696);
DegrpngOut(854) <= DegrpngIn(691); -- word --> 169
```

```
DegrpngOut(598) <= DegrpngIn(690);
DegrpngOut(342) <= DegrpngIn(689);
DegrpngOut(86) <= DegrpngIn(688);
DegrpngOut(853) <= DegrpngIn(683); -- word --> 170
DegrpngOut(597) <= DegrpngIn(682);
DegrpngOut(341) <= DegrpngIn(681);
DegrpngOut(85) <= DegrpngIn(680);
DegrpngOut(852) <= DegrpngIn(675); -- word --> 171
DegrpngOut(596) <= DegrpngIn(674);
DegrpngOut(340) <= DegrpngIn(673);
DegrpngOut(84) <= DegrpngIn(672);
DegrpngOut(851) <= DegrpngIn(667); -- word --> 172
DegrpngOut(595) <= DegrpngIn(666);
DegrpngOut(339) <= DegrpngIn(665);
DegrpngOut(83) <= DegrpngIn(664);
DegrpngOut(850) <= DegrpngIn(659); -- word --> 173
DegrpngOut(594) <= DegrpngIn(658);
DegrpngOut(338) <= DegrpngIn(657);
DegrpngOut(82) <= DegrpngIn(656);
DegrpngOut(849) <= DegrpngIn(651); -- word --> 174
DegrpngOut(593) <= DegrpngIn(650);
DegrpngOut(337) <= DegrpngIn(649);
DegrpngOut(81) <= DegrpngIn(648);
DegrpngOut(848) <= DegrpngIn(643); -- word --> 175
DegrpngOut(592) <= DegrpngIn(642);
DegrpngOut(336) <= DegrpngIn(641);
DegrpngOut(80) <= DegrpngIn(640);
DegrpngOut(847) <= DegrpngIn(635); -- word --> 176
DegrpngOut(591) <= DegrpngIn(634);
DegrpngOut(335) <= DegrpngIn(633);
DegrpngOut(79) <= DegrpngIn(632);
DegrpngOut(846) <= DegrpngIn(627); -- word --> 177
DegrpngOut(590) <= DegrpngIn(626);
DegrpngOut(334) <= DegrpngIn(625);
DegrpngOut(78) <= DegrpngIn(624);
DegrpngOut(845) <= DegrpngIn(619); -- word --> 178
DegrpngOut(589) <= DegrpngIn(618);
DegrpngOut(333) <= DegrpngIn(617);
DegrpngOut(77) <= DegrpngIn(616);
DegrpngOut(844) <= DegrpngIn(611); -- word --> 179
DegrpngOut(588) <= DegrpngIn(610);
DegrpngOut(332) <= DegrpngIn(609);
DegrpngOut(76) <= DegrpngIn(608);
DegrpngOut(843) <= DegrpngIn(603); -- word --> 180
DegrpngOut(587) <= DegrpngIn(602);
DegrpngOut(331) <= DegrpngIn(601);
DegrpngOut(75) <= DegrpngIn(600);
DegrpngOut(842) <= DegrpngIn(595); -- word --> 181
DegrpngOut(586) <= DegrpngIn(594);
DegrpngOut(330) <= DegrpngIn(593);
DegrpngOut(74) <= DegrpngIn(592);
DegrpngOut(841) <= DegrpngIn(587); -- word --> 182
DegrpngOut(585) <= DegrpngIn(586);
DegrpngOut(329) <= DegrpngIn(585);
DegrpngOut(73) <= DegrpngIn(584);
DegrpngOut(840) <= DegrpngIn(579); -- word --> 183
DegrpngOut(584) <= DegrpngIn(578);
DegrpngOut(328) <= DegrpngIn(577);
DegrpngOut(72) <= DegrpngIn(576);
DegrpngOut(839) <= DegrpngIn(571); -- word --> 184
```

```
DegrpngOut(583) <= DegrpngIn(570);
DegrpngOut(327) <= DegrpngIn(569);
DegrpngOut(71) <= DegrpngIn(568);
DegrpngOut(838) <= DegrpngIn(563); -- word --> 185
DegrpngOut(582) <= DegrpngIn(562);
DegrpngOut(326) <= DegrpngIn(561);
DegrpngOut(70) <= DegrpngIn(560);
DegrpngOut(837) <= DegrpngIn(555); -- word --> 186
DegrpngOut(581) <= DegrpngIn(554);
DegrpngOut(325) <= DegrpngIn(553);
DegrpngOut(69) <= DegrpngIn(552);
DegrpngOut(836) <= DegrpngIn(547); -- word --> 187
DegrpngOut(580) <= DegrpngIn(546);
DegrpngOut(324) <= DegrpngIn(545);
DegrpngOut(68) <= DegrpngIn(544);
DegrpngOut(835) <= DegrpngIn(539); -- word --> 188
DegrpngOut(579) <= DegrpngIn(538);
DegrpngOut(323) <= DegrpngIn(537);
DegrpngOut(67) <= DegrpngIn(536);
DegrpngOut(834) <= DegrpngIn(531); -- word --> 189
DegrpngOut(578) <= DegrpngIn(530);
DegrpngOut(322) <= DegrpngIn(529);
DegrpngOut(66) <= DegrpngIn(528);
DegrpngOut(833) <= DegrpngIn(523); -- word --> 190
DegrpngOut(577) <= DegrpngIn(522);
DegrpngOut(321) <= DegrpngIn(521);
DegrpngOut(65) <= DegrpngIn(520);
DegrpngOut(832) <= DegrpngIn(515); -- word --> 191
DegrpngOut(576) <= DegrpngIn(514);
DegrpngOut(320) <= DegrpngIn(513);
DegrpngOut(64) <= DegrpngIn(512);
DegrpngOut(831) <= DegrpngIn(507); -- word --> 192
DegrpngOut(575) <= DegrpngIn(506);
DegrpngOut(319) <= DegrpngIn(505);
DegrpngOut(63) <= DegrpngIn(504);
DegrpngOut(830) <= DegrpngIn(499); -- word --> 193
DegrpngOut(574) <= DegrpngIn(498);
DegrpngOut(318) <= DegrpngIn(497);
DegrpngOut(62) <= DegrpngIn(496);
DegrpngOut(829) <= DegrpngIn(491); -- word --> 194
DegrpngOut(573) <= DegrpngIn(490);
DegrpngOut(317) <= DegrpngIn(489);
DegrpngOut(61) <= DegrpngIn(488);
DegrpngOut(828) <= DegrpngIn(483); -- word --> 195
DegrpngOut(572) <= DegrpngIn(482);
DegrpngOut(316) <= DegrpngIn(481);
DegrpngOut(60) <= DegrpngIn(480);
DegrpngOut(827) <= DegrpngIn(475); -- word --> 196
DegrpngOut(571) <= DegrpngIn(474);
DegrpngOut(315) <= DegrpngIn(473);
DegrpngOut(59) <= DegrpngIn(472);
DegrpngOut(826) <= DegrpngIn(467); -- word --> 197
DegrpngOut(570) <= DegrpngIn(466);
DegrpngOut(314) <= DegrpngIn(465);
DegrpngOut(58) <= DegrpngIn(464);
DegrpngOut(825) <= DegrpngIn(459); -- word --> 198
DegrpngOut(569) <= DegrpngIn(458);
DegrpngOut(313) <= DegrpngIn(457);
DegrpngOut(57) <= DegrpngIn(456);
DegrpngOut(824) <= DegrpngIn(451); -- word --> 199
```

```
DegrpngOut(568) <= DegrpngIn(450);
DegrpngOut(312) <= DegrpngIn(449);
DegrpngOut(56) <= DegrpngIn(448);
DegrpngOut(823) <= DegrpngIn(443); -- word --> 200
DegrpngOut(567) <= DegrpngIn(442);
DegrpngOut(311) <= DegrpngIn(441);
DegrpngOut(55) <= DegrpngIn(440);
DegrpngOut(822) <= DegrpngIn(435); -- word --> 201
DegrpngOut(566) <= DegrpngIn(434);
DegrpngOut(310) <= DegrpngIn(433);
DegrpngOut(54) <= DegrpngIn(432);
DegrpngOut(821) <= DegrpngIn(427); -- word --> 202
DegrpngOut(565) <= DegrpngIn(426);
DegrpngOut(309) <= DegrpngIn(425);
DegrpngOut(53) <= DegrpngIn(424);
DegrpngOut(820) <= DegrpngIn(419); -- word --> 203
DegrpngOut(564) <= DegrpngIn(418);
DegrpngOut(308) <= DegrpngIn(417);
DegrpngOut(52) <= DegrpngIn(416);
DegrpngOut(819) <= DegrpngIn(411); -- word --> 204
DegrpngOut(563) <= DegrpngIn(410);
DegrpngOut(307) <= DegrpngIn(409);
DegrpngOut(51) <= DegrpngIn(408);
DegrpngOut(818) <= DegrpngIn(403); -- word --> 205
DegrpngOut(562) <= DegrpngIn(402);
DegrpngOut(306) <= DegrpngIn(401);
DegrpngOut(50) <= DegrpngIn(400);
DegrpngOut(817) <= DegrpngIn(395); -- word --> 206
DegrpngOut(561) <= DegrpngIn(394);
DegrpngOut(305) <= DegrpngIn(393);
DegrpngOut(49) <= DegrpngIn(392);
DegrpngOut(816) <= DegrpngIn(387); -- word --> 207
DegrpngOut(560) <= DegrpngIn(386);
DegrpngOut(304) <= DegrpngIn(385);
DegrpngOut(48) <= DegrpngIn(384);
DegrpngOut(815) <= DegrpngIn(379); -- word --> 208
DegrpngOut(559) <= DegrpngIn(378);
DegrpngOut(303) <= DegrpngIn(377);
DegrpngOut(47) <= DegrpngIn(376);
DegrpngOut(814) <= DegrpngIn(371); -- word --> 209
DegrpngOut(558) <= DegrpngIn(370);
DegrpngOut(302) <= DegrpngIn(369);
DegrpngOut(46) <= DegrpngIn(368);
DegrpngOut(813) <= DegrpngIn(363); -- word --> 210
DegrpngOut(557) <= DegrpngIn(362);
DegrpngOut(301) <= DegrpngIn(361);
DegrpngOut(45) <= DegrpngIn(360);
DegrpngOut(812) <= DegrpngIn(355); -- word --> 211
DegrpngOut(556) <= DegrpngIn(354);
DegrpngOut(300) <= DegrpngIn(353);
DegrpngOut(44) <= DegrpngIn(352);
DegrpngOut(811) <= DegrpngIn(347); -- word --> 212
DegrpngOut(555) <= DegrpngIn(346);
DegrpngOut(299) <= DegrpngIn(345);
DegrpngOut(43) <= DegrpngIn(344);
DegrpngOut(810) <= DegrpngIn(339); -- word --> 213
DegrpngOut(554) <= DegrpngIn(338);
DegrpngOut(298) <= DegrpngIn(337);
DegrpngOut(42) <= DegrpngIn(336);
DegrpngOut(809) <= DegrpngIn(331); -- word --> 214
```

```
DegrpngOut(553) <= DegrpngIn(330);
DegrpngOut(297) <= DegrpngIn(329);
DegrpngOut(41) <= DegrpngIn(328);
DegrpngOut(808) <= DegrpngIn(323); -- word --> 215
DegrpngOut(552) <= DegrpngIn(322);
DegrpngOut(296) <= DegrpngIn(321);
DegrpngOut(40) <= DegrpngIn(320);
DegrpngOut(807) <= DegrpngIn(315); -- word --> 216
DegrpngOut(551) <= DegrpngIn(314);
DegrpngOut(295) <= DegrpngIn(313);
DegrpngOut(39) <= DegrpngIn(312);
DegrpngOut(806) <= DegrpngIn(307); -- word --> 217
DegrpngOut(550) <= DegrpngIn(306);
DegrpngOut(294) <= DegrpngIn(305);
DegrpngOut(38) <= DegrpngIn(304);
DegrpngOut(805) <= DegrpngIn(299); -- word --> 218
DegrpngOut(549) <= DegrpngIn(298);
DegrpngOut(293) <= DegrpngIn(297);
DegrpngOut(37) <= DegrpngIn(296);
DegrpngOut(804) <= DegrpngIn(291); -- word --> 219
DegrpngOut(548) <= DegrpngIn(290);
DegrpngOut(292) <= DegrpngIn(289);
DegrpngOut(36) <= DegrpngIn(288);
DegrpngOut(803) <= DegrpngIn(283); -- word --> 220
DegrpngOut(547) <= DegrpngIn(282);
DegrpngOut(291) <= DegrpngIn(281);
DegrpngOut(35) <= DegrpngIn(280);
DegrpngOut(802) <= DegrpngIn(275); -- word --> 221
DegrpngOut(546) <= DegrpngIn(274);
DegrpngOut(290) <= DegrpngIn(273);
DegrpngOut(34) <= DegrpngIn(272);
DegrpngOut(801) <= DegrpngIn(267); -- word --> 222
DegrpngOut(545) <= DegrpngIn(266);
DegrpngOut(289) <= DegrpngIn(265);
DegrpngOut(33) <= DegrpngIn(264);
DegrpngOut(800) <= DegrpngIn(259); -- word --> 223
DegrpngOut(544) <= DegrpngIn(258);
DegrpngOut(288) <= DegrpngIn(257);
DegrpngOut(32) <= DegrpngIn(256);
DegrpngOut(799) <= DegrpngIn(251); -- word --> 224
DegrpngOut(543) <= DegrpngIn(250);
DegrpngOut(287) <= DegrpngIn(249);
DegrpngOut(31) <= DegrpngIn(248);
DegrpngOut(798) <= DegrpngIn(243); -- word --> 225
DegrpngOut(542) <= DegrpngIn(242);
DegrpngOut(286) <= DegrpngIn(241);
DegrpngOut(30) <= DegrpngIn(240);
DegrpngOut(797) <= DegrpngIn(235); -- word --> 226
DegrpngOut(541) <= DegrpngIn(234);
DegrpngOut(285) <= DegrpngIn(233);
DegrpngOut(29) <= DegrpngIn(232);
DegrpngOut(796) <= DegrpngIn(227); -- word --> 227
DegrpngOut(540) <= DegrpngIn(226);
DegrpngOut(284) <= DegrpngIn(225);
DegrpngOut(28) <= DegrpngIn(224);
DegrpngOut(795) <= DegrpngIn(219); -- word --> 228
DegrpngOut(539) <= DegrpngIn(218);
DegrpngOut(283) <= DegrpngIn(217);
DegrpngOut(27) <= DegrpngIn(216);
DegrpngOut(794) <= DegrpngIn(211); -- word --> 229
```

```
DegrpngOut(538) <= DegrpngIn(210);
DegrpngOut(282) <= DegrpngIn(209);
DegrpngOut(26) <= DegrpngIn(208);
DegrpngOut(793) <= DegrpngIn(203); -- word --> 230
DegrpngOut(537) <= DegrpngIn(202);
DegrpngOut(281) <= DegrpngIn(201);
DegrpngOut(25) <= DegrpngIn(200);
DegrpngOut(792) <= DegrpngIn(195); -- word --> 231
DegrpngOut(536) <= DegrpngIn(194);
DegrpngOut(280) <= DegrpngIn(193);
DegrpngOut(24) <= DegrpngIn(192);
DegrpngOut(791) <= DegrpngIn(187); -- word --> 232
DegrpngOut(535) <= DegrpngIn(186);
DegrpngOut(279) <= DegrpngIn(185);
DegrpngOut(23) <= DegrpngIn(184);
DegrpngOut(790) <= DegrpngIn(179); -- word --> 233
DegrpngOut(534) <= DegrpngIn(178);
DegrpngOut(278) <= DegrpngIn(177);
DegrpngOut(22) <= DegrpngIn(176);
DegrpngOut(789) <= DegrpngIn(171); -- word --> 234
DegrpngOut(533) <= DegrpngIn(170);
DegrpngOut(277) <= DegrpngIn(169);
DegrpngOut(21) <= DegrpngIn(168);
DegrpngOut(788) <= DegrpngIn(163); -- word --> 235
DegrpngOut(532) <= DegrpngIn(162);
DegrpngOut(276) <= DegrpngIn(161);
DegrpngOut(20) <= DegrpngIn(160);
DegrpngOut(787) <= DegrpngIn(155); -- word --> 236
DegrpngOut(531) <= DegrpngIn(154);
DegrpngOut(275) <= DegrpngIn(153);
DegrpngOut(19) <= DegrpngIn(152);
DegrpngOut(786) <= DegrpngIn(147); -- word --> 237
DegrpngOut(530) <= DegrpngIn(146);
DegrpngOut(274) <= DegrpngIn(145);
DegrpngOut(18) <= DegrpngIn(144);
DegrpngOut(785) <= DegrpngIn(139); -- word --> 238
DegrpngOut(529) <= DegrpngIn(138);
DegrpngOut(273) <= DegrpngIn(137);
DegrpngOut(17) <= DegrpngIn(136);
DegrpngOut(784) <= DegrpngIn(131); -- word --> 239
DegrpngOut(528) <= DegrpngIn(130);
DegrpngOut(272) <= DegrpngIn(129);
DegrpngOut(16) <= DegrpngIn(128);
DegrpngOut(783) <= DegrpngIn(123); -- word --> 240
DegrpngOut(527) <= DegrpngIn(122);
DegrpngOut(271) <= DegrpngIn(121);
DegrpngOut(15) <= DegrpngIn(120);
DegrpngOut(782) <= DegrpngIn(115); -- word --> 241
DegrpngOut(526) <= DegrpngIn(114);
DegrpngOut(270) <= DegrpngIn(113);
DegrpngOut(14) <= DegrpngIn(112);
DegrpngOut(781) <= DegrpngIn(107); -- word --> 242
DegrpngOut(525) <= DegrpngIn(106);
DegrpngOut(269) <= DegrpngIn(105);
DegrpngOut(13) <= DegrpngIn(104);
DegrpngOut(780) <= DegrpngIn(99); -- word --> 243
DegrpngOut(524) <= DegrpngIn(98);
DegrpngOut(268) <= DegrpngIn(97);
DegrpngOut(12) <= DegrpngIn(96);
DegrpngOut(779) <= DegrpngIn(91); -- word --> 244
```

```
        DegrpngOut(523) <= DegrpngIn(90);
        DegrpngOut(267) <= DegrpngIn(89);
        DegrpngOut(11) <= DegrpngIn(88);
        DegrpngOut(778) <= DegrpngIn(83); -- word --> 245
        DegrpngOut(522) <= DegrpngIn(82);
        DegrpngOut(266) <= DegrpngIn(81);
        DegrpngOut(10) <= DegrpngIn(80);
        DegrpngOut(777) <= DegrpngIn(75); -- word --> 246
        DegrpngOut(521) <= DegrpngIn(74);
        DegrpngOut(265) <= DegrpngIn(73);
        DegrpngOut(9) <= DegrpngIn(72);
        DegrpngOut(776) <= DegrpngIn(67); -- word --> 247
        DegrpngOut(520) <= DegrpngIn(66);
        DegrpngOut(264) <= DegrpngIn(65);
        DegrpngOut(8) <= DegrpngIn(64);
        DegrpngOut(775) <= DegrpngIn(59); -- word --> 248
        DegrpngOut(519) <= DegrpngIn(58);
        DegrpngOut(263) <= DegrpngIn(57);
        DegrpngOut(7) <= DegrpngIn(56);
        DegrpngOut(774) <= DegrpngIn(51); -- word --> 249
        DegrpngOut(518) <= DegrpngIn(50);
        DegrpngOut(262) <= DegrpngIn(49);
        DegrpngOut(6) <= DegrpngIn(48);
        DegrpngOut(773) <= DegrpngIn(43); -- word --> 250
        DegrpngOut(517) <= DegrpngIn(42);
        DegrpngOut(261) <= DegrpngIn(41);
        DegrpngOut(5) <= DegrpngIn(40);
        DegrpngOut(772) <= DegrpngIn(35); -- word --> 251
        DegrpngOut(516) <= DegrpngIn(34);
        DegrpngOut(260) <= DegrpngIn(33);
        DegrpngOut(4) <= DegrpngIn(32);
        DegrpngOut(771) <= DegrpngIn(27); -- word --> 252
        DegrpngOut(515) <= DegrpngIn(26);
        DegrpngOut(259) <= DegrpngIn(25);
        DegrpngOut(3) <= DegrpngIn(24);
        DegrpngOut(770) <= DegrpngIn(19); -- word --> 253
        DegrpngOut(514) <= DegrpngIn(18);
        DegrpngOut(258) <= DegrpngIn(17);
        DegrpngOut(2) <= DegrpngIn(16);
        DegrpngOut(769) <= DegrpngIn(11); -- word --> 254
        DegrpngOut(513) <= DegrpngIn(10);
        DegrpngOut(257) <= DegrpngIn(9);
        DegrpngOut(1) <= DegrpngIn(8);
        DegrpngOut(768) <= DegrpngIn(3); -- word --> 255
        DegrpngOut(512) <= DegrpngIn(2);
        DegrpngOut(256) <= DegrpngIn(1);
        DegrpngOut(0) <= DegrpngIn(0);

end Behavioral;
```

Κώδικας υλοποίησης του κυκλώματος Constant Value (Σχήματος 6.29).

```
entity constantsValueComp is
        port(
                        round : in integer range 0 to 36;
                        constantVal : out std_logic_vector(255 downto 0)
                );
end constantsValueComp;
```

```
architecture Behavioral of constantsValueComp is
begin

constantVal <=
        X"6a09e667f3bcc908b2fb1366ea957d3e3adec17512775099da2f590b0667322a" when round = 1 else
        X"bb896bf05955abcd5281828d66e7d99ac4203494f89bf12817deb43288712231" when round = 2 else
        X"1836e76b12d79c55118a1139d2417df52a2021225ff6350063d88e5f1f91631c" when round = 3 else
        X"263085a7000fa9c3317c6ca8ab65f7a7713cf4201060ce886af855a90d6a4eed" when round = 4 else
        X"1cebafd51a156aeb62a11fb3be2e14f60b7e48de85814270fd62e97614d7b441" when round = 5 else
        X"e5564cb574f7e09c75e2e244929e9549279ab224a28e445d57185e7d7a09fdc1" when round = 6 else
        X"5820f0f0d764cff3a5552a5e41a82b9eff6ee0aa615773bb07e8603424c3cf8a" when round = 7 else
        X"b126fb741733c5bfcef6f43a62e8e5706a26656028aa897ec1ea4616ce8fd510" when round = 8 else
        X"dbf0de32bca77254bb4f562581a3bc991cf94f225652c27f14eae958ae6aa616" when round = 9 else
        X"e6113be617f45f3de53cff03919a94c32c927b093ac8f23b47f7189aadb9bc67" when round = 10 else
        X"80d0d26052ca45d593ab5fb3102506390083afb5ffe107dacfcba7dbe601a12b" when round = 11 else
        X"43af1c76126714dfa950c368787c81ae3beecf956c85c962086ae16e40ebb0b4" when round = 12 else
        X"9aee8994d2d74a5cdb7b1ef294eed5c1520724dd8ed58c92d3f0e174b0c32045" when round = 13 else
        X"0b2aa58ceb3bdb9e1eef66b376e0c565d5d8fe7bacb8da866f859ac521f3d571" when round = 14 else
        X"7a1523ef3d970a3a9b0b4d610e02749d37b8d57c1885fe4206a7f338e8356866" when round = 15 else
        X"2c2db8f7876685f2cd9a2e0ddb64c9d5bf13905371fc39e0fa86e1477234a297" when round = 16 else
        X"9df085eb2544ebf62b50686a71e6e828dfed9dbe0b106c9452ceddff3d138990" when round = 17 else
        X"e6e5c42cb2d460c9d6e4791a1681bb2e222e54558eb78d5244e217d1bfcf5058" when round = 18 else
        X"8f1f57e44e126210f00763ff57da208a5093b8ff7947534a4c260a17642f72b2" when round = 19 else
        X"ae4ef4792ea148608cf116cb2bff66e8fc74811266cd641112cd17801ed38b59" when round = 20 else
        X"91a744efbf68b192d0549b608bdb3191fc12a0e83543cec5f882250b244f78e4" when round = 21 else
        X"4b5d27d3368f9c17d4b2a2b216c7e74e7714d2cc03e1e44588cd9936de74357c" when round = 22 else
        X"0ea17cafb8286131bda9e3757b3610aa3f77a6d0575053fc926eea7e237df289" when round = 23 else
        X"848af9f57eb1a616e2c342c8cea528b8a95a5d16d9d87be9bb3784d0c351c32b" when round = 24 else
        X"c0435cc3654fb85dd9335ba91ac3dbde1f85d567d7ad16f9de6e009bca3f95b5" when round = 25 else
        X"927547fe5e5e45e2fe99f1651ea1cbf097dc3a3d40ddd21cee260543c288ec6b" when round = 26 else
        X"c117a3770d3a34469d50dfa7db020300d306a365374fa828c8b780ee1b9d7a34" when round = 27 else
        X"8ff2178ae2dbe5e872fac789a34bc228debf54a882743caad14f3a550fdbe68f" when round = 28 else
        X"abd06c52ed58ff091205d0f627574c8cbc1fe7cf79210f5a2286f6e23a27efa0" when round = 29 else
        X"631f4acb8d3ca4253e301849f157571d3211b6c1045347befb7c77df3c6ca7bd" when round = 30 else
        X"ae88f2342c23344590be2014fab4f179fd4bf7c90db14fa4018fcce689d2127b" when round = 31 else
        X"93b89385546d71379fe41c39bc602e8b7c8b2f78ee914d1f0af0d437a189a8a4" when round = 32 else
        X"1d1e036abeef3f44848cd76ef6baa889fcec56cd7967eb909a464bfc23c72435" when round = 33 else
        X"a8e4ede4c5fe5e88d4fb192e0a0821e935ba145bbfc59c2508282755a5df53a5" when round = 34 else
        X"8e4e37a3b970f079ae9d22a499a714c875760273f74a9398995d32c05027d810" when round = 35 else
        X"61cfa42792f93b9fde36eb163e978709fafa7616ec3c7dad0135806c3d91a21b" when round = 36 else
        (others => '0');
end Behavioral;
```

Κώδικας υλοποίησης ενός Sbox (Σχήμα 6.29).

```
entity SBOX is
        port(dataIn : in std_logic_vector(3 downto 0);
             selSbox : in std_logic;
             dataOut : out std_logic_vector(3 downto 0));
end SBOX;
architecture Behavioral of SBOX is
begin

        process(selSbox,dataIn)
        begin
                if(selSbox = '0')then
                        case dataIn is
                                when X"0" => dataOut <= X"9";
```

```
                    when X"1" => dataOut <= X"0";
                    when X"2" => dataOut <= X"4";
                    when X"3" => dataOut <= X"B";
                    when X"4" => dataOut <= X"D";
                    when X"5" => dataOut <= X"C";
                    when X"6" => dataOut <= X"3";
                    when X"7" => dataOut <= X"F";
                    when X"8" => dataOut <= X"1";
                    when X"9" => dataOut <= X"A";
                    when X"A" => dataOut <= X"2";
                    when X"B" => dataOut <= X"6";
                    when X"C" => dataOut <= X"7";
                    when X"D" => dataOut <= X"5";
                    when X"E" => dataOut <= X"8";
                    when X"F" => dataOut <= X"E";
                    when others => dataOut <= X"0";
                end case;
            else
                case dataIn is
                    when X"0" => dataOut <= X"3";
                    when X"1" => dataOut <= X"C";
                    when X"2" => dataOut <= X"6";
                    when X"3" => dataOut <= X"D";
                    when X"4" => dataOut <= X"5";
                    when X"5" => dataOut <= X"7";
                    when X"6" => dataOut <= X"1";
                    when X"7" => dataOut <= X"9";
                    when X"8" => dataOut <= X"F";
                    when X"9" => dataOut <= X"2";
                    when X"A" => dataOut <= X"0";
                    when X"B" => dataOut <= X"4";
                    when X"C" => dataOut <= X"B";
                    when X"D" => dataOut <= X"A";
                    when X"E" => dataOut <= X"E";
                    when X"F" => dataOut <= X"8";
                    when others => dataOut <= X"0";
                end case;
            end if;
    end process;
end Behavioral;
```

Κώδικας υλοποίησης κυκλώματος διασύνδεσης των SBoxes με το κύκλωμα Constant Value (Σχήμα 6.29).

```
entity trnsfrmtnSBOX is
    port(resetSboxes : in std_logic;
         dataIn : in std_logic_vector(1023 downto 0);
         round  : in integer range 0 to 36;
         dataOut : out std_logic_vector(1023 downto 0));
end trnsfrmtnSBOX;
architecture Behavioral of trnsfrmtnSBOX is
    Component constantsValueComp
    port(round : in integer range 0 to 36;
         constantVal : out std_logic_vector(255 downto 0));
    end component;

    Component SBOX
    port(dataIn : in std_logic_vector(3 downto 0);
```

```
            selSbox : in std_logic;
            dataOut : out std_logic_vector(3 downto 0));
        end component;

SIGNAL sig : std_logic_vector(1023          downto 0);
SIGNAL constantVal : std_logic_vector(255 downto 0);
begin
constantsValueComp_Comp : constantsValueComp port map(round,constantVal);

Gen: FOR i IN 0 TO 255 Generate
        SBOX_Compi: SBOX port map(sig(1023-i*4 downto 1020-i*4),constantVal(255-i),
                                                        dataOut(1023-i*4 downto 1020-i*4));
        End Generate;

        sig <= dataIn when (resetSboxes = '0') else (others => '1');
end Behavioral;
```

Κώδικας υλοποίησης κυκλώματος L Function (Σχήμα 6.30).

```
entity function_L is
        port(dataIn : in std_logic_vector(7 downto 0);
            dataOut : out std_logic_vector(7 downto 0));
end function_L;
architecture Behavioral of function_L is
SIGNAL sigD : std_logic_vector(3 downto 0);
SIGNAL sig,sig2 : std_logic;
begin
        sigD(3) <= dataIn(3) xor dataIn(6);
        sigD(2) <= dataIn(2) xor dataIn(5);
        sig <= dataIn(1) xor dataIn(4);
        sigD(1) <= sig xor dataIn(7);
        sigD(0) <= dataIn(0) xor dataIn(7);

        dataOut(7) <= dataIn(7) xor sigD(2);
        dataOut(6) <= dataIn(6) xor sigD(1);
        sig2 <=  sigD(3) xor sigD(0);
        dataOut(5) <= sig2 xor dataIn(5);
        dataOut(4) <= dataIn(4) xor sigD(3);

        dataOut(3) <= sigD(3);
        dataOut(2) <= sigD(2);
        dataOut(1) <= sigD(1);
        dataOut(0) <= sigD(0);
end Behavioral;
```

Κώδικας υλοποίησης κυκλώματος διασύνδεσης των κυκλωμάτων L Function.

```
entity mapping_L_Blocks is
        port(dataIn : in std_logic_vector(1023 downto 0);
            dataOut : out std_logic_vector(1023 downto 0));
end mapping_L_Blocks;
architecture Behavioral of mapping_L_Blocks is
        component function_L
        port(dataIn : in std_logic_vector(7 downto 0);
            dataOut : out std_logic_vector(7 downto 0));
        end component;
begin
--
```

```
Gen: FOR i IN 0 TO 127 Generate
        function_L_Compi: function_L port map(dataIn(1023-i*8 downto 1016-i*8),
                                              dataOut(1023-i*8 downto 1016-i*8));
        End Generate;
end Behavioral;
```

Κώδικας υλοποίησης κυκλώματος P Function (Σχήμα 6.31).

```
entity function_P is
        port(dataIn : in std_Logic_vector(1023 downto 0);
             dataOut : out std_Logic_vector(1023 downto 0));
end function_P;
architecture Behavioral of function_P is

 signal sig_p,sigp_2 : std_Logic_vector(1023 downto 0);
begin
        sig_p(1023 downto 1020) <= dataIn(1023 downto 1020); -- word -> 0
        sig_p(1019 downto 1016) <= dataIn(1019 downto 1016); -- word -> 1
        sig_p(1015 downto 1012) <= dataIn(1011 downto 1008); -- word -> 2
        sig_p(1011 downto 1008) <= dataIn(1015 downto 1012); -- word -> 3
        sig_p(1007 downto 1004) <= dataIn(1007 downto 1004); -- word -> 4
        sig_p(1003 downto 1000) <= dataIn(1003 downto 1000); -- word -> 5
        sig_p(999 downto 996) <= dataIn(995 downto 992); -- word -> 6
        sig_p(995 downto 992) <= dataIn(999 downto 996); -- word -> 7
        sig_p(991 downto 988) <= dataIn(991 downto 988); -- word -> 8
        sig_p(987 downto 984) <= dataIn(987 downto 984); -- word -> 9
        sig_p(983 downto 980) <= dataIn(979 downto 976); -- word -> 10
        sig_p(979 downto 976) <= dataIn(983 downto 980); -- word -> 11
        sig_p(975 downto 972) <= dataIn(975 downto 972); -- word -> 12
        sig_p(971 downto 968) <= dataIn(971 downto 968); -- word -> 13
        sig_p(967 downto 964) <= dataIn(963 downto 960); -- word -> 14
        sig_p(963 downto 960) <= dataIn(967 downto 964); -- word -> 15
        sig_p(959 downto 956) <= dataIn(959 downto 956); -- word -> 16
        sig_p(955 downto 952) <= dataIn(955 downto 952); -- word -> 17
        sig_p(951 downto 948) <= dataIn(947 downto 944); -- word -> 18
        sig_p(947 downto 944) <= dataIn(951 downto 948); -- word -> 19
        sig_p(943 downto 940) <= dataIn(943 downto 940); -- word -> 20
        sig_p(939 downto 936) <= dataIn(939 downto 936); -- word -> 21
        sig_p(935 downto 932) <= dataIn(931 downto 928); -- word -> 22
        sig_p(931 downto 928) <= dataIn(935 downto 932); -- word -> 23
        sig_p(927 downto 924) <= dataIn(927 downto 924); -- word -> 24
        sig_p(923 downto 920) <= dataIn(923 downto 920); -- word -> 25
        sig_p(919 downto 916) <= dataIn(915 downto 912); -- word -> 26
        sig_p(915 downto 912) <= dataIn(919 downto 916); -- word -> 27
        sig_p(911 downto 908) <= dataIn(911 downto 908); -- word -> 28
        sig_p(907 downto 904) <= dataIn(907 downto 904); -- word -> 29
        sig_p(903 downto 900) <= dataIn(899 downto 896); -- word -> 30
        sig_p(899 downto 896) <= dataIn(903 downto 900); -- word -> 31
        sig_p(895 downto 892) <= dataIn(895 downto 892); -- word -> 32
        sig_p(891 downto 888) <= dataIn(891 downto 888); -- word -> 33
        sig_p(887 downto 884) <= dataIn(883 downto 880); -- word -> 34
        sig_p(883 downto 880) <= dataIn(887 downto 884); -- word -> 35
        sig_p(879 downto 876) <= dataIn(879 downto 876); -- word -> 36
        sig_p(875 downto 872) <= dataIn(875 downto 872); -- word -> 37
        sig_p(871 downto 868) <= dataIn(867 downto 864); -- word -> 38
        sig_p(867 downto 864) <= dataIn(871 downto 868); -- word -> 39
        sig_p(863 downto 860) <= dataIn(863 downto 860); -- word -> 40
        sig_p(859 downto 856) <= dataIn(859 downto 856); -- word -> 41
        sig_p(855 downto 852) <= dataIn(851 downto 848); -- word -> 42
        sig_p(851 downto 848) <= dataIn(855 downto 852); -- word -> 43
```

```
sig_p(847 downto 844) <= dataIn(847 downto 844); -- word -> 44
sig_p(843 downto 840) <= dataIn(843 downto 840); -- word -> 45
sig_p(839 downto 836) <= dataIn(835 downto 832); -- word -> 46
sig_p(835 downto 832) <= dataIn(839 downto 836); -- word -> 47
sig_p(831 downto 828) <= dataIn(831 downto 828); -- word -> 48
sig_p(827 downto 824) <= dataIn(827 downto 824); -- word -> 49
sig_p(823 downto 820) <= dataIn(819 downto 816); -- word -> 50
sig_p(819 downto 816) <= dataIn(823 downto 820); -- word -> 51
sig_p(815 downto 812) <= dataIn(815 downto 812); -- word -> 52
sig_p(811 downto 808) <= dataIn(811 downto 808); -- word -> 53
sig_p(807 downto 804) <= dataIn(803 downto 800); -- word -> 54
sig_p(803 downto 800) <= dataIn(807 downto 804); -- word -> 55
sig_p(799 downto 796) <= dataIn(799 downto 796); -- word -> 56
sig_p(795 downto 792) <= dataIn(795 downto 792); -- word -> 57
sig_p(791 downto 788) <= dataIn(787 downto 784); -- word -> 58
sig_p(787 downto 784) <= dataIn(791 downto 788); -- word -> 59
sig_p(783 downto 780) <= dataIn(783 downto 780); -- word -> 60
sig_p(779 downto 776) <= dataIn(779 downto 776); -- word -> 61
sig_p(775 downto 772) <= dataIn(771 downto 768); -- word -> 62
sig_p(771 downto 768) <= dataIn(775 downto 772); -- word -> 63
sig_p(767 downto 764) <= dataIn(767 downto 764); -- word -> 64
sig_p(763 downto 760) <= dataIn(763 downto 760); -- word -> 65
sig_p(759 downto 756) <= dataIn(755 downto 752); -- word -> 66
sig_p(755 downto 752) <= dataIn(759 downto 756); -- word -> 67
sig_p(751 downto 748) <= dataIn(751 downto 748); -- word -> 68
sig_p(747 downto 744) <= dataIn(747 downto 744); -- word -> 69
sig_p(743 downto 740) <= dataIn(739 downto 736); -- word -> 70
sig_p(739 downto 736) <= dataIn(743 downto 740); -- word -> 71
sig_p(735 downto 732) <= dataIn(735 downto 732); -- word -> 72
sig_p(731 downto 728) <= dataIn(731 downto 728); -- word -> 73
sig_p(727 downto 724) <= dataIn(723 downto 720); -- word -> 74
sig_p(723 downto 720) <= dataIn(727 downto 724); -- word -> 75
sig_p(719 downto 716) <= dataIn(719 downto 716); -- word -> 76
sig_p(715 downto 712) <= dataIn(715 downto 712); -- word -> 77
sig_p(711 downto 708) <= dataIn(707 downto 704); -- word -> 78
sig_p(707 downto 704) <= dataIn(711 downto 708); -- word -> 79
sig_p(703 downto 700) <= dataIn(703 downto 700); -- word -> 80
sig_p(699 downto 696) <= dataIn(699 downto 696); -- word -> 81
sig_p(695 downto 692) <= dataIn(691 downto 688); -- word -> 82
sig_p(691 downto 688) <= dataIn(695 downto 692); -- word -> 83
sig_p(687 downto 684) <= dataIn(687 downto 684); -- word -> 84
sig_p(683 downto 680) <= dataIn(683 downto 680); -- word -> 85
sig_p(679 downto 676) <= dataIn(675 downto 672); -- word -> 86
sig_p(675 downto 672) <= dataIn(679 downto 676); -- word -> 87
sig_p(671 downto 668) <= dataIn(671 downto 668); -- word -> 88
sig_p(667 downto 664) <= dataIn(667 downto 664); -- word -> 89
sig_p(663 downto 660) <= dataIn(659 downto 656); -- word -> 90
sig_p(659 downto 656) <= dataIn(663 downto 660); -- word -> 91
sig_p(655 downto 652) <= dataIn(655 downto 652); -- word -> 92
sig_p(651 downto 648) <= dataIn(651 downto 648); -- word -> 93
sig_p(647 downto 644) <= dataIn(643 downto 640); -- word -> 94
sig_p(643 downto 640) <= dataIn(647 downto 644); -- word -> 95
sig_p(639 downto 636) <= dataIn(639 downto 636); -- word -> 96
sig_p(635 downto 632) <= dataIn(635 downto 632); -- word -> 97
sig_p(631 downto 628) <= dataIn(627 downto 624); -- word -> 98
sig_p(627 downto 624) <= dataIn(631 downto 628); -- word -> 99
sig_p(623 downto 620) <= dataIn(623 downto 620); -- word -> 100
sig_p(619 downto 616) <= dataIn(619 downto 616); -- word -> 101
sig_p(615 downto 612) <= dataIn(611 downto 608); -- word -> 102
sig_p(611 downto 608) <= dataIn(615 downto 612); -- word -> 103
```

```
sig_p(607 downto 604) <= dataIn(607 downto 604); -- word -> 104
sig_p(603 downto 600) <= dataIn(603 downto 600); -- word -> 105
sig_p(599 downto 596) <= dataIn(595 downto 592); -- word -> 106
sig_p(595 downto 592) <= dataIn(599 downto 596); -- word -> 107
sig_p(591 downto 588) <= dataIn(591 downto 588); -- word -> 108
sig_p(587 downto 584) <= dataIn(587 downto 584); -- word -> 109
sig_p(583 downto 580) <= dataIn(579 downto 576); -- word -> 110
sig_p(579 downto 576) <= dataIn(583 downto 580); -- word -> 111
sig_p(575 downto 572) <= dataIn(575 downto 572); -- word -> 112
sig_p(571 downto 568) <= dataIn(571 downto 568); -- word -> 113
sig_p(567 downto 564) <= dataIn(563 downto 560); -- word -> 114
sig_p(563 downto 560) <= dataIn(567 downto 564); -- word -> 115
sig_p(559 downto 556) <= dataIn(559 downto 556); -- word -> 116
sig_p(555 downto 552) <= dataIn(555 downto 552); -- word -> 117
sig_p(551 downto 548) <= dataIn(547 downto 544); -- word -> 118
sig_p(547 downto 544) <= dataIn(551 downto 548); -- word -> 119
sig_p(543 downto 540) <= dataIn(543 downto 540); -- word -> 120
sig_p(539 downto 536) <= dataIn(539 downto 536); -- word -> 121
sig_p(535 downto 532) <= dataIn(531 downto 528); -- word -> 122
sig_p(531 downto 528) <= dataIn(535 downto 532); -- word -> 123
sig_p(527 downto 524) <= dataIn(527 downto 524); -- word -> 124
sig_p(523 downto 520) <= dataIn(523 downto 520); -- word -> 125
sig_p(519 downto 516) <= dataIn(515 downto 512); -- word -> 126
sig_p(515 downto 512) <= dataIn(519 downto 516); -- word -> 127
sig_p(511 downto 508) <= dataIn(511 downto 508); -- word -> 128
sig_p(507 downto 504) <= dataIn(507 downto 504); -- word -> 129
sig_p(503 downto 500) <= dataIn(499 downto 496); -- word -> 130
sig_p(499 downto 496) <= dataIn(503 downto 500); -- word -> 131
sig_p(495 downto 492) <= dataIn(495 downto 492); -- word -> 132
sig_p(491 downto 488) <= dataIn(491 downto 488); -- word -> 133
sig_p(487 downto 484) <= dataIn(483 downto 480); -- word -> 134
sig_p(483 downto 480) <= dataIn(487 downto 484); -- word -> 135
sig_p(479 downto 476) <= dataIn(479 downto 476); -- word -> 136
sig_p(475 downto 472) <= dataIn(475 downto 472); -- word -> 137
sig_p(471 downto 468) <= dataIn(467 downto 464); -- word -> 138
sig_p(467 downto 464) <= dataIn(471 downto 468); -- word -> 139
sig_p(463 downto 460) <= dataIn(463 downto 460); -- word -> 140
sig_p(459 downto 456) <= dataIn(459 downto 456); -- word -> 141
sig_p(455 downto 452) <= dataIn(451 downto 448); -- word -> 142
sig_p(451 downto 448) <= dataIn(455 downto 452); -- word -> 143
sig_p(447 downto 444) <= dataIn(447 downto 444); -- word -> 144
sig_p(443 downto 440) <= dataIn(443 downto 440); -- word -> 145
sig_p(439 downto 436) <= dataIn(435 downto 432); -- word -> 146
sig_p(435 downto 432) <= dataIn(439 downto 436); -- word -> 147
sig_p(431 downto 428) <= dataIn(431 downto 428); -- word -> 148
sig_p(427 downto 424) <= dataIn(427 downto 424); -- word -> 149
sig_p(423 downto 420) <= dataIn(419 downto 416); -- word -> 150
sig_p(419 downto 416) <= dataIn(423 downto 420); -- word -> 151
sig_p(415 downto 412) <= dataIn(415 downto 412); -- word -> 152
sig_p(411 downto 408) <= dataIn(411 downto 408); -- word -> 153
sig_p(407 downto 404) <= dataIn(403 downto 400); -- word -> 154
sig_p(403 downto 400) <= dataIn(407 downto 404); -- word -> 155
sig_p(399 downto 396) <= dataIn(399 downto 396); -- word -> 156
sig_p(395 downto 392) <= dataIn(395 downto 392); -- word -> 157
sig_p(391 downto 388) <= dataIn(387 downto 384); -- word -> 158
sig_p(387 downto 384) <= dataIn(391 downto 388); -- word -> 159
sig_p(383 downto 380) <= dataIn(383 downto 380); -- word -> 160
sig_p(379 downto 376) <= dataIn(379 downto 376); -- word -> 161
sig_p(375 downto 372) <= dataIn(371 downto 368); -- word -> 162
sig_p(371 downto 368) <= dataIn(375 downto 372); -- word -> 163
```

```
sig_p(367 downto 364) <= dataIn(367 downto 364); -- word -> 164
sig_p(363 downto 360) <= dataIn(363 downto 360); -- word -> 165
sig_p(359 downto 356) <= dataIn(355 downto 352); -- word -> 166
sig_p(355 downto 352) <= dataIn(359 downto 356); -- word -> 167
sig_p(351 downto 348) <= dataIn(351 downto 348); -- word -> 168
sig_p(347 downto 344) <= dataIn(347 downto 344); -- word -> 169
sig_p(343 downto 340) <= dataIn(339 downto 336); -- word -> 170
sig_p(339 downto 336) <= dataIn(343 downto 340); -- word -> 171
sig_p(335 downto 332) <= dataIn(335 downto 332); -- word -> 172
sig_p(331 downto 328) <= dataIn(331 downto 328); -- word -> 173
sig_p(327 downto 324) <= dataIn(323 downto 320); -- word -> 174
sig_p(323 downto 320) <= dataIn(327 downto 324); -- word -> 175
sig_p(319 downto 316) <= dataIn(319 downto 316); -- word -> 176
sig_p(315 downto 312) <= dataIn(315 downto 312); -- word -> 177
sig_p(311 downto 308) <= dataIn(307 downto 304); -- word -> 178
sig_p(307 downto 304) <= dataIn(311 downto 308); -- word -> 179
sig_p(303 downto 300) <= dataIn(303 downto 300); -- word -> 180
sig_p(299 downto 296) <= dataIn(299 downto 296); -- word -> 181
sig_p(295 downto 292) <= dataIn(291 downto 288); -- word -> 182
sig_p(291 downto 288) <= dataIn(295 downto 292); -- word -> 183
sig_p(287 downto 284) <= dataIn(287 downto 284); -- word -> 184
sig_p(283 downto 280) <= dataIn(283 downto 280); -- word -> 185
sig_p(279 downto 276) <= dataIn(275 downto 272); -- word -> 186
sig_p(275 downto 272) <= dataIn(279 downto 276); -- word -> 187
sig_p(271 downto 268) <= dataIn(271 downto 268); -- word -> 188
sig_p(267 downto 264) <= dataIn(267 downto 264); -- word -> 189
sig_p(263 downto 260) <= dataIn(259 downto 256); -- word -> 190
sig_p(259 downto 256) <= dataIn(263 downto 260); -- word -> 191
sig_p(255 downto 252) <= dataIn(255 downto 252); -- word -> 192
sig_p(251 downto 248) <= dataIn(251 downto 248); -- word -> 193
sig_p(247 downto 244) <= dataIn(243 downto 240); -- word -> 194
sig_p(243 downto 240) <= dataIn(247 downto 244); -- word -> 195
sig_p(239 downto 236) <= dataIn(239 downto 236); -- word -> 196
sig_p(235 downto 232) <= dataIn(235 downto 232); -- word -> 197
sig_p(231 downto 228) <= dataIn(227 downto 224); -- word -> 198
sig_p(227 downto 224) <= dataIn(231 downto 228); -- word -> 199
sig_p(223 downto 220) <= dataIn(223 downto 220); -- word -> 200
sig_p(219 downto 216) <= dataIn(219 downto 216); -- word -> 201
sig_p(215 downto 212) <= dataIn(211 downto 208); -- word -> 202
sig_p(211 downto 208) <= dataIn(215 downto 212); -- word -> 203
sig_p(207 downto 204) <= dataIn(207 downto 204); -- word -> 204
sig_p(203 downto 200) <= dataIn(203 downto 200); -- word -> 205
sig_p(199 downto 196) <= dataIn(195 downto 192); -- word -> 206
sig_p(195 downto 192) <= dataIn(199 downto 196); -- word -> 207
sig_p(191 downto 188) <= dataIn(191 downto 188); -- word -> 208
sig_p(187 downto 184) <= dataIn(187 downto 184); -- word -> 209
sig_p(183 downto 180) <= dataIn(179 downto 176); -- word -> 210
sig_p(179 downto 176) <= dataIn(183 downto 180); -- word -> 211
sig_p(175 downto 172) <= dataIn(175 downto 172); -- word -> 212
sig_p(171 downto 168) <= dataIn(171 downto 168); -- word -> 213
sig_p(167 downto 164) <= dataIn(163 downto 160); -- word -> 214
sig_p(163 downto 160) <= dataIn(167 downto 164); -- word -> 215
sig_p(159 downto 156) <= dataIn(159 downto 156); -- word -> 216
sig_p(155 downto 152) <= dataIn(155 downto 152); -- word -> 217
sig_p(151 downto 148) <= dataIn(147 downto 144); -- word -> 218
sig_p(147 downto 144) <= dataIn(151 downto 148); -- word -> 219
sig_p(143 downto 140) <= dataIn(143 downto 140); -- word -> 220
sig_p(139 downto 136) <= dataIn(139 downto 136); -- word -> 221
sig_p(135 downto 132) <= dataIn(131 downto 128); -- word -> 222
sig_p(131 downto 128) <= dataIn(135 downto 132); -- word -> 223
```

```
sig_p(127 downto 124) <= dataIn(127 downto 124); -- word -> 224
sig_p(123 downto 120) <= dataIn(123 downto 120); -- word -> 225
sig_p(119 downto 116) <= dataIn(115 downto 112); -- word -> 226
sig_p(115 downto 112) <= dataIn(119 downto 116); -- word -> 227
sig_p(111 downto 108) <= dataIn(111 downto 108); -- word -> 228
sig_p(107 downto 104) <= dataIn(107 downto 104); -- word -> 229
sig_p(103 downto 100) <= dataIn(99 downto 96); -- word -> 230
sig_p(99 downto 96) <= dataIn(103 downto 100); -- word -> 231
sig_p(95 downto 92) <= dataIn(95 downto 92); -- word -> 232
sig_p(91 downto 88) <= dataIn(91 downto 88); -- word -> 233
sig_p(87 downto 84) <= dataIn(83 downto 80); -- word -> 234
sig_p(83 downto 80) <= dataIn(87 downto 84); -- word -> 235
sig_p(79 downto 76) <= dataIn(79 downto 76); -- word -> 236
sig_p(75 downto 72) <= dataIn(75 downto 72); -- word -> 237
sig_p(71 downto 68) <= dataIn(67 downto 64); -- word -> 238
sig_p(67 downto 64) <= dataIn(71 downto 68); -- word -> 239
sig_p(63 downto 60) <= dataIn(63 downto 60); -- word -> 240
sig_p(59 downto 56) <= dataIn(59 downto 56); -- word -> 241
sig_p(55 downto 52) <= dataIn(51 downto 48); -- word -> 242
sig_p(51 downto 48) <= dataIn(55 downto 52); -- word -> 243
sig_p(47 downto 44) <= dataIn(47 downto 44); -- word -> 244
sig_p(43 downto 40) <= dataIn(43 downto 40); -- word -> 245
sig_p(39 downto 36) <= dataIn(35 downto 32); -- word -> 246
sig_p(35 downto 32) <= dataIn(39 downto 36); -- word -> 247
sig_p(31 downto 28) <= dataIn(31 downto 28); -- word -> 248
sig_p(27 downto 24) <= dataIn(27 downto 24); -- word -> 249
sig_p(23 downto 20) <= dataIn(19 downto 16); -- word -> 250
sig_p(19 downto 16) <= dataIn(23 downto 20); -- word -> 251
sig_p(15 downto 12) <= dataIn(15 downto 12); -- word -> 252
sig_p(11 downto 8) <= dataIn(11 downto 8); -- word -> 253
sig_p(7 downto 4) <= dataIn(3 downto 0); -- word -> 254
sig_p(3 downto 0) <= dataIn(7 downto 4); -- word -> 255

--Ends first For loop p

sigp_2(1023 downto 1020) <= sig_p(1023 downto 1020); -- word -> 0
sigp_2(1019 downto 1016) <= sig_p(1015 downto 1012); -- word -> 1
sigp_2(1015 downto 1012) <= sig_p(1007 downto 1004); -- word -> 2
sigp_2(1011 downto 1008) <= sig_p(999 downto 996); -- word -> 3
sigp_2(1007 downto 1004) <= sig_p(991 downto 988); -- word -> 4
sigp_2(1003 downto 1000) <= sig_p(983 downto 980); -- word -> 5
sigp_2(999 downto 996) <= sig_p(975 downto 972); -- word -> 6
sigp_2(995 downto 992) <= sig_p(967 downto 964); -- word -> 7
sigp_2(991 downto 988) <= sig_p(959 downto 956); -- word -> 8
sigp_2(987 downto 984) <= sig_p(951 downto 948); -- word -> 9
sigp_2(983 downto 980) <= sig_p(943 downto 940); -- word -> 10
sigp_2(979 downto 976) <= sig_p(935 downto 932); -- word -> 11
sigp_2(975 downto 972) <= sig_p(927 downto 924); -- word -> 12
sigp_2(971 downto 968) <= sig_p(919 downto 916); -- word -> 13
sigp_2(967 downto 964) <= sig_p(911 downto 908); -- word -> 14
sigp_2(963 downto 960) <= sig_p(903 downto 900); -- word -> 15
sigp_2(959 downto 956) <= sig_p(895 downto 892); -- word -> 16
sigp_2(955 downto 952) <= sig_p(887 downto 884); -- word -> 17
sigp_2(951 downto 948) <= sig_p(879 downto 876); -- word -> 18
sigp_2(947 downto 944) <= sig_p(871 downto 868); -- word -> 19
sigp_2(943 downto 940) <= sig_p(863 downto 860); -- word -> 20
sigp_2(939 downto 936) <= sig_p(855 downto 852); -- word -> 21
sigp_2(935 downto 932) <= sig_p(847 downto 844); -- word -> 22
```

```
sigp_2(931 downto 928) <= sig_p(839 downto 836); -- word -> 23
sigp_2(927 downto 924) <= sig_p(831 downto 828); -- word -> 24
sigp_2(923 downto 920) <= sig_p(823 downto 820); -- word -> 25
sigp_2(919 downto 916) <= sig_p(815 downto 812); -- word -> 26
sigp_2(915 downto 912) <= sig_p(807 downto 804); -- word -> 27
sigp_2(911 downto 908) <= sig_p(799 downto 796); -- word -> 28
sigp_2(907 downto 904) <= sig_p(791 downto 788); -- word -> 29
sigp_2(903 downto 900) <= sig_p(783 downto 780); -- word -> 30
sigp_2(899 downto 896) <= sig_p(775 downto 772); -- word -> 31
sigp_2(895 downto 892) <= sig_p(767 downto 764); -- word -> 32
sigp_2(891 downto 888) <= sig_p(759 downto 756); -- word -> 33
sigp_2(887 downto 884) <= sig_p(751 downto 748); -- word -> 34
sigp_2(883 downto 880) <= sig_p(743 downto 740); -- word -> 35
sigp_2(879 downto 876) <= sig_p(735 downto 732); -- word -> 36
sigp_2(875 downto 872) <= sig_p(727 downto 724); -- word -> 37
sigp_2(871 downto 868) <= sig_p(719 downto 716); -- word -> 38
sigp_2(867 downto 864) <= sig_p(711 downto 708); -- word -> 39
sigp_2(863 downto 860) <= sig_p(703 downto 700); -- word -> 40
sigp_2(859 downto 856) <= sig_p(695 downto 692); -- word -> 41
sigp_2(855 downto 852) <= sig_p(687 downto 684); -- word -> 42
sigp_2(851 downto 848) <= sig_p(679 downto 676); -- word -> 43
sigp_2(847 downto 844) <= sig_p(671 downto 668); -- word -> 44
sigp_2(843 downto 840) <= sig_p(663 downto 660); -- word -> 45
sigp_2(839 downto 836) <= sig_p(655 downto 652); -- word -> 46
sigp_2(835 downto 832) <= sig_p(647 downto 644); -- word -> 47
sigp_2(831 downto 828) <= sig_p(639 downto 636); -- word -> 48
sigp_2(827 downto 824) <= sig_p(631 downto 628); -- word -> 49
sigp_2(823 downto 820) <= sig_p(623 downto 620); -- word -> 50
sigp_2(819 downto 816) <= sig_p(615 downto 612); -- word -> 51
sigp_2(815 downto 812) <= sig_p(607 downto 604); -- word -> 52
sigp_2(811 downto 808) <= sig_p(599 downto 596); -- word -> 53
sigp_2(807 downto 804) <= sig_p(591 downto 588); -- word -> 54
sigp_2(803 downto 800) <= sig_p(583 downto 580); -- word -> 55
sigp_2(799 downto 796) <= sig_p(575 downto 572); -- word -> 56
sigp_2(795 downto 792) <= sig_p(567 downto 564); -- word -> 57
sigp_2(791 downto 788) <= sig_p(559 downto 556); -- word -> 58
sigp_2(787 downto 784) <= sig_p(551 downto 548); -- word -> 59
sigp_2(783 downto 780) <= sig_p(543 downto 540); -- word -> 60
sigp_2(779 downto 776) <= sig_p(535 downto 532); -- word -> 61
sigp_2(775 downto 772) <= sig_p(527 downto 524); -- word -> 62
sigp_2(771 downto 768) <= sig_p(519 downto 516); -- word -> 63
sigp_2(767 downto 764) <= sig_p(511 downto 508); -- word -> 64
sigp_2(763 downto 760) <= sig_p(503 downto 500); -- word -> 65
sigp_2(759 downto 756) <= sig_p(495 downto 492); -- word -> 66
sigp_2(755 downto 752) <= sig_p(487 downto 484); -- word -> 67
sigp_2(751 downto 748) <= sig_p(479 downto 476); -- word -> 68
sigp_2(747 downto 744) <= sig_p(471 downto 468); -- word -> 69
sigp_2(743 downto 740) <= sig_p(463 downto 460); -- word -> 70
sigp_2(739 downto 736) <= sig_p(455 downto 452); -- word -> 71
sigp_2(735 downto 732) <= sig_p(447 downto 444); -- word -> 72
sigp_2(731 downto 728) <= sig_p(439 downto 436); -- word -> 73
sigp_2(727 downto 724) <= sig_p(431 downto 428); -- word -> 74
sigp_2(723 downto 720) <= sig_p(423 downto 420); -- word -> 75
sigp_2(719 downto 716) <= sig_p(415 downto 412); -- word -> 76
sigp_2(715 downto 712) <= sig_p(407 downto 404); -- word -> 77
sigp_2(711 downto 708) <= sig_p(399 downto 396); -- word -> 78
sigp_2(707 downto 704) <= sig_p(391 downto 388); -- word -> 79
sigp_2(703 downto 700) <= sig_p(383 downto 380); -- word -> 80
sigp_2(699 downto 696) <= sig_p(375 downto 372); -- word -> 81
sigp_2(695 downto 692) <= sig_p(367 downto 364); -- word -> 82
```

```
sigp_2(691 downto 688) <= sig_p(359 downto 356); -- word -> 83
sigp_2(687 downto 684) <= sig_p(351 downto 348); -- word -> 84
sigp_2(683 downto 680) <= sig_p(343 downto 340); -- word -> 85
sigp_2(679 downto 676) <= sig_p(335 downto 332); -- word -> 86
sigp_2(675 downto 672) <= sig_p(327 downto 324); -- word -> 87
sigp_2(671 downto 668) <= sig_p(319 downto 316); -- word -> 88
sigp_2(667 downto 664) <= sig_p(311 downto 308); -- word -> 89
sigp_2(663 downto 660) <= sig_p(303 downto 300); -- word -> 90
sigp_2(659 downto 656) <= sig_p(295 downto 292); -- word -> 91
sigp_2(655 downto 652) <= sig_p(287 downto 284); -- word -> 92
sigp_2(651 downto 648) <= sig_p(279 downto 276); -- word -> 93
sigp_2(647 downto 644) <= sig_p(271 downto 268); -- word -> 94
sigp_2(643 downto 640) <= sig_p(263 downto 260); -- word -> 95
sigp_2(639 downto 636) <= sig_p(255 downto 252); -- word -> 96
sigp_2(635 downto 632) <= sig_p(247 downto 244); -- word -> 97
sigp_2(631 downto 628) <= sig_p(239 downto 236); -- word -> 98
sigp_2(627 downto 624) <= sig_p(231 downto 228); -- word -> 99
sigp_2(623 downto 620) <= sig_p(223 downto 220); -- word -> 100
sigp_2(619 downto 616) <= sig_p(215 downto 212); -- word -> 101
sigp_2(615 downto 612) <= sig_p(207 downto 204); -- word -> 102
sigp_2(611 downto 608) <= sig_p(199 downto 196); -- word -> 103
sigp_2(607 downto 604) <= sig_p(191 downto 188); -- word -> 104
sigp_2(603 downto 600) <= sig_p(183 downto 180); -- word -> 105
sigp_2(599 downto 596) <= sig_p(175 downto 172); -- word -> 106
sigp_2(595 downto 592) <= sig_p(167 downto 164); -- word -> 107
sigp_2(591 downto 588) <= sig_p(159 downto 156); -- word -> 108
sigp_2(587 downto 584) <= sig_p(151 downto 148); -- word -> 109
sigp_2(583 downto 580) <= sig_p(143 downto 140); -- word -> 110
sigp_2(579 downto 576) <= sig_p(135 downto 132); -- word -> 111
sigp_2(575 downto 572) <= sig_p(127 downto 124); -- word -> 112
sigp_2(571 downto 568) <= sig_p(119 downto 116); -- word -> 113
sigp_2(567 downto 564) <= sig_p(111 downto 108); -- word -> 114
sigp_2(563 downto 560) <= sig_p(103 downto 100); -- word -> 115
sigp_2(559 downto 556) <= sig_p(95 downto 92); -- word -> 116
sigp_2(555 downto 552) <= sig_p(87 downto 84); -- word -> 117
sigp_2(551 downto 548) <= sig_p(79 downto 76); -- word -> 118
sigp_2(547 downto 544) <= sig_p(71 downto 68); -- word -> 119
sigp_2(543 downto 540) <= sig_p(63 downto 60); -- word -> 120
sigp_2(539 downto 536) <= sig_p(55 downto 52); -- word -> 121
sigp_2(535 downto 532) <= sig_p(47 downto 44); -- word -> 122
sigp_2(531 downto 528) <= sig_p(39 downto 36); -- word -> 123
sigp_2(527 downto 524) <= sig_p(31 downto 28); -- word -> 124
sigp_2(523 downto 520) <= sig_p(23 downto 20); -- word -> 125
sigp_2(519 downto 516) <= sig_p(15 downto 12); -- word -> 126
sigp_2(515 downto 512) <= sig_p(7 downto 4); -- word -> 127
sigp_2(511 downto 508) <= sig_p(1019 downto 1016); -- word -> 128
sigp_2(507 downto 504) <= sig_p(1011 downto 1008); -- word -> 129
sigp_2(503 downto 500) <= sig_p(1003 downto 1000); -- word -> 130
sigp_2(499 downto 496) <= sig_p(995 downto 992); -- word -> 131
sigp_2(495 downto 492) <= sig_p(987 downto 984); -- word -> 132
sigp_2(491 downto 488) <= sig_p(979 downto 976); -- word -> 133
sigp_2(487 downto 484) <= sig_p(971 downto 968); -- word -> 134
sigp_2(483 downto 480) <= sig_p(963 downto 960); -- word -> 135
sigp_2(479 downto 476) <= sig_p(955 downto 952); -- word -> 136
sigp_2(475 downto 472) <= sig_p(947 downto 944); -- word -> 137
sigp_2(471 downto 468) <= sig_p(939 downto 936); -- word -> 138
sigp_2(467 downto 464) <= sig_p(931 downto 928); -- word -> 139
sigp_2(463 downto 460) <= sig_p(923 downto 920); -- word -> 140
sigp_2(459 downto 456) <= sig_p(915 downto 912); -- word -> 141
sigp_2(455 downto 452) <= sig_p(907 downto 904); -- word -> 142
```

```
sigp_2(451 downto 448) <= sig_p(899 downto 896); -- word -> 143
sigp_2(447 downto 444) <= sig_p(891 downto 888); -- word -> 144
sigp_2(443 downto 440) <= sig_p(883 downto 880); -- word -> 145
sigp_2(439 downto 436) <= sig_p(875 downto 872); -- word -> 146
sigp_2(435 downto 432) <= sig_p(867 downto 864); -- word -> 147
sigp_2(431 downto 428) <= sig_p(859 downto 856); -- word -> 148
sigp_2(427 downto 424) <= sig_p(851 downto 848); -- word -> 149
sigp_2(423 downto 420) <= sig_p(843 downto 840); -- word -> 150
sigp_2(419 downto 416) <= sig_p(835 downto 832); -- word -> 151
sigp_2(415 downto 412) <= sig_p(827 downto 824); -- word -> 152
sigp_2(411 downto 408) <= sig_p(819 downto 816); -- word -> 153
sigp_2(407 downto 404) <= sig_p(811 downto 808); -- word -> 154
sigp_2(403 downto 400) <= sig_p(803 downto 800); -- word -> 155
sigp_2(399 downto 396) <= sig_p(795 downto 792); -- word -> 156
sigp_2(395 downto 392) <= sig_p(787 downto 784); -- word -> 157
sigp_2(391 downto 388) <= sig_p(779 downto 776); -- word -> 158
sigp_2(387 downto 384) <= sig_p(771 downto 768); -- word -> 159
sigp_2(383 downto 380) <= sig_p(763 downto 760); -- word -> 160
sigp_2(379 downto 376) <= sig_p(755 downto 752); -- word -> 161
sigp_2(375 downto 372) <= sig_p(747 downto 744); -- word -> 162
sigp_2(371 downto 368) <= sig_p(739 downto 736); -- word -> 163
sigp_2(367 downto 364) <= sig_p(731 downto 728); -- word -> 164
sigp_2(363 downto 360) <= sig_p(723 downto 720); -- word -> 165
sigp_2(359 downto 356) <= sig_p(715 downto 712); -- word -> 166
sigp_2(355 downto 352) <= sig_p(707 downto 704); -- word -> 167
sigp_2(351 downto 348) <= sig_p(699 downto 696); -- word -> 168
sigp_2(347 downto 344) <= sig_p(691 downto 688); -- word -> 169
sigp_2(343 downto 340) <= sig_p(683 downto 680); -- word -> 170
sigp_2(339 downto 336) <= sig_p(675 downto 672); -- word -> 171
sigp_2(335 downto 332) <= sig_p(667 downto 664); -- word -> 172
sigp_2(331 downto 328) <= sig_p(659 downto 656); -- word -> 173
sigp_2(327 downto 324) <= sig_p(651 downto 648); -- word -> 174
sigp_2(323 downto 320) <= sig_p(643 downto 640); -- word -> 175
sigp_2(319 downto 316) <= sig_p(635 downto 632); -- word -> 176
sigp_2(315 downto 312) <= sig_p(627 downto 624); -- word -> 177
sigp_2(311 downto 308) <= sig_p(619 downto 616); -- word -> 178
sigp_2(307 downto 304) <= sig_p(611 downto 608); -- word -> 179
sigp_2(303 downto 300) <= sig_p(603 downto 600); -- word -> 180
sigp_2(299 downto 296) <= sig_p(595 downto 592); -- word -> 181
sigp_2(295 downto 292) <= sig_p(587 downto 584); -- word -> 182
sigp_2(291 downto 288) <= sig_p(579 downto 576); -- word -> 183
sigp_2(287 downto 284) <= sig_p(571 downto 568); -- word -> 184
sigp_2(283 downto 280) <= sig_p(563 downto 560); -- word -> 185
sigp_2(279 downto 276) <= sig_p(555 downto 552); -- word -> 186
sigp_2(275 downto 272) <= sig_p(547 downto 544); -- word -> 187
sigp_2(271 downto 268) <= sig_p(539 downto 536); -- word -> 188
sigp_2(267 downto 264) <= sig_p(531 downto 528); -- word -> 189
sigp_2(263 downto 260) <= sig_p(523 downto 520); -- word -> 190
sigp_2(259 downto 256) <= sig_p(515 downto 512); -- word -> 191
sigp_2(255 downto 252) <= sig_p(507 downto 504); -- word -> 192
sigp_2(251 downto 248) <= sig_p(499 downto 496); -- word -> 193
sigp_2(247 downto 244) <= sig_p(491 downto 488); -- word -> 194
sigp_2(243 downto 240) <= sig_p(483 downto 480); -- word -> 195
sigp_2(239 downto 236) <= sig_p(475 downto 472); -- word -> 196
sigp_2(235 downto 232) <= sig_p(467 downto 464); -- word -> 197
sigp_2(231 downto 228) <= sig_p(459 downto 456); -- word -> 198
sigp_2(227 downto 224) <= sig_p(451 downto 448); -- word -> 199
sigp_2(223 downto 220) <= sig_p(443 downto 440); -- word -> 200
sigp_2(219 downto 216) <= sig_p(435 downto 432); -- word -> 201
sigp_2(215 downto 212) <= sig_p(427 downto 424); -- word -> 202
```

```
sigp_2(211 downto 208) <= sig_p(419 downto 416); -- word -> 203
sigp_2(207 downto 204) <= sig_p(411 downto 408); -- word -> 204
sigp_2(203 downto 200) <= sig_p(403 downto 400); -- word -> 205
sigp_2(199 downto 196) <= sig_p(395 downto 392); -- word -> 206
sigp_2(195 downto 192) <= sig_p(387 downto 384); -- word -> 207
sigp_2(191 downto 188) <= sig_p(379 downto 376); -- word -> 208
sigp_2(187 downto 184) <= sig_p(371 downto 368); -- word -> 209
sigp_2(183 downto 180) <= sig_p(363 downto 360); -- word -> 210
sigp_2(179 downto 176) <= sig_p(355 downto 352); -- word -> 211
sigp_2(175 downto 172) <= sig_p(347 downto 344); -- word -> 212
sigp_2(171 downto 168) <= sig_p(339 downto 336); -- word -> 213
sigp_2(167 downto 164) <= sig_p(331 downto 328); -- word -> 214
sigp_2(163 downto 160) <= sig_p(323 downto 320); -- word -> 215
sigp_2(159 downto 156) <= sig_p(315 downto 312); -- word -> 216
sigp_2(155 downto 152) <= sig_p(307 downto 304); -- word -> 217
sigp_2(151 downto 148) <= sig_p(299 downto 296); -- word -> 218
sigp_2(147 downto 144) <= sig_p(291 downto 288); -- word -> 219
sigp_2(143 downto 140) <= sig_p(283 downto 280); -- word -> 220
sigp_2(139 downto 136) <= sig_p(275 downto 272); -- word -> 221
sigp_2(135 downto 132) <= sig_p(267 downto 264); -- word -> 222
sigp_2(131 downto 128) <= sig_p(259 downto 256); -- word -> 223
sigp_2(127 downto 124) <= sig_p(251 downto 248); -- word -> 224
sigp_2(123 downto 120) <= sig_p(243 downto 240); -- word -> 225
sigp_2(119 downto 116) <= sig_p(235 downto 232); -- word -> 226
sigp_2(115 downto 112) <= sig_p(227 downto 224); -- word -> 227
sigp_2(111 downto 108) <= sig_p(219 downto 216); -- word -> 228
sigp_2(107 downto 104) <= sig_p(211 downto 208); -- word -> 229
sigp_2(103 downto 100) <= sig_p(203 downto 200); -- word -> 230
sigp_2(99 downto 96) <= sig_p(195 downto 192); -- word -> 231
sigp_2(95 downto 92) <= sig_p(187 downto 184); -- word -> 232
sigp_2(91 downto 88) <= sig_p(179 downto 176); -- word -> 233
sigp_2(87 downto 84) <= sig_p(171 downto 168); -- word -> 234
sigp_2(83 downto 80) <= sig_p(163 downto 160); -- word -> 235
sigp_2(79 downto 76) <= sig_p(155 downto 152); -- word -> 236
sigp_2(75 downto 72) <= sig_p(147 downto 144); -- word -> 237
sigp_2(71 downto 68) <= sig_p(139 downto 136); -- word -> 238
sigp_2(67 downto 64) <= sig_p(131 downto 128); -- word -> 239
sigp_2(63 downto 60) <= sig_p(123 downto 120); -- word -> 240
sigp_2(59 downto 56) <= sig_p(115 downto 112); -- word -> 241
sigp_2(55 downto 52) <= sig_p(107 downto 104); -- word -> 242
sigp_2(51 downto 48) <= sig_p(99 downto 96); -- word -> 243
sigp_2(47 downto 44) <= sig_p(91 downto 88); -- word -> 244
sigp_2(43 downto 40) <= sig_p(83 downto 80); -- word -> 245
sigp_2(39 downto 36) <= sig_p(75 downto 72); -- word -> 246
sigp_2(35 downto 32) <= sig_p(67 downto 64); -- word -> 247
sigp_2(31 downto 28) <= sig_p(59 downto 56); -- word -> 248
sigp_2(27 downto 24) <= sig_p(51 downto 48); -- word -> 249
sigp_2(23 downto 20) <= sig_p(43 downto 40); -- word -> 250
sigp_2(19 downto 16) <= sig_p(35 downto 32); -- word -> 251
sigp_2(15 downto 12) <= sig_p(27 downto 24); -- word -> 252
sigp_2(11 downto 8) <= sig_p(19 downto 16); -- word -> 253
sigp_2(7 downto 4) <= sig_p(11 downto 8); -- word -> 254
sigp_2(3 downto 0) <= sig_p(3 downto 0); -- word -> 255
```

--Ends second For loop p

```
dataOut(1023 downto 1020) <= sigp_2(1023 downto 1020); -- word -> 0
dataOut(1019 downto 1016) <= sigp_2(1019 downto 1016); -- word -> 1
```

```
dataOut(1015 downto 1012) <= sigp_2(1015 downto 1012); -- word -> 2
dataOut(1011 downto 1008) <= sigp_2(1011 downto 1008); -- word -> 3
dataOut(1007 downto 1004) <= sigp_2(1007 downto 1004); -- word -> 4
dataOut(1003 downto 1000) <= sigp_2(1003 downto 1000); -- word -> 5
dataOut(999 downto 996) <= sigp_2(999 downto 996); -- word -> 6
dataOut(995 downto 992) <= sigp_2(995 downto 992); -- word -> 7
dataOut(991 downto 988) <= sigp_2(991 downto 988); -- word -> 8
dataOut(987 downto 984) <= sigp_2(987 downto 984); -- word -> 9
dataOut(983 downto 980) <= sigp_2(983 downto 980); -- word -> 10
dataOut(979 downto 976) <= sigp_2(979 downto 976); -- word -> 11
dataOut(975 downto 972) <= sigp_2(975 downto 972); -- word -> 12
dataOut(971 downto 968) <= sigp_2(971 downto 968); -- word -> 13
dataOut(967 downto 964) <= sigp_2(967 downto 964); -- word -> 14
dataOut(963 downto 960) <= sigp_2(963 downto 960); -- word -> 15
dataOut(959 downto 956) <= sigp_2(959 downto 956); -- word -> 16
dataOut(955 downto 952) <= sigp_2(955 downto 952); -- word -> 17
dataOut(951 downto 948) <= sigp_2(951 downto 948); -- word -> 18
dataOut(947 downto 944) <= sigp_2(947 downto 944); -- word -> 19
dataOut(943 downto 940) <= sigp_2(943 downto 940); -- word -> 20
dataOut(939 downto 936) <= sigp_2(939 downto 936); -- word -> 21
dataOut(935 downto 932) <= sigp_2(935 downto 932); -- word -> 22
dataOut(931 downto 928) <= sigp_2(931 downto 928); -- word -> 23
dataOut(927 downto 924) <= sigp_2(927 downto 924); -- word -> 24
dataOut(923 downto 920) <= sigp_2(923 downto 920); -- word -> 25
dataOut(919 downto 916) <= sigp_2(919 downto 916); -- word -> 26
dataOut(915 downto 912) <= sigp_2(915 downto 912); -- word -> 27
dataOut(911 downto 908) <= sigp_2(911 downto 908); -- word -> 28
dataOut(907 downto 904) <= sigp_2(907 downto 904); -- word -> 29
dataOut(903 downto 900) <= sigp_2(903 downto 900); -- word -> 30
dataOut(899 downto 896) <= sigp_2(899 downto 896); -- word -> 31
dataOut(895 downto 892) <= sigp_2(895 downto 892); -- word -> 32
dataOut(891 downto 888) <= sigp_2(891 downto 888); -- word -> 33
dataOut(887 downto 884) <= sigp_2(887 downto 884); -- word -> 34
dataOut(883 downto 880) <= sigp_2(883 downto 880); -- word -> 35
dataOut(879 downto 876) <= sigp_2(879 downto 876); -- word -> 36
dataOut(875 downto 872) <= sigp_2(875 downto 872); -- word -> 37
dataOut(871 downto 868) <= sigp_2(871 downto 868); -- word -> 38
dataOut(867 downto 864) <= sigp_2(867 downto 864); -- word -> 39
dataOut(863 downto 860) <= sigp_2(863 downto 860); -- word -> 40
dataOut(859 downto 856) <= sigp_2(859 downto 856); -- word -> 41
dataOut(855 downto 852) <= sigp_2(855 downto 852); -- word -> 42
dataOut(851 downto 848) <= sigp_2(851 downto 848); -- word -> 43
dataOut(847 downto 844) <= sigp_2(847 downto 844); -- word -> 44
dataOut(843 downto 840) <= sigp_2(843 downto 840); -- word -> 45
dataOut(839 downto 836) <= sigp_2(839 downto 836); -- word -> 46
dataOut(835 downto 832) <= sigp_2(835 downto 832); -- word -> 47
dataOut(831 downto 828) <= sigp_2(831 downto 828); -- word -> 48
dataOut(827 downto 824) <= sigp_2(827 downto 824); -- word -> 49
dataOut(823 downto 820) <= sigp_2(823 downto 820); -- word -> 50
dataOut(819 downto 816) <= sigp_2(819 downto 816); -- word -> 51
dataOut(815 downto 812) <= sigp_2(815 downto 812); -- word -> 52
dataOut(811 downto 808) <= sigp_2(811 downto 808); -- word -> 53
dataOut(807 downto 804) <= sigp_2(807 downto 804); -- word -> 54
dataOut(803 downto 800) <= sigp_2(803 downto 800); -- word -> 55
dataOut(799 downto 796) <= sigp_2(799 downto 796); -- word -> 56
dataOut(795 downto 792) <= sigp_2(795 downto 792); -- word -> 57
dataOut(791 downto 788) <= sigp_2(791 downto 788); -- word -> 58
dataOut(787 downto 784) <= sigp_2(787 downto 784); -- word -> 59
dataOut(783 downto 780) <= sigp_2(783 downto 780); -- word -> 60
dataOut(779 downto 776) <= sigp_2(779 downto 776); -- word -> 61
```

```
dataOut(775 downto 772) <= sigp_2(775 downto 772); -- word -> 62
dataOut(771 downto 768) <= sigp_2(771 downto 768); -- word -> 63
dataOut(767 downto 764) <= sigp_2(767 downto 764); -- word -> 64
dataOut(763 downto 760) <= sigp_2(763 downto 760); -- word -> 65
dataOut(759 downto 756) <= sigp_2(759 downto 756); -- word -> 66
dataOut(755 downto 752) <= sigp_2(755 downto 752); -- word -> 67
dataOut(751 downto 748) <= sigp_2(751 downto 748); -- word -> 68
dataOut(747 downto 744) <= sigp_2(747 downto 744); -- word -> 69
dataOut(743 downto 740) <= sigp_2(743 downto 740); -- word -> 70
dataOut(739 downto 736) <= sigp_2(739 downto 736); -- word -> 71
dataOut(735 downto 732) <= sigp_2(735 downto 732); -- word -> 72
dataOut(731 downto 728) <= sigp_2(731 downto 728); -- word -> 73
dataOut(727 downto 724) <= sigp_2(727 downto 724); -- word -> 74
dataOut(723 downto 720) <= sigp_2(723 downto 720); -- word -> 75
dataOut(719 downto 716) <= sigp_2(719 downto 716); -- word -> 76
dataOut(715 downto 712) <= sigp_2(715 downto 712); -- word -> 77
dataOut(711 downto 708) <= sigp_2(711 downto 708); -- word -> 78
dataOut(707 downto 704) <= sigp_2(707 downto 704); -- word -> 79
dataOut(703 downto 700) <= sigp_2(703 downto 700); -- word -> 80
dataOut(699 downto 696) <= sigp_2(699 downto 696); -- word -> 81
dataOut(695 downto 692) <= sigp_2(695 downto 692); -- word -> 82
dataOut(691 downto 688) <= sigp_2(691 downto 688); -- word -> 83
dataOut(687 downto 684) <= sigp_2(687 downto 684); -- word -> 84
dataOut(683 downto 680) <= sigp_2(683 downto 680); -- word -> 85
dataOut(679 downto 676) <= sigp_2(679 downto 676); -- word -> 86
dataOut(675 downto 672) <= sigp_2(675 downto 672); -- word -> 87
dataOut(671 downto 668) <= sigp_2(671 downto 668); -- word -> 88
dataOut(667 downto 664) <= sigp_2(667 downto 664); -- word -> 89
dataOut(663 downto 660) <= sigp_2(663 downto 660); -- word -> 90
dataOut(659 downto 656) <= sigp_2(659 downto 656); -- word -> 91
dataOut(655 downto 652) <= sigp_2(655 downto 652); -- word -> 92
dataOut(651 downto 648) <= sigp_2(651 downto 648); -- word -> 93
dataOut(647 downto 644) <= sigp_2(647 downto 644); -- word -> 94
dataOut(643 downto 640) <= sigp_2(643 downto 640); -- word -> 95
dataOut(639 downto 636) <= sigp_2(639 downto 636); -- word -> 96
dataOut(635 downto 632) <= sigp_2(635 downto 632); -- word -> 97
dataOut(631 downto 628) <= sigp_2(631 downto 628); -- word -> 98
dataOut(627 downto 624) <= sigp_2(627 downto 624); -- word -> 99
dataOut(623 downto 620) <= sigp_2(623 downto 620); -- word -> 100
dataOut(619 downto 616) <= sigp_2(619 downto 616); -- word -> 101
dataOut(615 downto 612) <= sigp_2(615 downto 612); -- word -> 102
dataOut(611 downto 608) <= sigp_2(611 downto 608); -- word -> 103
dataOut(607 downto 604) <= sigp_2(607 downto 604); -- word -> 104
dataOut(603 downto 600) <= sigp_2(603 downto 600); -- word -> 105
dataOut(599 downto 596) <= sigp_2(599 downto 596); -- word -> 106
dataOut(595 downto 592) <= sigp_2(595 downto 592); -- word -> 107
dataOut(591 downto 588) <= sigp_2(591 downto 588); -- word -> 108
dataOut(587 downto 584) <= sigp_2(587 downto 584); -- word -> 109
dataOut(583 downto 580) <= sigp_2(583 downto 580); -- word -> 110
dataOut(579 downto 576) <= sigp_2(579 downto 576); -- word -> 111
dataOut(575 downto 572) <= sigp_2(575 downto 572); -- word -> 112
dataOut(571 downto 568) <= sigp_2(571 downto 568); -- word -> 113
dataOut(567 downto 564) <= sigp_2(567 downto 564); -- word -> 114
dataOut(563 downto 560) <= sigp_2(563 downto 560); -- word -> 115
dataOut(559 downto 556) <= sigp_2(559 downto 556); -- word -> 116
dataOut(555 downto 552) <= sigp_2(555 downto 552); -- word -> 117
dataOut(551 downto 548) <= sigp_2(551 downto 548); -- word -> 118
dataOut(547 downto 544) <= sigp_2(547 downto 544); -- word -> 119
dataOut(543 downto 540) <= sigp_2(543 downto 540); -- word -> 120
dataOut(539 downto 536) <= sigp_2(539 downto 536); -- word -> 121
```

```
dataOut(535 downto 532) <= sigp_2(535 downto 532); -- word -> 122
dataOut(531 downto 528) <= sigp_2(531 downto 528); -- word -> 123
dataOut(527 downto 524) <= sigp_2(527 downto 524); -- word -> 124
dataOut(523 downto 520) <= sigp_2(523 downto 520); -- word -> 125
dataOut(519 downto 516) <= sigp_2(519 downto 516); -- word -> 126
dataOut(515 downto 512) <= sigp_2(515 downto 512); -- word -> 127
dataOut(511 downto 508) <= sigp_2(507 downto 504); -- word -> 128
dataOut(507 downto 504) <= sigp_2(511 downto 508); -- word -> 129
dataOut(503 downto 500) <= sigp_2(499 downto 496); -- word -> 130
dataOut(499 downto 496) <= sigp_2(503 downto 500); -- word -> 131
dataOut(495 downto 492) <= sigp_2(491 downto 488); -- word -> 132
dataOut(491 downto 488) <= sigp_2(495 downto 492); -- word -> 133
dataOut(487 downto 484) <= sigp_2(483 downto 480); -- word -> 134
dataOut(483 downto 480) <= sigp_2(487 downto 484); -- word -> 135
dataOut(479 downto 476) <= sigp_2(475 downto 472); -- word -> 136
dataOut(475 downto 472) <= sigp_2(479 downto 476); -- word -> 137
dataOut(471 downto 468) <= sigp_2(467 downto 464); -- word -> 138
dataOut(467 downto 464) <= sigp_2(471 downto 468); -- word -> 139
dataOut(463 downto 460) <= sigp_2(459 downto 456); -- word -> 140
dataOut(459 downto 456) <= sigp_2(463 downto 460); -- word -> 141
dataOut(455 downto 452) <= sigp_2(451 downto 448); -- word -> 142
dataOut(451 downto 448) <= sigp_2(455 downto 452); -- word -> 143
dataOut(447 downto 444) <= sigp_2(443 downto 440); -- word -> 144
dataOut(443 downto 440) <= sigp_2(447 downto 444); -- word -> 145
dataOut(439 downto 436) <= sigp_2(435 downto 432); -- word -> 146
dataOut(435 downto 432) <= sigp_2(439 downto 436); -- word -> 147
dataOut(431 downto 428) <= sigp_2(427 downto 424); -- word -> 148
dataOut(427 downto 424) <= sigp_2(431 downto 428); -- word -> 149
dataOut(423 downto 420) <= sigp_2(419 downto 416); -- word -> 150
dataOut(419 downto 416) <= sigp_2(423 downto 420); -- word -> 151
dataOut(415 downto 412) <= sigp_2(411 downto 408); -- word -> 152
dataOut(411 downto 408) <= sigp_2(415 downto 412); -- word -> 153
dataOut(407 downto 404) <= sigp_2(403 downto 400); -- word -> 154
dataOut(403 downto 400) <= sigp_2(407 downto 404); -- word -> 155
dataOut(399 downto 396) <= sigp_2(395 downto 392); -- word -> 156
dataOut(395 downto 392) <= sigp_2(399 downto 396); -- word -> 157
dataOut(391 downto 388) <= sigp_2(387 downto 384); -- word -> 158
dataOut(387 downto 384) <= sigp_2(391 downto 388); -- word -> 159
dataOut(383 downto 380) <= sigp_2(379 downto 376); -- word -> 160
dataOut(379 downto 376) <= sigp_2(383 downto 380); -- word -> 161
dataOut(375 downto 372) <= sigp_2(371 downto 368); -- word -> 162
dataOut(371 downto 368) <= sigp_2(375 downto 372); -- word -> 163
dataOut(367 downto 364) <= sigp_2(363 downto 360); -- word -> 164
dataOut(363 downto 360) <= sigp_2(367 downto 364); -- word -> 165
dataOut(359 downto 356) <= sigp_2(355 downto 352); -- word -> 166
dataOut(355 downto 352) <= sigp_2(359 downto 356); -- word -> 167
dataOut(351 downto 348) <= sigp_2(347 downto 344); -- word -> 168
dataOut(347 downto 344) <= sigp_2(351 downto 348); -- word -> 169
dataOut(343 downto 340) <= sigp_2(339 downto 336); -- word -> 170
dataOut(339 downto 336) <= sigp_2(343 downto 340); -- word -> 171
dataOut(335 downto 332) <= sigp_2(331 downto 328); -- word -> 172
dataOut(331 downto 328) <= sigp_2(335 downto 332); -- word -> 173
dataOut(327 downto 324) <= sigp_2(323 downto 320); -- word -> 174
dataOut(323 downto 320) <= sigp_2(327 downto 324); -- word -> 175
dataOut(319 downto 316) <= sigp_2(315 downto 312); -- word -> 176
dataOut(315 downto 312) <= sigp_2(319 downto 316); -- word -> 177
dataOut(311 downto 308) <= sigp_2(307 downto 304); -- word -> 178
dataOut(307 downto 304) <= sigp_2(311 downto 308); -- word -> 179
dataOut(303 downto 300) <= sigp_2(299 downto 296); -- word -> 180
dataOut(299 downto 296) <= sigp_2(303 downto 300); -- word -> 181
```

```
dataOut(295 downto 292) <= sigp_2(291 downto 288); -- word -> 182
dataOut(291 downto 288) <= sigp_2(295 downto 292); -- word -> 183
dataOut(287 downto 284) <= sigp_2(283 downto 280); -- word -> 184
dataOut(283 downto 280) <= sigp_2(287 downto 284); -- word -> 185
dataOut(279 downto 276) <= sigp_2(275 downto 272); -- word -> 186
dataOut(275 downto 272) <= sigp_2(279 downto 276); -- word -> 187
dataOut(271 downto 268) <= sigp_2(267 downto 264); -- word -> 188
dataOut(267 downto 264) <= sigp_2(271 downto 268); -- word -> 189
dataOut(263 downto 260) <= sigp_2(259 downto 256); -- word -> 190
dataOut(259 downto 256) <= sigp_2(263 downto 260); -- word -> 191
dataOut(255 downto 252) <= sigp_2(251 downto 248); -- word -> 192
dataOut(251 downto 248) <= sigp_2(255 downto 252); -- word -> 193
dataOut(247 downto 244) <= sigp_2(243 downto 240); -- word -> 194
dataOut(243 downto 240) <= sigp_2(247 downto 244); -- word -> 195
dataOut(239 downto 236) <= sigp_2(235 downto 232); -- word -> 196
dataOut(235 downto 232) <= sigp_2(239 downto 236); -- word -> 197
dataOut(231 downto 228) <= sigp_2(227 downto 224); -- word -> 198
dataOut(227 downto 224) <= sigp_2(231 downto 228); -- word -> 199
dataOut(223 downto 220) <= sigp_2(219 downto 216); -- word -> 200
dataOut(219 downto 216) <= sigp_2(223 downto 220); -- word -> 201
dataOut(215 downto 212) <= sigp_2(211 downto 208); -- word -> 202
dataOut(211 downto 208) <= sigp_2(215 downto 212); -- word -> 203
dataOut(207 downto 204) <= sigp_2(203 downto 200); -- word -> 204
dataOut(203 downto 200) <= sigp_2(207 downto 204); -- word -> 205
dataOut(199 downto 196) <= sigp_2(195 downto 192); -- word -> 206
dataOut(195 downto 192) <= sigp_2(199 downto 196); -- word -> 207
dataOut(191 downto 188) <= sigp_2(187 downto 184); -- word -> 208
dataOut(187 downto 184) <= sigp_2(191 downto 188); -- word -> 209
dataOut(183 downto 180) <= sigp_2(179 downto 176); -- word -> 210
dataOut(179 downto 176) <= sigp_2(183 downto 180); -- word -> 211
dataOut(175 downto 172) <= sigp_2(171 downto 168); -- word -> 212
dataOut(171 downto 168) <= sigp_2(175 downto 172); -- word -> 213
dataOut(167 downto 164) <= sigp_2(163 downto 160); -- word -> 214
dataOut(163 downto 160) <= sigp_2(167 downto 164); -- word -> 215
dataOut(159 downto 156) <= sigp_2(155 downto 152); -- word -> 216
dataOut(155 downto 152) <= sigp_2(159 downto 156); -- word -> 217
dataOut(151 downto 148) <= sigp_2(147 downto 144); -- word -> 218
dataOut(147 downto 144) <= sigp_2(151 downto 148); -- word -> 219
dataOut(143 downto 140) <= sigp_2(139 downto 136); -- word -> 220
dataOut(139 downto 136) <= sigp_2(143 downto 140); -- word -> 221
dataOut(135 downto 132) <= sigp_2(131 downto 128); -- word -> 222
dataOut(131 downto 128) <= sigp_2(135 downto 132); -- word -> 223
dataOut(127 downto 124) <= sigp_2(123 downto 120); -- word -> 224
dataOut(123 downto 120) <= sigp_2(127 downto 124); -- word -> 225
dataOut(119 downto 116) <= sigp_2(115 downto 112); -- word -> 226
dataOut(115 downto 112) <= sigp_2(119 downto 116); -- word -> 227
dataOut(111 downto 108) <= sigp_2(107 downto 104); -- word -> 228
dataOut(107 downto 104) <= sigp_2(111 downto 108); -- word -> 229
dataOut(103 downto 100) <= sigp_2(99 downto 96); -- word -> 230
dataOut(99 downto 96) <= sigp_2(103 downto 100); -- word -> 231
dataOut(95 downto 92) <= sigp_2(91 downto 88); -- word -> 232
dataOut(91 downto 88) <= sigp_2(95 downto 92); -- word -> 233
dataOut(87 downto 84) <= sigp_2(83 downto 80); -- word -> 234
dataOut(83 downto 80) <= sigp_2(87 downto 84); -- word -> 235
dataOut(79 downto 76) <= sigp_2(75 downto 72); -- word -> 236
dataOut(75 downto 72) <= sigp_2(79 downto 76); -- word -> 237
dataOut(71 downto 68) <= sigp_2(67 downto 64); -- word -> 238
dataOut(67 downto 64) <= sigp_2(71 downto 68); -- word -> 239
dataOut(63 downto 60) <= sigp_2(59 downto 56); -- word -> 240
dataOut(59 downto 56) <= sigp_2(63 downto 60); -- word -> 241
```

```
        dataOut(55 downto 52) <= sigp_2(51 downto 48); -- word -> 242
        dataOut(51 downto 48) <= sigp_2(55 downto 52); -- word -> 243
        dataOut(47 downto 44) <= sigp_2(43 downto 40); -- word -> 244
        dataOut(43 downto 40) <= sigp_2(47 downto 44); -- word -> 245
        dataOut(39 downto 36) <= sigp_2(35 downto 32); -- word -> 246
        dataOut(35 downto 32) <= sigp_2(39 downto 36); -- word -> 247
        dataOut(31 downto 28) <= sigp_2(27 downto 24); -- word -> 248
        dataOut(27 downto 24) <= sigp_2(31 downto 28); -- word -> 249
        dataOut(23 downto 20) <= sigp_2(19 downto 16); -- word -> 250
        dataOut(19 downto 16) <= sigp_2(23 downto 20); -- word -> 251
        dataOut(15 downto 12) <= sigp_2(11 downto 8); -- word -> 252
        dataOut(11 downto 8) <= sigp_2(15 downto 12); -- word -> 253
        dataOut(7 downto 4) <= sigp_2(3 downto 0); -- word -> 254
        dataOut(3 downto 0) <= sigp_2(7 downto 4); -- word -> 255
end Behavioral;
```

Κώδικας υλοποίησης κυκλώματος Truncating H.

```
entity truncating_H is
port(hashLength : in std_logic_vector(1 downto 0);
     dataIn : in std_logic_vector(511 downto 0);
     dataOut : out std_logic_vector(511 downto 0));
end truncating_H;
architecture Behavioral of truncating_H is
begin

trctng_H_pr : process(hashLength,dataIn)
        begin
        case hashLength is
        when "00" => dataOut <= X"00000000000000000000000000000000000000000000000000000000
                                  000000000000000" & dataIn(223 downto 0);
        when "01" => dataOut <= X"00000000000000000000000000000000000000000000000000000000
                                  0000000" & dataIn(255 downto 0);
        when "10" => dataOut <= X"00000000000000000000000000000000" & dataIn(383 downto 0);
        when "11" => dataOut <= dataIn;
        when others => dataOut <= (others => '1');
        end case;
         end process;
end Behavioral;
```

Κώδικας υλοποίησης κυκλώματος Counter (Σχήμα 6.32).

```
entity counterComp is
        port(clk,reset,enable : in std_logic;
             computingCrcls : in std_logic_vector(6 downto 0);
             counterOut : out integer range 0 to 74);
end counterComp;
architecture Behavioral of counterComp is
SIGNAL counter : integer range 0 to 74;
begin
counterOut <= counter;
counterProcess : process(clk,reset,enable,computingCrcls,counter)
        begin
        if(reset = '1')then
                counter <= 0;
        else
                if(enable = '1' and counter < computingCrcls)then
```

```
                        if(clk' event and clk = '1')then
                                counter <= counter + 1;
                        end if;
                end if;
        end if;
        end process;
end Behavioral;
```

Κώδικας υλοποίησης κυκλώματος Controller (Σχήμα 6.33).

```
entity Controller is
        port( clk,reset,enable : in std_logic;
                        computingCrcls : in std_logic_vector(6 downto 0);
                        resetSboxes,hashReady : out std_logic;
                        selectionMUX2x1 : out std_logic;
                        selectionMUX4x1 : out std_logic_vector(1 downto 0);
                        constantRound : out integer range 0 to 36
                );
end Controller;
architecture Behavioral of Controller is
        Component counterComp
        port( clk,reset,enable : in std_logic;
                computingCrcls : in std_logic_vector(6 downto 0);
                        counterOut : out integer range 0 to 74
          );
        end component;
SIGNAL counter : integer range 0 to 74;
begin
counter_Comp : counterComp port map(clk,reset,enable,computingCrcls,counter);

resetSboxes <= '1' when (counter = 0) else '0';
hashReady <= '1' when (counter = computingCrcls) else '0';

Controller : process(reset,counter)
        begin
                if(reset = '0')then
                        case counter is
                                when 0 => selectionMUX2x1 <= '0';
                                        selectionMUX4x1 <= "00";
                                        constantRound <= 0;

                                when 1 => selectionMUX2x1 <= '0';
                                        selectionMUX4x1 <= "01";
                                        constantRound <= 0;
----------STARTS FIRST MWORD 512-bit----------

                                when 2 => selectionMUX2x1 <= '0';
                                        selectionMUX4x1 <= "10";
                                        constantRound <= 1;

                                when 3 => selectionMUX2x1 <= '0';
                                        selectionMUX4x1 <= "10";
                                        constantRound <= 2;

                                when 4 => selectionMUX2x1 <= '0';
                                        selectionMUX4x1 <= "10";
                                        constantRound <= 3;
```

```
when 5 => selectionMUX2x1 <= '0';
          selectionMUX4x1 <= "10";
          constantRound <= 4;

when 6 => selectionMUX2x1 <= '0';
          selectionMUX4x1 <= "10";
          constantRound <= 5;

when 7 => selectionMUX2x1 <= '0';
          selectionMUX4x1 <= "10";
          constantRound <= 6;

when 8 => selectionMUX2x1 <= '0';
          selectionMUX4x1 <= "10";
          constantRound <= 7;

when 9 => selectionMUX2x1 <= '0';
          selectionMUX4x1 <= "10";
          constantRound <= 8;

when 10 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 9;

when 11 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 10;

when 12 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 11;

when 13 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 12;

when 14 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 13;

when 15 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 14;

when 16 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 15;

when 17 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 16;

when 18 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 17;

when 19 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 18;
```

```
when 20 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 19;

when 21 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 20;

when 22 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 21;

when 23 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 22;

when 24 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 23;

when 25 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 24;

when 26 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 25;

when 27 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 26;

when 28 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 27;

when 29 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 28;

when 30 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 29;

when 31 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 30;

when 32 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 31;

when 33 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 32;

when 34 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 33;
```

```
when 35 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 34;

when 36 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "10";
           constantRound <= 35;

when 37 => selectionMUX2x1 <= '0';
           selectionMUX4x1 <= "11";
           constantRound <= 36;

----------STARTS SECOND MWORD 512-bit----------
when 38 => selectionMUX2x1 <= '1';
           selectionMUX4x1 <= "01";
           constantRound <= 0;

when 39 => selectionMUX2x1 <= '1';
           selectionMUX4x1 <= "10";
           constantRound <= 1;

when 40 => selectionMUX2x1 <= '1';
           selectionMUX4x1 <= "10";
           constantRound <= 2;

when 41 => selectionMUX2x1 <= '1';
           selectionMUX4x1 <= "10";
           constantRound <= 3;

when 42 => selectionMUX2x1 <= '1';
           selectionMUX4x1 <= "10";
           constantRound <= 4;

when 43 => selectionMUX2x1 <= '1';
           selectionMUX4x1 <= "10";
           constantRound <= 5;

when 44 => selectionMUX2x1 <= '1';
           selectionMUX4x1 <= "10";
           constantRound <= 6;

when 45 => selectionMUX2x1 <= '1';
           selectionMUX4x1 <= "10";
           constantRound <= 7;

when 46 => selectionMUX2x1 <= '1';
           selectionMUX4x1 <= "10";
           constantRound <= 8;

when 47 => selectionMUX2x1 <= '1';
           selectionMUX4x1 <= "10";
           constantRound <= 9;

when 48 => selectionMUX2x1 <= '1';
           selectionMUX4x1 <= "10";
           constantRound <= 10;

when 49 => selectionMUX2x1 <= '1';
           selectionMUX4x1 <= "10";
```

```
                constantRound <= 11;

        when 50 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 12;

        when 51 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 13;

        when 52 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 14;

        when 53 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 15;

        when 54 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 16;

        when 55 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 17;

        when 56 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 18;

        when 57 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 19;

        when 58 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 20;

        when 59 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 21;

        when 60 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 22;

        when 61 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 23;

        when 62 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 24;

        when 63 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
                constantRound <= 25;

        when 64 => selectionMUX2x1 <= '1';
                selectionMUX4x1 <= "10";
```

```
                    constantRound <= 26;

                when 65 => selectionMUX2x1 <= '1';
                    selectionMUX4x1 <= "10";
                    constantRound <= 27;

                when 66 => selectionMUX2x1 <= '1';
                    selectionMUX4x1 <= "10";
                    constantRound <= 28;

                when 67 => selectionMUX2x1 <= '1';
                    selectionMUX4x1 <= "10";
                    constantRound <= 29;

                when 68 => selectionMUX2x1 <= '1';
                    selectionMUX4x1 <= "10";
                    constantRound <= 30;

                when 69 => selectionMUX2x1 <= '1';
                    selectionMUX4x1 <= "10";
                    constantRound <= 31;

                when 70 => selectionMUX2x1 <= '1';
                    selectionMUX4x1 <= "10";
                    constantRound <= 32;

                when 71 => selectionMUX2x1 <= '1';
                    selectionMUX4x1 <= "10";
                    constantRound <= 33;

                when 72 => selectionMUX2x1 <= '1';
                    selectionMUX4x1 <= "10";
                    constantRound <= 34;

                when 73 => selectionMUX2x1 <= '1';
                    selectionMUX4x1 <= "10";
                    constantRound <= 35;

                when 74 => selectionMUX2x1 <= '1';
                    selectionMUX4x1 <= "11";
                    constantRound <= 36;
----------------------------------ENDS SECOND MWORD 512-bit----------------
                when others => selectionMUX2x1 <= '0';  selectionMUX4x1 <= "00";
constantRound <= 0;
            end case;
        else
            selectionMUX2x1 <= '0';  selectionMUX4x1 <= "00"; constantRound <= 0;
        end if;
    end process;
end Behavioral;
```