ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Υλοποίηση Αποκωδικοποιητή Reed-Solomon σε Υλικό

Κοσμάς Καριώτης Α.Μ. 2115069

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων	**Συνεπιβλέπων**
Σταμούλης Γεώργιος	**Άχμεντ Μάχντι**
Καθηγητής	**Πανεπιστημιακός Υπότροφος**

Λαμία, 2019

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [1], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. *Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. *Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. *Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια*

4. *Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: ……/…../20……

Ο – Η Δηλ.

(Υπογραφή)

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

Σύνοψη

Οι κώδικες ανίχνευσης και διόρθωσης λαθών εφαρμόζονται όλο και πιο συχνά στα σύγχρονα τηλεπικοινωνιακά συστήματα για την εξασφάλιση της ορθής λειτουργίας τους. Για τους κώδικες Reed-Solomon, οι οποίοι ανήκουν σ' αυτήν την κατηγορία, πραγματοποιήθηκαν αμέτρητες μελέτες που οδήγησαν σε διαφόρους αλγορίθμους, βελτιστοποιήσεις τους και υλοποιήσεις τους στο υλικό και το λογισμικό. Στην πτυχιακή εργασία αυτή παρουσιάζεται αναλυτικά η διαδικασία κατασκευής ενός τέτοιου αποκωδικοποιητή Reed-Solomon, ο οποίος βασίζεται σε αποδοτικούς αλγορίθμους της βιβλιογραφίας.

Μετά από σύντομη εισαγωγή στην θεωρία διόρθωσης λαθών στο Κεφάλαιο 2, πραγματοποιείται λεπτομερής περιγραφή των μαθηματικών πεπερασμένων πεδίων Galois (Κεφάλαιο 3) και των αρχιτεκτονικών πράξεων τους στο υλικό και το λογισμικό (Κεφάλαιο 4). Έπειτα, στα αντίστοιχα Κεφάλαια 5 και 6, περιγράφονται βήμα-προς-βήμα οι διαδικασίες κωδικοποίησης και αποκωδικοποίησης των κωδίκων Reed-Solomon. Στην συνέχεια, στο Κεφάλαιο 7 παρουσιάζεται κώδικας επαλήθευσης λογισμικού στην γλώσσα προγραμματισμού Python. Συμπληρώνοντας το Κεφάλαιο 6, στο Κεφάλαιο 8 περιγράφεται αποδοτική αρχιτεκτονική αποκωδικοποιητή, στην οποία επιτυγχάνεται και περαιτέρω βελτιστοποίηση με την μορφή μιας δεύτερης υλοποίησης. Στο Κεφάλαιο 9 επαληθεύεται η λειτουργικότητα των υλοποιημένων αποκωδικοποιητών με χρήση προσομοιώσεων στο ModelSim. Επίσης, πραγματοποιείται και σύνθεση των αποκωδικοποιητών σε διάφορα FPGA της Xilinx. Τέλος, στο Κεφάλαιο 10 συγκρίνονται οι υλοποιήσεις της εργασίας με άλλες αντίστοιχες υλοποιήσεις του χώρου της βιβλιογραφίας, καθώς και μεταξύ τους και μέσω της σύγκρισης αυτής διεξάγονται χρήσιμα συμπεράσματα.

Abstract

Error detection and correction codes are increasingly applied to modern telecommunication systems to ensure their proper operation. Numerous studies have been conducted for the case of Reed-Solomon codes, which fall into this category, and led to various algorithms and optimizations for hardware and software implementations. This thesis describes the process of making such a Reed-Solomon decoder that is based on efficient bibliographic algorithms.

After a brief introduction to the theory of error correction in Chapter 2, a detailed description of finite or Galois field mathematics (Chapter 3) and architectures for their operations in hardware and software is made (Chapter 4). Next, in the corresponding Chapters 5 and 6, step-by-step procedures for coding and decoding Reed-Solomon codes are described. Next, Chapter 7 describes the software verification code in the programming language Python. Complementing Chapter 6, Chapter 8 describes an efficient Reed-Solomon decoder architecture in hardware, which is also further optimized creating a second decoder implementation. In Chapter 9, a functionality verification of the implemented decoders using ModelSim simulations is made. Furthermore, the decoders are also synthesized for various Xilinx FPGAs. Lastly, in Chapter 10 a comparison of the present implementations with other corresponding implementations of the literature is presented. The two implementations are also compared to each other and through all these comparisons useful conclusions are drawn.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Υλοποίηση Αποκωδικοποιητή Reed-Solomon σε Υλικό

Κοσμάς Καριώτης Α.Μ. 2115069

Περιεχόμενα

1. ΕΙΣΑΓΩΓΗ 1
2. ΘΕΩΡΙΑ ΔΙΟΡΘΩΣΗΣ ΛΑΘΩΝ 2
2.1 Τηλεπικοινωνιακό σύστημα 2
2.1.1 Τύποι σημάτων 2
2.2 Μετάδοση σε κανάλι με θόρυβο 2
2.2.1 Μετάδοση χωρίς κωδικοποίηση 3
2.2.2 Ανίχνευση λαθών μη κωδικοποιημένης μετάδοσης 3
2.2.3 Ήπιες αποφάσεις ανάθεσης τιμής 3
2.3 Θεώρημα Shannon 4
2.4 Κώδικες Διόρθωσης Λαθών (ECC) 4
2.4.1 Κριτήρια σχεδιασμού 4
2.4.2 Εφαρμογές κωδίκων διόρθωσης λαθών 5
2.4.3 Πλεονεκτήματα χρήσης κωδίκων ανίχνευσης και διόρθωσης λαθών 5
2.4.4 Τύποι κωδίκων 5
3. ΠΕΔΙΑ GALOIS 8
3.1 Στοιχεία Πεδίων Galois 8
3.2 Τρόποι Αναπαράστασης Στοιχείων 8
3.2.1 Αναπαράσταση πολυωνύμου 8
3.2.2 Αναπαράσταση διανύσματος 8
3.2.3 Αναπαράσταση θετικού ακεραίου 9
3.3 Πρόσθεση στα Πεδία Galois 9
3.4 Αφαίρεση στα Πεδία Galois 10
3.5 Κατασκευή Πεδίων Galois 10
3.5.1 Πολυώνυμο κατασκευής πεδίου 10
3.5.2 Διαδικασία κατασκευής πεδίου 10
3.6 Πολλαπλασιασμός στα Πεδία Galois 11

3.6.1 Πολλαπλασιασμός στα GF(p) 12
3.6.2 Πολλαπλασιασμός στα GF(p^m) 12
3.6.3 Μια μέθοδος πολλαπλασιασμού 12
3.7 Διαίρεση στα Πεδία Galois 12
3.8 Αντίστροφοι πρόσθεσης και πολλαπλασιασμού 13
3.9 Το πεδίο Galois GF(2^8) 13
4. ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ ΠΡΑΞΕΩΝ ΣΤΑ GF(2^m) 14
4.1 Αθροιστής 14
4.2 Πολλαπλασιαστής 14
4.2.1 Γενικός πολλαπλασιασμός 14
4.2.2 Πολλαπλασιασμός με σταθερά 15
4.2.3 Πολλαπλασιασμός βασισμένος σε LUT's 16
4.2.4 Πολλαπλασιασμός στο λογισμικό 17
4.3 Διαιρέτης 17
4.3.1 Διαίρεση βασισμένη στον πολλαπλασιασμό με αντίστροφο 17
4.3.2 Διαίρεση στο λογισμικό 18
5. ΚΩΔΙΚΟΠΟΙΗΣΗ REED-SOLOMON 19
5.1 Εισαγωγή στους κώδικες Reed-Solomon 19
5.2 Επεξήγηση αποδοτικότητας σε σφάλματα ριπών 19
5.2.1 Παράδειγμα ριπής θορύβου 19
5.3 Διαδικασία κωδικοποίησης 20
5.3.1 Πολυώνυμο μηνύματος 20
5.3.2 Σχηματισμός κωδικοποιημένου μηνύματος 20
5.3.3 Βάση της διόρθωσης λαθών 21
5.3.4 Παράδειγμα κωδικοποίησης 21
5.3.5 Κωδικοποίηση στο υλικό 22
5.4 Μελέτη απόδοσης BER 24
5.5 Πιθανότητα λάθους 25
6. ΑΠΟΚΩΔΙΚΟΠΟΙΗΣΗ REED-SOLOMON 26
6.1 Τα βασικά στάδια αποκωδικοποίησης 26
6.1.1 Υπολογισμός συνδρόμων 27
6.1.2 Επίλυση βασικής εξίσωσης (key equation) 27
6.1.3 Εύρεση θέσεων και τιμών λάθους 28
6.1.4 Διόρθωση λαθών 29
6.2 Αλγόριθμοι επίλυσης βασικής εξίσωσης 29

6.2.1 Αλγόριθμος Peterson-Gorenstein-Zierler (PGZ) 29
6.2.2 Αλγόριθμος Sugiyama 30
6.2.3 Αλγόριθμος Berlekamp-Massey 31
6.2.4 Βελτιστοποιήσεις Berlekamp-Massey 32
6.3 Αλγόριθμος Chien Search για εύρεση θέσεων λάθους 44
6.3.1 Βασικός αλγόριθμος Chien Search 45
6.3.2 Βελτιστοποιήσεις Chien Search 46
6.4 Αλγόριθμος Forney για εύρεση τιμών λάθους 49
6.4.1 Βασικός Αλγόριθμος Forney 49
6.4.2 Κοινή χρήση πόρων με Chien Search 50
7. ΚΩΔΙΚΑΣ ΕΠΑΛΗΘΕΥΣΗΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΣΕ ΛΟΓΙΣΜΙΚΟ 52
7.1 Υλοποίηση μαθηματικών Galois 52
7.2 Υλοποίηση κωδικοποίησης Reed-Solomon 53
7.3 Υλοποίηση Αποκωδικοποίησης Reed-Solomon 54
8. ΥΛΟΠΟΙΗΣΗ ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΗ ΣΕ ΥΛΙΚΟ 62
8.1 Αρχιτεκτονική αποκωδικοποιητή 62
8.2 Μπλοκ υπολογισμού συνδρόμων 64
8.3 Μπλοκ επίλυσης βασικής εξίσωσης με RiBM 66
8.4 Μπλοκ Chien Search 67
8.5 Μπλοκ Forney 68
8.6 Μνήμη FIFO 69
8.7 Μπλοκ Διόρθωσης Λαθών 69
8.8 2-Παράλληλο Μπλοκ Chien 70
8.9 Βελτιστοποιημένη Αρχιτεκτονική 72
9. ΕΠΑΛΗΘΕΥΣΗ 73
9.1. Προσομοίωση Μνήμης FIFO 73
9.2 Προσομοίωση Μπλοκ Υπολογισμού Συνδρόμων 73
9.3 Προσομοίωση Μπλοκ RiBM 74
9.4 Προσομοίωση Σειριακού Μπλοκ Chien 74
9.5 Προσομοίωση 2-Παράλληλου Μπλοκ Chien 75
9.6 Προσομοίωση Μπλοκ Forney 76
9.7 Προσομοίωση Μπλοκ Διόρθωσης Λαθών 76
9.8 Προσομοίωση Συνολικού Αποκωδικοποιητή με Σειριακό Chien 77
9.9 Προσομοίωση Συνολικού Αποκωδικοποιητή με 2-Παράλληλο Chien 77
9.10 Σύνθεση των Διαφόρων Μπλοκ σε Xilinx FPGA 78

9.11 Σύνθεση Σειριακής Αρχιτεκτονικής σε Xilinx FPGA 80
9.12 Σύνθεση 2-Παράλληλης Αρχιτεκτονικής σε Xilinx FPGA 80
10. ΣΥΓΚΡΙΣΗ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ 81
10.1 Σύγκριση με υλοποίηση της πηγής [XX] 81
10.2 Σύγκριση με υλοποίηση της πηγής [XXVIII] 82
10.3 Σύγκριση με υλοποίηση της πηγής [XXIX] 82
10.4 Σύγκριση Chien με την υλοποίηση της πηγής [XXIV] 82
10.5 Σύγκριση με υλοποίηση της πηγής [XXX] 83
10.6 Σύγκριση με υλοποίηση της πηγής [XXXI] 83
10.7 Σύγκριση με υλοποίηση της πηγής [XXXII] 84
11. ΠΗΓΕΣ ΚΑΙ ΒΙΒΛΙΟΓΡΑΦΙΑ 84
12. ΠΑΡΑΡΤΗΜΑΤΑ 86
Παράρτημα Α – Εκθετικά και Λογάριθμοι του GF(2^8) με p(x) = $(11D)_{hex}$ 86
Παράρτημα Β – Γενικός Πολλαπλασιαστής για το GF(2^8) με p(x) = $(11D)_{hex}$ 88
Παράρτημα Γ – Πολλαπλασιαστές μεταβλητής-σταθεράς στο GF(2^8) με p(x) = $(11D)_{hex}$ 90

Κατάλογος Πινάκων

Πίνακας 3-1 - Τα στοιχεία του πεδίου GF(16) με p(x) = x4 + x + 1 11
Πίνακας 3-2 - Ορισμένα στοιχεία του πεδίου GF(2^8) με p(x) = $x^8 + x^4 + x^3 + x^2 + 1$ 13
Πίνακας 6-1 - Αριθμός ελαχιστοποιημένων πυλών Chien Search [XXIV] 47
Πίνακας 8-1 - Παράμετροι και Αλγόριθμοι του Υλοποιημένου Αποκωδικοποιητή RS 62
Πίνακας 8-2 - Τιμές σημάτων εξόδου του μπλοκ διόρθωσης λαθών 70
Πίνακας 9-1 - Σύνθεση FIFO στο Xilinx Spartan3E-XC3500E-5 78
Πίνακας 9-2 - Σύνθεση Μπλοκ Συνδρόμων στο Xilinx Spartan3E-XC3500E-5 79
Πίνακας 9-3 - Σύνθεση Μπλοκ RiBM στο Xilinx Spartan3E-XC3500E-5 79
Πίνακας 9-4 - Σύνθεση Σειριακού Μπλοκ Chien στο Xilinx Spartan3E-XC3500E-5 79
Πίνακας 9-5 - Σύνθεση 2-Παράλληλου Μπλοκ Chien στο Xilinx Spartan3E-XC3500E-5 79
Πίνακας 9-6 - Σύνθεση Μπλοκ Forney στο Xilinx Spartan3E-XC3500E-5 80
Πίνακας 9-7 - Σύνθεση Μπλοκ Διόρθωσης Λαθών στο Xilinx Spartan3E-XC3500E-5 80
Πίνακας 9-8 - Σύνθεση Σειριακής Αρχιτεκτονικής στο Xilinx Spartan3E-XC3500E-5 80
Πίνακας 9-9 - Σύνθεση 2-Παράλληλης Αρχιτεκτονικής στο Xilinx Spartan3E-XC3500E-5 80
Πίνακας 10-1 - Σύνθεση στο Xilinx Spartan3-X3S50TQ155-5 για κώδικα RS(255,251) 81
Πίνακας 10-2 - Σύνθεση στο Xilinx Spartan3E-XC3500E-5 για κώδικα RS(255,251) 82
Πίνακας 10-3 - Σύνθεση στο Xilinx Spartan3E-XC3500E-5 για κώδικα RS(255,239) 82
Πίνακας 10-4 - Σύνθεση Chien στο Xilinx Spartan3E-XC3500E-5 για κώδικα RS(255,239) 82
Πίνακας 10-5 - Σύνθεση στο Xilinx Spartan3E-XC3500E-4 για κώδικα RS(255,239) 83
Πίνακας 10-6 - Σύνθεση στο Xilinx Virtex4-XC4VLX15-12SF363 για κώδικα RS(255,239) 83
Πίνακας 10-7 - Σύνθεση στο Xilinx Spartan3E-XC3500E-5 για κώδικα RS(255,239) 84
Πίνακας 12-1 – Εκθετικά του GF(2^8) [X] 86
Πίνακας 12-2 – Λογάριθμοι του GF(2^8) [X] 87

Κατάλογος Σχημάτων

Σχήμα 2-1 - Παράδειγμα επανάληψης του κάθε bit με ρυθμό n = 3 6
Σχήμα 2-2 - Παράδειγμα συνελικτικού κωδικοποιητή 7
Σχήμα 4-1 - Αρχιτεκτονική Αθροιστή στα πεδία GF(2^m) 14
Σχήμα 4-2 - Αρχιτεκτονική Πολλαπλασιασμού λογισμικού στα πεδία GF(2^m) 17
Σχήμα 4-3 - Αρχιτεκτονική Διαίρεσης με χρήση πίνακα αντιστρόφου στα πεδία GF(2^m) 18
Σχήμα 4-4 - Αρχιτεκτονική Διαίρεσης Λογισμικού στα πεδία GF(2^m) 18
Σχήμα 5-1 - Επηρεασμός συμβόλων RS(255,247) από θόρυβο ριπής 26-bit 20
Σχήμα 5-2 - Παράδειγμα πολυωνυμικής διαίρεσης στην περίπτωση του GF(2^4) [VI] 22
Σχήμα 5-3 - Παράδειγμα διαίρεσης διοχέτευσης στην περίπτωση του GF(2^4) [VI] 23
Σχήμα 5-4 - Κύκλωμα Κωδικοποίησης LFSR 24
Σχήμα 5-5 - Διάγραμμα P_b και απόδοσης E_b/N_o [XIII] 25
Σχήμα 6-1 - Βασική Αρχιτεκτονική αποκωδικοποίησης 26
Σχήμα 6-2 - Αρχιτεκτονική Αλγορίθμου Sugiyama για Υπολογισμό Λ(x) και Ω(x) 31
Σχήμα 6-3 - Αρχιτεκτονική Αλγορίθμου iBM για Υπολογισμό Λ(x) και Ω(x) 35
Σχήμα 6-4 - Συστολική Αρχιτεκτονική Αλγορίθμου riBM για Υπολογισμό Λ(x) και Ω(x) 38
Σχήμα 6-5 - Ομογενής Συστολική Αρχιτεκτονική Αλγορίθμου RiBM 39
Σχήμα 6-6 - Αρχιτεκτονική Αλγορίθμου DiBM για υπολογισμό Λ(x) 42
Σχήμα 6-7 - Αρχιτεκτονική Αλγορίθμου DiBM για υπολογισμό Ω(x) 42
Σχήμα 6-8 - Αρχιτεκτονική Berlekamp-Massey Διπλής Γραμμής για t = 5 43
Σχήμα 6-9 – Βασική Αρχιτεκτονική Αλγορίθμου Chien Search για εύρεση θέσεων λάθους 45
Σχήμα 6-10 - Αρχιτεκτονική Chien Search βελτιωμένη ως προς τον αριθμό λογικών πυλών 46
Σχήμα 6-11 - 2-Παράλληλη Αρχιτεκτονική Chien Search 47
Σχήμα 6-12 - 2-Παράλληλη Αρχιτεκτονική Chien Search με διάσπαση των a^{ji} 48
Σχήμα 6-13 - p-Παράλληλη Αρχιτεκτονική Chien Search 48
Σχήμα 6-14 - Αρχιτεκτονική Αλγορίθμου Forney για υπολογισμό τιμών λάθους 50
Σχήμα 6-15 - Αρχιτεκτονική Chien με διάσπαση αθροίσματος σε ζυγό και περιττό μέρος 51
Σχήμα 8-1 - Αρχιτεκτονική Υλοποιημένου Αποκωδικοποιητή 63
Σχήμα 8-2 - Μπλοκ Υπολογισμού Συνδρόμων 65
Σχήμα 8-3 - Λογική Μπλοκ Επίλυσης βασικής εξίσωσης με Αλγόριθμο RiBM 66
Σχήμα 8-4 - Μπλοκ εύρεσης θέσεων λάθους με Chien 68
Σχήμα 8-5 - Μπλοκ εύρεσης τιμών λάθους με Forney 69
Σχήμα 8-6 – Μπλοκ εύρεσης θέσεων λάθους με 2-Παράλληλο Chien 71
Σχήμα 8-7 Χρονισμός Σειριακής Αρχιτεκτονικής Αποκωδικοποιητή RS 72
Σχήμα 8-8 Χρονισμός Αρχιτεκτονικής Αποκωδικοποιητή RS με 2-παράλληλο Chien 72

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Υλοποίηση Αποκωδικοποιητή Reed-Solomon σε Υλικό

Κοσμάς Καριώτης Α.Μ. 2115069

1. ΕΙΣΑΓΩΓΗ

Στην σύγχρονη εποχή ανταλλάσσονται συνεχώς δεδομένα μέσα από διαφόρων μορφών κανάλια επικοινωνίας. Τα συστήματα αυτά οφείλουν να λειτουργούν με αξιοπιστία, καθώς και να προστατεύουν την ακεραιότητα των δεδομένων που αποστέλλονται. Στους διαύλους επικοινωνίας παρουσιάζεται θόρυβος και παρεμβολές, τα οποία τροποποιούν την μεταδιδόμενη πληροφορία προσθέτοντας λάθη ή σφάλματα. Χωρίς κάποια ενέργεια πριν την αποστολή της πληροφορίας, τα λάθη που πιθανόν εμφανίζονται δεν μπορούν να ανιχνευτούν εύκολα. Έτσι, πριν την αποστολή η πληροφορία κωδικοποιείται χρησιμοποιώντας κώδικες ανίχνευσης και διόρθωσης λαθών, οι οποίοι συνήθως προσθέτουν ένα αριθμό πλεονάζον συμβόλων ελέγχου. Με αυτόν τον τρόπο, κατά την λήψη ενός κωδικοποιημένου μηνύματος ο δέκτης μπορεί να ελέγξει αν παρουσιάστηκαν λάθη στο μήνυμα και να ζητήσει πιθανόν την επανεκπομπή του. Επίσης, μπορεί να διορθώσει και κάποια απ' αυτά τα λάθη σύμφωνα με την ικανότητα διόρθωσης λαθών που του παρέχει ο κώδικας που χρησιμοποιεί. Φυσικά υπάρχουν πολλές περιπτώσεις επικοινωνίας στις οποίες η επανεκπομπή είναι αδύνατη ή πολύ δύσκολη. Σ' αυτές τις επικοινωνίες η χρήση κωδίκων διόρθωσης λαθών, όπως οι κώδικες Reed-Solomon, είναι απαραίτητη.

Οι κώδικες Reed-Solomon εφαρμόζονται συνήθως στα στοιχεία των πεπερασμένων πεδίων $GF(2^m)$ ή πεδίων Galois, όπου οι πράξεις γίνονται στο δυαδικό σύστημα. Στα $GF(2^m)$ η πρόσθεση αποτελεί μια απλή πύλη XOR ανά bit, ενώ εφαρμόζοντας πράξεις και αλγορίθμους της βιβλιογραφίας επιτυγχάνεται και εύκολα βελτιστοποίηση των αρχιτεκτονικών πολλαπλασιασμού και διαίρεσης. Επιπρόσθετα, προτάθηκαν ποικίλοι αλγόριθμοι που χρησιμοποιούνται για την κωδικοποίηση των δεδομένων στην μορφή Reed-Solomon, καθώς και τροποποιήθηκαν οι βασικοί αλγόριθμοι που εφαρμόζονται κατά την αποκωδικοποίηση. Πιο συγκεκριμένα, οι βελτιστοποιήσεις αυτές μείωσαν τους πόρους και την καθυστέρηση κρίσιμου μονοπατιού, αυξάνοντας την συχνότητα εκτέλεσης των βασικών μπλοκ αποκωδικοποίησης.

Στους αποκωδικοποιητές υλικού που υλοποιήθηκαν κατά την διάρκεια της πτυχιακής εργασίας, και των οποίων οι αρχιτεκτονικές παρουσιάζονται στην Ενότητα 8, χρησιμοποιούνται αρχιτεκτονικές και βελτιστοποιήσεις των βασικών αλγορίθμων αποκωδικοποίησης, που θεωρούνται πως έχουν την πιο βέλτιστη απόδοση. Ωστόσο, πριν την υλοποίηση αποκωδικοποιητή Reed-Solomon στο υλικό, απαιτείται μια μικρή εισαγωγή στης θεωρία διόρθωσης λαθών, καθώς και περιγραφή των πράξεων στα πεπερασμένα πεδία Galois. Πριν την υλοποίηση πραγματοποιείται επίσης και λεπτομερής ανάλυση των διαδικασιών κωδικοποίησης και αποκωδικοποίησης μαζί με αλγορίθμους και βελτιστοποιήσεις τους, που χρησιμοποιούνται στα διάφορα στάδια.

2. ΘΕΩΡΙΑ ΔΙΟΡΘΩΣΗΣ ΛΑΘΩΝ

2.1 Τηλεπικοινωνιακό σύστημα

Τηλεπικοινωνιακό σύστημα αποτελεί κάθε είδους επικοινωνία μεγάλης απόστασης, η οποία χρησιμοποιεί συνηθισμένα κανάλια μετάδοσης, όπως τηλέφωνο, τηλεόραση και ραδιόφωνο. Ένα υποσύνολο των τηλεπικοινωνιακών συστημάτων αποτελούν οι επικοινωνίες μετάδοσης δεδομένων, οι οποίες εκμεταλλεύονται τεχνολογίες τηλεπικοινωνιών. Έτσι, γενικεύοντας, τα τηλεπικοινωνιακά συστήματα ορίζονται ως συλλογές συμβατού υλικού και λογισμικού, διατεταγμένα με τρόπο τέτοιο που επιτρέπει την μετάδοση πληροφορίας από ένα σημείο σε κάποιο άλλο. Αυτή η πληροφορία μπορεί να είναι κείμενο, δεδομένα, γραφικά, φωνή, έγγραφα, βίντεο κ.α [I].

Ένα τηλεπικοινωνιακό σύστημα οφείλει να εκτελεί συγκεκριμένες διαφανείς προς τον χρήστη λειτουργίες [I]:

- μετάδοση της πληροφορίας
- δημιουργία διασύνδεσης αποστολέα-δέκτη
- αποτελεσματική δρομολόγηση πληροφορίας
- επεξεργασία πληροφορίας και εξασφάλιση παραλαβής σωστής πληροφορίας
- έλεγχος πληροφορίας για σφάλματα και αναδιάταξη μορφοποίησης όταν απαιτείται
- μετατροπή πληροφορίας σε διαφορετική μορφοποίηση όταν αλλάζει το μέσο
- έλεγχος της ροής πληροφοριών, διατηρώντας πληροφορίες σχετικά με το δίκτυο
- εξασφάλιση ασφάλειας πληροφορίας

2.1.1 Τύποι σημάτων

Τα τηλεπικοινωνιακά μέσα μεταφέρουν δύο τύπων σήματα, αναλογικά και ψηφιακά. Τα αναλογικά αποτελούν συνεχή κύματα που μεταδίδουν την πληροφορία με την μορφή αλλοιώσεων του πλάτους και της συχνότητας του φέροντος. Τέτοιου είδους σήματα χρησιμοποιούνται σήμερα στην ραδιοφωνία, ενώ χρησιμοποιήθηκαν παλιότερα και στην τηλεφωνία και τα συστήματα ηχοληψίας. Στην σημερινή εποχή πραγματοποιήθηκε μετάβαση στα ψηφιακά σήματα, τα οποία αποτελούν διακριτούς παλμούς, που μεταδίδουν την πληροφορία σε «1» και «0». Τα ψηφιακά συστήματα τείνουν να επηρεάζονται λιγότερο από παρεμβολές και θόρυβο. Επίσης, εφόσον είναι εξαιρετικά εύκολη η διάκριση μεταξύ 0 και 1, ακόμα και με την παρουσία θορύβου, η αναμετάδοση για ενίσχυση του σήματος σε μεταδόσεις μεγάλης απόστασης, ελαχιστοποιεί την επίδραση του θορύβου [I].

2.2 Μετάδοση σε κανάλι με θόρυβο

Κατά την μετάδοση, στο κανάλι επικοινωνίας παρουσιάζεται διαφόρων ειδών θόρυβος. Στην περίπτωση των ψηφιακών συστημάτων ο θόρυβος αυτός προκαλεί σφάλματα που εντάσσονται σε τρείς ευρείς κατηγορίες:

1) Τυχαία σφάλματα – όπου τα σφάλματα εμφανίζονται σε μεμονωμένα bits και οι πιθανότητες λαθών στα bit είναι ανεξάρτητες ή σχεδόν ανεξάρτητες μεταξύ τους.
2) Σφάλματα ριπής – όπου τα σφάλματα bit εμφανίζονται διαδοχικά στο χρόνο και σε ομάδες
3) Κρουστικά Σφάλματα – όπου μεγάλα τμήματα δεδομένων είναι γεμάτα σφάλματα.

Τα τυχαία σφάλματα προκαλούνται συνήθως από τον θερμικό θόρυβο των καναλιών επικοινωνίας. Κύριες αιτίες σφάλματων ριπής αποτελούν η εξασθένιση του σήματος κατά την μετάδοσή του και μηχανικά ελαττώματα συστημάτων αποθήκευσης. Τα κρουστικά σφάλματα

προκαλούνται από μεγάλες αστοχίες συστήματος και προκαλούν καταστροφικές αποτυχίες που είναι σοβαρές και μη αναγνωρίσιμες. Οι σημερινές τεχνικές διόρθωσης λαθών μπορούν μόνο να ανιχνεύσουν τέτοιου είδους σφάλματα και όχι να τα διορθώσουν. Η επισήμανση τέτοιου είδους σφαλμάτων από οποιοδήποτε σύστημα αποκωδικοποιητή κρίνεται ως εξαιρετικής σημασίας και χρησιμότητας λειτουργία. Μέσω της επισήμανσης τους, τμήματα με ανεπανόρθωτα σφάλματα αναμεταδίδονται και λαμβάνονται πιθανά σωστά κατά την επόμενη λήψη τους[V].

2.2.1 Μετάδοση χωρίς κωδικοποίηση

Κατά την μετάδοση πληροφορίας, με την μορφή αναλογικού σήματος, από ένα σημείο σε κάποιο άλλο, το οποίο συνδέεται μέσω θορυβώδους καναλιού επικοινωνίας συγκεκριμένου εύρους ζώνης και στο οποίο χρησιμοποιείται σταθερή ισχύ εκπομπής, κύριος στόχος είναι η επίτευξη κάποιου βαθμού αξιοπιστίας. Λαμβάνοντας την πληροφορία στο δέκτη χωρίς τροποποίηση, μαζί με την χρήσιμη πληροφορία πραγματοποιείται και λήψη μεγάλου θορύβου. Η διάκριση μεταξύ θορύβου και πληροφορίας είναι πρακτικώς ανέφικτη, καθώς η πληροφορία είναι «θαμμένη» κάτω από τον θόρυβο [II].

Για την εξάλειψη του μεγαλύτερου μέρους του θορύβου, προτού ληφθεί η πληροφορία περνιέται το αναλογικό σήμα μέσα από ζωνοπερατό φίλτρο, το οποίο φιλτράρει συχνότητες πάνω από κάποια τιμή. Με αυτόν τον τρόπο βελτιώνεται το σήμα που λαμβάνει ο δέκτης κατά μεγάλο βαθμό. Ταχείς εναλλαγές του αναλογικού φέροντος διαγράφονται και ο δέκτης λαμβάνει πεπερασμένο αριθμό τάσεων για κάθε bit [II].

2.2.2 Ανίχνευση λαθών μη κωδικοποιημένης μετάδοσης

Ακόμα και μετά το φιλτράρισμα του σήματος, το σήμα περιέχει ακόμη θόρυβο που βρίσκεται μέσα στο εύρος ζώνης του σήματος. Αυτός ο εναπομείναν θόρυβος επιδρά σοβαρά στο λαβών σήμα και έτσι ο δέκτης πρέπει να «μαντέψει» ποια ήταν αρχική πληροφορία. Για παράδειγμα έστω πως θεωρείτε κάθε τιμή τάσης πάνω από 0V πως αποτελεί λογικό 1, ενώ κάθε τιμή τάσης από 0V και κάτω αποτελεί λογικό 0. Όταν η επίδραση του θορύβου είναι αρκετά μεγάλη, η τιμή της τάσης του μεταδιδόμενου bit μπορεί να αντιστραφεί πολύ εύκολα σε πολικότητα, προκαλώντας έτσι το λεγόμενο ‘σφάλμα bit’ στον δέκτη. Στην περίπτωση που ένα ή και περισσότερα bit της πληροφορίας αποκωδικοποιηθούν λάθος, το λαμβανόμενο μήνυμα είναι λάθος και έχουμε ‘σφάλμα μηνύματος’. Δεν μπορεί να βελτιστοποιηθεί παραπάνω το ποσοστό επιτυχίας σωστής ανίχνευσης, όσο αποστέλλεται η πληροφορία μη κωδικοποιημένη. Οι μοναδικές παράμετροι που μπορούν να τροποποιηθούν είναι το πλάτος της μετάδοσης και η διάρκεια αποστολής κάθε bit, η οποία επιτρέπει να χρησιμοποιηθεί φίλτρο στενότερης ζώνης κατά την λήψη, αφαιρώντας έτσι περισσότερο θόρυβο από το λαμβανόμενο μήνυμα [II].

2.2.3 Ήπιες αποφάσεις ανάθεσης τιμής

Στην προηγούμενη Ενότητα θεωρήθηκε πώς ο δέκτης κατηγοριοποιεί το λαμβανόμενο σήμα σε 1 ή 0, με κριτήριο το θετικό ή αρνητικό πλάτος του σήματος. Αυτού του είδους η απόφαση για κάθε πλάτος είναι εξαιρετικά «σκληρή», καθώς έτσι απορρίπτονται πιθανά χρήσιμες πληροφορίες που περιέχονται μέσα στα θορυβώδη πλάτη. Για την ακρίβεια, είναι πιο πιθανό να αποτελεί λογικό 1 ένα εξαιρετικά θετικό πλάτος, παρά ένα πλάτος που είναι απλώς θετικό (αντίστοιχα για αρνητικό). Επομένως, πρέπει να πραγματοποιούνται πιο «ήπιες» αποφάσεις στον δέκτη, λαμβάνοντας υπόψη και την συσχέτιση μεταξύ κάθε λέξης και του συνολικού κώδικα-λέξεων. Ως συσχέτιση ορίζουμε την μέτρηση της ομοιότητας μεταξύ δύο σημάτων.

Μία προσέγγιση συσχέτισης που μπορεί να εφαρμοστεί στον δέκτη είναι η σύγκριση της ληφθείσας λέξης με όλες τις πιθανές κωδικές λέξεις, χρησιμοποιώντας ως μέτρο απόφασης τον

αριθμό συμβόλων που διαφέρουν (απόσταση Hamming). Ακόμη, καλή προσέγγιση αποτελεί και η εύρεση της μικρότερης Ευκλείδειας απόστασης μεταξύ του πλάτους και των πιθανών κωδικών λέξεων. Σε και τις δύο περιπτώσεις όμως προσδιορίζουμε την μέγιστη πιθανή λέξη [II].

2.3 Θεώρημα Shannon

Η χωρητικότητα καναλιού, C, σε bits ανά δευτερόλεπτο, που υπολογίζεται με το λεγόμενο θεώρημα Shannon-Hartley, ορίζει ένα άνω όριο στον αριθμό bits ανά δευτερόλεπτο που μπορούν να μεταδοθούν αξιόπιστα μέσα από ένα συγκεκριμένο κανάλι με παραμέτρους:

- W – εύρος ζώνης σε Hz
- S – λαμβανόμενη ισχύς σήματος
- N – ισχύς θορύβου

Το θεώρημα ορίζεται ως εξής:

$$C = W \log_2[\,1 + \frac{S}{N}] \tag{1}$$

Ο Shannon απέδειξε πως σε ένα κανάλι απεριόριστου εύρους ζώνης, με αρκετά περίπλοκη κωδικοποίηση, είναι δυνατή η μετάδοση πληροφοριών με εξαιρετικά μικρό ποσοστό σφάλματος. Αυτό ισχύει μόνο στις περιπτώσεις όπου ο ρυθμός μετάδοσης R σε bits/sec είναι μικρότερος από την χωρητικότητα του καναλιού, δηλ. $R < C$ [V].

Επιπλέον, αξίζει να σημειωθεί πως το ποσοστό επιτυχίας αναγνώρισης σωστού bit αυξάνεται όσο μεταδίδεται περισσότερη πληροφορία, κωδικοποιώντας δηλαδή περισσότερα bits ανά σύμβολο. Σύμφωνα με τον Shannon, υπό ορισμένες προϋποθέσεις, η δυνατότητα διόρθωσης λαθών ενός κώδικα αυξάνεται πιο γρήγορα από την δημιουργία σφαλμάτων σε μικρότερα σύμβολα με μεγαλύτερο εύρος ζώνης. Το ποσοστό επιτυχίας αναγνώρισης αγγίζει το 100% ενώ αυξάνεται το μήκος των απεσταλμένων τμημάτων και όσο διατηρείται η αναλογία σήματος προς θόρυβο (SNR) πάνω από ορισμένη κρίσιμη τιμή. Η βέλτιστη θεωρητικά απόδοση επιτυγχάνεται με άπειρο μήκος μηνύματος και κώδικα. Επομένως, δεν έχει σημασία πόσο εύρους ζώνης καταλαμβάνει η κωδικοποιημένη μετάδοση, αλλά απαιτείται οι διάφορες λέξεις-κλειδιά να διαφέρουν μεταξύ τους κατά όσο το δυνατόν περισσότερο [II].

2.4 Κώδικες Διόρθωσης Λαθών (ECC)

2.4.1 Κριτήρια σχεδιασμού

Κατά τον σχεδιασμό ενός συστήματος επικοινωνίας, πρέπει να λαμβάνονται υπόψη και τα λάθη που μπορούν να παρουσιαστούν στο κανάλι. Στο κανάλι αυτό παρουσιάζονται λάθη με συγκεκριμένο ρυθμό εμφάνισης (error-rate) και εξασθενίσεις. Ανάλογα με το είδος της επικοινωνίας επιτρέπεται μικρή ή μεγάλη καθυστέρηση. Στην περίπτωση της φωνητικής επικοινωνίας επιτρέπεται μικρή καθυστέρηση, ενώ κατά την αποστολή ενός γράμματος ηλεκτρονικού ταχυδρομείου η καθυστέρηση μπορεί να είναι και μεγαλύτερη. Επιπρόσθετα, σημαντικότατη είναι και η χρήση των υπολογιστικών πόρων, το πόσο χρόνο θα χρειαστεί δηλαδή η κωδικοποίηση/αποκωδικοποίηση και πόσους πόρους του συστήματος θα καταλαμβάνει κατά την στιγμή αυτή [III].

Κατά την επιλογή του κατάλληλου κώδικα καθορίζονται, επομένως, και διάφορες παραμέτρους αυτού του κώδικα. Αυτές είναι [III]:

- Το αλφάβητο – συνήθως πεπερασμένο πεδίο Galois

- Το μήκος του κωδικοποιημένου μηνύματος σε bits/bytes
- Πόσα λάθη διορθώνονται ανά κωδικοποιημένο μήνυμα.
- Το πλήθος των διαθέσιμων κωδικοποιημένων λέξεων
- Διαθέσιμοι αλγόριθμοι κωδικοποίησης/αποκωδικοποίησης

Πέρα από τις παραμέτρους απαιτείται και ο υπολογισμός άνω όριων για την επικοινωνία. Χρειάζεται να υπολογιστεί το όριο (limit) Shannon που υπολογίζει την ποσότητα πληροφορίας που μπορεί να μεταδοθεί στο κανάλι, καθώς και το λεγόμενο Singleton bound:

$$d \leq n - k + 1 \tag{2}$$

που υπολογίζει το πλήθος των κωδικών λέξεων (codewords) για δεδομένη ικανότητα διόρθωσης λαθών (error-correcting capacity) και μήκος κωδικών λέξεων (codeword length) [III].

2.4.2 Εφαρμογές κωδίκων διόρθωσης λαθών

Οι κώδικες διόρθωσης λαθών βρίσκουν εφαρμογή στην κωδικοποίηση καναλιών (channel coding), κρυπτογραφία (cryptography) και τον ομαδικό έλεγχο (group testing). Τα δεδομένα μεταδίδονται μέσα από κανάλια γεμάτα θορύβους και παρεμβολές. Σε πολλές περιπτώσεις δεν μπορούν να ξανασταλούν τα δεδομένα μέσω επανεκπομπής τους. Η επανεκπομπή αυτή είτε είναι ακριβή σε κόστος είτε αδύνατη. Έτσι, απαιτείται κωδικοποίηση των δεδομένων αυτών η οποία να επιτρέπει την ανίχνευση και διόρθωση των λαθών [III].

Παραδείγματα τέτοιου είδους αποτελούν [III]:

- Η επικοινωνία με το διάστημα – όπου αν χαθούν δεδομένα είναι πια αργά να ξανασταλούν.
- Η ασύρματη επικοινωνία – όπου αν χαθούν δεδομένα αναμένεται ήδη το επόμενο πακέτο δεδομένων.
- Ο χώρος αποθήκευσης – όπου αν χαθούν δεδομένα δεν μπορούν να ανακτηθούν.

2.4.3 Πλεονεκτήματα χρήσης κωδίκων ανίχνευσης και διόρθωσης λαθών

Οι κώδικες ανίχνευσης και διόρθωσης λαθών παρουσιάζουν πολλά πλεονεκτήματα. Πρώτα απ' όλα, χρησιμοποιώντας ΚΔΛ ένα σύστημα επιτυγχάνει μεγάλο βαθμό αξιοπιστίας της επικοινωνίας, ακόμα και με παρουσία θορύβου στο κανάλι. Όταν η βελτιστοποίηση ενός συστήματος είναι ανέφικτη ή ακριβή με άλλα μέσα, οι ΚΔΛ παρέχουν έλεγχο λαθών και αύξηση της επίδοσης του συστήματος. Η μείωση του κόστους του συστήματος δεν επηρεάζει όμως την υπάρχουσα επίδοση, καθώς εφαρμόζοντας ΚΔΛ ανακτούμε την «χαμένη» απόδοση [V].

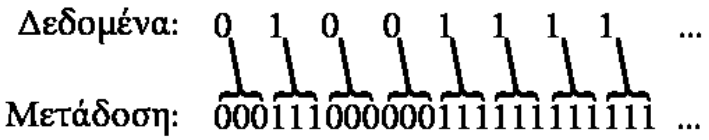
2.4.4 Τύποι κωδίκων

Για την ανίχνευση και διόρθωση των λαθών, απαιτείται η εισαγωγή αρκετών συμβόλων πλεονασμού (redundancy symbols). Ο αριθμός αυτών των συμβόλων καθορίζεται από τον βαθμό διόρθωσης που απαιτείται. Γενικότερα, αυτά τα επιπρόσθετα σύμβολα πρέπει να παρέχουν αρκετές πληροφορίες για τον εντοπισμό των θέσεων αλλά και τον καθορισμό της τιμής των εσφαλμένων συμβόλων του λαμβάνον μηνύματος [IV].

Η γενική κατηγοριοποίηση των κωδίκων διόρθωσης λαθών είναι σε συνεχούς (continuous) και βασισμένους-σε-μπλοκ (block-based). Οι διάφοροι κώδικες μπορούν να συνδυαστούν για την παραγωγή σύνθετων (composite) κωδίκων [IV].

2.4.4.1 Επανάληψη συμβόλων

Η πιο απλή μορφή ΚΔΛ είναι η επανάληψη συμβόλων (repetition). Κάθε bit αποστέλλεται n-φορές και στον δέκτη χρησιμοποιείται σύστημα ψηφοφορίας που προσδιορίζει το πιο πιθανό bit [IV]. Παράδειγμα της διαδικασίας αυτής παρουσιάζεται στο Σχήμα 2-1.



Σχήμα 2-1 - Παράδειγμα επανάληψης του κάθε bit με ρυθμό n = 3

Για να επιτευχθεί αξιόπιστη επικοινωνία με αυτόν τον τρόπο απαιτείται μείωση του ρυθμού επανάληψης κοντά στο 0. Ωστόσο, σύμφωνα με την Ενότητα 2.3 ο θεωρητικός ρυθμός μετάδοσης R πρέπει να είναι λίγο μικρότερος από την χωρητικότητα καναλιού C ($R < C$). Άρα, η μέθοδος αυτή είναι μη αποτελεσματική εφόσον βέλτιστη απόδοση επιτυγχάνεται όταν $R \cong C$ [IV].

2.4.4.2 Κώδικας Hamming

Ένας αρκετά απλός κώδικας μπλοκ είναι ο Κώδικας Hamming. Για κάθε bit υπολογίζεται η λεγόμενη απόσταση Hamming (Hamming distance), που υπολογίζει τον αριθμό των bits που είναι διαφορετικά σε δύο συμβολοσειρές bit. Για παράδειγμα, υποθέτοντας δύο συμβολοσειρές X και Y, και πως η μέγιστη απόσταση Hamming είναι d, μπορεί να μετατραπεί το X σε Y με d λάθη bit. Επομένως, σε περιβάλλον κωδικοποίησης λαθών, με ελάχιστη απόσταση Hamming μεταξύ δύο κωδικών λέξεων $d + 1$, μπορούν να ανιχνευτούν τα d από αυτά τα λάθη. Μπορούν να διορθωθούν μέχρι d λάθη bit μόνο όταν η ελάχιστη απόσταση Hamming είναι $2d + 1$ [IV].

2.4.4.3 Κώδικες Reed-Solomon

Οι κώδικες Reed-Solomon λειτουργούν με την ίδια λογική όπως και οι κώδικες Hamming. Η μόνη διαφορά είναι πως οι Reed-Solomon λειτουργούν με σύμβολα πολλών-bits και όχι μεμονωμένα bits [V].

Οι μπλοκ-κώδικες Reed-Solomon είναι εξαιρετικά αποδοτικοί στην περίπτωση ριπών λάθους και βρίσκουν εφαρμογή σε μεγάλο πλήθος συστημάτων ψηφιακών επικοινωνιών και αποθήκευσης. Ένας κώδικας Reed-Solomon ορίζεται ως: $RS(n, k)$ με s-bit σύμβολα. Ο κωδικοποιητής λαμβάνει k-σύμβολα δεδομένων με s-bits το καθένα και εισάγει αρκετά bits ισοτιμίας (parity bits), ώστε να προκύψει κωδικοποιημένη λέξη n-συμβόλων. Δηλαδή εισάγονται στην ουσία $n - k$ σύμβολα ισοτιμίας (parity symbols) με s-bits το καθένα. Ο αντίστοιχος αποκωδικοποιητής Reed-Solomon μπορεί να διορθώσει μέχρι t-σύμβολα, όπου $2t = n - k$ [IV].

Για δεδομένο μήκος συμβόλου s, το μέγιστο μήκος κωδικοποιημένης λέξης n είναι [IV]:

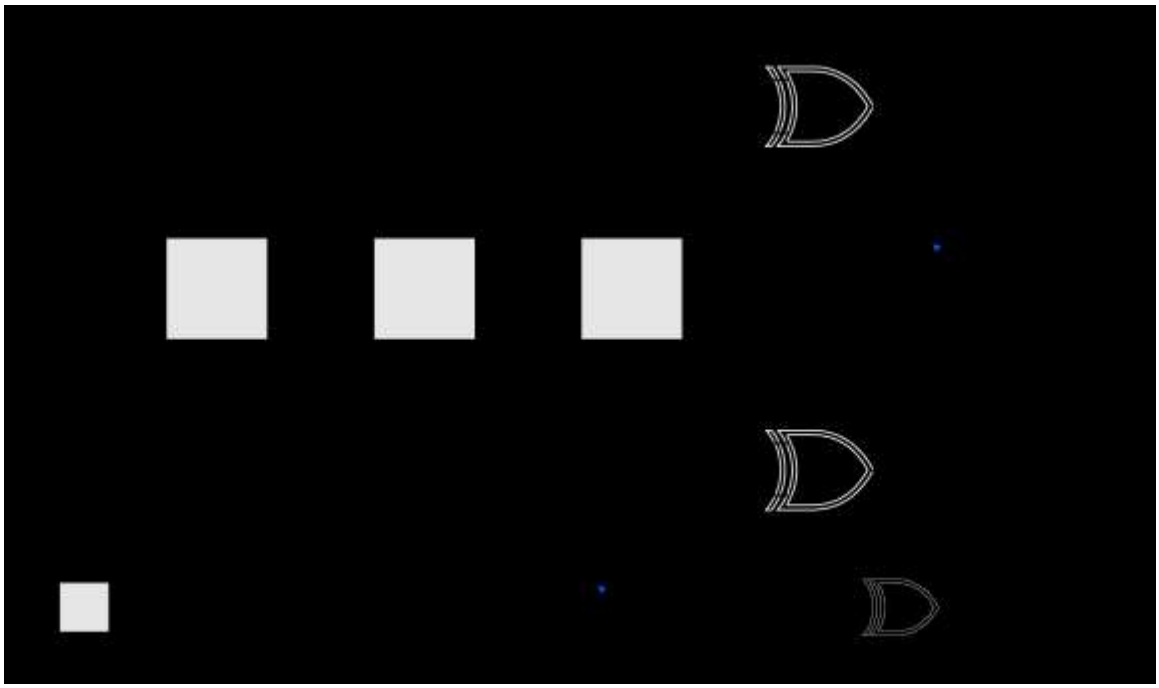
$$n = 2^s - 1 \quad (3)$$

Πιο ολοκληρωμένες πληροφορίες για τους κώδικες Reed-Solomon καθώς και επεξηγήσεις των διαδικασιών κωδικοποίησης και αποκωδικοποίησής τους θα δοθούν από την Ενότητα 5 και μετά.

2.4.4.4 Συνελικτικοί Κώδικες

Οι συνελικτικοί κώδικες είναι σχεδιασμένοι για συνεχή λειτουργία που είναι εξαιρετικά χρήσιμη στα συστήματα μετάδοσης δεδομένων. Χρησιμοποιώντας καταχωρητές ολίσθησης (shift registers)

ο κωδικοποιητής παράγει μια συνεχή κωδικοποιημένη ροή εξόδου. Η έξοδος κάθε ακολουθίας bit εξαρτάται από τις προηγούμενες ακολουθίες bit. Η τελική ακολουθία που προκύπτει, αποτελεί την πιο πιθανή ακολουθία εξόδου, ακόμα και με την παρουσία αυθαίρετου πλήθους λαθών. Τέτοιοι κώδικες είναι υπολογιστικά φθηνοί και αποδοτικοί σε συγκεκριμένα λάθη bits, ειδικά σε αυτά που προκαλούνται από Γκαουσιανό θόρυβο [IV].



Σχήμα 2-2 - Παράδειγμα συνελικτικού κωδικοποιητή

Το μήκος του καταχωρητή ολίσθησης είναι γνωστό ως μήκος περιορισμού (constraint length) και καθορίζει το μέγιστο αριθμό από διαδοχικά bits που μπορούν να επηρεάσουν την έξοδο. Ο ρυθμός του κώδικα R, ορίζεται ως το πηλίκο του μήκους συμβόλων εισόδου προς το μήκους κωδικοποιημένης εξόδου [IV]:

$$R = \frac{k}{n} \tag{4}$$

2.4.4.5 Αποκωδικοποίηση Viterbi

Αρκετά δημοφιλές τρόπο αποκωδικοποίησης αποτελεί ο λεγόμενος αλγόριθμος ή αποκωδικοποιητής Viterbi. Ο αλγόριθμος προσπαθεί να βρει την πιο πιθανή λαμβανόμενη ακολουθία δεδομένων, διατηρώντας τα τέσσερα πιο πιθανά μονοπάτια. Για κάθε μονοπάτι, υπολογίζεται και διατηρείται η απόσταση Hamming μεταξύ της λαμβανόμενης ακολουθίας και το μονοπάτι. Μετά την λήψη των πρώτων τριών κωδίκων, επιλέγονται τα μονοπάτια με την μικρότερη απόσταση Hamming. Όταν δύο μονοπάτια τα οποία συγχωνεύονται έχουν την ίδια απόσταση Hamming, επιλέγεται συνήθως το πάνω μονοπάτι. Η διαδικασία μπορεί να τερματιστεί σε κάθε βήμα της εκτέλεσής της. Το πιο πιθανό μήνυμα εξόδου είναι πάντοτε αυτό με την μικρότερη απόσταση Hamming [IV].

3. ΠΕΔΙΑ GALOIS

Για την πλήρη κατανόηση των κωδίκων Reed-Solomon απαιτούνται θεωρητικές γνώσεις των πεπερασμένων πεδίων (Finite Fields) ή αλλιώς πεδίων Galois (Galois Fields).

3.1 Στοιχεία Πεδίων Galois

Ένα πεδίο Galois αποτελείται από ένα σύνολο στοιχείων (αριθμών) με καθορισμένες λειτουργίες πρόσθεσης και πολλαπλασιασμού. Τα στοιχεία αυτά είναι βασισμένα σε ένα πρωτόγονο στοιχείο (primitive element), που συνήθως δηλώνεται ως α, και λαμβάνουν τις τιμές:

$$0,\ \alpha^0,\ \alpha^1,\ \alpha^2, \dots, \alpha^{q-2}$$

κατασκευάζοντας έτσι ένα σύνολο από $q = p^m$ στοιχεία, όπου p είναι ένας πρώτος αριθμός που ορίζεται ως χαρακτηριστικό του πεδίου (field characteristic) και m είναι ο βαθμός επέκτασης (extension order) του πεδίου. Ως εκ τούτου, το πεδίο είναι γνωστό ως $GF(p^m)$ [VI][VII].

Η τιμή του α επιλέγεται συνήθως να είναι 2, επειδή οι τιμές μπορούν αντιστοιχιστούν εύκολα σε bits, ωστόσο μπορούν να χρησιμοποιηθούν και άλλες τιμές που όμως πρέπει να αποτελούν ρίζες του πρωτόγονου πολυώνυμου (περισσότερα για αυτό στην Ενότητα 3.5). Οι υψηλότερες δυνάμεις του α αποκτώνται σε κάθε βήμα με το χαρακτηριστικό του πεδίου, α [VII]. Το πώς ακριβώς λειτουργεί η διαδικασία πολλαπλασιασμού θα εξηγηθεί στην Ενότητα 3.6.

3.2 Τρόποι Αναπαράστασης Στοιχείων

Εκτός από την αναπαράσταση τους ως δύναμη του α, τα στοιχεία των πεδίων Galois $GF(2^m)$ μπορούν να εκφραστούν εύκολα ως δυαδικά διανύσματα (binary vectors), καθώς και ως πολυώνυμα (polynomials) ή και θετικοί ακέραιοι (positive integers). Οι αναπαράσταση πολυωνύμου μπορεί να εφαρμοστεί εύκολα για οποιοδήποτε α, ενώ οι άλλες δύο αναπαραστάσεις βρίσκουν συχνότερα εφαρμογή στην περίπτωση που α = 2, δηλ. μόνο στα $GF(2^m)$ [VII].

3.2.1 Αναπαράσταση πολυωνύμου

Κάθε στοιχείο του $GF(p^m)$ μπορεί να αναπαρασταθεί μέσω πολυωνυμικής έκφρασης (polynomial expression) της γενικής μορφής:

$$a_{m-1}x^{m-1} + \cdots + a_1x + a_0$$

όπου στην περίπτωση του $GF(2^m)$ οι συντελεστές a_{m-1} έως a_0 λαμβάνουν τις τιμές 0 ή 1.

Στην γενική περίπτωση όπου το α έχει άλλη τιμή, οι συντελεστές a_i λαμβάνουν τιμές από το πεδίο $GF(p)$. Το $GF(2)$ αποτελείται μόνο από τις τιμές 0 και 1. Έτσι εξηγείται και γιατί στην περίπτωση που α = 2, οι τιμές των συντελεστών είναι 0 ή 1 [VI].

Για παράδειγμα, στην περίπτωση του $GF(2^4)$, τα στοιχεία αναπαρίστανται με το πολυώνυμο:

$$a_3x^3 + a_2x^2 + a_1x + a_0$$

Αυτοί οι συντελεστές μπορούν να αντιστοιχιστούν εύκολα στους δυαδικούς αριθμούς 0000 έως 1111, οι οποίοι στην συνέχεια μπορούν να μετατραπούν στους δεκαδικούς αριθμούς 0 έως 15. Αυτές είναι και οι αναπαραστάσεις που αναλύονται στις επόμενες Ενότητες.

3.2.2 Αναπαράσταση διανύσματος

Από την αναπαράσταση πολυωνύμου, συμπεραίνεται εύκολα πώς οι συντελεστές των στοιχείων $a_{m-1}, \cdots, a_1, a_0$ του $GF(2^m)$ αντιστοιχίζονται στις δυαδικές τιμές 000 ... 0 έως 111 ... 1. Έτσι,

τελικώς κάθε στοιχείο του $GF(2^m)$ μπορεί να αναπαρασταθεί και ως δυαδικό διάνυσμα της μορφής:

$$(a_{m-1}, \cdots, a_1, a_0)$$

Επομένως, κάθε στοιχείο του $GF(2^m)$ μπορεί να αναπαρασταθεί ως διάνυσμα m-διαστάσεων [VI].

Στην γενική περίπτωση του $GF(p^m)$ οι τιμές του διανύσματος δίνονται από την περιοδική ακολουθία:

$$s_{j+m} = s_{j+m-1}p_{j+m-1} + s_{j+m-2}p_{j+m-2} + \cdots + s_j p_j \quad (5)$$

όπου p_i είναι οι συντελεστές του πρωτογόνου πολυωνύμου (με το οποίο ασχολείται η Ενότητα 3.5). Πριν την εκτέλεση του αλγορίθμου, τα αρχικά $m-1$ σε πλήθος s_i πρέπει να αρχικοποιηθούν με τιμές του $GF(p)$. Για παράδειγμα, μπορούν να χρησιμοποιηθούν οι τιμές:

$$s_0 = 1, s_1 = 0, \cdots, s_{m-1} = 0$$

Σύμφωνα με τα παραπάνω, το στοιχείο a^i αναπαρίσταται από το διάνυσμα:

$$(s_i, s_{i+1}, s_{i+2}, \cdots, s_{i+m-1}), \;\; 0 \le i \le 2^m - 2$$

Το στοιχείο 0 αποτελεί διάνυσμα από μηδενικά - $(0,\ 0,\ \cdots, 0)$ [VII].

3.2.3 Αναπαράσταση θετικού ακεραίου

Όπως αναφέρθηκε πρωτύτερα, τα στοιχεία μπορούν να αναπαρασταθούν ως θετικοί ακέραιοι δεκαδικοί αριθμοί μετατρέποντας την δυαδική αναπαράσταση σε δεκαδική. Γενικά, μπορούν να εκφραστούν τα στοιχεία και με χρήση της παρακάτω σχέσης:

$$a^i \to N^{(a^i)} = \log_a(a^i) + 1 = i + 1 \quad (6)$$

Το στοιχείο 0 εκφράζεται ως $N^{(0)} = 0$ [VII].

3.3 Πρόσθεση στα Πεδία Galois

Η πρόσθεση δυο στοιχείων a και b του $GF(p)$ εκτελείται ως:

$$a + b = (a + b)\ mod\ p \quad (7)$$

Επομένως, για την εκτέλεση μιας πρόσθεσης χρησιμοποιείται αριθμητική υπολοίπων (modular arithmetic). Πιο συγκεκριμένα το αποτέλεσμα της πρόσθεσης ισούται με το υπόλοιπο της διαίρεσης του αθροίσματος $a + b$ με το p [VIII].

Το 0 αποτελεί ουδέτερο στοιχείο της πρόσθεσης δηλαδή [I]:

$$a^i + 0 = 0 + a^i = a^i \quad (8)$$

Στα πεδία $GF(p^m)$ εκτελείται και πάλι η ίδια πράξη, αλλά μεταξύ δύο πολυωνύμων. Για την ακρίβεια εφαρμόζεται η πράξη της πρόσθεσης στο $GF(p)$ στους συντελεστές ιδίου βαθμού των πολυωνύμων [VIII]:

$$P_1(x) + P_2(x) = \sum_{i=0}^{n-1} \big((a_i + b_i)\ mod\ p\big)x^i \quad (9)$$

Στην ειδική περίπτωση όπου $p = 2$, δηλ. στα $GF(2^m)$, η πράξη $mod2$ έχει το ίδιο αποτέλεσμα με την πράξη αποκλειστικού-Ή (XOR) [VI]. Έτσι, η πράξη πρόσθεσης στα πεδία $GF(2^m)$ εκφράζεται ως εξής:

$$P_1(x) + P_2(x) = \sum_{i=0}^{n-1} (a_i \oplus b_i)\, x^i \tag{10}$$

Για παράδειγμα, στο πεδίο $GF(16) = GF(2^4)$ η πρόσθεση των στοιχείων $p_1(x) = x^3 + x$ και $p_2(x) = x^3 + x^2 + 1$ μας δίνει $x^2 + x + 1$. Η πράξη μπορεί να εφαρμοστεί, διατυπώνοντας τα στοιχεία σε δυαδική μορφή και εκτελώντας την πράξη XOR ανά bit:

$$1010 + 1101 = 0111$$

Ακόμη, μπορεί να βρεθεί το ίδιο αποτέλεσμα και μέσω της αναπαράστασης θετικού ακεραίου δεκαδικού αριθμού [VI]:

$$(10 + 13) \bmod 16 = 23 \bmod 16 = 7$$

3.4 Αφαίρεση στα Πεδία Galois

Η αφαίρεση είναι το ακριβώς αντίστοιχο με πρόσημο – ανάμεσα στα *a* και *b*:

$$P_1(x) - P_2(x) = \sum_{i=0}^{n-1} (a_i - b_i)\, x^i \tag{11}$$

Αξίζει να σημειωθεί επίσης, πως στην ειδική περίπτωση των $GF(2^m)$ η πράξη της αφαίρεσης είναι ταυτόσημη με την πράξη της πρόσθεσης. Άρα, μπορεί να διατυπωθεί η πράξη πρόσθεσης/αφαίρεσης πολυωνύμων συνολικά ως [VIII]:

$$P_1(x) \pm P_2(x) = \sum_{i=0}^{n-1} (a_i \oplus b_i)\, x^i \tag{12}$$

3.5 Κατασκευή Πεδίων Galois

3.5.1 Πολυώνυμο κατασκευής πεδίου

Ένα πεδίο Galois κατασκευάζεται από το λεγόμενο πολυώνυμο παραγωγής πεδίου (field generator polynomial), γνωστό και ως πρωτόγονο πολυώνυμο (primitive polynomial). Για ένα δεδομένο πεδίο $GF(p^m)$ το πολυώνυμο αυτό κρίνετε απαραίτητο να είναι βαθμού m και ανάγωγο (irreducible), δηλ. μη απλοποιήσιμο πολυώνυμο. Το πολυώνυμο αυτό δεν είναι μοναδικό. Σε πολλά πεδία, μπορούν να υπάρξουν πολλά κατάλληλα πολυώνυμα για τον ρόλο.

Για παράδειγμα στο πεδίο $GF(16)$ ως πολυώνυμο παραγωγής πεδίου p(x) μπορεί να χρησιμοποιηθεί το εξής πολυώνυμο [VI]:

$$\mathrm{p(x)} = x^4 + x + 1$$

3.5.2 Διαδικασία κατασκευής πεδίου

Το πρωτόγονο στοιχείο (primitive element) α αποτελεί ρίζα του πολυωνύμου p(x), και άρα:

$$\mathrm{p}(\alpha) = \alpha^4 + \alpha + 1 = 0 \rightarrow \alpha^4 = \alpha + 1$$

Χρησιμοποιώντας αυτήν την εξίσωση για απλοποίηση μπορεί να κατασκευαστεί πολύ εύκολα το πεδίο $GF(16)$. Σε κάθε βήμα πολλαπλασιάζεται η προηγούμενη τιμή που υπολογίστηκε με το α

και απλοποιούνται τα a^4 που εμφανίζονται, αντικαθιστώντας τα με $a+1$ [VI]. Τα βήματα της διαδικασίας κατασκευής του $GF(16)$ παρουσιάζονται λεπτομερώς στον Πίνακα 3-1.

Μορφή Δείκτη	Πολυωνυμική αναπαράσταση	Δυαδική αναπαράσταση	Δεκαδική αναπαράσταση
0	0	0000	0
a^0	1	0001	1
a^1	a	0010	2
a^2	a^2	0100	4
a^3	a^3	1000	8
a^4	$a+1$	0011	3
a^5	a^2+a	0110	6
a^6	a^3+a^2	1100	12
a^7	a^3+a+1	1011	11
a^8	a^2+1	0101	5
a^9	a^3+a	1010	10
a^{10}	a^2+a+1	0111	7
a^{11}	a^3+a^2+a	1110	14
a^{12}	a^3+a^2+a+1	1111	15
a^{13}	a^3+a^2+1	1101	13
a^{14}	a^3+1	1001	9

Πίνακας 3-1 - Τα στοιχεία του πεδίου GF(16) με p(x) = x4 + x + 1

Από το a^{15} και μετά επαναλαμβάνονται τα προηγούμενα στοιχεία δηλ.

$$a^{15}=a^0,\ a^{16}=a^1,\cdots$$

Στα $GF(2^m)$, μπορεί να διατυπωθεί η διαδικασία κατασκευής αλγοριθμικά ως:

GFEXP[0] ← 1, ByteValue = 1
for i = 1 → $2^m - 1$ **do**
  ByteValue = ByteValue ≪ 1
  if ByteValue[m] =′ 1′**then**
    ByteValue = ByteValue ⊕ p(x)
  end if
  GFEXP[i] = ByteValue
end for

Χρησιμοποιώντας διαφορετικό p(x) κατασκευάζεται φυσικά διαφορετικό πεδίο, εφόσον το a^m αντικαθίσταται με διαφορετικό πολυώνυμο [VI].

3.6 Πολλαπλασιασμός στα Πεδία Galois

Σε πλήρη αντιστοιχία με την πράξη της πρόσθεσης, η πράξη πολλαπλασιασμού αποτελεί και αυτή με τη σειρά της πράξη αριθμητικής υπολοίπων.

3.6.1 Πολλαπλασιασμός στα GF(p)

Στην περίπτωση των $GF(p)$ υπολογίζουμε το «κλασσικό» γινόμενο μεταξύ των δύο στοιχείων και στην συνέχεια το υπόλοιπο της διαίρεσης αυτού του γινομένου με το χαρακτηριστικό του πεδίου p, ώστε το αποτέλεσμα να αποτελεί και αυτό με την σειρά του στοιχείο του $GF(p)$ [VIII].

$$a \times b = (a \times b) \bmod p \tag{13}$$

3.6.2 Πολλαπλασιασμός στα GF(p^m)

Από την άλλη, στην περίπτωση των $GF(p^m)$ η πράξη του πολλαπλασιασμού εκτελείται ως πολλαπλασιασμός μεταξύ πολυωνύμων. Σε αυτήν την πράξη χρησιμοποιούνται δύο ειδών πράξεις υπολοίπων. Μια ειδική πράξη υπολοίπου που διαβεβαιώνει πώς ο βαθμός του προκύπτον πολυωνύμου είναι μικρότερος του n, και μια πράξη αριθμητικής υπολοίπων που εφαρμόζεται στους συντελεστές αυτού του πολυωνύμου [VIII].

Για την πρώτη πράξη απαιτείται το πρωτόγονο πολυώνυμο $p(x)$, πολυώνυμο με το οποίο διαιρείται το αποτέλεσμα του πολλαπλασιασμού μεταξύ δυο πολυωνύμων $p_1(x)$ και $p_2(x)$, εξασφαλίζοντας έτσι πως το αποτέλεσμα είναι βαθμού μικρότερου του n. Πιο συγκεκριμένα, μέσω αυτής της διαίρεσης εξασφαλίζεται πώς το αποτέλεσμα της πράξης θα είναι βαθμού $n-1$ ή μικρότερου. Έπειτα εφαρμόζεται η άλλη πράξη στους συντελεστές αυτού του πολυωνύμου [VIII].

3.6.3 Μια μέθοδος πολλαπλασιασμού

Έναν πιο εύκολο τρόπο εκτέλεσης πολλαπλασιασμού αποτελεί η αντικατάσταση κάθε εμφάνισης του x^m χρησιμοποιώντας το πρωτόγονο πολυώνυμο $p(x)$. Με αυτόν τον τρόπο, μπορεί να αντιστοιχιστεί πολύ εύκολα οποιοδήποτε πολυώνυμο με βαθμό μεγαλύτερο του $n-1$ σε πολυώνυμο με βαθμό του πολύ $n-1$ [VIII].

Η μέθοδος αυτή συνοψίζεται ως εξής [VIII]:

1) Αντικατέστησε τους όρους $x^i, \forall i \geq n$, με την ταυτότητα $x^m \times x^{i-m}$
2) Αντικατέστησε τον όρο x^m χρησιμοποιώντας το πρωτόγονο πολυώνυμο
3) Απλοποίησε το πολυώνυμο
4) Επανέλαβε μέχρις ότου κανένας όρος να μην έχει βαθμό μεγαλύτερο του $m-1$

Για παράδειγμα, έστω πως πολλαπλασιάζουμε τα στοιχεία $2x+1$ και $x+1$ του πεδίου $GF(3^2)$, χρησιμοποιώντας το πρωτόγονο πολυώνυμο:

$$p(x) = x^2 + 2x + 1 \rightarrow x^2 = 2x + 1$$

Σύμφωνα με την μέθοδο αυτή, τα βήματα του πολλαπλασιασμού είναι τα εξής [VIII]:

	$(x+1) \times (2x+1) = (2x^2 + 3x + 1) \bmod 3$
1	$2x^2 + 1$
2	$2 \times (2x+1) + 1$
3	$4x + 2 + 1 = (4x+3) \bmod 3$
4	x

3.7 Διαίρεση στα Πεδία Galois

Η πράξη της διαίρεσης μεταξύ δύο στοιχείων a^i και a^j πραγματοποιείται συνήθως ως πολλαπλασιασμός με τον αντίστροφο πολλαπλασιασμού του διαιρέτη $a^{-j} = a^{2m-1-i}$. Στην περίπτωση όπου j = 0, ο αντίστροφος είναι το στοιχείο 1. Το στοιχείο 0 δεν έχει αντίστροφο [VII].

Επομένως η πράξη διαίρεσης διατυπώνεται ως εξής:

$$a^i \div a^j = a^i \times a^{-j} \tag{14}$$

3.8 Αντίστροφοι πρόσθεσης και πολλαπλασιασμού

Για κάθε στοιχείο a^i του πεδίου $GF(2^m)$ ο αντίστροφος της πράξης της πρόσθεσης ορίζεται ως:

$$a^i = -a^i \tag{15}$$

Ο αντίστροφος της πράξης πολλαπλασιασμού ορίζεται ως [VII]:

$$a^{-i} = \begin{cases} a^{2^m-1-i}, & 1 \le i \le 2^m - 2 \\ 1, & i = 0 \end{cases} \tag{16}$$

3.9 Το πεδίο Galois GF(2^8)

Η κωδικοποίηση Reed-Solomon $RS(255, 255-2t)$ που χρησιμοποιείται στην υλοποίηση του αποκωδικοποιητή εφαρμόζεται στο πεδίο $GF(256) = GF(2^8)$. Δηλαδή, τα p και m λαμβάνουν τις τιμές p = 2 και m = 8.

Ως πρωτόγονο πολυώνυμο χρησιμοποιείται το πολυώνυμο:

$$p(x) = x\text{^}8 + x\text{^}4 + x\text{^}3 + x\text{^}2 + 1$$

Ο Πίνακας 3-2 περιέχει ορισμένα από τα στοιχεία του πεδίο που κατασκευάζεται με την χρήση αυτού του πολυωνύμου [X].

Μορφή Δείκτη	Πολυωνυμική αναπαρ.	Δυαδική αναπαρ.	Δεκαδική αναπαρ.
0	0	00000000	0
a^0	1	00000001	1
a^1	a	00000010	2
a^2	a^2	00000100	4
a^3	a^3	00001000	8
a^4	a^4	00010000	16
a^5	a^5	00100000	32
a^6	a^6	01000000	64
a^7	a^7	10000000	128
a^8	$a^4 + a^3 + a^2 + 1$	00011101	29
a^9	$a^5 + a^4 + a^3 + a$	00111010	58

Πίνακας 3-2 - Ορισμένα στοιχεία του πεδίου GF(2^8) με $p(x) = x^8 + x^4 + x^3 + x^2 + 1$

Στο Παράρτημα Α μπορεί να βρεθεί πίνακας όλων των στοιχείων του $GF(2^8)$ για την περίπτωση του επιλεγμένου πρωτόγονου πολυωνύμου $p(x)$ [X].

4. ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ ΠΡΑΞΕΩΝ ΣΤΑ GF(2^m)

4.1 Αθροιστής

Στην ειδική περίπτωση των $GF(2^m)$ οι αθροιστές (και αφαιρέτες) υλοποιούνται ως πύλες XOR. Όπως αναφέρθηκε και στην Ενότητα 3.3 η πράξη XOR εκτελείται ανάμεσα σε κάθε bit (ή συντελεστή πολυωνύμου) των δύο στοιχείων που συμμετέχουν στην πράξη αυτή. Έτσι, τελικώς, απαιτούνται m πύλες XOR για κάθε πράξη πρόσθεσης/αφαίρεσης.

Σχήμα 4-1 - Αρχιτεκτονική Αθροιστή στα πεδία GF(2^m)

4.2 Πολλαπλασιαστής

GF-Πολλαπλασιαστής μπορεί να υλοποιηθεί με ποικίλους τρόπους. Ωστόσο σύμφωνα με την βιβλιογραφία, πιο αποτελεσματικός κρίνετε ο γενικός πολλαπλασιαστής και οι διάφορες τροποποιήσεις της αρχιτεκτονικής του.

4.2.1 Γενικός πολλαπλασιασμός

Έστω δύο στοιχεία A και B του $GF(2^m)$. Χρησιμοποιώντας την αναπαράσταση πολυωνύμου που αναφέρθηκε στην Ενότητα 3.2, τα στοιχεία αυτά αναπαρίστανται με τον εξής τρόπο:

$$A = a_0 + a_1 a + \cdots + a_{m-1} a^{m-1} = \sum_{i=0}^{m-1} a_i a^i$$

$$B = b_0 + b_1 a + \cdots + b_{m-1} a^{m-1} = \sum_{i=0}^{m-1} b_i a^i$$

Το γινόμενό τους, C, εκφράζεται ως εξής:

$$C = A \cdot B = A \cdot \sum_{i=0}^{m-1} b_i \, a^i = \sum_{i=0}^{m-1} b_i \, (A a^i) \qquad (17)$$

Αυτή η τελευταία εξίσωση διατυπώνεται εύκολα ως γραμμικό σύστημα στην μορφή πινάκων. Λύνοντας αυτό το σύστημα υπολογίζονται οι m-συντελεστές του πολυωνύμου C, c_i [IX][XI].

Σε μορφή πινάκων, το σύστημα διατυπώνεται ως εξής:

$$[c_0 \;\; c_1 \cdots c_{m-1}] = [b_0 \;\; b_1 \cdots b_{m-1}] \begin{bmatrix} A \\ A\,a \\ A\,a^2 \\ \vdots \\ A\,a^{m-1} \end{bmatrix} \tag{18}$$

Στην περίπτωση του $GF(2^4)$ χρησιμοποιώντας ως πρωτόγονο πολυώνυμο το:

$$p(x) = x^4 + x^3 + x^2 + x + 1$$

προκύπτουν τελικώς οι εξής συντελεστές:

$$\begin{aligned} c_0 &= t_0 + t_4 + t_5 \\ c_1 &= t_1 + t_4 + t_6 \\ c_2 &= t_2 + t_4 \\ c_3 &= t_3 + t_4 \end{aligned} \quad \text{, όπου:} \quad \begin{aligned} t_0 &= a_0 b_0 \\ t_1 &= a_0 b_1 + a_1 b_0 \\ t_2 &= a_0 b_2 + a_1 b_1 + a_2 b_0 \\ t_3 &= a_0 b_3 + a_1 b_2 + a_2 b_1 + a_3 b_0 \\ t_4 &= a_1 b_3 + a_2 b_2 + a_3 b_1 \\ t_5 &= a_2 b_3 + a_3 b_2 \\ t_6 &= a_3 b_3 \end{aligned}$$

Ο αριθμός των λογικών πυλών που απαιτούνται για μια τέτοιου είδους υλοποίηση είναι m^2 πύλες AND και $(m-1)^2$ πύλες XOR για τον υπολογισμό των t_i. Επιπλέον απαιτούνται και επιπρόσθετες πύλες XOR οι οποίες εξαρτώνται από την επιλογή του πρωτόγονου πολυωνύμου $p(x)$ [IX][XI]. Το κύκλωμα γενικού πολλαπλασιαστή $GF(2^8)$ που χρησιμοποιείται στην υλοποίηση του αποκωδικοποιητή μπορεί να βρεθεί με την μορφή κώδικα υλικού VHDL στο Παράρτημα Β [IX][XI].

4.2.2 Πολλαπλασιασμός με σταθερά

Στους αλγορίθμους κωδικοποίησης και αποκωδικοποίησης εμφανίζονται πολύ συχνά πολλαπλασιασμοί με σταθερές τιμές $a, a^2, \ldots, \alpha^v$. Σε τέτοιες περιπτώσεις η χρήση ενός γενικού πολλαπλασιαστή σαν αυτόν που αναλύθηκε στην προηγούμενη Ενότητα επιφέρει πολύ μεγάλη απαίτηση σε υλικό που δεν είναι απαραίτητη στην περίπτωση σταθερού τελούμενου. Όταν ένα από τα τελούμενα του πολλαπλασιασμού είναι σταθερά χρησιμοποιείται πολλαπλασιαστής του τύπου μεταβλητή-σταθερά (variable-constant). [IX]

Έστω δύο στοιχεία a και b του $GF(2^m)$ όπου το b είναι σταθερά. Οι συντελεστές του γινομένου c υπολογίζονται με τον εξής τρόπο:

$$\begin{aligned} c = a \cdot b &= (a_0 + a_1 a + \cdots + a_{m-1} a^{m-1}) \cdot b \\ &= a_0 \cdot (\alpha^0 \cdot b) + a_1 (\alpha \cdot b) + \cdots + a_{m-1} (a^{m-1} \cdot b) \end{aligned} \tag{19}$$

Τα γινόμενα $b_i = a^i b, \;\; 0 \le i < m$, μπορούν να προϋπολογιστούν και να αποθηκευτούν σε δυαδικό πίνακα B. Οι συντελεστές του πολυωνύμου C, c_j, υπολογίζονται πολλαπλασιάζοντας την γραμμή a με κάθε στήλη j του πίνακα B [IX].

Το εσωτερικό γινόμενο γραμμής-στήλης ορίζεται ως εξής:

$$c = a \cdot b = a \cdot B = [a_0 \quad a_1 \quad \cdots \quad a_{m-1}] \begin{bmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,m-1} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m-1,0} & b_{m-1,1} & \cdots & b_{m-1,m-1} \end{bmatrix} \tag{20}$$

Επομένως, οι συντελεστές c_j υπολογίζονται από το άθροισμα:

$$c_j = \sum_{i=0}^{m-1} a_i \cdot b_{i,j} = \sum_{\substack{i=0 \\ b_{i,j}=1}}^{m-1} a_i \tag{21}$$

Αναφέρθηκε στην Ενότητα 3 πως στην αριθμητική $GF(2)$ το γινόμενο αποτελεί AND, ενώ το άθροισμα XOR. Ως εκ τούτου, ένας πολλαπλασιαστής σταθεράς (scaler) χρειάζεται $\leq m(m-1)$ πύλες XOR 2-εισόδων. Ένας τυπικός πολλαπλασιαστής αυτού του τύπου απαιτεί συνήθως $\approx \frac{1}{2}m^2$ πύλες XOR. Φυσικά μπορούν να απαλειφθούν και κοινές υποεκφράσεις που τυχόν υπάρχουν μεταξύ των συναρτήσεων εξόδου των συντελεστών c_j και να ελαττωθεί έτσι ο αριθμός των απαιτούμενων πυλών ακόμα περισσότερο. Γενικά η εύρεση του μικρότερου δυνατού κυκλώματος αποτελεί πρόβλημα NP-complete [IX].

Έστω πως απαιτείται να κατασκευαστεί πολλαπλασιαστής τύπου μεταβλητής-σταθεράς με σταθερά $b = [110001]$ στο πεδίο $GF(2^6)$ με πρωτόγονο πολυώνυμο $p(x) = x^6 + x + 1$. Πρώτη γραμμή του πίνακα B αποτελεί το ίδιο το b. Οι υπόλοιπες γραμμές του πίνακα B αποκτώνται εύκολα από δεξιά ολίσθηση της προηγούμενης γραμμής. Κάθε φορά που κάποιο '1' «φεύγει» από τα δεξιά ακολουθεί πρόσθεση (XOR) με πρότυπο ανατροφοδότησης (feedback pattern), που στην περίπτωσή μας είναι το $[110000]$. Με αυτόν τον τρόπο κατασκευάζεται ο παρακάτω πίνακας B, ενώ υπολογίζονται και οι αντίστοιχοι συντελεστές του γινομένου c_j [IX]:

$$B = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = a_0 \oplus \alpha_1 \oplus \alpha_5 \\ c_1 = a_0 \oplus \alpha_2 \oplus \alpha_5 \\ c_2 = \alpha_1 \oplus \alpha_3 \\ c_3 = \alpha_2 \oplus \alpha_4 \\ c_4 = \alpha_3 \oplus \alpha_5 \\ c_5 = \alpha_0 \oplus \alpha_4 \end{array}$$

Εφόσον το γινόμενο εκτελείται ανά στήλη j, εμφανίζονται στις εξισώσεις μόνο οι συντελεστές του α, a_i, στους οποίους το αντίστοιχο $b_{i,j}$ δεν είναι μηδενικό. Ο αριθμός των απαιτούμενων πυλών XOR στην περίπτωση του πολλαπλασιαστή αυτού είναι 8, ενώ αν χρησιμοποιούνταν γενικός πολλαπλασιαστής θα χρειάζονταν $(m-1)^2 = (6-1)^2 = 25$ πύλες XOR και επιπλέον $m^2 = 6^2 = 36$ πύλες AND. Στο Παράρτημα Γ αποδεικνύονται οι απαιτούμενες συναρτήσεις ορισμένων πολλαπλασιαστών του τύπου μεταβλητής-σταθεράς για το πεδίο $GF(2^8)$ [IX].

4.2.3 Πολλαπλασιασμός βασισμένος σε LUT's

Αρκετά ενδιαφέρουσα αρχιτεκτονική πολλαπλασιασμού είναι αυτή που χρησιμοποιεί μνήμη ROM με προ-υπολογισμένα αποτελέσματα πολλαπλασιασμού. Ωστόσο, το μεγάλο πρόβλημα αυτής της αρχιτεκτονικής αποτελεί το μέγεθός της. Ο αριθμός των στοιχείων που πρέπει να αποθηκευτούν σε ένα πλήρως βελτιστοποιημένο σε κόστος LUT είναι [VII]:

$$N_{LUT} = 2^{2m-1} - 3 \cdot 2^{m-1} + 1 \tag{22}$$

Ο αλγόριθμος πολλαπλασιασμού έχει ως εξής [VII]:

(1) Αν ένα από τα στοιχεία 0, επέστρεψε 0.
(2) Αν ένα από τα στοιχεία 1, επέστρεψε το άλλο στοιχείο.
(3) Αλλιώς γενικά στην πράξη $a^i \times a^j$ επέστρεψε το στοιχείο $\begin{cases} LUT[a^i][a^j], & i \geq j \\ LUT[a^j][a^i], & i < j \end{cases}$

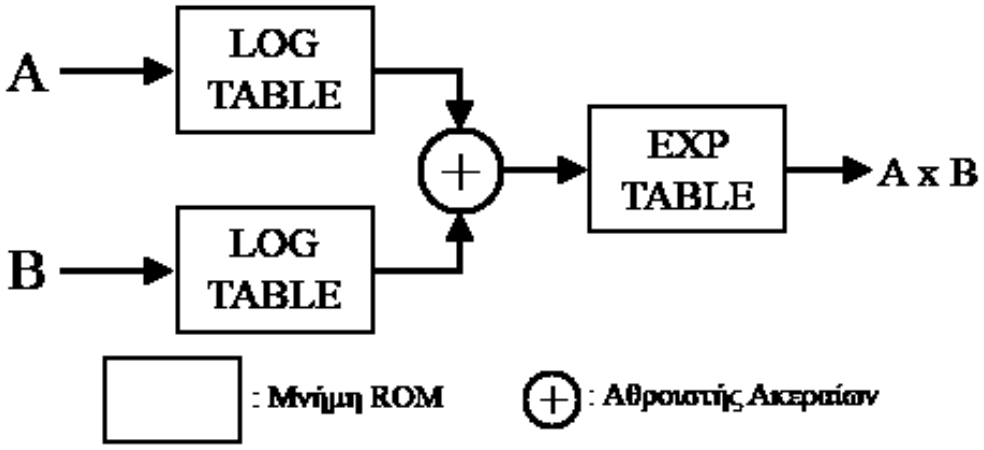
4.2.4 Πολλαπλασιασμός στο λογισμικό

Μια προσέγγιση που χρησιμοποιείται πολύ συχνά στο λογισμικό βασίζεται σε πράξεις λογαρίθμων. Το γινόμενο δυο στοιχείων σε μορφή δεικτών είναι ίσο με το στοιχείο που έχει εκθέτη το άθροισμα των εκθετών τους modulo $2^m - 1$. Μαθηματικά, ορίζεται με τον εξής τρόπο:

$$a^i \times a^j = a^{(i+j)\ mod\ 2^{m-1}} \tag{23}$$

Επομένως, προ-υπολογίζοντας πίνακα λογαρίθμων με μήκος $2^m - 1$ για την εύρεση του δείκτη κάθε στοιχείου και υπολογίζοντας στην συνέχεια το αποτέλεσμα της πρόσθεσής αυτών των δεικτών, χρησιμοποιώντας αυτό το αποτέλεσμα ως δείκτη σε προ-υπολογισμένο πίνακα εκθετών μήκους $2 \cdot (2^m - 1)$, υπολογίζεται πολύ εύκολα και γρήγορα το αποτέλεσμα του πολλαπλασιασμού σε λογισμικό. Φυσικά μια τέτοια υλοποίηση είναι πολύ ακριβή στο υλικό καθώς απαιτούνται δύο πίνακες λογαρίθμων και ένας πίνακας εκθετών, δηλ. $4 \cdot (2^m - 1)$ κελία [VI].

Σχηματικά μπορεί να διατυπωθεί ο πολλαπλασιασμός αυτός ως εξής:



Σχήμα 4-2 - Αρχιτεκτονική Πολλαπλασιασμού λογισμικού στα πεδία GF(2^m)

4.3 Διαιρέτης

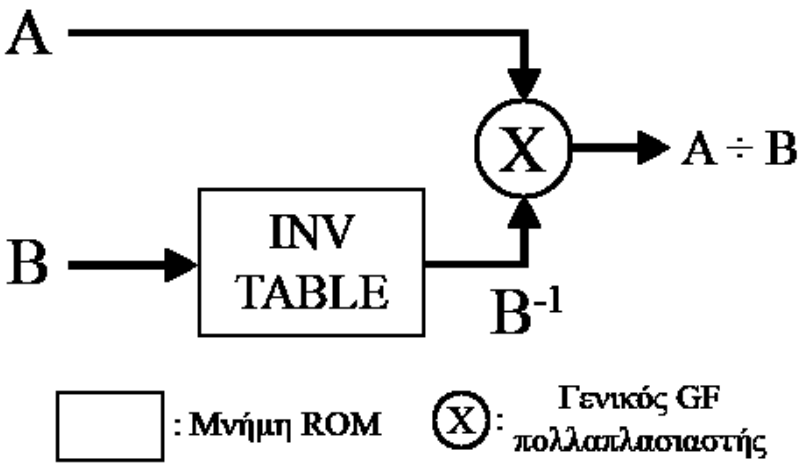
4.3.1 Διαίρεση βασισμένη στον πολλαπλασιασμό με αντίστροφο

Όπως αναφέρθηκε και στην Ενότητα 3.7, δεν υλοποιείται κύκλωμα διαίρεσης, καθώς έχει μεγάλο κόστος σε υλικό και υψηλή πολυπλοκότητα που συνεπάγεται μεγάλη καθυστέρηση. Αντί αυτού υλοποιείται η διαίρεση ως πολλαπλασιασμός με τον αντίστροφο του διαιρέτη. Ο αντίστροφος αυτός υπολογίζεται μέσα από LUT ή ROM με μέγεθος $2^m \times m$ bit.

Στην περίπτωση του πεδίου $GF(2^8)$ ένας τέτοιος πίνακας κοστίζει περίπου όσο και 256 πύλες. Για το ίδιο πεδίο μπορεί να χρησιμοποιηθεί για τον υπολογισμό του αντίστροφου και γενικός πολλαπλασιαστής που κοστίζει περίπου 260 πύλες. Έτσι, προτιμάται μνήμη ROM με περίπου ίδιο κόστος σε υλικό, η οποία επιτυγχάνει πιο βέλτιστη ανταπόκριση [IX].

Όσο μεγαλώνει η τιμή του εκθέτη m η χρήση LUT γίνετε όλο και πιο ανέφικτη. Για παράδειγμα ήδη στα $GF(2^{10})$ η υλοποίηση LUT απαιτεί ισοδύναμο αριθμό 1280 πύλες, ενώ ο γενικός πολλαπλασιασμός περίπου 400 [IX].

Στο Σχήμα 4-3 παρουσιάζεται η πιο συνηθισμένη αρχιτεκτονική υλοποίησης διαίρεσης που χρησιμοποιεί κύκλωμα πολλαπλασιασμού και LUT αντίστροφου για τον υπολογισμό του αντίστροφου του διαιρέτη.



Σχήμα 4-3 - Αρχιτεκτονική Διαίρεσης με χρήση πίνακα αντιστρόφου στα πεδία GF(2^m)

4.3.2 Διαίρεση στο λογισμικό

Η τεχνική λογαρίθμων του πολλαπλασιασμού μπορεί να χρησιμοποιηθεί και στην περίπτωση της πράξης της διαίρεσης. Το πηλίκο δυο στοιχείων σε μορφή δεικτών είναι ίσο με το στοιχείο που έχει εκθέτη την αφαίρεση των εκθετών τους modulo $2^m - 1$. Μαθηματικά διατυπώνεται αυτό ως εξής [VI]:

$$a^i \div a^j = a^{(i-j)\ mod\ 2^m-1} \tag{24}$$

Η ανάγκη σε υλικό είναι σχεδόν ίδια με αυτήν του πολλαπλασιασμού, μόνο που αυτήν την φορά δεν απαιτείται μόνο ένας αθροιστής, αλλά και επιπρόσθετος αφαιρέτης. Δηλαδή, η υλοποίηση του διαιρέτη είναι και πάλι μη αποτελεσματική στην περίπτωση του υλικού [VI]. Στο Σχήμα 4-4 παρουσιάζεται η αρχιτεκτονική υλοποίησης διαιρέτη στο λογισμικό.

Σχήμα 4-4 - Αρχιτεκτονική Διαίρεσης Λογισμικού στα πεδία GF(2^m)

5. ΚΩΔΙΚΟΠΟΙΗΣΗ REED-SOLOMON

5.1 Εισαγωγή στους κώδικες Reed-Solomon

Οι κώδικες Reed-Solomon αποτελούν κώδικες διόρθωσης λαθών. Όπως πρωτοαναφέρθηκε και στην Ενότητα 2.4, τέτοιου είδους κώδικες χρησιμοποιούνται για την ανίχνευση και διόρθωση των λαθών που συμβαίνουν στην μετάδοση της πληροφορίας. Επιπρόσθετα, αξίζει να σημειωθεί και πως οι κώδικες Reed-Solomon αποτελούν [VII]:

- γραμμικούς κώδικες (linear code) - κάθε γραμμικός συνδυασμός μεταξύ δύο κωδικών λέξεων (codewords) αποτελεί και πάλι κωδική λέξη.
- κυκλικούς κώδικες (cyclic code) - κάθε κυκλική ολίσθηση μιας κωδικής λέξης αποτελεί και αυτή με την σειρά της κωδική λέξη.
- μπλοκ-κώδικες (block codes) – η ακολουθία δεδομένων χωρίζεται σε μπλοκ ίσου μήκους που το καθένα από αυτά είναι κωδικοποιημένο ανεξάρτητα από τα άλλα μπλοκ.
- μη-δυαδικούς κώδικες (non-binary code) – τα κωδικά σύμβολα δεν είναι δυαδικά 0 ή 1.

Τα σύμβολα των κωδικών λέξεων αποτελούν στοιχεία του πεπερασμένου πεδίου $GF(q)$, το οποίο χρησιμοποιείται στην διαδικασία κατασκευής του κώδικα. Κάθε κώδικας Reed-Solomon δηλώνεται ως $RS(n,k)$ και έχει τις εξής ιδιότητες [VII]:

- αποτελείται από n κωδικά σύμβολα
- κωδικοποιεί k στοιχεία πληροφορίας
- περιέχει $r = n - k$ πλεονάζοντα στοιχεία ελέγχου
- διορθώνει το πολύ $t = \frac{r}{2}$ λάθη (άρα $2t = n - k$)
- η ελάχιστη απόσταση Hamming του είναι $d_{min} = r + 1$

Οι κώδικες Reed-Solomon μπορούν να διορθώσουν κάθε είδους λάθος. Ο μόνος περιορισμός τους είναι ο αριθμός των κωδικών συμβόλων όπου παρουσιάζονται λάθη, ο οποίος πρέπει να είναι μικρότερος ή ίσος του t. Στην περίπτωση που ο κώδικας κατασκευάζεται για πεδίο $GF(2^m)$, ο κώδικας μπορεί να δηλωθεί και ως [VII]:

$$RS(2^m - 1, 2^m - 1 - 2t)$$

5.2 Επεξήγηση αποδοτικότητας σε σφάλματα ριπών

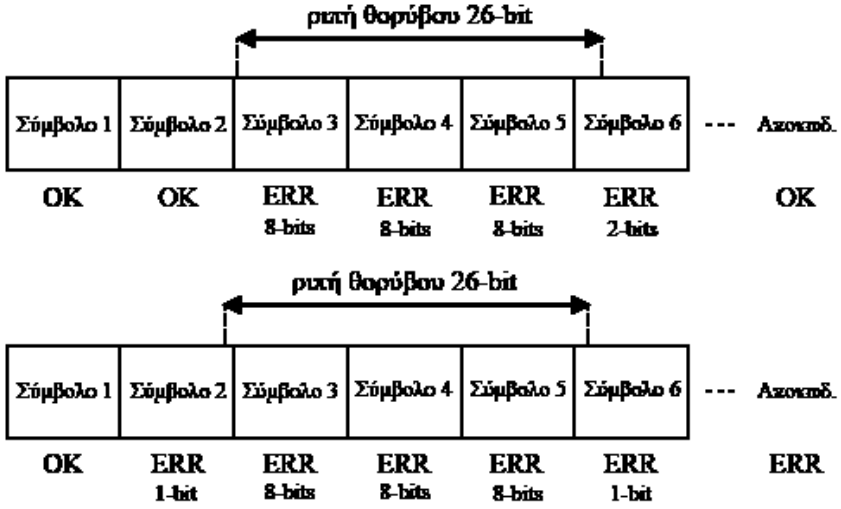
Στην Ενότητα 2.4 αναφέρθηκε σύντομα πως οι κώδικες Reed-Solomon είναι εξαιρετικά αποδοτικοί και αποτελεσματικοί σε σφάλματα που προκαλούνται από ριπές θορύβου. Σε αυτήν την Ενότητα εξηγείται γιατί συμβαίνει αυτό.

Οι κώδικες Reed-Solomon μπορούν να διορθώσουν μέχρι t λάθη συμβόλων, ανεξάρτητα από την «ζημία» που προκαλείται σε καθένα από αυτά τα σύμβολα. Δεν έχει σημασία αν επηρεάστηκε ολόκληρο το σύμβολο, ένα byte ή και ένα bit του κωδικού συμβόλου. Εφόσον κάθε σύμβολο έχει μήκος m-bits, η παρουσία ριπής θορύβου θα προκαλέσει σφάλματα σε ένα αριθμό διαδοχικών bits. Αυτά τα bits ανήκουν σε κάποιο πλήθος διαδοχικών συμβόλων και επομένως μόνο αυτά επηρεάζονται. Τα υπόλοιπα σύμβολα παραμένουν ανεπηρέαστα [XIV].

5.2.1 Παράδειγμα ριπής θορύβου

Για παράδειγμα, έστω πως τα σύμβολα ενός κώδικα αποτελούνται από $m = 8$ bits και πως ο κώδικας αυτός έχει δυνατότητα διόρθωσης $t = 4$ λαθών. Ο κώδικας Reed-Solomon που αντιστοιχεί σε αυτά τα κριτήρια δηλώνεται ως $RS(255, 247)$. Έστω πως παρουσιάζεται θόρυβος

ριπής σε 26 διαδοχικά bits επηρεάζοντας ένα μπλοκ πληροφορίας κατά την μετάδοση [XIV]. Ο θόρυβος αυτός μπορεί να επηρεάσει τα κωδικά σύμβολα με ποικίλους τρόπους, κάποιοι από τους οποίους παρουσιάζονται στο Σχήμα 5-1.



Σχήμα 5-1 - Επηρεασμός συμβόλων RS(255,247) από θόρυβο ριπής 26-bit

Όπως διαπιστώνεται από τα δύο αυτά παραδείγματα, δεν έχει σημασία ποια ή πόσα από τα m bit του κάθε συμβόλου επηρεάζονται. Στην πρώτη περίπτωση που επηρεάζονται 2 από τα 8 bits του 6ου συμβόλου θεωρείται πως το συνολικό σύμβολο παρουσιάζει σφάλματα. Ωστόσο, εφόσον ο συνολικός αριθμός συμβόλων με λάθη είναι ίσος με 4 ($4 \leq 4$), και θεωρώντας επίσης πως δεν παρουσιάζονται σφάλματα στα υπόλοιπα σύμβολα, η αποκωδικοποίηση εκτελείται επιτυχώς. Στην δεύτερη περίπτωση όμως επηρεάζεται ένα bit από το 2ο και ένα bit από το 6ο σύμβολο, εμφανίζοντας στον αποκωδικοποιητή λάθη σε 5 σύμβολα και καθιστώντας έτσι την διόρθωση αυτών των λαθών ανέφικτη. Φυσικά αν τα λάθη αυτά παρουσιαστούν σε τυχαία bits που ανήκουν σε διάφορα σύμβολα της κωδικής λέξης, αυξάνεται κατά πολύ η πιθανότητα επηρεασμού περισσότερων από $t = 4$ συμβόλων της κωδικής λέξης. Τελικώς, οι κώδικες Reed-Solomon αποτυγχάνουν να διορθώσουν τα λάθη που προκλήθηκαν στην παρουσία τυχαίου θορύβου. Βασικό συμπέρασμα είναι το επομένως πως οι κώδικες Reed-Solomon παρουσιάζουν μεγάλη αποδοτικότητα μόνο έναντι ριπών θορύβου [XIV].

5.3 Διαδικασία κωδικοποίησης

5.3.1 Πολυώνυμο μηνύματος

Τα k στοιχεία πληροφορίας που αποτελούν το μήνυμα προς κωδικοποίηση σε μορφή ενός μπλοκ μπορούν να αναπαρασταθούν ως πολυώνυμο $M(x)$ με βαθμό $k-1$ με τον εξής τρόπο:

$$M(x) = M_{k-1}x^{k-1} + \cdots + M_1 x + M_0$$

όπου καθένας από τους συντελεστές $M_{k-1}, \cdots, M_1, M_0$ αποτελεί m-bit σύμβολο του $GF(2^m)$ [VI].

5.3.2 Σχηματισμός κωδικοποιημένου μηνύματος

Για την κωδικοποίηση του μηνύματος το πολυώνυμο μηνύματος πολλαπλασιάζεται αρχικά με $x^{n-k} = x^{2t}$ και διαιρείται με το πολυώνυμο παραγωγής πεδίου $g(x)$. Η διαίρεση αυτή παράγει

πολυώνυμο πηλίκου $q(x)$ και πολυώνυμο υπολοίπου $r(x)$, όπου για το $r(x)$ ο βαθμός πολυωνύμου είναι το πολύ $n-k-1$.

Επομένως:

$$\frac{M(x) \times x^{n-k}}{g(x)} = q(x) + \frac{r(x)}{g(x)} \tag{25}$$

Υπολογίζοντας το πολυώνυμο $r(x)$, το προς μετάδοση κωδικοποιημένο μήνυμα $T(x)$ κατασκευάζεται συνδυάζοντας τα $M(x)$ και $r(x)$ με τον εξής τρόπο:

$$\begin{aligned} T(x) &= M(x) \times x^{n-k} + r(x) \\ &= M_{k-1}x^{n-1} + \cdots + M_0 x^{n-k} + r_{n-k-1}x^{n-k-1} + \cdots + r_0 \end{aligned} \tag{26}$$

Με αυτόν τον τρόπο το κωδικοποιημένο μήνυμα που παράγεται βρίσκεται πια στην σωστή μορφοποίηση. Ο αριθμός των συμβόλων είναι δηλαδή $n = k + 2t$, όπου $2t$ είναι ο αριθμός των πλεονάζων στοιχείων ελέγχου [VI].

5.3.3 Βάση της διόρθωσης λαθών

Προσθέτοντας το υπόλοιπο $r(x)$ εξασφαλίζεται πως το πολυώνυμο του κωδικοποιημένου μηνύματος είναι πάντοτε διαιρέσιμο με το πολυώνυμο παραγωγής πεδίου $g(x)$ χωρίς να αφήσει υπόλοιπο (τέλεια διαίρεση). Ο πολλαπλασιασμός της εξίσωσης διαίρεσης του $M(x)$ με $g(x)$ δίνει:

$$M(x) \times x^{n-k} = g(x) \times q(x) + r(x)$$

ή (27)

$$M(x) \times x^{n-k} + r(x) = g(x) \times q(x)$$

Παρατηρείται πως το αριστερό μέρος αποτελεί το κωδικοποιημένο μήνυμα μετάδοσης $T(x)$. Από την άλλη, το δεξιό μέρος έχει παράγοντα το $g(x)$, πολυώνυμο που έχει επιλεχθεί με συγκεκριμένο τρόπο, ώστε να αποτελείται από ένα αριθμό παραγόντων (factors). Ο καθένας από τους παράγοντες του $g(x)$ αποτελεί και παράγοντα του $T(x)$ και έτσι η διαίρεση με καθένα από αυτούς τους παράγοντες αποτελεί τέλεια διαίρεση. Ως εκ τούτου, κάθε φορά που το κριτήριο αυτό δεν αληθεύει για το κωδικοποιημένο μήνυμα που λαμβάνεται, είναι σίγουρο πως προκλήθηκαν ένα ή περισσότερα λάθη στο μήνυμα κάτα την μετάδοσή του [VI].

5.3.4 Παράδειγμα κωδικοποίησης

Έστω μήνυμα προς μετάδοση που αποτελείται από 11 σύμβολα των 4-bit. Το μήνυμα αυτό κωδικοποιείται με κώδικα $RS(15, 11)$. Έστω ότι το μήνυμα αυτό αποτελείται από τους αριθμούς 1 ως 11. Το πολυώνυμο μηνύματος $M(x)$ είναι το εξής:

$$M(x) = x^{10} + 2x^9 + 3x^8 + 4x^7 + 5x^6 + 6x^5 + 7x^4 + 8x^3 + 9x^2 + 10x + 11$$

Όπως αναφέρθηκε πρωτύτερα, αρχικά πραγματοποιείται πολλαπλασιασμός του $M(x)$ με $x^{n-k} = x^{15-11} = x^4$. Ο πολλαπλασιασμός αυτός παράγει το πολυώνυμο [VI]:

$$x^{14} + 2x^{13} + 3x^{12} + 4x^{11} + 5x^{10} + 6x^9 + 7x^8 + 8x^7 + 9x^6 + 10x^5 + 11x^4$$

Έτσι, ελευθερώνεται χώρος για $2t = 4$ πλεονάζοντα σύμβολα ελέγχου. Σύμφωνα με τον 2ο τύπο της Ενότητας 5.3.2, ακολουθεί διαίρεση αυτού του πολυωνύμου με το πολυώνυμο παραγωγής του πεδίου $GF(2^4)$. Στη περίπτωση του πεδίου $GF(2^4)$ ένα πολυώνυμο $g(x)$ είναι το:

$$g(x) = x^4 + 15x^3 + 3x^2 + x + 12$$

Η κλασσική Πολυωνυμική διαίρεση μεταξύ των πολυωνύμων παρουσιάζεται στο Σχήμα 5-2, όπου το πολυώνυμο υπολοίπου υπολογίζεται πως είναι [VI].:

$$r(x) = 3x^3 + 3x^2 + 12x + 12$$

	x^{14}	x^{13}	x^{12}	x^{11}	x^{10}	x^9	x^8	x^7	x^6	x^5	x^4	x^3	x^2	x^1	x^0
	1	2	3	4	5	6	7	8	9	10	11	0	0	0	0
$\times x^{10}$	1	15	3	1	12										
		13	0	5	9	6									
$\times 13x^9$		13	7	4	13	3									
			7	1	4	5	7								
$\times 7x^8$			7	11	9	7	2								
				10	13	2	5	8							
$\times 10x^7$				10	12	13	10	1							
					1	15	15	9	9						
$\times 1x^6$					1	15	3	1	12						
						0	12	8	5	10					
$\times 0x^5$						0	0	0	0	0					
							12	8	5	10	11				
$\times 12x^4$							12	8	7	12	15				
								0	2	6	4	0			
$\times 0x^3$								0	0	0	0	0			
									2	6	4	0	0		
$\times 2x^2$									2	13	6	2	11		
										11	2	2	11	0	
$\times 11x$										11	3	14	11	13	
											1	12	0	13	0
$\times 1$											1	15	3	1	12
												3	3	12	12

Σχήμα 5-2 - Παράδειγμα πολυωνυμικής διαίρεσης στην περίπτωση του GF(2^4) [VI]

Επομένως, το τελικό κωδικοποιημένο μήνυμα είναι το εξής [VI]:

$$T(x) = x^{14} + 2x^{13} + 3x^{12} + 4x^{11} + 5x^{10} + 6x^9 + 7x^8 + 8x^7$$
$$+ 9x^6 + 10x^5 + 11x^4 + 3x^3 + 3x^2 + 12x + 12$$

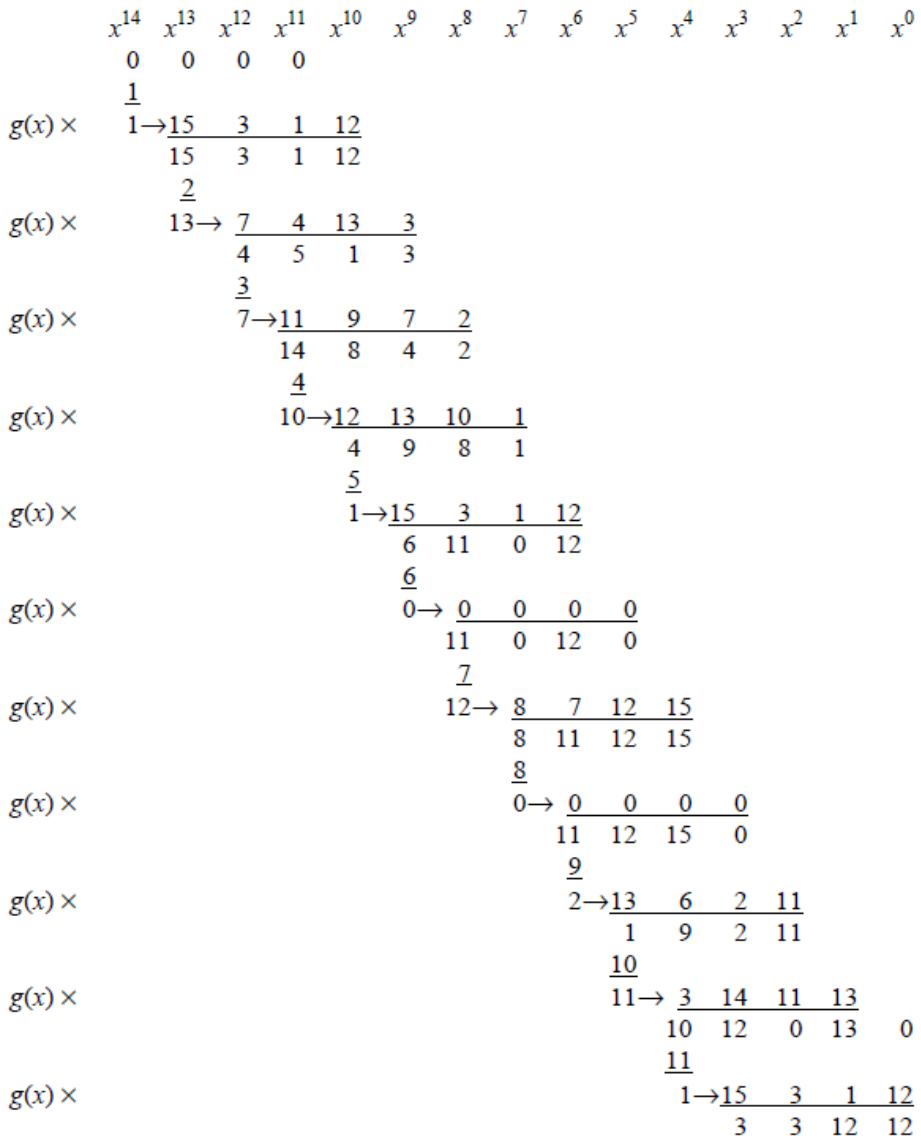
Μεταδίδεται δηλαδή η ακολουθία συμβόλων [VI]:

$$1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11,\ 3,\ 3,\ 12,\ 12$$

5.3.5 Κωδικοποίηση στο υλικό

Στο υλικό, τα σύμβολα του μηνύματος λαμβάνονται συνήθως ένα-ένα, ξεκινώντας από το M_{k-1} που αποτελεί το πρώτο σύμβολο του πολυωνύμου μηνύματος $M(x)$. Έτσι, η διαδικασία κωδικοποίησης και πιο συγκεκριμένα η πράξη της διαίρεσης υλοποιείται ως διοχέτευση (pipeline).

Παράδειγμα εκτέλεσης αυτής της διοχετευμένης διαίρεσης για το παράδειγμα της προηγούμενης Ενότητας παρουσιάζεται στο Σχήμα 5-3 της επόμενης σελίδας.

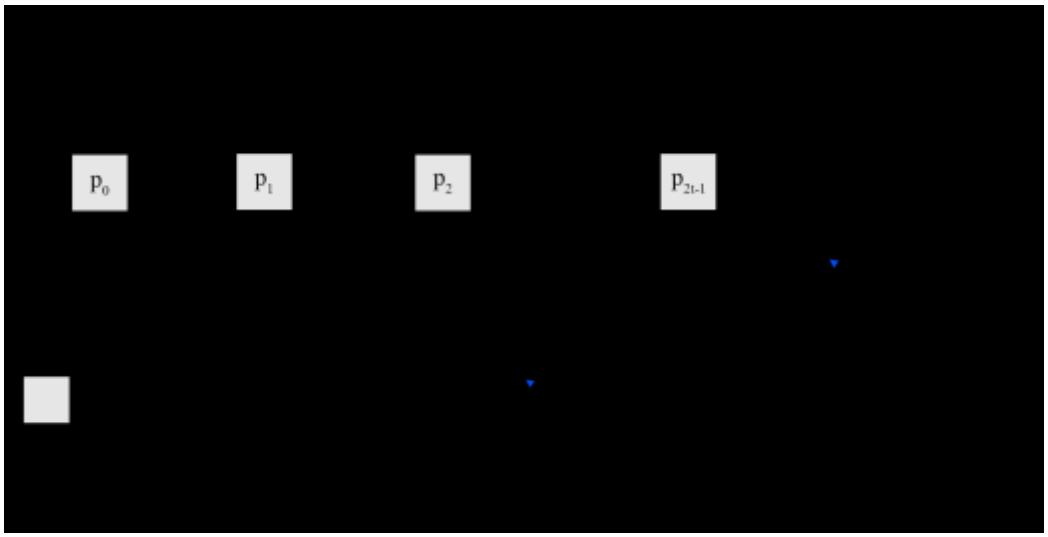


Σχήμα 5-3 - Παράδειγμα διαίρεσης διοχέτευσης στην περίπτωση του GF(2^4) [VI]

Όπως παρατηρείται και στο Σχήμα 5-3, το πρώτο σύμβολο του μηνύματος 1 προστίθεται στο περιεχόμενο της πιο σημαντικής στήλης, που αρχικά λαμβάνει την τιμή 0. Το αποτέλεσμα που προκύπτει από αυτήν την πράξη έχει τιμή 1. Αυτή η τιμή πολλαπλασιάζεται στην συνέχεια με τους υπόλοιπους συντελεστές του πολυωνύμου $g(x)$ που έχουν τις τιμές 15, 3, 1 και 12 αντίστοιχα. Οι τιμές που προκύπτουν από αυτόν τον πολλαπλασιασμό προστίθενται στις υπόλοιπες στήλες που επίσης λαμβάνουν αρχικά τιμή 0. Στην συνέχεια, η τιμή του 2ου συμβόλου, 2, προστίθεται στην επόμενη σημαντική στήλη, 15, δίνοντας 13. Αυτή η τιμή πολλαπλασιάζεται στην συνέχεια με τους συντελεστές του πολυωνύμου $g(x)$, δίνοντας τις τιμές 7, 4, 13 και 3 που προστίθενται στις υπόλοιπες στήλες. Η διαδικασία συνεχίζεται με αυτόν τον τρόπο παράγοντας τελικώς το $r(x)$ [VI].

Ένα βασικό κύκλωμα υπολογισμού των $2t$ συμβόλων πλεονασμού που προστίθενται κατά την κωδικοποίηση παρουσιάζεται στο Σχήμα 5-4. Αποτελεί κύκλωμα καταχωρητή ολίσθησης

γραμμικής ανατροφοδότησης ή LFSR (Linear Feedback Shift Register), το οποίο θεωρείται και κύκλωμα διαίρεσης. Ακολουθεί πλήρως την λογική διοχετευμένης διαίρεσης που παρουσιάστηκε πρωτύτερα. Μετά την πλήρη μετατόπιση της πληροφορίας μέσα στο LFSR, τα περιεχόμενα των καταχωρητών σχηματίζουν τα σύμβολα ισοτιμίας με μορφή πολυωνύμου $p(x)$ και μπορούν μέσω του κυκλώματος ελέγχου (με χρήση του «διακόπτη») να προστεθούν στο τέλος του πολυωνύμου κωδικοποιημένης λέξης $c(x)$ [XVIII].



Σχήμα 5-4 - Κύκλωμα Κωδικοποίησης LFSR

5.4 Μελέτη απόδοσης BER

Κατά την ψηφιακή μετάδοση, καθώς και στις λειτουργίες αποθήκευσης, σημαντικό κριτήριο απόδοσης αποτελεί το γνωστό BER (Bit-Error-Rate) που ορίζεται ως:

$$BER = \frac{Αριθμός\ σφαλμένων\ bit}{Συνολικός\ αριθμός\ bit} \tag{28}$$

Ακόμη, χρησιμοποιείται πολύ συχνά και το SNR (Signal-to-Noise-Ratio) που ορίζει την σχέση ανάμεσα στο σήμα και τον θόρυβο. Το SNR είναι αντιστρόφως ανάλογο του BER και έτσι όσο πιο χαμηλή είναι η τιμή του BER τόσο πιο υψηλή θα είναι η τιμή του SNR, και κατά συνέπεια τόσο πιο βέλτιστη θα είναι η ποιότητα της επικοινωνίας. Μαθηματικά διατυπώνεται η σχέση μεταξύ των SNR και BER ως:

$$BER \propto \frac{1}{SNR} \tag{29}$$

Ο ακριβής τύπος εξαρτάται κατά πολύ από τον δίαυλο επικοινωνίας. Στην προσομοίωση Matlab της πηγής [XII] χρησιμοποιείται διαμόρφωση και αποδιαμόρφωση BPSK (Binary Phase Shift Keying). Το κωδικοποιημένο μήνυμα περνάει από Γκαουσιανό κανάλι που προσθέτει λευκό θόρυβο τύπου AWGN. Υπολογίζοντας την σχέση της ενέργειας ανά bit προς την πυκνότητα του θορύβου E_b/N_o, διαπιστώθηκε πως η μείωση του ρυθμού του κώδικα, διατηρώντας σταθερό το μήκος του μπλοκ, βελτιώνει την απόδοση BER. Επίσης, παρατηρήθηκε και πως η απόδοση βελτιώνεται και κατά την αύξηση του μήκους του μπλοκ. Επομένως η απόδοση των κωδίκων Reed-Solomon βελτιώνεται κατά την μείωση του ρυθμού κώδικα και κατά την αύξηση του μήκους μπλοκ, μέχρι ένα βαθμό. Γενικά θεωρείται πως οι κώδικες Reed-Solomon, καθώς και οι πολλοί άλλοι ΚΔΛ, παρουσιάζουν κακή απόδοση BER στην περίπτωση χαμηλού SNR [XII].

5.5 Πιθανότητα λάθους

Στους κώδικες Reed-Solomon, η πιθανότητα λάθους σε κάποιο σύμβολο, P_E, διατυπώνεται μαθηματικά με την σχέση:

$$P_E \approx \frac{1}{2^m - 1} \sum_{j=t+1}^{2^m-1} j \binom{2^m - 1}{j} p^j (1-p)^{2^m-1-j} \tag{30}$$

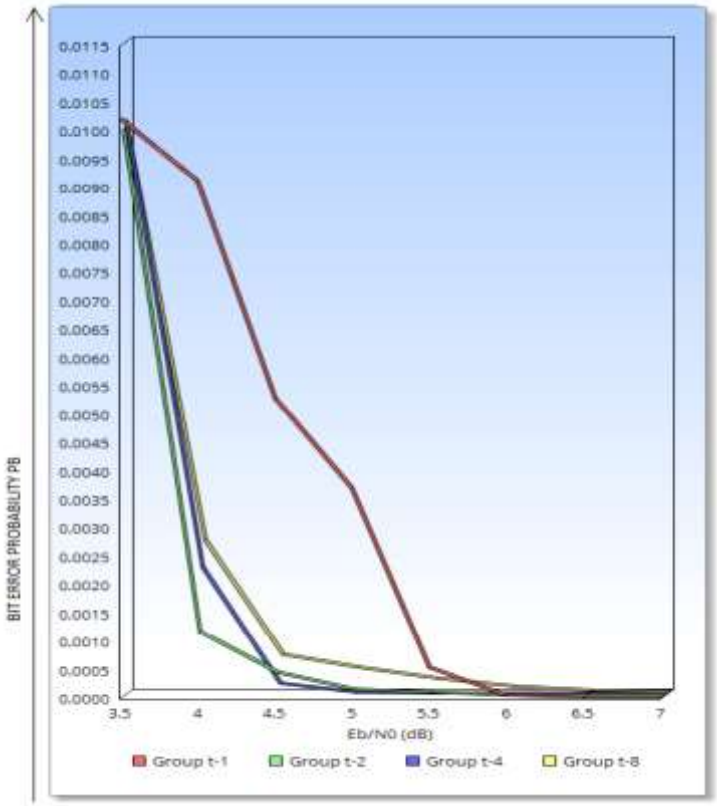
όπου p είναι η πιθανότητα να προκληθεί λάθος στο κανάλι.

Η πιθανότητα λάθους σε κάποιο bit, P_B, μπορεί να εκφραστεί με άνω όριο την πιθανότητα λάθους ανά σύμβολο. Η σχέση μεταξύ των P_B και P_E μαθηματικά διατυπώνεται έτσι ως:

$$\frac{P_B}{P_E} = \frac{2^{m-1}}{2^m - 1} \tag{31}$$

Στην μελέτη της πηγής [XIII] συγκρίθηκε η πιθανότητα λάθους ανά bit P_B με την απόδοση E_b/N_o. Στην περίπτωση μήκους συμβόλου 5-bit, όπου δηλαδή $m = 5$, διαπιστώθηκε πως η απόδοση σε λάθη βελτιώνεται καθώς αυξάνεται η τιμή του t στο διάστημα $[1, 4]$. Ωστόσο, κατά την περαιτέρω αύξηση του t στην τιμή $t = 8$, διαπιστώθηκε μείωση της απόδοσης P_B κατά περίπου 0.5dB του E_b/N_o, σε σύγκριση με την τιμή της περίπτωσης $t = 4$. Επομένως, όπως και στην περίπτωση του BER, ο ρυθμός βελτίωσης της απόδοσης δεν είναι σταθερός [XIII],

Το Σχήμα 5-5 αποτελεί διάγραμμα πιθανότητας λάθους ανά bit, P_B, προς την απόδοση ενέργεια bit προς θόρυβο, E_b/N_o, από την έρευνα της πηγής [XIII].



Σχήμα 5-5 - Διάγραμμα P_b και απόδοσης E_b/N_o [XIII]

6. ΑΠΟΚΩΔΙΚΟΠΟΙΗΣΗ REED-SOLOMON

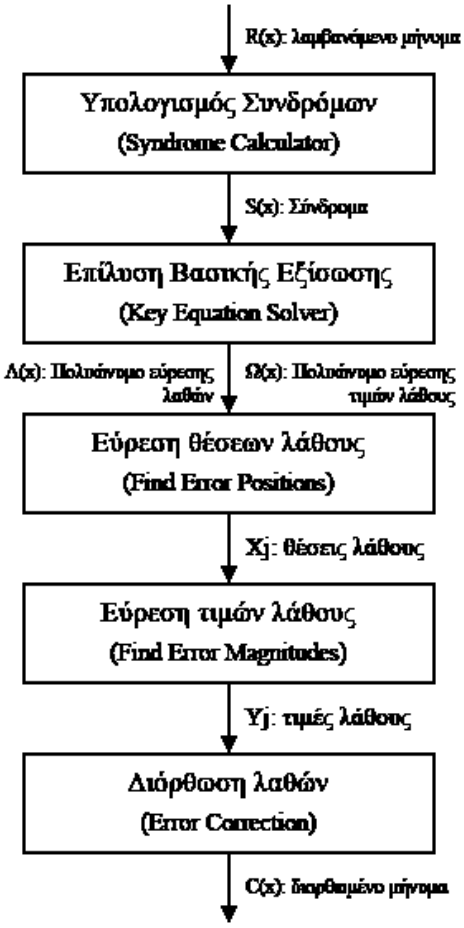
Διαπερνώντας το κωδικοποιημένο μήνυμα $c(x)$ από κανάλι με ποικίλες πηγές παρεμβολών και θορύβου υπάρχει μεγάλη πιθανότητα να παρουσιαστούν λάθη στα σύμβολα του. Έτσι, το λαμβανόμενο μήνυμα $r(x)$ διαφέρει από το αρχικό μήνυμα $c(x)$ σε κάποιο αριθμό συμβόλων. Η διαφορά μεταξύ αυτών των δύο πολυωνύμων μπορεί να εκφραστεί μέσω ενός πολυωνύμου λαθών $e(x)$. Άρα, τελικώς, το λαμβανόμενο μήνυμα διατυπώνεται μαθηματικά ως [XVII]:

$$r(x) = c(x) + e(x) \tag{32}$$

Κύριος στόχος του αποκωδικοποιητή είναι η εύρεση των συντελεστών του πολυωνύμου $e(x)$ που δεν έχουν μηδενική τιμή, δηλαδή των θέσεων με λάθη, καθώς και των τιμών τους. Ο αποκωδικοποιητής βρίσκεται πάντοτε στην θέση να εντοπίσει τα λάθη αυτά (εντοπισμός λαθών), ωστόσο έχει την δυνατότητα να διορθώσει μόνο τόσα όσα του επιτρέπει η ικανότητα διόρθωσης λαθών του t. Όταν ο αποκωδικοποιητής εντοπίσει πως υπάρχουν περισσότερα από t λάθη στο λαμβανόμενο μήνυμα τότε «δηλώνει» πως αυτά τα λάθη είναι ανεπανόρθωτα. Στην συνέχεια του Κεφαλαίου εικάζεται πως τα λάθη που παρουσιάζονται είναι το πολύ t [XVII].

6.1 Τα βασικά στάδια αποκωδικοποίησης

Το Σχήμα 6-1 απεικονίζει τα βασικά στάδια της διαδικασίας αποκωδικοποίησης.



Σχήμα 6-1 - Βασική Αρχιτεκτονική αποκωδικοποίησης

6.1.1 Υπολογισμός συνδρόμων

Πρώτο στάδιο της διαδικασίας αποκωδικοποίησης αποτελεί ο υπολογισμός του λεγόμενου πολυωνύμου συνδρόμων λάθους ή πιο απλά των συνδρόμων. Τα σύνδρομα είναι 2t σε πλήθος και υπολογίζονται βρίσκοντας την τιμή του λαμβανόμενου πολυωνύμου $r(x)$ για τα 2t πρώτα εκθετικά του $GF(2^m)$ που είναι τα $1, a, a^2, a^3, \cdots, a^{2t-1}$. Έτσι, τα σύνδρομα ορίζονται ως [XV][XVII]:

$$s_i = r(a^i), \qquad i = 0, 1, \cdots, 2t-1 \tag{33}$$

Η αξιολόγηση του πολυωνύμου $r(x) = r_0 + r_1 x + \cdots + r_{n-1}x^{n-1}$ για τα εκθετικά a^i, που εκτελείται για τον υπολογισμό των 2t συνδρόμων, ορίζεται ως το εξής άθροισμα [XV][XVII]:

$$s_i = \sum_{j=0}^{n-1} r_j (a^i)^j, \qquad i = 0, 1, \cdots, 2t-1 \tag{34}$$

Όταν δεν υπάρχουν σφάλματα στο λαμβανόμενο μήνυμα όλα τα σύνδρομα υπολογίζονται ως μηδέν. Μη-μηδενικά σύνδρομα υπονοούν λάθος στα σύμβολα του λαμβανόμενου μηνύματος. Τα υπολογισμένα σύνδρομα δίνονται στην συνέχεια στα επόμενα μπλοκ του αποκωδικοποιητή για τον υπολογισμό των θέσεων και τιμών λάθους [XV][XVII].

Το πολυώνυμο συνδρόμων μπορεί να οριστεί ως [XV]:

$$S(x) = \sum_{i=0}^{2t-1} s_i x^i = s_0 + s_1 x + \cdots + s_{2t-1}x^{2t-1} \tag{35}$$

Συναρτήσει των θέσεων λάθους X_j και των αντίστοιχων τιμών λάθους τους Y_j ορίζουμε τα σύνδρομα και με χρήση της σχέσης [XVI]:

$$s_i = \sum_{j=0}^{n-1} Y_j (X_j)^{i+1}, \qquad i = 0, 1, \cdots, 2t-1 \tag{36}$$

6.1.2 Επίλυση βασικής εξίσωσης (key equation)

Μετά τον υπολογισμό των συνδρόμων λάθους ακολουθεί η επίλυση της λεγόμενης βασικής εξίσωσης, μέσω της οποίας υπολογίζονται οι συντελεστές του πολυωνύμου εύρεσης λαθών (error locator polynomial) $\Lambda(x)$, καθώς και του πολυωνύμου εύρεσης τιμών/μεγεθών λάθους (error evaluator/magnitude polynomial) $\Omega(x)$. Χρησιμοποιώντας αυτά τα δύο πολυώνυμα απλοποιείται η διαδικασία εύρεσης των θέσεων και τιμών λάθους κατά μεγάλο βαθμό [XV].

6.1.2.1 Πολυώνυμο εύρεσης λαθών (Error locator polynomial)

Για πλήθος λαθών v, το πολυώνυμο εύρεσης λαθών $\Lambda(x)$ ορίζεται μαθηματικά ως [XVI]:

$$\begin{aligned} \Lambda(x) &= \prod_{i=1}^{v} (1 - xX_i) \\ &= \prod_{i=1}^{v} (-X_i) \cdot \prod_{i=1}^{v} (x - X_i^{-1}) \\ &= 1 + \lambda_1 x + \cdots + \lambda_v x^v \end{aligned} \tag{37}$$

Οι λύσεις του $\Lambda(x)$ αποτελούν τους αντίστροφους των θέσεων λάθους δηλαδή τα [XVI]:

$$X_1^{-1}, X_2^{-1}, \cdots, X_v^{-1}$$

Ο βαθμός του $\Lambda(x)$ είναι ίσος με τον αριθμό των λαθών. Έτσι, ο αποκωδικοποιητής χρειάζεται να υπολογίσει την τιμή του v, καθώς και τις θέσεις και τιμές των λαθών αυτών [XVI].

6.1.2.2 Πολυώνυμο εύρεσης τιμών λάθους (Error evaluator polynomial)

Το πολυώνυμο εύρεσης τιμών λάθους $\Omega(x)$ υπολογίζεται από την βασική εξίσωση, η οποία το συνδέει με τα πολυώνυμα $S(x)$ και $\Lambda(x)$ και διατυπώνεται με τον εξής τρόπο:

$$\Lambda(x) \cdot S(x) = \Omega(x) \; mod \; x^{2t} \tag{38}$$

Σύμφωνα με αυτήν την εξίσωση, οι συντελεστές του $\Omega(x)$ υπολογίζονται με τις πράξεις [XVI]:

$$\begin{aligned} \omega_0 &= S_b \\ \omega_1 &= S_{b+1} + S_b\lambda_1 \\ \omega_2 &= S_{b+2} + S_{b+1}\lambda_1 + S_b\lambda_2 \\ &\vdots \\ \omega_{v-1} &= S_{b+v-1} + S_{b+v-2}\lambda_1 + \cdots + S_b\lambda_{v-1} \end{aligned} \tag{39}$$

Απαιτούνται επομένως $\frac{1}{2}(v-1)(v-2) \approx \frac{1}{2}v^2$ πολλαπλασιασμοί-συσσώρευσης [XVI].

Με την μορφή πολυωνύμου το $\Omega(x)$ ορίζεται ως εξής:

$$\Omega(x) = 1 + \omega_1 x + \omega_2 x^2 + \; \cdots + \omega_{v-1}x^{v-1} \tag{40}$$

Ο βαθμός του πολυωνύμου $\Omega(x)$ είναι επομένως πάντοτε μικρότερος του v, $\deg\big(\Omega(x)\big) < v$, για πλήθος λαθών $v \leq t$ [XVI][XVII].

Η επίλυση αυτής της εξίσωσης για τον υπολογισμό των πολυωνύμων $\Lambda(x)$ και $\Omega(x)$ αποτελεί ίσως το πιο δύσκολο κομμάτι της υλοποίησης ενός αποκωδικοποιητή. Έχουν σχεδιαστεί πολλοί αποτελεσματικοί αλγόριθμοι για την επίλυση της βασικής εξίσωσης [XVII]. Αλγόριθμο επίλυσης βασικής εξίσωσης, μαζί με διάφορες βελτιστοποιήσεις τους, αναφέρονται στην Ενότητα 6.2.

6.1.3 Εύρεση θέσεων και τιμών λάθους

Γνωρίζοντας την τιμή των συντελεστών του $\Lambda(x)$ και πως οι λύσεις του αποτελούν τους αντίστροφους των θέσεων λάθους, η εύρεση των θέσεων αυτών ενέχει μόνο τον υπολογισμό της τιμής του $\Lambda(x)$ για κάθε a^{-j}, όπου $0 \leq j \leq n-1$. Με άλλα λόγια υπολογίζεται το $\Lambda(a^{-j})$. Αυτή η διαδικασία είναι γνωστή ως Αναζήτηση Chien (Chien Search). Περισσότερη λεπτομέρεια για αυτήν την διαδικασία θα δοθεί στην Ενότητα 6.3 [XVII].

Μετά την εύρεση των θέσεων λάθους, τις οποίες ορίζουμε ως X_i, επόμενο βήμα αποτελεί η εύρεση της τιμής αυτών των λαθών. Η τιμή κάθε λάθους, Y_i, υπολογίζεται με την εξίσωση Forney:

$$Y_i = -\frac{\Omega\left(X_i^{-1}\right)}{x\Lambda'\left(X_i^{-1}\right)} \tag{41}$$

Το $\Lambda'(x)$ αποτελεί την τυπική παράγωγο του $\Lambda(x)$, που στην περίπτωση της αριθμητικής πεδίων Galois υπολογίζεται με την σχέση:

$$\Lambda'(x) = \lambda_1 + \lambda_3 x^2 + \cdots \tag{42}$$

που επαναδιατυπώνεται εύκολα και ως:

$$x\Lambda'(x) = \lambda_1 x + \lambda_3 x^3 + \cdots \tag{43}$$

Με αυτήν την σχέση υπολογίζεται πρακτικώς το άθροισμα των συντελεστών περιττού βαθμού, κάτι το οποίο υπολογίζεται ήδη κατά την Αναζήτηση Chien και επιτρέπει έτσι την βελτιστοποίηση χρήσης πόρων υλικού [XVII]. Για τον υπολογισμό όλων των τιμών λάθους ο αλγόριθμος του Forney απαιτεί $2v^2$ πολλαπλασιασμούς [XVI].

6.1.4 Διόρθωση λαθών

Οι τιμές των λαθών Y_i που υπολογίζονται μέσω του αλγορίθμου Forney προστίθενται τελικώς στις αντίστοιχες θέσεις X_i του λαμβανόμενου μηνύματος $r(x)$, ώστε να διορθωθούν τα λάθη του μηνύματος. Μετά το πέρας της διαδικασίας αποκωδικοποίησης είναι γνωστό πλέον το πολυώνυμο λαθών $e(x)$, το οποίο στους συντελεστές που αντιστοιχούν στις θέσεις λαθών X_i έχει πια τις αντίστοιχες τιμές τους Y_i. Οι υπόλοιποι συντελεστές του $e(x)$ έχουν φυσικά τιμή μηδέν. Έτσι, η διόρθωση των λαθών αποτελεί μόνο το ακόλουθο άθροισμα μεταξύ των πολυωνύμων $r(x)$ και $e(x)$, όπως αναφέρθηκε και στην αρχή του Κεφαλαίου [VI]:

$$c(x) = r(x) + e(x)$$

6.2 Αλγόριθμοι επίλυσης βασικής εξίσωσης

6.2.1 Αλγόριθμος Peterson-Gorenstein-Zierler (PGZ)

Τον ίσως πιο εύκολα κατανοητό αλγόριθμο αποκωδικοποίησης αποτελεί ο αλγόριθμος των Peterson-Gorenstein-Zierler (PGZ). Είναι αποδοτικός για τιμές $t < 4$, ωστόσο δεν κλιμακώνεται για μεγαλύτερες τιμές του t [XVII].

Σύμφωνα με όσα αναφέρθηκαν πρωτύτερα για τα σύνδρομα και το πολυώνυμο εύρεσης λαθών $\Lambda(x)$, για ένα μέγιστο πλήθος λαθών v το σύνδρομο s_k ορίζεται και ως [VII]:

$$\Lambda_v s_{k-v} + \Lambda_{v-1} s_{k-v+1} + \cdots + \Lambda_1 s_{k-1} = -s_k \tag{44}$$

Η επίλυση για τους συντελεστές του πολυωνύμου $\Lambda(x)$ διατυπώνεται ως σύστημα [VII]:

$$M^{(v)}\Lambda^{(v)} = \begin{bmatrix} s_1 & s_2 & \cdots & s_v \\ s_2 & s_3 & \cdots & s_{v+1} \\ \vdots & \vdots & \vdots & \vdots \\ s_v & s_{v+1} & \cdots & s_{2v-1} \end{bmatrix} \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -s_{v+1} \\ -s_{v+2} \\ \vdots \\ -s_{2v} \end{bmatrix} \tag{45}$$

Υπολογίζοντας τα $2t$ σύνδρομα και εφόσον δεν είναι μηδενικά, τα βήματα του PGZ συνοψίζονται [VII]:

1. Θεωρούμε πως $v = t$
2. Έλεγχος μοναδικότητας του πίνακα $M^{(v)}$ υπολογίζοντας την ορίζουσα του
   - Αν $\det M^{(v)} = 0$ τότε ο πίνακας είναι μοναδικός και μειώνεται η τιμή του v κατά 1, $v = v - 1$. Στην περίπτωση που $v = 0$ τα λάθη είναι ανεπανόρθωτα και έτσι τερματίζεται η διαδικασία. Αλλιώς, επαναλαμβάνεται το βήμα 2.
   - Αν $\det M^{(v)} \neq 0$ τότε ο πίνακας είναι μη μοναδικός και εκτελείται το βήμα 3.
3. Επιλύεται το ακόλουθο σύστημα εξισώσεων:

$$\begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -s_{v+1} \\ -s_{v+2} \\ \vdots \\ -s_{2v} \end{bmatrix} \begin{bmatrix} s_1 & s_2 & \cdots & s_v \\ s_2 & s_3 & \cdots & s_{v+1} \\ \vdots & \vdots & \vdots & \vdots \\ s_v & s_{v+1} & \cdots & s_{2v-1} \end{bmatrix}^{-1} \tag{46}$$

4. Λύση του συστήματος είναι το πολυώνυμο εύρεσης λαθών $\Lambda(x)$

Ο υπολογισμός της ορίζουσας γίνετε μέσω της επέκτασης Laplace. Το σύστημα επιλύεται εύκολα με την μέθοδο απαλοιφής Gauss [VII].

Στην συνέχεια, υπολογίζονται κανονικά οι ρίζες του $\Lambda(x)$, που όπως αναφέρθηκε και στην αντίστοιχη Ενότητα αποτελούν τους αντίστροφους των θέσεων λάθους $X_1^{-1}, X_2^{-1}, \cdots, X_v^{-1}$. Για τον υπολογισμό των θέσεων λάθους προτιμάται ο αλγόριθμος Forney. Εφόσον δεν υπολογίζεται ταυτόχρονα και το πολυώνυμο εύρεσης τιμών λάθους $\Omega(x)$ που απαιτείται στον αλγόριθμο Forney, μπορούν να βρεθούν οι τιμές λάθους $Y_1, Y_2, \cdots, Y_v$ και λύνοντας το σύστημα εξισώσεων [XVI]:

$$\begin{aligned} Y_1X_1 + \cdots Y_vX_v &= S_1 \\ Y_1X_1^2 + \cdots Y_vX_v^2 &= S_2 \\ &\vdots \\ Y_1X_1^v + \cdots Y_vX_v^v &= S_v \\ &\vdots \\ Y_1X_1^{2t} + \cdots Y_vX_v^{2t} &= S_{2t} \end{aligned} \tag{47}$$

6.2.2 Αλγόριθμος Sugiyama

Ο αλγόριθμος των Sugiyama, Kasahara, Hirasawa, Namekawa που προτάθηκε το 1975 βασίζεται στον επεκταμένο αλγόριθμο του Ευκλείδη, και για αυτόν τον λόγω είναι γνωστός και ως αλγόριθμος αποκωδικοποίησης Ευκλείδη (Euclid) [XVI].

Η βασική εξίσωση επαναδιατυπώνεται ως εξής:

$$\Omega(x) = \Lambda(x)S(x) + b(x)x^{2t} \tag{48}$$

όπου $b(x)$ πολυώνυμο με βαθμό μικρότερο του ν [XVI].

Η ταυτότητα του Bézout, για δύο πολυώνυμα $a(x)$ και $b(x)$ με μέγιστο κοινό διαιρέτη το πολυώνυμο $GCD\big(a(x), b(x)\big) = c(x)$, ορίζει:

$$c(x) = a(x)m(x) + b(x)n(x) \tag{49}$$

όπου $m(x)$ και $n(x)$ κάποια πολυώνυμα [VII].

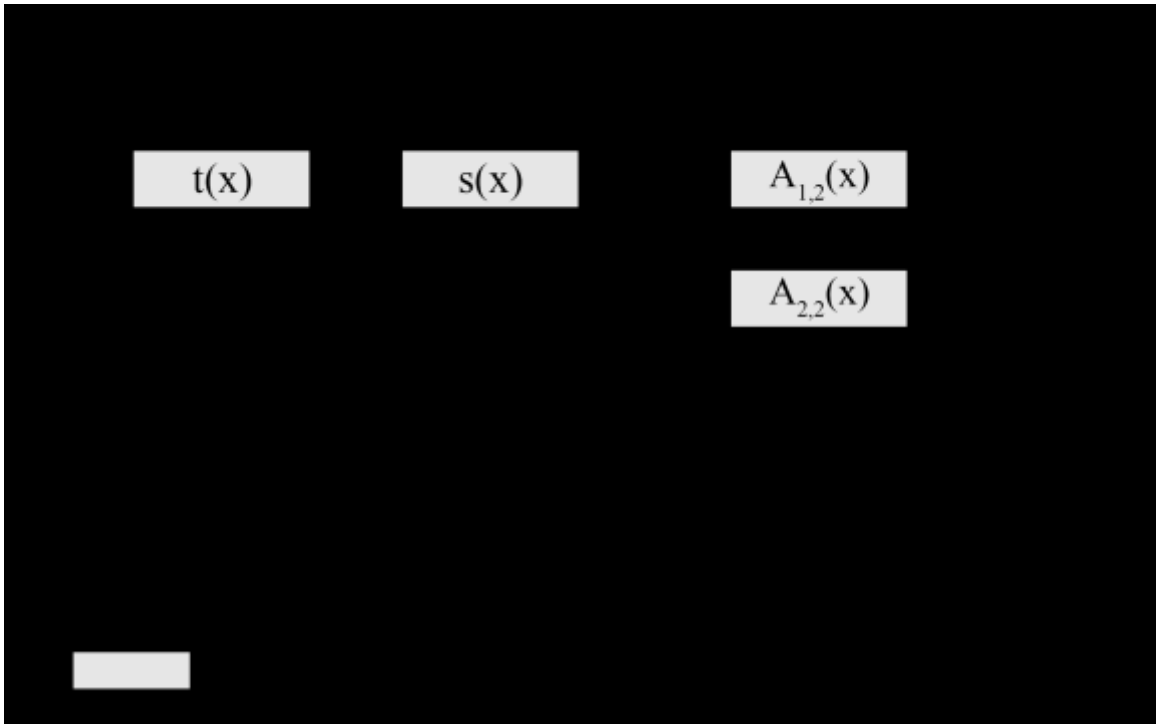
Αυτή η σχέση λύνεται με χρήση του επεκταμένου αλγόριθμου Ευκλείδη. Αποτέλεσμα της διαδικασίας αυτής στην περίπτωση της βασικής εξίσωσης είναι το πολυώνυμο εύρεσης λαθών $\Lambda(x)$, καθώς και το πολυώνυμο εύρεσης τιμών λάθους $\Omega(x)$ [VII].

Τα βήματα του αλγορίθμου είναι τα εξής [XVI]:

1. Υπολογισμός συνδρόμων: $S_j = r\big(a^{j-1}\big), j = 1, \cdots, 2t$
2. Αρχικοποίηση:
$$s(x) \leftarrow x^{2t}, \qquad t(x) \leftarrow \sum_{j=1}^{2t} S_j x^{j-1}, \qquad A(x) \leftarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
3. Όσο ο βαθμός του $t(x)$ είναι μεγαλύτερος του t ($\deg(t(x) \geq t$):
$$\mathrm{Q(x)} \leftarrow \left\lfloor \frac{s(x)}{t(x)} \right\rfloor, \qquad \begin{bmatrix} s(x) \\ t(x) \end{bmatrix} \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -Q(x) \end{bmatrix} \begin{bmatrix} s(x) \\ t(x) \end{bmatrix}, \qquad A(x) \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -Q(x) \end{bmatrix} A(x)$$
4. Οριστικοποίηση λύσης:
$$\Delta \leftarrow A_{2,2}(0), \qquad \Lambda(x) \leftarrow \Delta^{-1}A_{2,2}(x), \qquad \Omega(x) \leftarrow \Delta^{-1}t(x)$$

Η υλοποίηση αυτού του αλγορίθμου σε υλικό απαιτεί $2t$ διαιρέτες και t πολλαπλασιαστές. Για τον υπολογισμό του αντιστρόφου απαιτείται μνήμη ROM. Έτσι, συνυπολογίζοντας και αθροιστές, πολυπλέκτες και πύλη AND, η καθυστέρηση του κρίσιμου μονοπατιού του αλγορίθμου είναι $(ROM + AND + 2 \times MULT + ADD + 2 \times MUX)$. Η καθυστέρηση ολόκληρου του μπλοκ επίλυσης της βασικής εξίσωσης (block latency) είναι $2t$ κύκλοι ρολογιού [XVII].

Στο Σχήμα 6-2 παρουσιάζεται η αρχιτεκτονική του αλγορίθμου Sugiyama με χρήση μπλοκ διαγράμματος.



Σχήμα 6-2 - Αρχιτεκτονική Αλγορίθμου Sugiyama για Υπολογισμό Λ(x) και Ω(x)

6.2.3 Αλγόριθμος Berlekamp-Massey

Έστω κώδικας RS με ικανότητα διόρθωσης λαθών t και κωδικοποιημένο μήνυμα με αριθμό λαθών έστω v. Για $v < j \leq 2t$, γνωρίζοντας ένα αριθμό συνδρόμων v, υπολογίζονται τα επόμενα σύνδρομα με τις ακόλουθες διαδοχικές πράξεις [VII]:

$$\begin{aligned} -s_{v+1} &= \Lambda_1 s_v + \Lambda_2 s_{v-1} + \cdots + \Lambda_v s_1 \\ -s_{v+2} &= \Lambda_1 s_{v+1} + \Lambda_2 s_v + \cdots + \Lambda_v s_2 \\ &\vdots \\ -s_{2t} &= \Lambda_1 s_{2t-1} + \Lambda_2 s_{2t-2} + \cdots + \Lambda_v s_{2t-v} \end{aligned} \tag{50}$$

Αυτή η διαδικασία υλοποιείται με χρήση καταχωρητή ολίσθησης γραμμικής ανατροφοδότησης ή LFSR, κάτι που αναφέρθηκε και στην Ενότητα 5 στην διαδικασία υλοποίησης της κωδικοποίησης Reed-Solomon στο υλικό. Με παρόμοιο τρόπο υπολογίζονται επομένως και τα σύνδρομα από τους συντελεστές του $\Lambda(x)$ [VII].

Οι καταχωρητές αρχικοποιούνται με τις τιμές των αρχικά γνωστών συνδρόμων $s_1, s_2, \cdots, s_v$ και πολλαπλασιάζονται σε κάθε κύκλο με μια σταθερά τιμή συντελεστή του $\Lambda(x)$. Σε κάθε κύκλο το επόμενο σύνδρομο ισούται με το άθροισμα όλων των γινομένων αυτών [VII].

Σύμφωνα με τον αλγόριθμο Berlekamp-Massey, το πρόβλημα εύρεσης των συντελεστών του πολυωνύμου εύρεσης λαθών $\Lambda(x)$ μπορεί να εκφραστεί ως πρόβλημα κατασκευής ενός τέτοιου LFSR. Οι συντελεστές αυτοί υπολογίζονται σε $2t$ επαναλήψεις του αλγορίθμου. Με είσοδο τα $2t$ σύνδρομα που υπολογίζονται στο μπλοκ υπολογισμού συνδρόμων, και για τα οποία δεν ισχύει $s_1 = s_2 = \cdots = s_{2t}$, τα βήματα του Berlekamp-Massey συνοψίζονται ως εξής [VII]:

1. Αρχικοποίηση: $L_0 = 0$ - μήκος LFSR
   $r = 0$ - μετρητής επαναλήψεων
   $\Lambda^{(0)}(x) = 1$ - πολυώνυμο εύρεσης λαθών
   $B(x) = 1$ - βοηθητικό πολυώνυμο
2. Αύξηση μετρητή: $r = r + 1$
3. Υπολογισμός διαφοράς (discrepancy) μεταξύ της τιμής που πρέπει να υπολογιστεί από το LFSR, s_r, και τις τιμής που υπολογίζεται πραγματικά από το LFSR:
$$\Delta^{(r)} = s_r - \sum_{j=0}^{L_{r-1}} \Lambda_j^{(r-1)} s_{r-j}$$
4. Αν $\Delta^{(r)} = 0$ τότε το LFSR παράγει σωστά το σύνδρομο s_r και έτσι:
$$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x), \quad L_r = L_{r-1}$$
   Αν $\Delta^{(r)} \neq 0$ τότε το LFSR πρέπει να τροποποιηθεί και εκτελούνται:
$$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x) - \Delta^{(r)} x B(x)$$
   Αν $2L_{r-1} \leq r - 1$
   $L_r = r - L_{r-1}$
   $B(x) = \frac{\Lambda^{(r-1)}(x)}{\Delta^{(r)}}$
   Αλλιώς αν $2L_{r-1} > r - 1$
   $L_r = L_{r-1}$
   $B(x) = xB(x)$
5. Αν ο βαθμός του $\Lambda^{(r)}(x) > t$, τότε τα λάθη είναι ανεπανόρθωτα.
6. Αν $r = 2t$ τότε τερματίζει ο αλγόριθμος και $\Lambda(x) = \Lambda^{(2t)}(x)$
   Αλλιώς αν $r < 2t$ επιστροφή στο Βήμα 2.

Ο αλγόριθμος αυτός έχει την χαμηλότερη πολυπλοκότητα υλικού, καθώς οι ίδιοι πόροι υλικού μπορούν μετά το πέρας του αλγορίθμου να χρησιμοποιηθούν και για την εύρεσης του $\Omega(x)$. Ένας τρόπος υπολογισμού των συντελεστών του $\Omega(x)$ αναφέρθηκε στην Ενότητα 6.1.2.2 [XVII].

6.2.4 Βελτιστοποιήσεις Berlekamp-Massey

6.2.4.1 Διατύπωση BM του Blahut

Μια αρκετά δημοφιλές βελτιστοποίηση του αλγορίθμου Berlekamp-Massey αποτελεί η διατύπωση του Blahut, στην οποία υπολογίζονται τα πολυώνυμα $\Lambda^{(r)}(x)$ και $B^{(r)}(x)$ επαναληπτικά. Μετά το πέρας $2t$ επαναλήψεων, το πολυώνυμο $\Omega(x)$ υπολογίζεται από το γινόμενο των συντελεστών βαθμού μικρότερου του $t - 1$ του πολλαπλασιασμού $\Lambda(x)S(x)$. Μια τέτοια υλοποίηση χρειάζεται να αποθηκεύσει μόνο $4t$ στοιχεία του πεδίου GF. Ο χρόνος που απαιτείται για τον υπολογισμού του $\Omega(x)$ είναι t κύκλοι ρολογιού. Έτσι, η επίλυση της βασικής

εξίσωσης απαιτεί τελικώς $3t$ κύκλους. Η διατύπωση του Blahut κερδίζει επομένως σε υλικό αυξάνοντας τον χρόνο εκτέλεσης [XXII].

Δυστυχώς η διατύπωση του Blahut παρουσιάζει και πάλι το ίδιο πρόβλημα με τον κλασσικό BM, το οποίο είναι η ανάγκη διαίρεσης κάθε συντελεστή του $\Lambda^{(r)}(x)$ με το $\Delta^{(r)}$. Αυτή η διαίρεση μπορεί να υλοποιηθεί με χρήση πίνακα αντιστρόφου, πολλαπλασιάζοντας τον κάθε συντελεστή του $\Lambda^{(r)}(x)$ με τον αντίστροφο του $\Delta^{(r)}$. Ωστόσο, απαιτούνται κυκλώματα διαιρέτη για κάθε συντελεστή του $\Lambda^{(r)}(x)$, τα οποία λόγω της χρήσης μνήμης ROM είναι πιο χρονοβόρα από το αντίστοιχο κύκλωμα πολλαπλασιασμού. Οι διαιρέτες αυξάνουν την καθυστέρηση του κρίσιμου τμήματος και μειώνουν έτσι την δυνατότητα εκτέλεσης σε μεγάλες συχνότητες ρολογιού. Φυσικά η αντικατάσταση των διαιρέσεων του Berlekamp-Massey από πολλαπλασιασμούς δεν θα απαιτεί υπολογισμό αντίστροφου και θα επιτρέψει την εκτέλεση του μπλοκ επίλυσης βασικής εξίσωσης σε πολύ μεγαλύτερες συχνότητες [XXII].

6.2.4.2 Berlekamp-Massey χωρίς αντίστροφο (iBM)

Με την χρήση αλγορίθμου BM χωρίς αντίστροφο (ή χωρίς διαίρεσης) υπολογίζονται πολλαπλάσια των πολυωνύμων $\Lambda(x)$ και $\Omega(x)$, $\beta \cdot \Lambda(x)$ και $\beta \cdot \Omega(x)$. Οι λύσεις του $\beta \cdot \Lambda(x)$ είναι ίδιες με τις λύσεις του $\Lambda(x)$ και επομένως υπολογίζονται οι ίδιες θέσεις λάθους . Κάτι αντίστοιχο ισχύει και στην περίπτωση του πολυωνύμου $\Omega(x)$, όπου απαιτείται μόνο δεξιά ολίσθηση των συντελεστών του $\beta \cdot \Omega(x)$ για την εύρεση των ίδιων τιμών λάθους με το αρχικό $\Omega(x)$. Στον κλασσικό BM ο $1^{ος}$ συντελεστής του $\Lambda(x)$ έχει την τιμή $\Lambda(0) = 1$ και δεν απαιτείται η αποθήκευσή του σε καταχωρητή. Στην περίπτωση του iBM όμως, ο συντελεστής αυτός λαμβάνει στο πέρας της διαδικασίας την τιμή $\Lambda(0) = \beta$ [XXII].

Τα βήματα του αλγορίθμου iBM είναι τα εξής:

1. Αρχικοποίηση: $\lambda^{(0)}(0) = 1, \quad \lambda^{(0)}(i) = 0, \quad i = 1, 2, \cdots, t$
   $b^{(0)}(0) = 1, \quad b^{(0)}(i) = 0, \quad i = 1, 2, \cdots, t$
   $k^{(0)} = 0, \qquad \gamma^{(0)} = 1$
2. Είσοδος: $s_i, \ i = 0, 1, \cdots, 2t - 1$
3. Για $r = 0$ εώς $2t - 1$:

   Βήμα iBM.1 $\quad \delta^{(r)} = s_r \cdot \lambda_0^{(r)} + s_{r-1} \cdot \lambda_1^{(r)} + \cdots + s_{r-t} \cdot \lambda_t^{(r)}$

   Βήμα iBM.2 $\quad \lambda_i^{(r+1)} = \gamma^{(r)} \cdot \lambda_i^{(r)} - \delta^{(r)} \cdot {b_{i-1}}^{(r)}, \quad (i = 0, 1, \cdots, t)$

   Βήμα iBM.3 $\quad$ Αν $\delta^{(r)} \neq 0$ και $k^{(r)} \geq 0$ τότε :

   $$b_i^{(r+1)} = \lambda_i^{(r)}, \quad (i = 0, 1, \cdots, t)$$
   $$\gamma^{(r+1)} = \delta^{(r)}$$
   $$k^{(r+1)} = -k^{(r)} - 1$$

   Αλλιώς :

   $$b_i^{(r+1)} = b_{i-1}^{(r)}, \quad (i = 0, 1, \cdots, t)$$
   $$\gamma^{(r+1)} = \gamma^{(r)}$$

$$k^{(r+1)} = k^{(r)} + 1$$

4. Για $i = 0$ εώς $t - 1$:

   Βήμα iBM.4 $\quad \omega_i^{(2t)} = s_i \cdot \lambda_0^{(2t)} + s_{i-1} \cdot \lambda_1^{(2t)} + \cdots + s_0 \cdot \lambda_i^{(2t)}$

5. Έξοδος: $\lambda_i^{(2t)}, \quad i = 0, 1, \cdots, t$

   $\omega_i^{(2t)}, \quad i = 0, 1, \cdots, t - 1$

Η τιμή του $b_{-1}^{(r)}$ που εμφανίζεται στα βήματα iBM.2 και iBM.3 έχει τιμή 0 για όλα τα r. Επίσης, για $r < t$ το βήμα iBM.1 περιέχει τους όρους $s_{-1} \cdot \lambda_{r+1}^{(r)},\ s_{-2} \cdot \lambda_{r+2}^{(r)}, \cdots,\ s_{r-t} \cdot \lambda_t^{(r)}$, με άγνωστες τιμές $s_{-1},\ s_{-2}, \cdots,\ s_{r-t}$. Ωστόσο, εφόσον ο βαθμός του $\lambda^{(r)}(x)$ είναι $\deg \lambda^{(r)}(x) \leq r$, τα $\lambda_{r+1}^{(r)}$, $\lambda_{r+2}^{(r)}, \cdots,\ \lambda_t^{(r)}$ έχουν όλα την τιμή μηδέν. Έτσι, το αποτέλεσμα $\delta^{(r)}$ δεν επηρεάζεται [XXII].

Τα βήματα iBM.1 και iBM.4 παρουσιάζουν μεγάλη ομοιότητα και έτσι απαιτούνται μόνο δύο βασικές δομές πράξεων:

- μπλοκ υπολογισμού διαφοράς (discrepancy computation - DC) για την υλοποίηση των βημάτων iBM.1 και iBM.4
- μπλοκ ενημέρωσης του πολυωνύμου $\lambda_i^{(2t)}$(error locator update – ELU) για την παράλληλη υλοποίηση των βημάτων iBM.2 και iBM.3

Ο τρόπος με τον οποίο επικοινωνούν αυτά τα μπλοκ μεταξύ τους, αλλά και με τα μπλοκ υπολογισμού συνδρόμων (SC) και εύρεσης θέσεων και τιμών λάθους (CSEE) παρουσιάζεται στο Σχήμα 6-3 της επόμενης σελίδας [XXII].

Η συνολική απαίτηση υλικού του iBM, αγνοώντας το κύκλωμα ελέγχου, είναι $4t + 2$ μανταλωτές (latches), $3t + 3$ πολλαπλασιαστές, $2t + 1$ αθροιστές και $t + 1$ πολυπλέκτες. Ο συνολικός χρόνος επίλυσης της βασικής εξίσωσης είναι $3t$ κύκλοι. Υπολογίζοντας το $\Omega(x)$ επαναληπτικά μειώνεται ο χρόνος εκτέλεσης σε $2t$ κύκλους, ωστόσο διπλασιάζεται η ανάγκη υλικού, καθώς απαιτούνται σχεδόν 2 ELU. Η απαίτηση υλικού αυξάνεται σε $6t + 2$ μανταλωτές, $5t + 3$ πολλαπλασιαστές, $3t + 1$ αθροιστές και $2t + 1$ πολυπλέκτες [XXII].

Σε και τις δύο περιπτώσεις όμως, η καθυστέρηση του κρίσιμου τμήματος είναι [XXII]:

$$\begin{aligned} T_{iBM} &= 2 \cdot MULT + (1 + \log_2(t+1)) \cdot ADD \\ &> 2 \cdot (MULT + ADD) \end{aligned}$$

Αναδιατυπώνοντας τον αλγόριθμο iBM η καθυστέρηση μειώνεται κατά πολύ μεγάλο βαθμό, επιτρέποντας έτσι εκτέλεση σε μεγαλύτερες συχνότητες [XXII].

Σχήμα 6-3 - Αρχιτεκτονική Αλγορίθμου iBM για Υπολογισμό Λ(x) και Ω(x)

6.2.4.3 Αναδιατυπωμένος BM χωρίς αντίστροφο (riBM)

Το βήμα iBM.2 του αλγορίθμου iBM, σε μορφή πολυωνύμων μπορεί να διατυπωθεί και ως:

$$\Lambda^{(r+1)}(x) = \gamma^{(r)} \cdot \Lambda^{(r)}(x) - x\delta^{(r)} \cdot B^{(r)}(x) \tag{51}$$

Στο βήμα iBM.3 ο αλγόριθμος iBM αναθέτει στο $B^{(r+1)}(x)$ είτε το πολυώνυμο $\Lambda^{(r)}(x)$, είτε το πολυώνυμο $x \cdot B^{(r)}(x)$. Επίσης, αξίζει να σημειωθεί και πως το $\delta^{(r)}$ ισούται με τον συντελεστή $\delta_r^{(r)}$ του x^r στο πολυώνυμο $\Delta^{(r)}(x)$, που προκύπτει από τον πολλαπλασιασμό πολυωνύμων:

$$\Lambda^{(r)}(x) \cdot S(x) = \Delta^{(r)}(x) = \delta_0^{(r)} + \delta_1^{(r)} \cdot x + \cdots + \delta_r^{(r)} \cdot x^r + \cdots \tag{52}$$

Έτσι, μπορεί να υλοποιηθεί ο iBM ακόμα πιο αποδοτικά αν υπολογιστούν όλοι συντελεστές του $\Delta^{(r)}(x)$, καθώς και του $\Theta^{(r)}(x) = B^{(r)}(x) \cdot S(x)$, ακόμα και αν απαιτείται μόνο το $\delta_r^{(r)}$ για τον υπολογισμό του $\Lambda^{(r+1)}(x)$ [XXII].

Έστω πως στην εκκίνηση του κύκλου ρολογίου το μπλοκ επίλυσης βασικής εξίσωσης έχει στην διάθεσή του όλους τους συντελεστές των πολυωνύμων $\Delta^{(r)}(x)$, $\Theta^{(r)}(x)$, $\Lambda^{(r)}(x)$ και $B^{(r)}(x)$. Σε αυτήν την περίπτωση ο συντελεστής $\delta_r^{(r)} = \delta^{(r)}$ είναι επίσης διαθέσιμος. Ως εκ τούτου, είναι δυνατός ο υπολογισμός των πολυωνύμων $\Lambda^{(r+1)}(x)$ και $B^{(r+1)}(x)$ [XXII].

Αντικαθιστώντας το $B^{(r)}(x)$ από το πολυώνυμο $\Theta^{(r)}(x)$, το βήμα iBM.1 μπορεί να αναδιατυπωθεί με τον εξής τρόπο:

$$\begin{aligned} \Delta^{(r+1)}(x) &= \Lambda^{(r+1)}(x) \cdot S(x) \\ &= \left[\gamma^{(r)} \cdot \Lambda^{(r)}(x) - x\delta_r^{(r)} \cdot B^{(r)}(x)\right] \cdot S(x) \\ &= \gamma^{(r)} \cdot \Delta^{(r)}(x) - x\delta_r^{(r)} \cdot \Theta^{(r)}(x) \end{aligned} \tag{53}$$

Όπου στο $\Theta^{(r+1)}(x) = B^{(r+1)}(x) \cdot S(x)$ ανατίθεται είτε το $\Delta^{(r)}(x) = \Lambda^{(r)}(x) \cdot S(x)$, είτε το $x\Theta^{(r)}(x) = xB^{(r)}(x) \cdot S(x)$. Με άλλα λόγια τα πολυώνυμα $\Delta^{(r+1)}(x)$ και $\Theta^{(r+1)}(x)$ υπολογίζονται με τον ίδιο τρόπο που υπολογίζονταν και τα $\Lambda^{(r+1)}(x)$ και $B^{(r+1)}(x)$ στον iBM. Οι συντελεστές και των τεσσάρων πολυωνύμων μπορούν να υπολογιστούν ταυτόχρονα και έτσι στον επόμενο κύκλο ρολογιού είναι διαθέσιμοι όλοι οι συντελεστές τους, καθώς και το $\delta_{r+1}^{(r+1)}$ [XXII].

Μετά από $2t$ επαναλήψεις του αναδιατυπωμένου αλγορίθμου riBM κατασκευάζονται τα πολυώνυμα $\Lambda^{(2t)}(x)$ και $\Delta^{(2t)}(x)$. Εφόσον $\Omega(x) = \Lambda(x) \cdot S(x)$ οι συντελεστές χαμηλότερου βαθμού του πολυωνύμου $\Delta^{(2t)}(x)$ αποτελούν τους συντελεστές του $\Omega(x)$. Έτσι, δεν απαιτείται το βήμα iBM.4 και τα πολυώνυμα $\Lambda(x)$ και $\Omega(x)$ υπολογίζονται ταυτόχρονα σε $2t$ κύκλους [XXII].

Εφόσον η ενημέρωση των συντελεστών των τεσσάρων πολυωνύμων είναι πανομοιότυπη, οι διαφορές $\delta_i^{(r)}$ μπορούν να υπολογιστούν με μπλοκ ELU, όπως αυτό που χρησιμοποιείται και στον iBM. Η διαφορά $\delta_r^{(r)}$ υπολογίζεται στον επεξεργαστή $PE0_r$ και έτσι απαιτούνται πολυπλέκτες για την δρομολόγηση των τιμών στην μονάδα ελέγχου, καθώς και στο μπλοκ ELU που υπολογίζει τα πολυώνυμα $\Lambda^{(r+1)}(x)$ και $B^{(r+1)}(x)$. Με μια τελευταία αναδιατύπωση του iBM εξαλείφονται και αυτοί οι πολυπλέκτες [XXII].

Για κάθε $i < r$, οι συντελεστές $\delta_i^{(r)}$και $\theta_i^{(r)}$δεν επηρεάζουν τις τιμές των διαφορών $\delta_{r+j}^{(r+j)}$. Συνεπώς δεν χρειάζεται να αποθηκευτούν οι συντελεστές $\delta_i^{(r)}$και $\theta_i^{(r)}$για $i < r$. [XXII]

Για $r = 0, 1, \cdots, 2t-1$ ορίζονται τα $\hat{\delta}_i^{(r)} = \delta_{i+r}^{(r)}$ και $\hat{\theta}_i^{(r)} = \theta_{i+r}^{(r)}$, καθώς και τα πολυώνυμα:

$$\hat{\Delta}^{(r)}(x) = \sum_{i=0}^{2t-1} \hat{\delta}_i^{(r)} x^i, \qquad \hat{\Theta}^{(r)}(x) = \sum_{i=0}^{2t-1} \hat{\theta}_i^{(r)} x^i$$

που αρχικοποιούνται ως $\hat{\Delta}^{(0)}(x) = \hat{\Theta}^{(0)}(x) = S(x)$ [XXII].

Οι συντελεστές του πολυωνύμου $\hat{\Delta}^{(r+1)}(x)$ ενημερώνονται ως εξής:

$$\begin{aligned} \hat{\delta}_i^{(r+1)} &= \delta_{i+1+r}^{(r+1)} = \gamma^{(r)} \cdot \delta_{i+1+r}^{(r)} - \delta_r^{(r)} \cdot \theta_{i+r}^{(r)} \\ &= \gamma^{(r)} \cdot \hat{\delta}_{i+1}^{(r)} - \hat{\delta}_0^{(r)} \cdot \hat{\theta}_i^{(r)} \end{aligned} \tag{54}$$

ενώ οι συντελεστές του $\hat{\Theta}(x)$, $\hat{\theta}_i^{(r+1)} = \theta_{i+1+r}^{(r+1)}$ λαμβάνουν είτε την τιμή $\hat{\delta}_i^{(r)} = \delta_{i+1+r}^{(r+1)}$, είτε την τιμή $\hat{\theta}_i^{(r)} = \theta_{i+1+r}^{(r+1)}$. Αξίζει να αναφερθεί πως με αυτόν τον τρόπο ενημέρωσης των συντελεστών, η διαφορά $\delta_r^{(r)}$ που ισούται με $\hat{\delta}_0^{(r)}$ βρίσκεται σταθερά στην θέση 0. Έτσι, επιτυγχάνεται σημαντική απλοποίηση της υλοποίησης του αλγορίθμου στο υλικό [XXII].

Τα βήματα του αλγορίθμου riBM συνοψίζονται ως εξής [XXII]:

1. Αρχικοποίηση: $\lambda^{(0)}(0) = b^{(0)}(0) = 1, \;\; \lambda^{(0)}(i) = b^{(0)}(i) = 0, \;\; i = 1, 2, \cdots, t$
   $\hat{\delta}_i^{(0)} = \hat{\theta}_i^{(0)} = s_i, \;\; (i = 0, 1, \cdots, 2t-1) - s_i$ είσοδος
   $k^{(0)} = 0, \qquad \gamma^{(0)} = 1$
2. Για $r = 0$ εώς $2t - 1$:

   Βήμα riBM.1 $\lambda_i^{(r+1)} = \gamma^{(r)} \cdot \lambda_i^{(r)} - \hat{\delta}_0{}^{(r)} \cdot b_{i-1}{}^{(r)}, \;\; (i = 0, 1, \cdots, t)$

   $\hat{\delta}_i^{(r+1)} = \gamma^{(r)} \cdot \hat{\delta}_{i+1}^{(r)} - \hat{\delta}_0^{(r)} \cdot \hat{\theta}_i^{(r)}, \;\; (i = 0, 1, \cdots, 2t-1)$

   Βήμα riBM.2 Αν $\hat{\delta}_0^{(r)} \neq 0$ και $k^{(r)} \geq 0$ τότε :

   $b_i^{(r+1)} = \lambda_i^{(r)}, \;\; (i = 0, 1, \cdots, t)$

   $\hat{\theta}_i^{(r+1)} = \hat{\delta}_{i+1}^{(r)}, \;\; (i = 0, 1, \cdots, 2t-1)$

   $\gamma^{(r+1)} = \hat{\delta}_0^{(r)}$

   $k^{(r+1)} = -k^{(r)} - 1$

   Αλλιώς :

   $b_i^{(r+1)} = b_{i-1}^{(r)}, \;\; (i = 0, 1, \cdots, t)$

   $\hat{\theta}_i^{(r+1)} = \hat{\theta}_i^{(r)}, \;\; (i = 0, 1, \cdots, 2t-1)$

   $\gamma^{(r+1)} = \gamma^{(r)}$

   $k^{(r+1)} = k^{(r)} + 1$

3. Έξοδος:
   $\lambda_i^{(2t)}, \;\; (i = 0, 1, \cdots, t)$
   $\omega_i^{(2t)} = \hat{\delta}_i^{(2t)}, \;\; (i = 0, 1, \cdots, t-1)$

Οι απαιτήσεις σε υλικό του riBM είναι $2t$ επεξεργαστές PE1, που αντιστοιχούν σε $4t$ μανδαλωτές, $4t$ πολλαπλασιαστές, $2t$ αθροιστές και $2t$ πολυπλέκτες, καθώς και μια μονάδα ελέγχου. Η καθυστέρηση του κρίσιμου μονοπατιού είναι ίση με την καθυστέρηση του μπλοκ ELU:

$$T_{riBM} = T_{MULT} + T_{ADD}$$

που είναι λιγότερο από το μισό της καθυστέρησης του iBM [XXII].

Στο Σχήμα 6-4 της επόμενης σελίδας παρουσιάζεται συστολική αρχιτεκτονική υλοποίησης του αλγορίθμου riBM στο υλικό με αρκετά απλοποιημένο τρόπο.

Σχήμα 6-4 - Συστολική Αρχιτεκτονική Αλγορίθμου riBM για Υπολογισμό Λ(x) και Ω(x)

6.2.4.4 Αναδιατυπωμένος BM χωρίς αντίστροφο (RiBM)

Η αρχιτεκτονική riBM αποτελείται από δυο μπλοκ υπολογισμού: ELU και rDC. Μπορεί να εξαλειφθεί το μπλοκ ELU, αντικαθιστώντας το rDC με ένα βελτιωμένο μπλοκ στο οποίο η συστοιχία $2t$ επεξεργαστών PE1 επιμηκύνεται σε συστοιχία $3t+1$ επεξεργαστών. Με αυτόν τον τρόπο κατασκευάζεται πλήρως ομογενής συστολική αρχιτεκτονική, μέσω της οποίας υπολογίζονται και τα δύο πολυώνυμα, $\Lambda(x)$ και $\Omega(x)$, χρησιμοποιώντας μόνο μια συστοιχία επεξεργαστών. Με άλλα λόγια οι $t+1$ επεξεργαστές τύπου PE0 επανεμφανίζονται ως PE1 επεξεργαστές. Αυτή η τροποποιημένη αρχιτεκτονική ονομάζεται RiBM και χρησιμοποιείται στις υλοποιήσεις της πτυχιακής εργασίας που αναλύονται στο Κεφάλαιο 8. Η αρχιτεκτονική αυτή παρουσιάζει την ίδια πολυπλοκότητα υλικού, καθώς και την ίδια καθυστέρηση κρίσιμου μονοπατιού όπως η riBM. Ωστόσο, λόγω της δομής της προσφέρει πλεονεκτήματα στις διατάξεις κυκλωμάτων VLSI και έτσι προτιμάται [XXII]. Ομογενής συστολική αρχιτεκτονική του RiBM παρουσιάζεται στο Σχήμα 6-5 της επόμενης σελίδας .

Σχήμα 6-5 - Ομογενής Συστολική Αρχιτεκτονική Αλγορίθμου RiBM

Μια συστοιχία PE1 επεξεργαστών της αρχιτεκτονικής RiBM χρησιμοποιείται για την εκτέλεση των ίδιων πράξεων με την συστοιχία PE0 επεξεργαστών της αρχιτεκτονικής riBM, ωστόσο απαιτείται επιπλέον αριστερή ολίσθηση των συντελεστών σε κάθε παλμό ρολογιού. Οι επεξεργαστές $PE1_0$, $PE1_1$, …, $PE1_{2t-1}$ αρχικοποιούνται όπως και στην αρχιτεκτονική riBM με τις τιμές των $2t$ συνδρόμων, ενώ οι $PE1_{2t}$, $PE1_{2t+1}$, …, $PE1_{3t-1}$ με μηδενικά. Τέλος, ο επεξεργαστής $PE1_{3t}$ αρχικοποιείται με άσσους, 1 του $GF(2^m)$ [XXII].

Στις $2t$ επαναλήψεις του αλγορίθμου τα πολυώνυμα $\hat{\Delta}^{(r)}(x)$ και $\hat{\Theta}^{(r)}(x)$ ενημερώνονται ξεκινώντας από το αριστερό μέρος της συστοιχίας επεξεργαστών (ενημερώνονται και ολισθαίνουν αριστερά). Μετά από τους $2t$ κύκλους ρολογιού, οι συντελεστές του πολυωνύμου $\Omega(x)$ βρίσκονται στους επεξεργαστές $PE1^0$ - $PE1^{t-1}$. Ο επεξεργαστής $PE1_{3t}$ περιέχει αρχικά τους συντελεστές $\lambda_0^{(0)}$ και $b_0^{(0)}$. Κατά την διάρκεια των επαναλήψεων τα πολυώνυμα $\Lambda^{(r)}(x)$ και $B^{(r)}(x)$ ολισθαίνουν αριστερά και οι συντελεστές $\lambda_i^{(r)}$ και $b_i^{(r)}$ αποθηκεύονται στον επεξεργαστή $PE1_{3t-r+i}$. Μετά το πέρας $2t$ επαναλήψεων ο επεξεργαστής $PE1_{t+i}$ περιέχει έτσι τους συντελεστές $\lambda_i^{(2t)}$ και $b_i^{(2t)}$, όπου $i = 0, 1, \dots, t$ [XXII].

Εκτελούνται δύο παράλληλοι υπολογισμοί που δεν παρεμβάλλονται ο ένας στον άλλο. Τα πολυώνυμα $\Lambda^{(r)}(x)$ και $B^{(r)}(x)$ αποθηκεύονται στους επεξεργαστές PE1 με δείκτη $3t + r$ και

πάνω. Εφόσον ο βαθμός του $\Delta^{(r)}(x)$ ισούται με: $\deg\left(\Delta^{(r)}(x)\right) = \deg(S(x)) + \deg\left(\Lambda^{(r)}(x)\right)$, μπορούμε να συμπεράνουμε πως: $\deg\left(\hat{\Delta}^{(r)}(x)\right) \leq 2t - 1 - r + l(r)$, όπου $l(r) = \frac{r-k(r)}{2}$ είναι το γνωστό πάνω όριο του $\deg\left(\Lambda^{(r)}(x)\right)$. Το $l(r)$ αποτελεί αύξουσα συνάρτηση ως προς το r με μέγιστη τιμή $l(2t) = e$, όπου e ο αριθμός των λαθών. Έτσι, $2t - 1 - r + l(r) < 3t - r$ για $0 \leq r \leq 2t$ και καθώς τα πολυώνυμα $\Lambda^{(r)}(x)$ και $B^{(r)}(x)$ ολισθαίνουν αριστερά δεν επηρεάζουν τους συντελεστές των πολυωνύμων $\hat{\Delta}^{(r)}(x)$ και $\hat{\Theta}^{(r)}(x)$.

Τα βήματα του αλγορίθμου RiBM μπορούν να συνοψιστούν ως εξής [XXII]:

1. Αρχικοποίηση: $\hat{\delta}_{3t}^{(0)} = 1, \quad \hat{\delta}_i^{(0)} = 0, \quad (i = 2t, 2t+1, \dots, 3t-1)$
   $\hat{\delta}_i^{(0)} = \hat{\theta}_i^{(0)} = s_i, \quad (i = 0, 1, \cdots, 2t-1) - s_i$ είσοδος
   $k^{(0)} = 0, \qquad \gamma^{(0)} = 1$
2. Για $r = 0$ εώς $2t - 1$:

   Βήμα RiBM.1 $\hat{\delta}_i^{(r+1)} = \gamma^{(r)} \cdot \hat{\delta}_{i+1}^{(r)} - \hat{\delta}_0^{(r)} \cdot \hat{\theta}_i^{(r)}, \quad (i = 0, 1, \cdots, 3t)$

   Βήμα RiBM.2 Αν $\hat{\delta}_0^{(r)} \neq 0$ και $k^{(r)} \geq 0$ τότε :

   $\hat{\theta}_i^{(r+1)} = \hat{\delta}_{i+1}^{(r)}, \quad (i = 0, 1, \cdots, 3t)$

   $\gamma^{(r+1)} = \hat{\delta}_0^{(r)}$

   $k^{(r+1)} = -k^{(r)} - 1$

   Αλλιώς :

   $\hat{\theta}_i^{(r+1)} = \hat{\theta}_i^{(r)}, \quad (i = 0, 1, \cdots, 3t)$

   $\gamma^{(r+1)} = \gamma^{(r)}$

   $k^{(r+1)} = k^{(r)} + 1$

3. Έξοδος: $\lambda_i^{(2t)} = \hat{\delta}_{t+i}^{(2t)}, \quad (i = 0, 1, \cdots, t), \quad \omega_i^{(2t)} = \hat{\delta}_i^{(2t)}, \quad (i = 0, 1, \cdots, t-1)$

Ο συντελεστής $\delta_{3t+1}^{(r)}$ ισούται με 0 για όλες τις τιμές r και επομένως δεν χρειάζεται να αποθηκευτεί ή να ενημερωθεί [XXII].

6.2.4.5 Αποσυντεθειμένος BM χωρίς αντίστροφο (DiBM)

Ο αποσυντεθειμένος (decomposed) BM χωρίς αντίστροφο που συμβολίζεται ως DiBM υπολογίζει και αυτός με την σειρά του τα πολυώνυμα $\Lambda(x)$ και $\Omega(x)$.

Το πολυώνυμο $\Lambda(x)$ υπολογίζεται με τον ακόλουθο επαναληπτικό αλγόριθμο $2t$ κύκλων [XVIII]:

1. Αρχικοποίηση: $D^{(-1)} = 0, \quad \delta = 1, \quad \Lambda^{(-1)}(x) = T^{(-1)}(x) = 1, \quad \Delta^{(0)} = S_1$
2. Για $i = 0$ έως $2t - 1$:
   $\Lambda^{(i)}(x) = \delta \cdot \Lambda^{(i-1)}(x) + \Delta^{(i)} x T^{(i-1)}(x)$
   $\Delta^{(i+1)} = S_{i+2}\Lambda_0^{(i)} + S_{i+1}\Lambda_1^{(i)} + \cdots + S_{i-t+2}\Lambda_t^{(i)}$
   Αν $(\Delta^{(i)} = 0$ ή $2D^{(i-1)} \geq i + 1)$:

$$D^{(i)} = D^{(i-1)}, \quad T^{(i)}(x) = xT^{(i-1)}(x)$$

Αλλιώς:

$$D^{(i)} = i + 1 - D^{(i-1)}, \quad \delta = \Delta^{(i)}, \quad T^{(i)}(x) = \Lambda^{(i-1)}(x)$$

Οι συντελεστές του πολυωνύμου $\Omega(x)$ βρίσκονται από την βασική εξίσωση μέσω της εξίσωσης:

$$\omega_i = S_{i+1}\Lambda_0 + S_i\Lambda_1 + \cdots + S_1\Lambda_i, \quad i = 0, 1, \ldots, t-1 \tag{55}$$

Ο υπολογισμός των συντελεστών του $\Omega(x)$ παρουσιάζει μικρές διαφορές με τον υπολογισμό του $\Delta^{(i)}$. Συνεπώς, μπορεί να χρησιμοποιηθεί το ίδιο υλικό με το οποίο υπολογίζεται το $\Lambda(x)$ και για τον υπολογισμό του πολυωνύμου $\Omega(x)$. Ακόμη, μειώνετε αισθητά η πολυπλοκότητα υλικού σε σχέση με τον κλασσικό BM, καθώς χρησιμοποιούνται μόνο 3 πολλαπλασιαστές FFM για τον υπολογισμό των $\Lambda(x)$ και $\Omega(x)$ [XVIII].

Για την καλύτερη κατανόηση της αρχιτεκτονικής ορίζονται:

$$\Lambda_j^{(i)} = \begin{cases} \delta \cdot \Lambda_0^{(i-1)}, & j = 0 \\ \delta \cdot \Lambda_{j-1}^{(i-1)} + \Delta^{(i)} T_{j-1}^{(i-1)}, & 1 \le j \le t \end{cases} \tag{56}$$

$$\Lambda_j^{(i)} = \begin{cases} \delta \cdot \Lambda_0^{(i-1)}, & j = 0 \\ \delta \cdot \Lambda_{j-1}^{(i-1)} + \Delta^{(i)} T_{j-1}^{(i-1)}, & 1 \le j \le t \end{cases} \tag{57}$$

όπου $\Delta_j^{(i+1)}$ τα μερικά αποτελέσματα που προκύπτουν κατά τον υπολογισμό του $\Delta^{(i+1)}$ [XVIII].

Κατά τον υπολογισμό του $\Delta^{(i+1)}$ στον κύκλο υπολογισμό $i+1$:

$$\begin{aligned} \Delta^{(i+1)} &= \Delta_t^{(i+1)} + S_{i-t+2}\Lambda_t^{(i)} \\ &= \cdots \\ &= S_{i+2}\Lambda_0^{(i)} + S_{i+1}\Lambda_1^{(i)} + \cdots + S_{i-t+2}\Lambda_t^{(i)} \end{aligned} \tag{58}$$

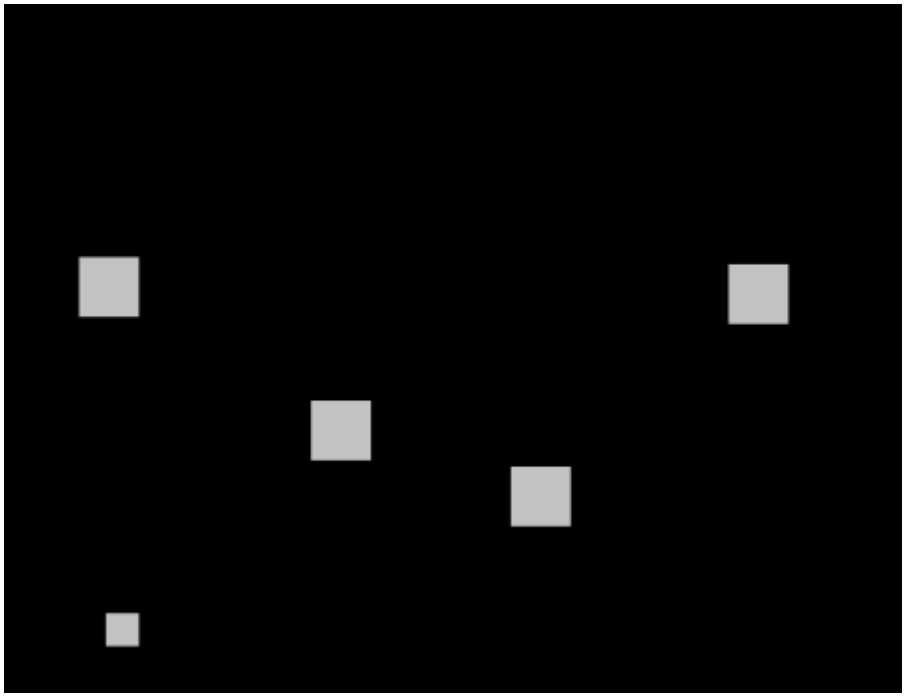
Με άλλα λόγια η κάθε επανάληψη μπορεί να αποσυντεθεί σε $t+2$ κύκλους υπολογισμού. Σε κάθε κύκλο ο υπολογισμό του $\Lambda_j^{(i)}$ απαιτεί το πολύ 2 πολλαπλασιαστές, ενώ ο υπολογισμός του $\Delta_j^{(i+1)}$ μόνο 1 πολλαπλασιαστή. Στον κύκλο j , για τον υπολογισμό του $\Delta_j^{(i+1)}$ χρειάζονται μόνο οι τιμές των $\Lambda_{j-1}^{(i)}$και $\Delta_{j-1}^{(i+1)}$ που έχουν υπολογιστεί στον κύκλο $(j-1)$. Κατά παρόμοιο τρόπο στον ίδιο κύκλο ο υπολογισμός του $\Lambda_j^{(i)}$απαιτεί μόνο τα $\Delta^{(i)}$ και $\Lambda_j^{(i-1)}$ που έχουν υπολογιστεί στον κύκλο 0 και το βήμα $(i-1)$ αντίστοιχα [XVIII]. Η αρχιτεκτονική του DiBM παρουσιάζεται στο Σχήμα 6-6 της επόμενης σελίδας, όπου υπολογίζεται το πολυώνυμο $\Lambda(x)$.

Στο Σχήμα 6-7 υπολογίζεται με το ίδιο υλικό και το πολυώνυμο $\Omega(x)$. Ο υπολογισμός του $\Omega(x)$ ξεκινά αμέσως μετά τον υπολογισμό του $\Lambda(x)$. Το υλικό του DiBM απαιτεί αναδιαμόρφωση για τον υπολογισμό του $\Omega(x)$. Τα $\Delta_j^{(i)}$και $\Omega_j^{(i)}$ αποτελούν ενδιάμεσα αποτελέσματα για τον υπολογισμό του $\Omega^{(i)}$ και ορίζουμε [XVIII]:

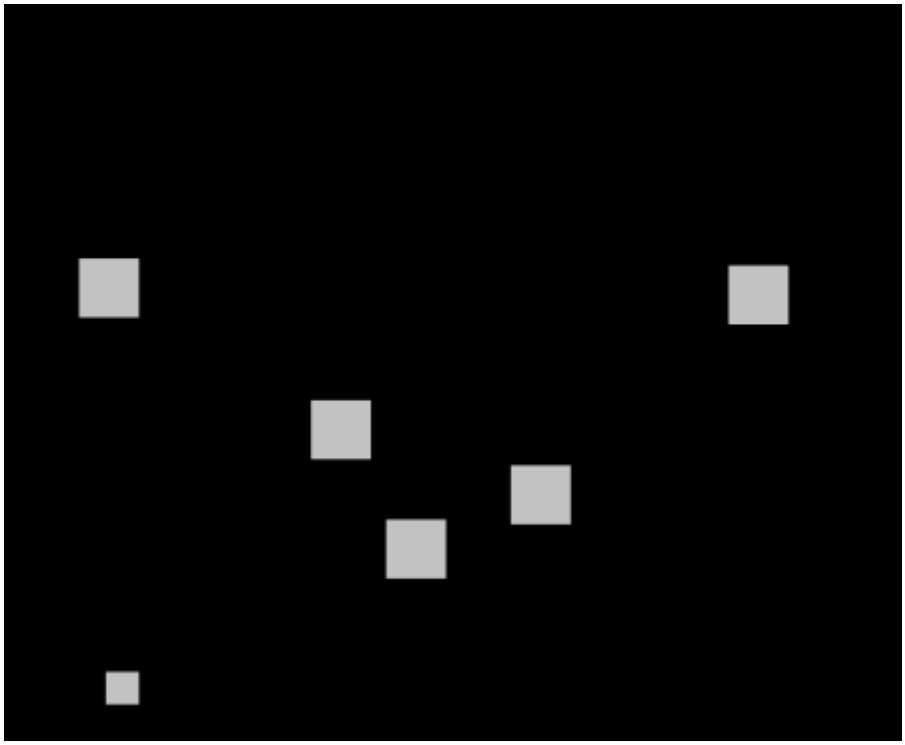
$$\Omega_j^{(i)} = \begin{cases} s_{i+1} \cdot \Lambda_0, & j = 0 \\ \Omega_{j-1}^{(i)} + s_{i-j+1} \cdot \Lambda_j, & 1 \le j \le i \end{cases} \tag{59}$$

Στην τελευταία επανάληψη ολοκληρώνεται ο υπολογισμός του πολυωνύμου $\Omega(x)$ [XVIII]:

$$\Omega_j^{(i)} = \Omega_{j-1}^{(i)} + s_1 \Lambda_i = \cdots = \Omega^{(i)}$$



Σχήμα 6-6 - Αρχιτεκτονική Αλγορίθμου DiBM για υπολογισμό Λ(x)



Σχήμα 6-7 - Αρχιτεκτονική Αλγορίθμου DiBM για υπολογισμό Ω(x)

6.2.4.6 BM διπλής γραμμής (Dual-Line)

Ο αλγόριθμος Berlekamp-Massey μπορεί να υλοποιηθεί με χρήση δύο διαφορετικών δομών. Η μια δομή είναι παράλληλη, το οποίο σημαίνει πως κάθε επανάληψη του αλγορίθμου απαιτεί ένα κύκλο ρολογιού. Η δεύτερη δομή είναι σειριακή. Σ 'αυτήν κάθε επανάληψη απαιτεί $2(t+1)$ κύκλους ρολογιού. Σε και τις δύο δομές όμως, τα σύνδρομα και το πολυώνυμο $\Lambda^{(i)}(x)$ αποθηκεύονται σε καταχωρητές και η διαφορά Δ_i υπολογίζεται από τις τιμές αυτές. Στην αρχιτεκτονική διπλής γραμμής οι καταχωρητές λαμβάνουν ενδιάμεσο τελούμενο $\Delta_k^{(i)}$ το οποίο γίνετε Δ_k όταν $\Lambda^{(k-1)}(x) = \Lambda^{(i)}(x)$. Με άλλα λόγια χρησιμοποιώντας αυτήν την νέα δομή απαλείφεται το άθροισμα στον υπολογισμό της διαφοράς Δ_i και μειώνεται τελικώς το κρίσιμο μονοπάτι. Τα $\Delta_k^{(i-1)}$ ορίζονται ως εξής [XXIII]:

$$\Delta_k^{(i-1)} = \sum_{j=0}^{k-1} \Lambda_j^{(i-1)} S_{k-j} \tag{60}$$

Τα $\Delta_k^{(i-1)}$ αποτελούν ενδιάμεσα τελούμενα Δ_k όταν $i = k$. Τα Δ_k υπολογίζονται από τα $\Lambda_j^{(i)}$και S_{k-j}, ενώ τα $\Delta_k^{(i)}$ υπολογίζονται από τα $\Delta_k^{(i-1)}$ σύμφωνα με την εξίσωση [XXIII]:

$$\Delta_k^{(i)} = \varepsilon^{(i-1)} \cdot \Delta_k^{(i-1)} + \sum_{j=0}^{k-1} \Delta_i \cdot B_{j-1}^{(i-1)} \cdot S_{k-j} \tag{61}$$

Αν η τελευταία τροποποίηση του πολυωνύμου $B^{(i)}(x)$ πραγματοποιείται στον l-οστό κύκλο ή μπορούμε να πούμε και πως $\Delta_l \neq 0$ και $2L_{l-1} \leq l-1$, τότε $B_{j-1}^{(i-1)} = \Lambda_{j-i+l}^{(l-1)}$ και η από πάνω εξίσωση αλλάζει σε [XXIII]:

$$\Delta_k^{(i)} = \varepsilon^{(i-1)} \cdot \Delta_k^{(i-1)} + \Delta_i \Delta_{k'}^{(l-1)}, \qquad k' = k - i + l \tag{62}$$

Σύμφωνα με αυτές τις εξισώσεις διαπιστώνεται πως τα $\Delta_k^{(i)}$ μπορούν να υπολογιστούν από τα $\Delta_k^{(i-1)}$ και $\Delta_k^{(l-1)}$. Οι επάνω εξισώσεις υλοποιούνται με δύο σειρές καταχωρητών C και D [XXIII]. Παράδειγμα για την περίπτωση κώδικα RS με ικανότητα διόρθωσης λαθών $t = 5$ παρουσιάζεται στο Σχήμα 6-8.

Σχήμα 6-8 - Αρχιτεκτονική Berlekamp-Massey Διπλής Γραμμής για t = 5

Οι τιμές $C_k^{(i)}$ των καταχωρητών C υπολογίζονται ως εξής [XXIII]:

$$C_k^{(i)} = \Delta_{k+i+1}^{(i)} = \varepsilon^{(i-1)} \cdot C_{k+1}^{(i-1)} + \Delta_i C_{k+1}^{(l-1)}, \qquad \Delta_i = \Delta_{k+i+1}^{(i-1)} = C_0^{(i-1)} \tag{63}$$

Οι τιμές $D_k^{(i)}$ των καταχωρητών D ορίζονται με τον εξής τρόπο [XXIII]:

$$D_k^{(i)} = \begin{cases} C_k^{(i-1)}, & \Delta_i \neq 0 \; και \; 2L_{i-1} \leq i-1 \\ D_k^{(i-1)}, & αλλιώς \end{cases} \tag{64}$$

Έτσι, τα $C_k^{(i)}$ τροποποιούνται ως εξής [XXIII]:

$$C_k^{(i)} = \begin{cases} \varepsilon^{(i-1)} \cdot C_{k+1}^{(i-1)}, & \Delta_i = 0 \\ \varepsilon^{(i-1)} \cdot C_{k+1}^{(i-1)} + \Delta_i D_{k+1}^{(l-1)}, & αλλιώς \end{cases} \tag{65}$$

Μια απευθείας υλοποίηση των παραπάνω εξισώσεων απαιτεί $6t$ καταχωρητές, $2t$ για τους C, $2t$ για τους D και $2t$ για τον υπολογισμό του $\Lambda_j^{(i)}$. Ωστόσο, με μια επιπλέον τροποποίηση απαιτούνται τελικώς μόνο $4t$ καταχωρητές. Τα $\Delta_k^{(i-1)}$ αποθηκεύονται σε $2t-(i-1)$ καταχωρητές και έτσι για $k \geq 2t-(i-1)$ τα $C_k^{(i-1)}$ έχουν τιμή 0. Αυτοί οι καταχωρητές μπορούν να χρησιμοποιηθούν επομένως για τον υπολογισμό των $\Lambda_j^{(i)}$ [XXIII].

Οι εξισώσεις των καταχωρητών D τροποποιούνται σε [XXIII]:

$$D_k^{(i)} = \begin{cases} 0, & k = 2t-i \\ C_k^{(i-1)}, & (k \neq 2t-i) \; και \; (\Delta_i \neq 0) \; και \; (2L_{i-1} \leq i-1) \\ D_k^{(i-1)}, & αλλιώς \end{cases} \tag{66}$$

Οι εξισώσεις των καταχωρητών C δεν τροποποιούνται περαιτέρω. Όταν $i = 2t$, το $\Lambda(x)$ λαμβάνεται από την ανάθεση [XXIII]:

$$\Lambda_j = C_j, \quad 0 \leq j \leq t.$$

Η υλοποίηση διπλής γραμμής παρουσιάζει μικρή καθυστέρηση $2t$ κύκλων. Επίσης, έχει πολύ χαμηλή καθυστέρηση κρίσιμου μονοπατιού $(T_{MULT} + T_{ADD})$ και είναι εξαιρετικά απλή στην υλοποίησή της. Με μικρή αύξηση των απαιτήσεων υλικού υπερτερεί της βασικής BM αρχιτεκτονικής, καθώς και τον σειριακών και παράλληλων υλοποιήσεων του βασικού αλγορίθμου, καθώς αυτές οι υλοποιήσεις δεν παρουσιάζουν ομοιομορφία και έτσι υλοποιούνται δύσκολα [XVII][XXIII].

6.3 Αλγόριθμος Chien Search για εύρεση θέσεων λάθους

Μετά την εύρεση του πολυωνύμου $\Lambda(x)$, επόμενο βήμα αποτελεί η εύρεση των θέσεων που παρουσιάζονται τα λάθη. Στην Ενότητα 6.1.3 οι θέσεις λάθους ορίστηκαν ως X_i. Αναφέρθηκε επίσης πως οι ρίζες του πολυωνύμου $\Lambda(x)$ αποτελούν τους αντίστροφους (reciprocals) των θέσεων λάθους, δηλαδή τα $X_i^{(-1)}$ή $1/X_i$. Άρα, φαίνεται απολύτως φυσιολογικό να υπολογιστούν οι ρίζες του $\Lambda(x)$ και να βρεθούν στην συνέχεια οι αντίστροφοί τους. Ωστόσο, η εύρεση των ριζών του πολυωνύμου $\Lambda(x)$ και των αντίστροφών τους στην συνέχεια αποτελεί μη αποδοτική μέθοδο. Έτσι, για την ανίχνευση των θέσεων λάθους χρησιμοποιείται ο αλγόριθμος του Chien που περιγράφεται στην συνέχεια [XV].

6.3.1 Βασικός αλγόριθμος Chien Search

Βασική ιδέα του Chien αποτελεί η ευκολία που μας παρέχει το σχετικό μικρό μέγεθος των πεπερασμένων πεδίων τα οποία χρησιμοποιούνται. Το μικρό μέγεθος επιτρέπει τον άπληστο (greedy) έλεγχο όλων των στοιχείων του πεδίου $GF(2^8)$ ένα-προς-ένα. Όσα απ' αυτά τα στοιχεία δίνουν αποτέλεσμα μηδέν, στην αξιολόγηση του πολυωνύμου, αποτελούν ρίζα του. Ωστόσο, ο τρόπος με τον οποίο υλοποιείται η αναζήτηση (search) είναι εξαιρετικά αποτελεσματικός [XV].

Για κάθε θέση a^i υπολογίζεται το εξής άθροισμα:

$$\Lambda(a^i) = 1 + \Lambda_1 a^i + \Lambda_2 a^{2i} + \cdots + \Lambda_t a^{ti} \tag{67}$$

που αποτελεί την αξιολόγηση (evaluation) του πολυωνύμου $\Lambda(x)$ για το στοιχείο a^i [XV].

Σύμφωνα με το πάνω άθροισμα ο αλγόριθμος του Chien αξιολογεί το πολυώνυμο $\Lambda(x)$ με χρήση t πολλαπλασιασμών με σταθερά (constant multiplications). Διατηρείται ένα πλήθος t μεταβλητών κατάστασης (state variables) $Q_1, Q_2, \dots, Q_t$, οι οποίες αρχικοποιούνται με τους αντιστοίχους συντελεστές του $\Lambda(x)$, Λ_j, έτσι ώστε στην επανάληψη i [XVI]:

$$Q_j = \Lambda_j a^{ji}, \quad j = 1, 2, \dots, t \tag{68}$$

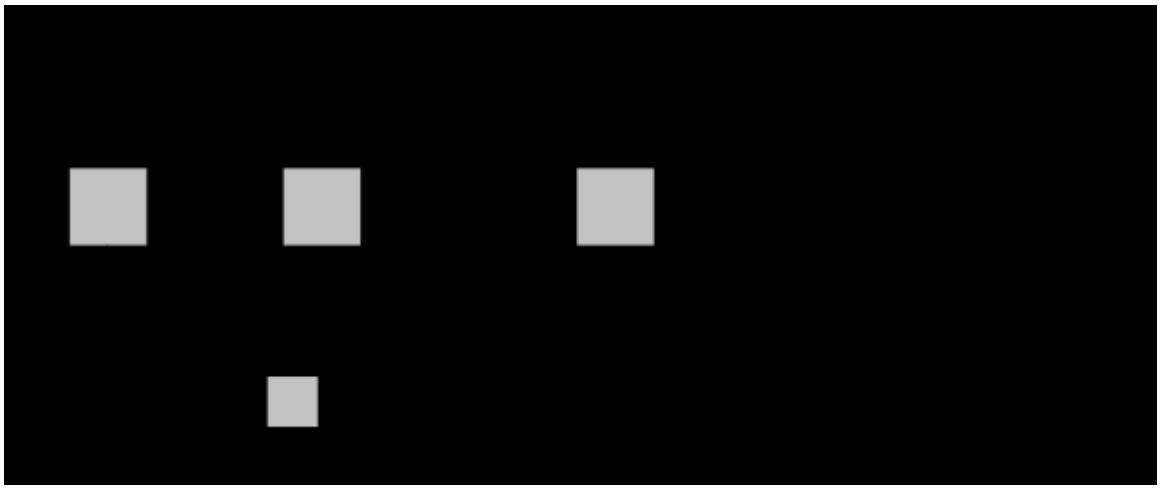
Κάθε μεταβλητή κατάστασης ενημερώνεται με πολλαπλασιασμό επί σταθερά ως εξής [XVI]:

$$Q_j \leftarrow Q_j a^j, \quad j = 1, 2, \dots, v \tag{69}$$

Στην επανάληψη i το άθροισμα των μεταβλητών κατάστασης ισούται με [XVI]:

$$\sum_{j=1}^{t} Q_j = \Lambda(a^i) - 1 \tag{70}$$

Θέση λάθους αναγνωρίζεται όταν το άθροισμα αυτό δώσει αποτέλεσμα -1. Απλή αρχιτεκτονική αυτού του αλγορίθμου, στην οποία προστίθεται το 1 στην αξιολόγηση του $\Lambda(x)$ και έτσι ελέγχει για αποτέλεσμα 0, παρουσιάζεται στο Σχήμα 6-9, όπου το σήμα εξόδου ERRLOC αληθεύει (τιμή 1) μόνο όταν $\Lambda(a^i) + 1 = 0$. Στην περίπτωση που το άθροισμα έχει αποτέλεσμα διαφορετικό του 0, το σήμα εξόδου λαμβάνει τιμή 0. [XVI].



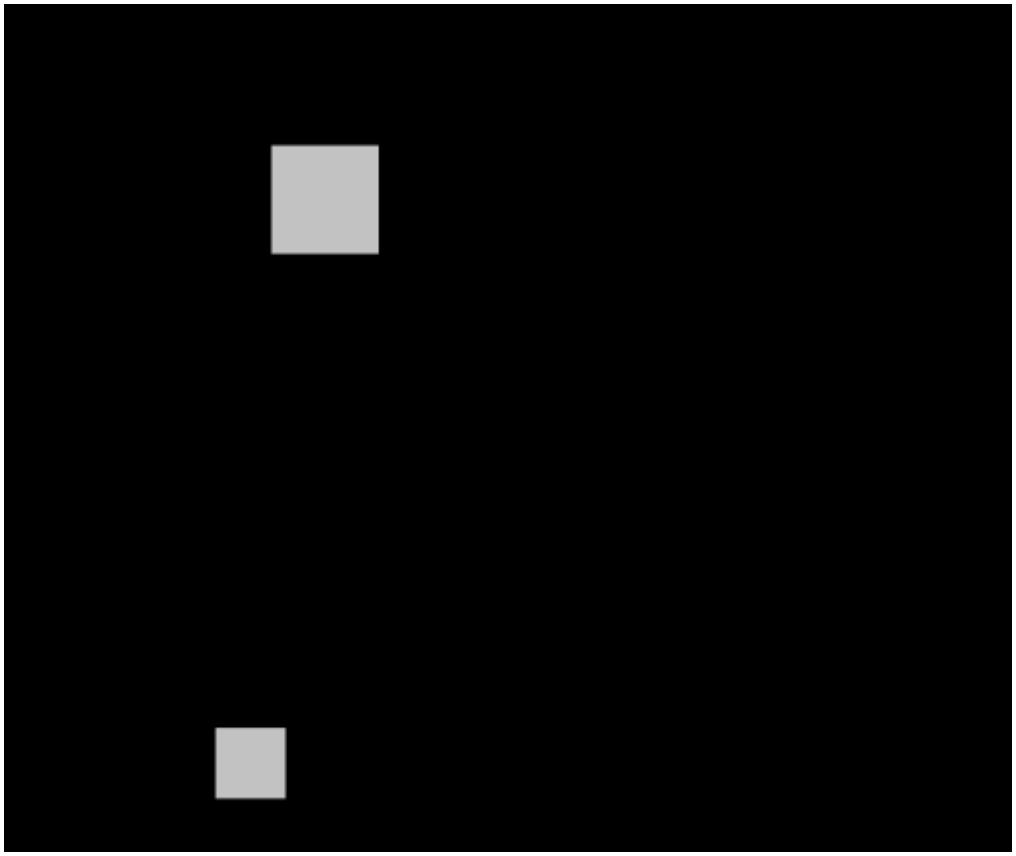
Σχήμα 6-9 – Βασική Αρχιτεκτονική Αλγορίθμου Chien Search για εύρεση θέσεων λάθους

Εφόσον τα μηδενικά του $\Lambda(x)$ αποτελούν τους αντίστροφους των θέσεων λάθους, το ERRLOC αληθεύει για τις θέσεις: $a^{-i} = a^{n-i} = X_i$. Καθώς το i αυξάνεται στο διάστημα $[1,\ n]$, οι θέσεις λάθους ανιχνεύονται από το περισσότερο σημαντικό (most-significant) έως το λιγότερο σημαντικό (least-significant) σύμβολο ή συντελεστή του μηνύματος. Η διαδικασία μπορεί να εκτελεστεί και αντίστροφα, χρησιμοποιώντας τις σταθερές $a^{-1}, \dots, a^{-t}$, αντί των $a^1, \dots, a^t$ [XVI].

6.3.2 Βελτιστοποιήσεις Chien Search

6.3.2.1 Βελτιστοποίηση χρήσης λογικών πυλών

Παραγοντοποιώντας το πολυώνυμο $\Lambda(x)$ ελαχιστοποιείται ο αριθμός των λογικών πυλών που απαιτούνται για την υλοποίηση του Chien, κάτι που επιφέρει και μείωση της κατανάλωσης ρεύματος, συγκριτικά με το βασικό κύκλωμα [XXIV]. Η τροποποιημένη αρχιτεκτονική αναζήτησης Chien παρουσιάζεται στο Σχήμα 6-10.



Σχήμα 6-10 - Αρχιτεκτονική Chien Search βελτιωμένη ως προς τον αριθμό λογικών πυλών

Το πολυώνυμο $\Lambda(x)$ ορίζεται ως:

$$\Lambda(x) = \Lambda_t x^t + \Lambda_{t-1} x^{t-1} + \cdots + \Lambda_1 x + \Lambda_0 \tag{71}$$

Με διαδοχική παραγοντοποίηση ως προς το x προκύπτει:

$$\begin{aligned} \Lambda(x) &= x(\Lambda_t x^{t-1} + \Lambda_{t-1} x^{t-2} + \cdots + \Lambda_1) + \Lambda_0 \\ &= x(x(\Lambda_t x^{t-2} + \Lambda_{t-1} x^{t-3} + \cdots + \Lambda_2) + \Lambda_1) + \Lambda_0 \\ &= x(x(x(\dots x(\Lambda_t\, x + \Lambda_{t-1}) + \Lambda_{t-2}\,) + \cdots) + \Lambda_2) + \Lambda_1) + \Lambda_0 \end{aligned} \tag{72}$$

όπου οι συντελεστές του $\Lambda(x)$, $\Lambda_1, \Lambda_2, \dots, \Lambda_t$ είναι όλοι μηδενικοί [XXIV].

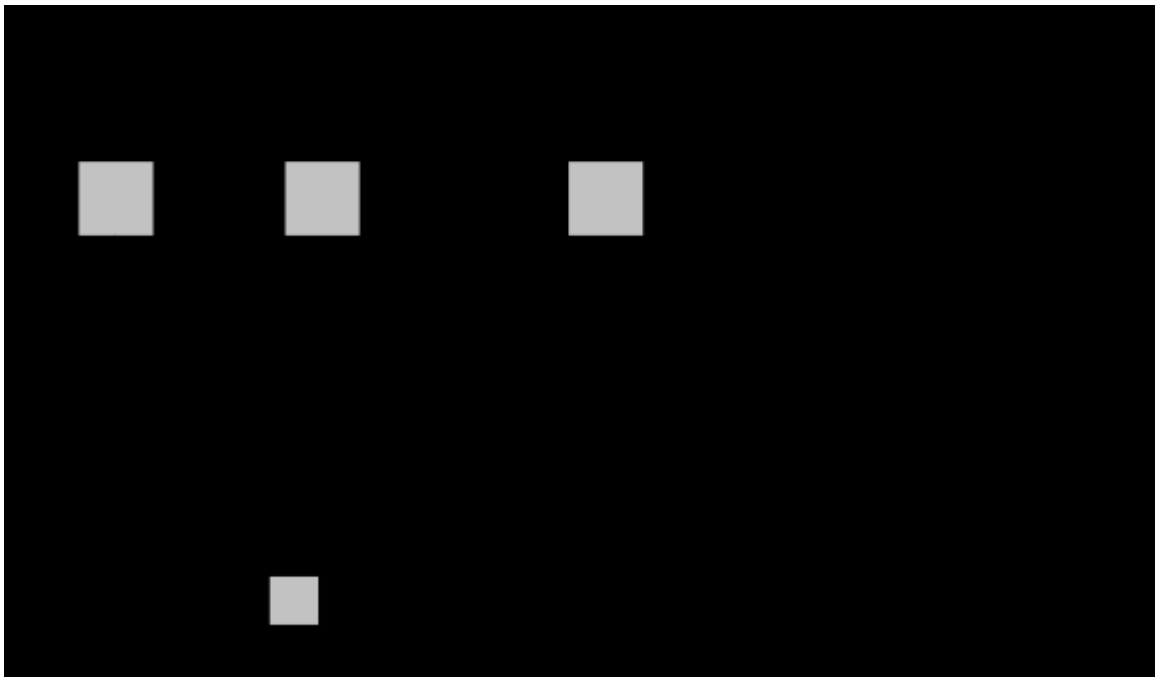
Στον Πίνακα 6-1 παρουσιάζεται η ελάττωση που επιτυγχάνεται για πολυώνυμα $\Lambda(x)$ με βαθμό t. Η μείωση αγγίζει το 50% σε σχέση με το βασικό κύκλωμα [XXIV].

Βαθμός Λ(x)	Λογικές Πύλες βασικού CS	Λογικές πύλες βελτιωμένου CS	Ελάττωση
2	11	7	4
3	15	9	6
⋮	⋮	⋮	⋮
t	$3+4t$	$3+2t$	2t

Πίνακας 6-1 - Αριθμός ελαχιστοποιημένων πυλών Chien Search [XXIV]

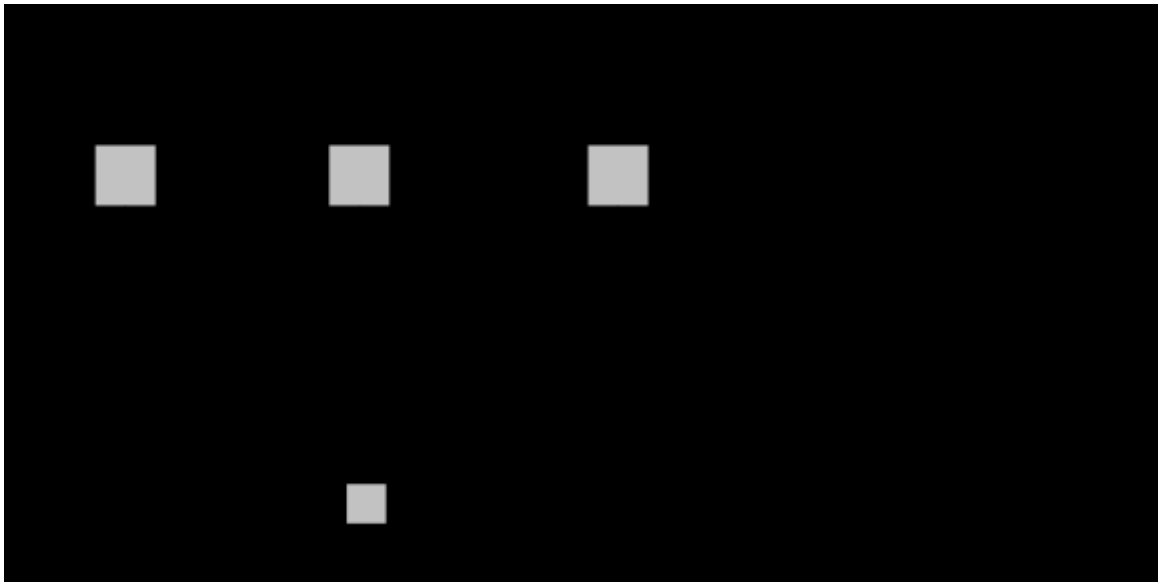
6.3.2.2 2-Παράλληλη Αναζήτηση Chien

Για επιπλέον επιτάχυνση της διαδικασία αναζήτησης θέσεων λάθους ή εύρεσης ριζών του πολυωνύμου $\Lambda(x)$, μπορούν να εκτελεστούν πολλές αναζητήσεις Chien Search παράλληλα. Μια 2-παράλληλη αναζήτηση Chien αξιολογεί ταυτόχρονα τα $\Lambda(a^{2i})$ και $\Lambda(a^{2i+1})$. Αυτό το κύκλωμα παράλληλης αναζήτησης ζυγών (even) και περιττών (odd) θέσεων λάθους υλοποιείται αποδοτικότερα αν δεν χρησιμοποιηθούν δύο αντίγραφα του κλασσικού CS, αλλά πραγματοποιείται κοινή χρήση των στοιχείων μνήμης μεταξύ των δύο παράλληλων αναζητήσεων. Η αρχιτεκτονική αυτή παρουσιάζεται στο Σχήμα 6-11. Το σήμα εξόδου ERRLOC0 αληθεύει για ζυγές θέσεις λάθους, ενώ το ERRLOC1 αληθεύει για περιττές θέσεις λάθους [XVI].



Σχήμα 6-11 - 2-Παράλληλη Αρχιτεκτονική Chien Search

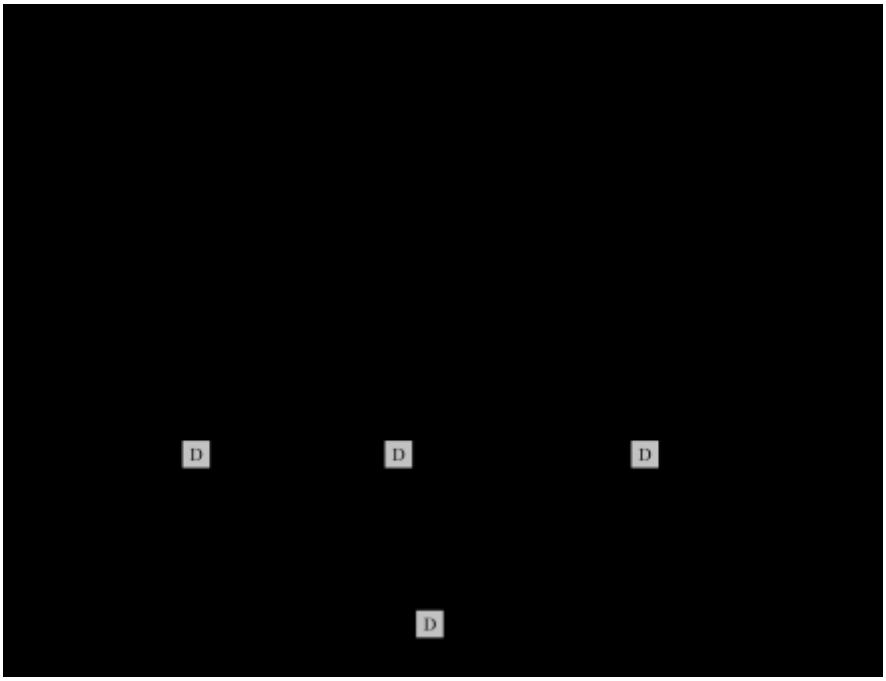
Για να μην χρησιμοποιηθούν σταθερές πολλαπλασιασμού του τύπου a^{2j} που απαιτούν συνήθως μεγαλύτερο αριθμό πυλών, διασπώνται οι μεγάλες δυνάμεις με τον εξής τρόπο: $a^{2j} = a^j \cdot a^j$. Ο υπολογισμός μέσω αυτής της αρχιτεκτονικής απαιτεί λιγότερες πύλες και μειώνει έτσι την καθυστέρηση του κρίσιμου μονοπατιού, επιτρέποντας μεγαλύτερη συχνότητα λειτουργίας [XVI]. Η βελτιστοποίηση αυτή παρουσιάζεται στο Σχήμα 6-12 της επόμενης σελίδας.



Σχήμα 6-12 - 2-Παράλληλη Αρχιτεκτονική Chien Search με διάσπαση των a^{ji}

6.3.2.3 p-Παράλληλη Αναζήτηση Chien

Με 2-παράλληλη αναζήτηση μειώνονται οι κύκλοι αναζήτησης από n σε $n/2$. Αν απαιτείται περαιτέρω μείωση της ολικής καθυστέρησης (latency) του μπλοκ αναζήτησης στην κλίμακα n/p χρησιμοποιούνται p-παράλληλες αρχιτεκτονικές [XVI]. Μια p-Παράλληλη αρχιτεκτονική Chien παρουσιάζεται στο Σχήμα 6-13 της επόμενης σελίδας.



Σχήμα 6-13 - p-Παράλληλη Αρχιτεκτονική Chien Search

Σ' αυτήν την αρχιτεκτονική υπολογίζονται ταυτόχρονα οι αξιολογήσεις του $\Lambda(x)$ για τις τιμές $a^i, a^{2i}, \dots, a^{pi}$, δηλαδή τα $\Lambda(a^i), \Lambda(a^{2i}), \dots, \Lambda(a^{pi})$ [XXV].

Για την επίτευξη κοινής χρήσης πόρων μεταξύ των πολλαπλασιαστών της αρχιτεκτονικής αυτής μπορεί να χρησιμοποιηθεί ο ακόλουθος αλγόριθμος [XXV]:

1. Μετασχηματισμός των συναρτήσεων εξόδου κάθε πολλαπλασιαστή σε μορφή πίνακα (equation matrix)
2. Υπολογισμός πίνακα ομοιότητας (similarity matrix) για κάθε γραμμή και βαθμού ομοιότητας με τις άλλες γραμμές.
3. Διαγραφή όμοιων γραμμών και γραμμών με λιγότερα από δύο '1'. Αύξουσα ταξινόμηση πίνακα ομοιότητας με βάση των αριθμό '1' σε κάθε γραμμή.
4. Εύρεση πλήρως καλυμμένων (covered) γραμμών του πίνακα ομοιότητας και διατύπωση καλυμμένων (covering) σε συνάρτηση με τις πλήρως καλυμμένες (covered) γραμμές.
5. Επανάληψη βήματος 4 για τις συναρτήσεις εξόδου.
6. Έλεγχος για μικρότερο μονοπάτι στις συναρτήσεις εξόδου, υπολογίζοντας το άθροισμα κάθε στήλες του πίνακα ομοιότητας και επιλέγοντας το μονοπάτι με το μικρότερο άθροισμα.

Με την χρήση αυτού του αλγορίθμου πραγματοποιείται βελτιστοποίηση ως προς το μέγεθος και την μέγιστη συχνότητα εκτέλεσης του κυκλώματος [XXV].

6.4 Αλγόριθμος Forney για εύρεση τιμών λάθους

Μετά την εύρεση των θέσεων λάθους, ακολουθεί η εύρεση των τιμών των λαθών αυτών. Όπως αναφέρθηκε και στην Ενότητα 6.1.3 οι τιμές των λαθών ορίζονται ως Y_i, με το Y_i να αντιστοιχεί στην θέση λάθους X_i. Ένας πολύ αποδοτικός αλγόριθμος που εφαρμόζεται για τον υπολογισμό των τιμών αυτών είναι ο αλγόριθμος Forney [XV].

6.4.1 Βασικός Αλγόριθμος Forney

Ο αλγόριθμος του Forney βασίζεται στην αξιολόγηση του πολυωνύμου $\Omega(x)$ για τον αντίστροφο της θέσης λάθους X_k, X_k^{-1}. Εφόσον το X_k^{-1} αποτελεί ρίζα, ισχύει $\Lambda(X_k^{-1}) = 0$, και έτσι:

$$\Omega(X_k^{-1}) = \Lambda(X_k^{-1}) + \sum_{i=1}^{t} X_k^{-1} X_i Y_i \prod_{\substack{j=1 \\ j\neq i}}^{t} (1 - X_j X_k^{-1})$$

$$= Y_k \prod_{\substack{j=1 \\ j\neq k}}^{t} (1 - X_j X_k^{-1}) \tag{73}$$

Σύμφωνα με το παραπάνω, το Y_k υπολογίζεται από την σχέση [XV]:

$$Y_k = \frac{\Omega(X_k^{-1})}{\prod_{\substack{j=1 \\ j\neq l}}^{v} (1 - X_j X_k^{-1})} \tag{74}$$

Ο παρονομαστής αποτελεί αξιολόγηση της παραγώγου του $\Lambda(x)$, $\Lambda'(x)$, για το X_k^{-1} και έτσι τελικώς προκύπτει η εξίσωση της Ενότητας 6.1.3 [XVI]:

$$Y_k = -\frac{\Omega(X_k^{-1})}{\Lambda'(X_k^{-1})} \tag{75}$$

Η παράγωγος $\Lambda'(x)$ υπολογίζεται με τον εξής τρόπο [XIX]:

$$\Lambda'(x) = \frac{d\Lambda(x)}{dx} = \frac{d(\Lambda_0 + \Lambda_1 x + \Lambda_2 x^2 + \cdots + \Lambda_t x^t)}{dx} = \Lambda_1 + 2\Lambda_2 x + 3\Lambda_3 x^2 + \cdots + t\Lambda_t x^{t-1} \quad (76)$$

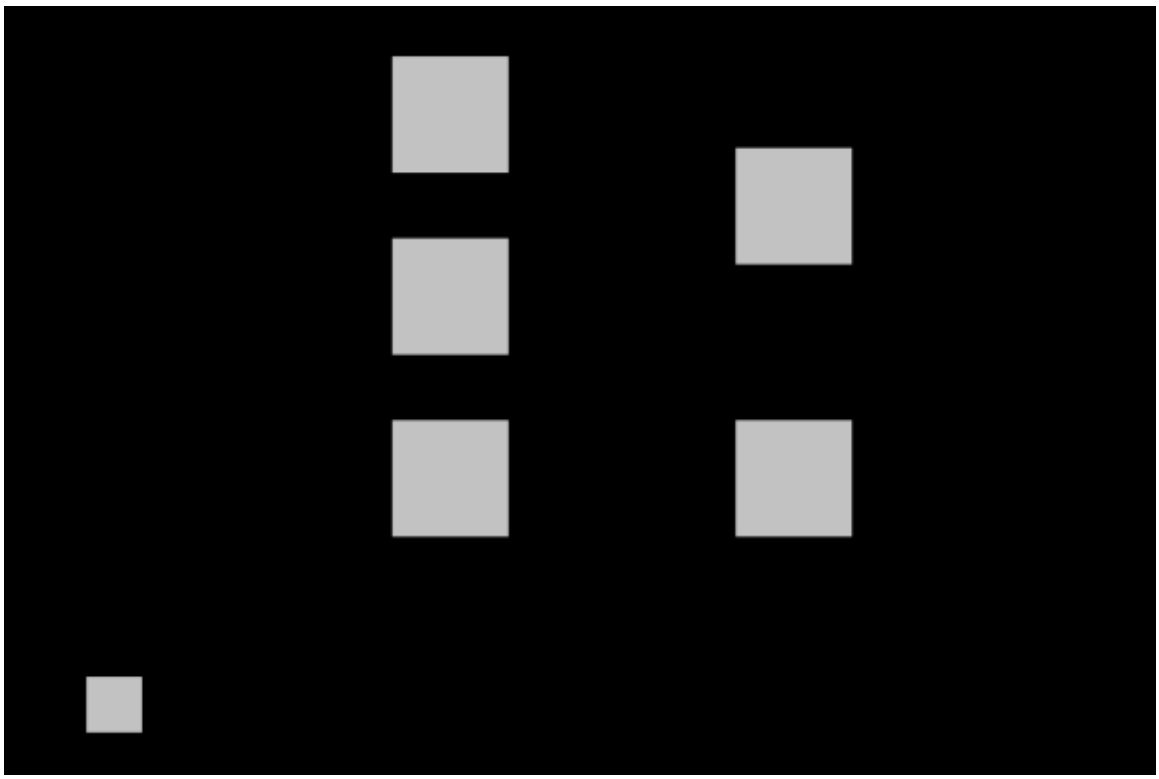
Εφόσον το πολυώνυμο ανήκει σε πεπερασμένο πεδίο $GF(2^m)$ ισχύει πώς [XVI]:

$$i \cdot \Lambda_i = \begin{cases} 0, & i\ ζυγός \\ \Lambda_i, & i\ περιττός \end{cases} \quad (77)$$

Άρα, η παράγωγος $\Lambda'(x)$ κατασκευάζεται από τους περιττούς συντελεστές του $\Lambda(x)$, αναθέτοντάς τους στον όρο με την αμέσως μικρότερη δύναμη:

$$\Lambda'(x) = \Lambda_1 + \Lambda_3 x^2 + \Lambda_5 x^4 + \cdots \quad (78)$$

Η διαίρεση μπορεί να υλοποιηθεί και πάλι ως πολλαπλασιασμός με τον αντίστροφο του παρονομαστή. Έτσι, προκύπτει η αρχιτεκτονική Forney που παρουσιάζεται στο Σχήμα 6-14. Το σήμα $ERRLOC$ αποτελεί έξοδο του μπλοκ του Chien και όπως αναφέρθηκε στην αντίστοιχη Ενότητα ορίζει αν η τωρινή θέση είναι θέση λάθους ή όχι [XVI].



Σχήμα 6-14 - Αρχιτεκτονική Αλγορίθμου Forney για υπολογισμό τιμών λάθους

6.4.2 Κοινή χρήση πόρων με Chien Search

Γνωρίζοντας πως το άθροισμα τον περιττών στοιχείων αποτελεί την παράγωγο $\Lambda'(x)$, το άθροισμα των μεταβλητών κατάστασης Q_j διασπάται σε δύο αθροίσματα. Το ένα αποτελεί άθροισμα των περιττών συντελεστών του $\Lambda(x)$, που ισούται με την παράγωγο $\Lambda'(x)$. Το δεύτερο αποτελεί άθροισμα των ζυγών συντελεστών του $\Lambda(x)$. Το περιττό άθροισμα μπορεί να δίνετε ως

είσοδος στο μπλοκ του Forney, καθώς αποτελεί την ζητούμενη παράγωγο $\Lambda'(a^i)$ στην επανάληψη i. Το ζυγό τμήμα προστίθεται στο περιττό για την αναγνώριση θέσης λάθους στον αλγόριθμο του Chien. [XXI].

Η αρχιτεκτονική βελτιστοποιημένου μπλοκ του αλγορίθμου Chien για την περίπτωση κώδικα Reed-Solomon με ικανότητα διόρθωσης λαθών $t = 8$ παρουσιάζεται στο Σχήμα 6-15.

Σχήμα 6-15 - Αρχιτεκτονική Chien με διάσπαση αθροίσματος σε ζυγό και περιττό μέρος

Αυτή είναι και η βελτιστοποίηση που χρησιμοποιείται στην υλοποίηση του αποκωδικοποιητή της παρούσας πτυχιακής εργασίας, φυσικά με ορισμένες τροποποιήσεις. Γενικά , με την χρήση της αρχιτεκτονικής αυτής τα μπλοκ του Chien και Forney λειτουργούν σαν ένα ενιαίο μπλοκ εύρεσης θέσης και τιμής λάθους. Σε πολλές υλοποιήσεις της βιβλιογραφίας αναφέρεται αυτό το ενιαίο μπλοκ με το όνομα CSEE δηλ. Chien Search Error Correction, κάτι που επιδεικνύει πως πραγματοποιείται στο μπλοκ αυτό και η διαδικασία διόρθωσης λαθών.

Φυσικά δεν απαιτείται το $ERRLOC$ ως σήμα εξόδου αν αποθηκευτούν οι τιμές των θέσεων λάθους, οι αξιολογήσεις του $\Omega(x)$ και οι παράγωγοι $\Lambda'(x)$ για τις θέσεις που ισχύει $\Lambda'(x_i) = 0$ σε καταχωρητές. Αυτοί οι πίνακες καταχωρητών, μαζί με τον συνολικό αριθμό λαθών που εντοπίστηκαν κατά την αναζήτηση από τον Chien ($errCount$) δίνονται στην συνέχεια στο μπλοκ του Forney που χρειάζεται να υπολογίσει πια μόνο τις διαιρέσεις $\Omega(x_i) \div \Lambda'(x_i)$ για το πολύ t θέσεις λάθους. Οι θέσεις και τιμές των λαθών αποστέλλονται τελικώς στο μπλοκ διόρθωσης λαθών.

7. ΚΩΔΙΚΑΣ ΕΠΑΛΗΘΕΥΣΗΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΣΕ ΛΟΓΙΣΜΙΚΟ

Η επαλήθευση των αποτελεσμάτων του υλικού πραγματοποιήθηκε με χρήση κώδικα στην Γλώσσα Προγραμματισμού Python που βασίζεται στην υλοποίηση της πηγής [XXVII]. Ωστόσο κατά την διάρκεια της πτυχιακής πραγματοποιήθηκαν πολλές αλλαγές στις συναρτήσεις, με κύριο σκοπό να χρησιμοποιηθούν οι ίδιοι αλγόριθμοι των βημάτων αποκωδικοποίησης με το υλικό.

7.1 Υλοποίηση μαθηματικών Galois

Όπως αναφέρθηκε και στην Ενότητα 4, οι πράξεις πολλαπλασιασμού και διαίρεσης υλοποιούνται στο υλικό πολύ εύκολα με χρήση προϋπολογισμένων πινάκων εκθετικών και λογαρίθμων. Αυτοί οι πίνακες υπολογίζονται ως εξής:

```
GFEXP = [0] * 512
GFLOG = [0] * 256

# initialising exponential and logarithmic fields
GFEXP[0] = 1
byteValue = 1
for bytePos in range(1, 255):
    byteValue <<= 1
    if (byteValue & 0x100):
        byteValue ^= 0x11d  # primitive polynomial: x^8 + x^4 + x^3 + x^2 + 1
    GFEXP[bytePos] = byteValue
    GFLOG[byteValue] = bytePos

for bytePos in range(255, 512):
    GFEXP[bytePos] = GFEXP[bytePos - 255]
```

Ο πίνακας των εκθετικών έχει διπλάσιο μήκος από τον αντίστοιχο των λογαρίθμων, ώστε να μπορεί να πραγματοποιηθεί ανάγνωση του εκθετικού της πρόσθεσης δύο λογαρίθμων χωρίς να απαιτείται η πράξη υπολοίπου $mod\ 255$.

Έτσι, οι πράξεις πολλαπλασιασμού και διαίρεσης ορίζονται με τις εξής συναρτήσεις:

```
# GALOIS PRIMITIVE OPERATIONS
def gfMult(argX, argY):  # Galois multiplication
    # parametre checks
    if ((argX == 0) or (argY == 0)):
        byteValue = 0
    else:
        # perform the operation
        byteValue = GFEXP[GFLOG[argX] + GFLOG[argY]]

    # return the product result
    return (byteValue)
```

```
def gfDivi(argX, argY):  # Galois division
    # validate the divisor
    if (argY == 0):
        raise ZeroDivisionError()

    # validate the dividend
    if (argX == 0):
        byteValue = 0
    else:
        # perform the division
        byteValue = GFEXP[GFLOG[argX] - GFLOG[argY] + 255]

    # return the division result
    return (byteValue)
```

Στην διαδικασία κωδικοποίησης της πηγής απαιτείται και η συνάρτηση πολλαπλασιασμού πολυωνύμων:

```
def gfPolyMult(polyA, polyB):  # Polynomial multiplication
    # initialise the product
    polyProd = len(polyA) + len(polyB) - 1
    polyProd = [0] * polyProd

    # start multiplying
    for posB in range(0, len(polyB)):
        for posA in range(0, len(polyA)):
            polyProd[posA + posB] ^= gfMult(polyA[posA], polyB[posB])

    # return the product result
    return (polyProd)
```

7.2 Υλοποίηση κωδικοποίησης Reed-Solomon

Κατά την διαδικασία κωδικοποίησης απαιτείται πολυώνυμο παραγωγής πεδίου που υπολογίζεται μέσω της συνάρτησης:

```
def rsGenPoly(errSize):
    polyValue = [1]

    for polyPos in range(0, 2*errSize):
        tempVal = [1, GFEXP[polyPos]]
        polyValue = gfPolyMult(polyValue, tempVal)

    # return the polynomial result
    return (polyValue)
```

Η καθ' αυτού κωδικοποίηση ορίζεται ως εξής:

```
def RSEncode(argMesg, errSize):

    # prepare the generator polynomial
    polyGen = rsGenPoly(errSize)

    # prepare the output buffer
    outBuffer = (len(argMesg) + 2*errSize)
    outBuffer = [0] * outBuffer

    # initialise the output buffer
    for mesgPos in range(0, len(argMesg)):
        mesgChar = argMesg[mesgPos]
        outBuffer[mesgPos] = ord(mesgChar)

    # begin encoding
    for mesgPos in range(0, len(argMesg)):
        mesgChar = outBuffer[mesgPos]
        if (mesgChar != 0):
            for polyPos in range(0, len(polyGen)):
                tempValue = gfMult(polyGen[polyPos], mesgChar)
                outBuffer[mesgPos + polyPos] ^= tempValue

    # finalise the output buffer
    for mesgPos in range(0, len(argMesg)):
        mesgChar = argMesg[mesgPos]
        outBuffer[mesgPos] = ord(mesgChar)
    # return the output buffer
    return (outBuffer)
```

Στην περίπτωση κώδικα $RS(255, 245)$ με $errSize = 5$, η είσοδος $argMesg$ έχει μήκος 245, ενώ η έξοδος $outBuffer$ μήκος $245 + 2 \times errSize = 255$.

7.3 Υλοποίηση Αποκωδικοποίησης Reed-Solomon

Τα σύνδρομα υπολογίζονται μέσω της συνάρτησης:

```
def rsSyndPoly(argCode, errSize):  # Generate the Syndrome Polynomial
    # initialise the polynomial
    polyValue = [0] * (2*errSize)

    # compute the polynomial terms
    for errPos in range(0, 2*errSize):
        for polyPos in range(0, len(argCode)):
            mulValue = gfMult(GFEXP[errPos], polyValue[errPos])
            polyValue[errPos] = argCode[polyPos] ^ mulValue

    # return the polynomial
    return (polyValue)
```

Ο αλγόριθμος RiBM που χρησιμοποιείται για την επίλυση της βασικής εξίσωσης ορίζεται ως εξής:

```
def RiBM(polySynd, errSize):
    # initialization
    k = 0
    gamma = 1

    delta = [0] * (3*errSize + 2)
    theta = [0] * (3*errSize + 1)

    delta[3*errSize] = theta[3*errSize] = 1

    for i in range(0, 2*errSize):
        delta[i] = theta[i] = polySynd[i]

    delta_next = [0] * (3*errSize + 2)
    for i in range(0, 3*errSize + 2):
        delta_next[i] = delta[i]

    # algorithm loop
    iterNum = 0
    while iterNum < 2*errSize:
        # step 1
        for i in range(0, 3*errSize + 2):
            delta[i] = delta_next[i]

        for i in range(0, 3*errSize + 1):
            delta_next[i] = gfMult(
                gamma, delta[i + 1]) ^ gfMult(delta[0], theta[i])

        # step 2
        if delta[0] != 0 and k >= 0:
            for i in range(0, 3*errSize + 1):
                theta[i] = delta[i + 1]
            gamma = delta[0]
            k = -k - 1
        else:
            k = k + 1

        iterNum += 1

    # final step
    lamda = delta_next[errSize: 2*errSize + 1]
    omega = delta_next[0:errSize]
    errPolyLength = errSize - int(k/2)

    return lamda, omega, errPolyLength
```

Αξίζει να σημειωθεί πως υπολογίζεται εδώ και το πλήθος των λαθών $errCount$, εφόσον η διαδικασία διόρθωσης λαθών λαμβάνει πίνακες με τις θέσεις και τιμές των λαθών.

Ο αλγόριθμος Chien ορίζεται ως εξής:

```
def rsFindErr(errPoly, errEvalPoly, errSize):

    errList = [0] * errSize
    sumOdd = [0] * errSize
    errEval = [0] * errSize
    curErr = 0

    # init lamda's
    lamda = [0] * (errSize + 1)
    for i in range(0, errSize + 1):
        lamda[i] = errPoly[i]

    # init omega's
    omega = [0] * errSize
    for i in range(0, errSize):
        omega[i] = errEvalPoly[i]

    for errPos in range(0, 255):
        # update lamda's
        for j in range(0, errSize + 1):
            lamda[j] = gfMult(lamda[j], GFEXP[j])

        # calculate odd sum
        sumOdd_tmp = 0
        for j in range(1, errSize + 1):
            if j % 2 == 1:
                sumOdd_tmp ^= lamda[j]

        # calculate even sum
        sumEven_tmp = 0
        for j in range(0, errSize + 1):
            if j % 2 == 0:
                sumEven_tmp ^= lamda[j]

        # calculate sum
        errZed = sumOdd_tmp ^ sumEven_tmp

        # update omega's
        for j in range(0, errSize):
            omega[j] = gfMult(omega[j], GFEXP[j+2*errSize])
```

```
            # calculate sum
            omega_eval = omega[0]
            for j in range(1, errSize):
                omega_eval = omega_eval ^ omega[j]

            # check if root
            if errZed == 0:
                errList[curErr] = errPos
                sumOdd[curErr] = sumOdd_tmp
                errEval[curErr] = omega_eval
                curErr += 1

    errCount = curErr

    return errList, sumOdd, errEval, errCount
```

Διατηρούνται πίνακες που αποθηκεύουν τις τιμές των λαθών, τις παραγώγους $\Lambda'(a^i)$ και τις αξιολογήσεις του $\Omega(x)$ για τις θέσεις που αποτελούν θέσεις λάθους ($\Lambda(a^i) = 0$). Οι τελευταίοι δύο πίνακες δίνονται στην συνέχεια ως είσοδος στην συνάρτηση του Forney.

Ο αλγόριθμος του Forney ορίζεται ως εξής:

```
def rsForney(sumOdd, errEval, errCount, errSize):
    div_inv_table = [0, 1, 142, 244, 71, 167, 122, 186, 173, 157, 221, 152, 61, 170, 93, 150,
                     216, 114, 192, 88, 224, 62, 76, 102, 144, 222, 85, 128, 160, 131, 75, 42,
                     108, 237, 57, 81, 96, 86, 44, 138, 112, 208, 31, 74, 38, 139, 51, 110,
                     72, 137, 111, 46, 164, 195, 64, 94, 80, 34, 207, 169, 171, 12, 21, 225,
                     54, 95, 248, 213, 146, 78, 166, 4, 48, 136, 43, 30, 22, 103, 69, 147,
                     56, 35, 104, 140, 129, 26, 37, 97, 19, 193, 203, 99, 151, 14, 55, 65,
                     36, 87, 202, 91, 185, 196, 23, 77, 82, 141, 239, 179, 32, 236, 47, 50,
                     40, 209, 17, 217, 233, 251, 218, 121, 219, 119, 6, 187, 132, 205, 254, 252,
                     27, 84, 161, 29, 124, 204, 228, 176, 73, 49, 39, 45, 83, 105, 2, 245,
                     24, 223, 68, 79, 155, 188, 15, 92, 11, 220, 189, 148, 172, 9, 199, 162,
                     28, 130, 159, 198, 52, 194, 70, 5, 206, 59, 13, 60, 156, 8, 190, 183,
                     135, 229, 238, 107, 235, 242, 191, 175, 197, 100, 7, 123, 149, 154, 174, 182,
                     18, 89, 165, 53, 101, 184, 163, 158, 210, 247, 98, 90, 133, 125, 168, 58,
                     41, 113, 200, 246, 249, 67, 215, 214, 16, 115, 118, 120, 153, 10, 25, 145,
                     20, 63, 230, 240, 134, 177, 226, 241, 250, 116, 243, 180, 109, 33, 178, 106,
                     227, 231, 181, 234, 3, 143, 211, 201, 66, 212, 232, 117, 127, 255, 126, 253]

    # for i in range(0, 256):
    #     print("x\"" + hex(div_inv_table[i]).lstrip("0x") + "\",")

    err_mag = [0] * errSize

    for curErr in range(0, errSize):
        inv_sumOdd = div_inv_table[sumOdd[curErr]]

        err_mag[curErr] = gfMult(errEval[curErr], inv_sumOdd)

    return err_mag
```

Η μνήμη ROM που απαιτείται για τον υπολογισμό του αντίστροφου στο υλικό, μπορεί με εύκολο τρόπο να δοθεί στην μορφή που την απαιτεί η γλώσσα VHDL με την επανάληψη for που βρίσκεται σε σχόλια. Η τιμή λάθους υπολογίζεται χρησιμοποιώντας τον αντίστροφο της παραγώγου που υπολογίστηκε στον Chien και την αξιολόγηση του $\Omega(x)$. Οι τιμές λάθους αποθηκεύονται και πάλι σε πίνακα για να χρησιμοποιηθούν στην συνέχεια στην συνάρτηση διόρθωσης λαθών.

Η συνάρτηση διόρθωσης λαθών ορίζεται ως εξής:

```
def rsCorrect(argCode, errPos, errMag, errCount, errSize):  # Correct Errors
    n = len(argCode)

    for i in range(0, errCount):
        if errPos[i] < n:
            argCode[errPos[i]] = argCode[errPos[i]] ^ errMag[i]

    return argCode
```

Η συνάρτηση αποκωδικοποίησης καλεί τις προηγούμενες συναρτήσεις και τυπώνει μηνύματα με τα υπολογισμένα πολυώνυμα, με σκοπό την επαλήθευση των αποτελεσμάτων του υλικού. Ο κώδικας της συνάρτησης αυτής είναι ο εξής:

```
def RSDecode(argCode, errSize):  # Main decode routine

    # initialise the code buffer
    codeBuffer = list(argCode)

    # calculate the syndrome
    polySynd = rsSyndPoly(codeBuffer, errSize)
    if (max(polySynd) == 0):
        print("The message has no errors\n")
        return (codeBuffer)

    print("Syndrome Polynomial:", polySynd)
    binary_string = ""
    for i in range(len(polySynd) - 1, -1, -1):
        binary_string += format(polySynd[i], '08b')
    print(binary_string, "\n")

    # calculate the error locator & evaluator polynomials
    errPoly, errEvalPoly, ErrPolyLength = RiBM(polySynd, errSize)

    print("Error Locator Polynomial:", errPoly)
    binary_string = ""
    for i in range(len(errPoly) - 1, -1, -1):
        binary_string += format(errPoly[i], '08b')
    print(binary_string, "\n")
```

```
print("Error Evaluator Polynomial:", errEvalPoly)
binary_string = ""
for i in range(len(errEvalPoly) - 1, -1, -1):
    binary_string += format(errEvalPoly[i], '08b')
print(binary_string, "\n")

print("ErrPolyLength:", ErrPolyLength)

# find error locations
errList, sumOdd, errEval, errCount = rsFindErr(errPoly, errEvalPoly, errSize)

print("Error Locations:", errList)
binary_string = ""
for i in range(len(errList) - 1, -1, -1):
    binary_string += format(errList[i], '08b')
print(binary_string, "\n")

print("ErrPoly Derivatives:", sumOdd)
binary_string = ""
for i in range(len(sumOdd) - 1, -1, -1):
    binary_string += format(sumOdd[i], '08b')
print(binary_string, "\n")

print("ErrEvalPoly Evaluations:", errEval)
binary_string = ""
for i in range(len(errEval) - 1, -1, -1):
    binary_string += format(errEval[i], '08b')
print(binary_string, "\n")

if (ErrPolyLength != errCount):
    print(ErrPolyLength, "!=", errCount, "\nToo many errors!\n")
    return argCode

# find error magnitudes
errMag = rsForney(sumOdd, errEval, errCount, errSize)
print("Error Magnitudes:", errMag)
binary_string = ""
for i in range(len(errMag) - 1, -1, -1):
    binary_string += format(errMag[i], '08b')
print(binary_string, "\n")

# correct the errors
#outMesg = rsCorrect(codeBuffer, polySynd, errList)
outMesg = rsCorrect(codeBuffer, errList, errMag, errCount, errSize)
return outMesg
```

Ως μήνυμα πληροφορίας χρησιμοποιείται η τυχαία συμβολοσειρά:

```
tMesg = "XAwFHlYCXRmRjasIRWQJfUTOAunDwKbQsfAgIySSSIwoPWTrCFlsLU3pRmPIKPByyRhnGrOsHoOTiBPInRnQOoKNNNNwPYPIWTjWeSvpeyjAjgPRfmikiIanAIRAWNng

rMVlvbtVtfgVUoSqtTVXpiIQMfaSigGXXMCmJIUPPWIPKyauCCskSCwPQQopOCRtVdwWOheIEEndyrxpcKNEgXnVTnhLwJhisUbSJvbawQSiSSH"
```

Το μήνυμα κωδικοποιείται ως εξής:

```
errSize = 8  # error correction ability t

# encode the message
tCode = RSEncode(tMesg, errSize)
print("Original codeword: ", tCode)
print("\r\r")
```

Προστίθενται έως $t = 8$ τυχαία λάθη σε $COUNT$ εκτελέσεις αποκωδικοποίησης:

```
successCount = 0

COUNT = 100
for num in range(0, COUNT):
    rCode = [0] * len(tCode)
    for i in range(0, len(tCode)):
        rCode[i] = tCode[i]

    # introduce random errors
    errCount = random.randint(0, errSize)
    print("Adding", errCount, "Random Errors...\n")
    for i in range(0, errCount):
        randIndex = random.randint(0, 254)
        randMag = random.randint(1, 255)
        print("rCode[" + str(randIndex) + "] ^= ", str(randMag))
        rCode[randIndex] ^= randMag

    print("\nCodeword with error: ", rCode)
    print("\r\r")
```

Για να χρησιμοποιηθούν σε Testbench στην συνέχεια, τα σύμβολα των κωδικοποιημένων μηνυμάτων, με πιθανή παρουσία λαθών, γράφονται γραμμή-προς-γραμμή σε δυαδική μορφή σε αρχεία μέσω του κώδικα:

```
    # write to file
    file = open("input"+str(num)+".txt", "w")
    for i in range(0, len(rCode)):
        file.write(format(rCode[i], '08b') + "\n")
    file.close()
```

Η αποκωδικοποίηση αποτελεί κλήση της συνάρτησης:

```
    # decode the message
    tDec = RSDecode(rCode, errSize)
```

Τέλος, συγκρίνεται και το αποτέλεσμα της αποκωδικοποίησης με την αρχική πληροφορία για να διαπιστωθεί αν η αποκωδικοποίηση εκτελέστηκε με επιτυχία ή όχι:

```
msg = ""
for i in range(0, len(tMesg)):
    msg += chr(tDec[i])
print("Decoded message: ", msg)

failure = 0
for i in range(0, len(tMesg)):
    if tMesg[i] != chr(tDec[i]):
        print("Error at Pos", i)
        failure = 1

if failure == 1:
    print("\nError correction Failure!")
else:
    successCount += 1
```

Η εκτέλεση του κώδικα λογισμικού δίνει την ακόλουθη έξοδο:

```
Original codeword:  [88, 65, 97, 70, 72, 108, 89, 67, 90, 78, 109, 82, 106, 97, 122, 73, 82, 87, 81, 74, 102, 83, 84, 79, 65, 117, 110, 68, 101, 75, 98
, 81, 115, 102, 65, 113, 73, 121, 90, 69, 90, 90, 119, 111, 80, 87, 84, 114, 67, 70, 108, 111, 76, 68, 83, 112, 69, 109, 80, 73, 75, 80, 66, 121, 121,
83, 104, 120, 71, 114, 85, 122, 72, 122, 71, 84, 108, 66, 80, 73, 110, 82, 110, 81, 71, 111, 98, 78, 72, 78, 71, 119, 80, 89, 70, 90, 87, 84, 106, 87,
101, 90, 118, 112, 115, 121, 106, 65, 106, 103, 80, 88, 102, 109, 108, 107, 105, 73, 97, 110, 65, 73, 72, 65, 87, 72, 110, 103, 110, 77, 86, 108, 118,
98, 116, 86, 116, 102, 103, 86, 85, 111, 83, 113, 116, 84, 86, 88, 112, 105, 73, 81, 77, 102, 97, 90, 105, 113, 71, 90, 88, 77, 67, 109, 74, 73, 85, 80
, 70, 87, 73, 80, 75, 121, 120, 97, 67, 67, 120, 107, 90, 67, 119, 80, 81, 81, 111, 112, 83, 67, 82, 116, 86, 100, 119, 78, 79, 104, 101, 73, 69, 69, 1
10, 100, 121, 114, 120, 112, 114, 107, 78, 75, 113, 88, 110, 86, 70, 110, 104, 76, 101, 74, 98, 105, 115, 68, 98, 83, 74, 118, 98, 97, 119, 81, 90, 105
, 90, 90, 72, 113, 107, 10, 152, 99, 24, 31, 160, 252, 24, 50, 205, 159, 70, 134, 94]

RS(255, 239)

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

Adding 1 Random Errors...

rCode[204] ^=  22

Codeword with error:  [88, 65, 97, 70, 72, 108, 89, 67, 90, 78, 109, 82, 106, 97, 122, 73, 82, 87, 81, 74, 102, 83, 84, 79, 65, 117, 110, 68, 101, 75,
98, 81, 115, 102, 65, 113, 73, 121, 90, 69, 90, 90, 119, 111, 80, 87, 84, 114, 67, 70, 108, 111, 76, 68, 83, 112, 69, 109, 80, 73, 75, 80, 66, 121, 121
, 83, 104, 120, 71, 114, 85, 122, 72, 122, 71, 84, 108, 66, 80, 73, 110, 82, 110, 81, 71, 111, 98, 78, 72, 78, 71, 119, 80, 89, 70, 90, 87, 84, 106, 87
, 101, 90, 118, 112, 115, 121, 106, 65, 106, 103, 80, 88, 102, 109, 108, 107, 105, 73, 97, 110, 65, 73, 72, 65, 87, 72, 110, 103, 110, 77, 86, 108, 118
, 98, 116, 86, 116, 102, 103, 86, 85, 111, 83, 113, 116, 84, 86, 88, 112, 105, 73, 81, 77, 102, 97, 90, 105, 113, 71, 90, 88, 77, 67, 109, 74, 73, 85,
80, 70, 87, 73, 80, 75, 121, 120, 97, 67, 67, 120, 107, 90, 67, 119, 80, 81, 81, 111, 112, 83, 67, 82, 116, 86, 100, 119, 78, 79, 104, 101, 73, 69, 69,
 110, 100, 111, 114, 120, 112, 114, 107, 78, 75, 113, 88, 110, 86, 70, 110, 104, 76, 101, 74, 98, 105, 115, 68, 98, 83, 74, 118, 98, 97, 119, 81, 90, 1
05, 90, 90, 72, 113, 107, 10, 152, 99, 24, 31, 160, 252, 24, 50, 205, 159, 70, 134, 94]

Syndrome Polynomial: [22, 78, 107, 218, 149, 251, 48, 240, 23, 75, 122, 143, 137, 151, 241, 18]
0001001011110001100101111000100011000111101111010010010110001011111110000001100001111101110010101110110101011010011100001010110

Error Locator Polynomial: [38, 190, 0, 0, 0, 0, 0, 0, 0]
0000000000
```

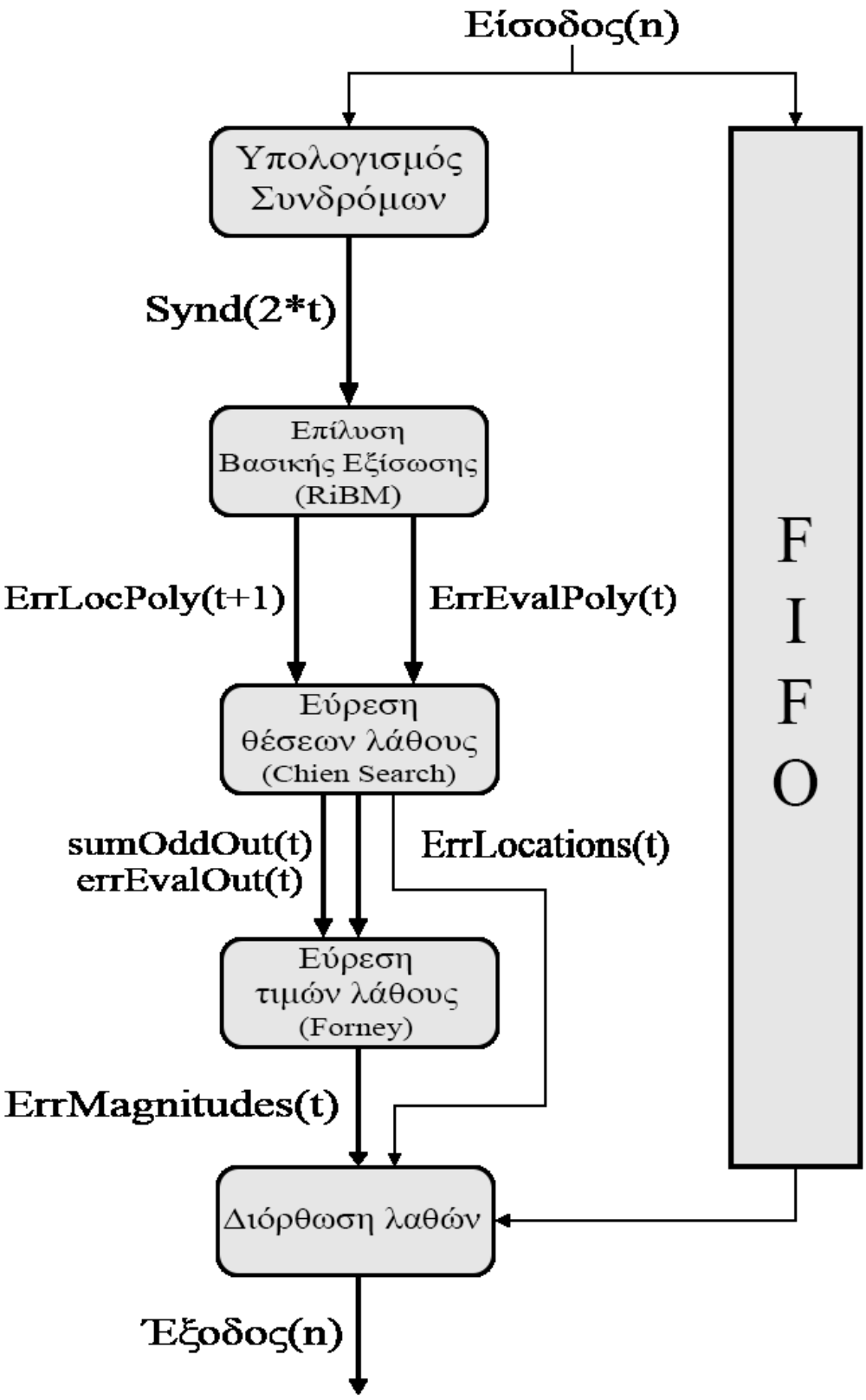
8. ΥΛΟΠΟΙΗΣΗ ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΗ ΣΕ ΥΛΙΚΟ

8.1 Αρχιτεκτονική αποκωδικοποιητή

Οι διάφορες παράμετροι του κώδικα RS, καθώς και οι αλγόριθμοι που χρησιμοποιούνται στα διάφορα στάδια του υλοποιημένου αποκωδικοποιητή παρουσιάζονται στο Πίνακα 8-1. Ακόμη, το Σχήμα 8-1 της επόμενης σελίδας αποτελεί διάγραμμα της συνολικής αρχιτεκτονικής του αποκωδικοποιητή με τις παραμέτρους αυτές.

Παράμετροι κώδικα RS	
Πεπερασμένο πεδίο Galois	$GF(2^8)$ με $p(x) = (11D)_{hex}$
Μήκος συμβόλου	$m = 8$
Μήκος κωδικής λέξης	$n = 255$
Ικανότητα διόρθωσης λαθών	$t = 1, 2, 3, \cdots, 8$
Σύμβολα πλεονασμού	$2t$
Μήκος πληροφορίας	$n - 2t$
Κώδικας	$RS(n, n-2t)$
Αλγόριθμοι Αποκωδικοποιητή	
Υπολογισμός Συνδρόμων	Σειριακό μπλοκ με πολλαπλασιαστές μεταβλητής-σταθεράς που δίνει ως έξοδο το πολυώνυμο Συνδρόμων, $S(x)$.
Επίλυση Βασικής Εξίσωσης	Αλγόριθμος RiBM με είσοδο: • $S(x)$ $(Synd)$ και έξοδο: • $\Lambda(x)$ $(ErrLocPoly)$ • $\Omega(x)$ $(ErrEvalPoly)$
Εύρεση θέσεων λάθους	Σειριακή Αναζήτηση Chien με είσοδο: • $\Lambda(x)$ • $\Omega(x)$ και έξοδο: • θέσεις λαθών $(ErrLocations)$ με τις αντίστοιχες: • παραγώγους $\Lambda'(a^i)$ $(sumOddOut)$ • αξιολογήσεις $\Omega(a^i)$ $(errEvalOut)$
Εύρεση τιμών λάθους	Σειριακός Αλγόριθμος Forney με είσοδο: • $\Lambda'(a^i)$ • $\Omega(a^i)$ και έξοδο: • τιμές λαθών $(ErrMagnitudes)$
Διόρθωση λαθών	Διόρθωση των λαθών με πρόσθεση των $ErrMag[i]$ που αντιστοιχούν στις θέσεις $ErrLoc[i]$ στην τιμή εξόδου της FIFO.
Μήκος μνήμης FIFO	$n + 2t + n + t + 5 = 2n + 3t + 5$

Πίνακας 8-1 - Παράμετροι και Αλγόριθμοι του Υλοποιημένου Αποκωδικοποιητή RS



Σχήμα 8-1 - Αρχιτεκτονική Υλοποιημένου Αποκωδικοποιητή

8.2 Μπλοκ υπολογισμού συνδρόμων

Όπως αναφέρθηκε και στην Ενότητα 6.1.1 τα $2t$ σύνδρομα υπολογίζονται από το άθροισμα:

$$s_i = \sum_{j=0}^{n-1} r_j \left(a^i\right)^j, \qquad i = 0, 1, \cdots, 2t-1 \tag{79}$$

όπου r_j είναι οι συντελεστές του πολυωνύμου μηνύματος που λαμβάνετε.

Σύμφωνα με αυτήν την σχέση για τον υπολογισμό των $2t$ συνδρόμων, στην περίπτωση κώδικα $RS(n, n-2t)$, απαιτούνται οι ακόλουθοι υπολογισμοί:

$$\begin{aligned} s_0 &= \sum_{j=0}^{n-1} r_j (a^0)^j \\ s_1 &= \sum_{j=0}^{n-1} r_j (a^1)^j \\ &\vdots \\ s_{2t-1} &= \sum_{j=0}^{n-1} r_j (a^{2t-1})^j \end{aligned} \tag{80}$$

Με άλλα λόγια, το πρώτο σύνδρομο ισούται με το άθροισμα όλων των συντελεστών του πολυωνύμου $r(x)$, ενώ στα υπόλοιπα απαιτείται σε κάθε κύκλο πολλαπλασιασμός επί μια σταθερά, και στην συνέχεια άθροισμα του αντίστοιχου συντελεστή r_j. Επομένως απαιτούνται $2t-1$ πολλαπλασιαστές του τύπου μεταβλητής-σταθεράς με σταθερές. Για παράδειγμα, στην περίπτωση κώδικα $RS(255,\ 245)$ απαιτούνται οι σταθερές:

$$a = 2_{hex},\ a^2 = 4_{hex},\ a^3 = 8_{hex},\ a^4 = 10_{hex},\ a^5 = 20_{hex},$$

$$a^6 = 40_{hex}, \qquad a^7 = 80_{hex}, \qquad a^8 = 1D_{hex}, \qquad a^9 = 3A_{hex}$$

Στο Παράρτημα Γ μπορούν να βρεθούν ολοκληρωμένες μαθηματικές αποδείξεις των συναρτήσεων εξόδου για τους πολλαπλασιαστές αυτούς.

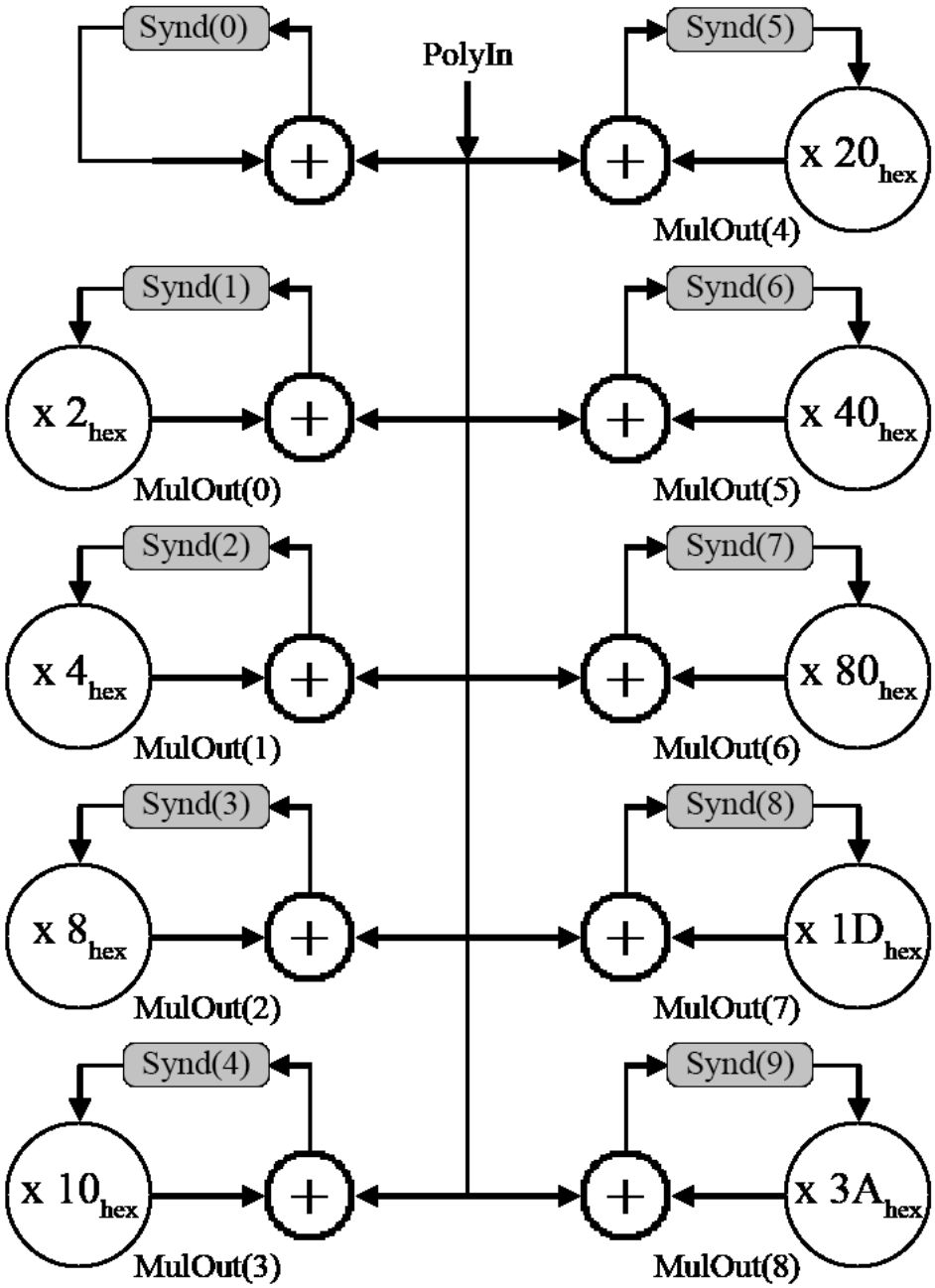
Οι τιμές των $2t$ συνδρόμων αποθηκεύονται σε ίσου αριθμού καταχωρητές. Οι καταχωρητές αυτοί αρχικοποιούνται με σήμα αρχικοποίησης (reset) ή σήμα εκκίνησης εισόδου ($inputStart$) και το άθροισμα εκτελείται μόνο όταν η τωρινή είσοδος είναι έγκυρη (valid).

Η διαδικασία υπολογισμού συνδρόμων τερματίζει όταν το σήμα $inputStart$ λάβει τιμή '1' και τότε τοποθετούνται στην έξοδο του μπλοκ τα $2t$ σύνδρομα ($SyndOut$), καθώς και λαμβάνει λογική τιμή '1' το σήμα τέλους $SyndFin$. Κατά την διάρκεια υπολογισμού, καθώς και όταν το μπλοκ βρίσκεται σε αναμονή για είσοδο μετά την πρώτη αρχικοποίηση μέσω του $reset$, το σήμα $SyndFin$ έχει τιμή '0'. Το $SyndFin$ αποτελεί «σκανδάλη εκκίνησης» (trigger) του μπλοκ επίλυσης βασικής εξίσωσης.

Εφόσον η εκκίνηση της επόμενης διαδικασίας υπολογισμού και η τοποθέτηση των συνδρόμων πραγματοποιείται ταυτόχρονα, είναι σημαντική η διάκριση του αν το σήμα εκκίνησης $inputStart$ σηματοδοτεί την πρώτη είσοδο, γιατί στην πρώτη είσοδο δεν υπάρχει ακόμα έξοδος.

Έναν εύκολο τρόπο υλοποίηση αποτελεί η χρήση ενός βοηθητικού σήματος $NoOutput$ με αρχική τιμή '1'. Το σήμα λαμβάνει την τιμή αυτή κατά την αρχικοποίηση του μπλοκ συνδρόμων μέσω του σήματος $reset$. Την πρώτη φορά που το μπλοκ θα λάβει σήμα $inputStart$ η τιμή του σήματος αυτού θα αλλάξει σε '0'. Ωστόσο, έξοδος υπάρχει μόνο όταν το σήμα είναι '0'. Άρα, η διαχείριση της εξόδου πραγματοποιείται ελέγχοντας τις τιμές των σημάτων $NoOutput$ και $inputStart$.

Το βασικό κύκλωμα υπολογισμού συνδρόμων, χωρίς τα διάφορα σήματα ελέγχου και αρχικοποίησης, για την περίπτωση κώδικα $RS(255, 245)$ παρουσιάζεται στο Σχήμα 8-2.



Σχήμα 8-2 - Μπλοκ Υπολογισμού Συνδρόμων

8.3 Μπλοκ επίλυσης βασικής εξίσωσης με RiBM

Για την επίλυση της βασικής εξίσωσης χρησιμοποιείται ο αλγόριθμος RiBM που μέσα σε $2t$ κύκλους υπολογίζει ταυτόχρονα τα πολυώνυμα $\Lambda(x)$ και $\Omega(x)$. Λεπτομερής ανάλυση του αλγορίθμου αυτού πραγματοποιήθηκε στην Ενότητα 6.2.4.4.

Για την υλοποίηση του αλγορίθμου αυτού στο υλικό απαιτούνται $6t + 4$ καταχωρητές, από τους οποίους $3t + 1$ χρησιμοποιούνται για την αποθήκευση του πολυωνύμου $\hat{\Delta}^{(r)}(x)$, $3t$ για το πολυώνυμο $\hat{\theta}^{(r)}(x)$, 1 για την αποθήκευση του $\gamma^{(r)}$ και 2 για τους μετρητές k και $iterNum$. Κατά τον υπολογισμό του $\hat{\Delta}^{(r+1)}(x)$ στο Βήμα RiBM.1 απαιτούνται $6t + 2$ γενικοί πολλαπλασιαστές και $3t + 1$ αθροιστές. Η ανάθεση των συντελεστών του $\hat{\theta}^{(r)}(x)$ στο Βήμα RiBM.2 υλοποιείται με $3t + 1$ πολυπλέκτες. Η ανάθεση του $\gamma^{(r+1)}$ στο ίδιο βήμα αποτελεί και αυτή με την σειρά της πολυπλέκτη. Επίσης, απαιτείται και πολυπλέκτης για τον μετρητή k, καθώς και για τον μετρητή επανάληψης $iterNum$. Έτσι, τελικώς απαιτούνται $3t + 4$ πολυπλέκτες [XXII].

Ο μετρητής k μπορεί να υλοποιηθεί με λογική συμπληρώματος ως προς 2. Ο μετρητής απαιτεί μόνο t bits. Με αυτόν τον τρόπο για τον έλεγχο $k \geq 0$ απαιτείται μόνο έλεγχος του πιο σημαντικού bit του k, k_{t-1}. Η αύξηση της τιμής του k κατά 1 υλοποιείται κανονικά ως $k^{(r+1)} = k^{(r)} + 1$, ενώ η ενημέρωση της 1ης περίπτωσης, $k^{(r+1)} = -k^{(r)} - 1$, είναι ταυτόσημη με την πράξη αντιστροφής όλων των bit του μετρητή $k^{(r+1)} = not\ k^{(r)}$, και έτσι δεν απαιτείται αφαίρεση [XXII]. Στο Σχήμα 8-3 παρουσιάζεται η λογική του μπλοκ RiBM χωρίς αναλυτική περιγραφή της μονάδας ελέγχου και των σημάτων της.

Σχήμα 8-3 - Λογική Μπλοκ Επίλυσης βασικής εξίσωσης με Αλγόριθμο RiBM

Η διαδικασία επίλυσης ξεκινά κατά την λήψη του σήματος $BMStart$, που αποτελεί πρακτικώς το σήμα εξόδου $SyndFin$ του μπλοκ υπολογισμού συνδρόμων. Σε εκείνο τον κύκλο ανατίθενται και τα σύνδρομα στους καταχωρητές που αντιστοιχούν στους πρώτους $2t$ συντελεστές των πολυωνύμων $\hat{\Delta}^{(r)}(x)$ και $\hat{\theta}^{(r)}(x)$, καθώς και αρχικοποιείται ο μετρητής επαναλήψεων σε 0. Μετά από $2t$ κύκλους το μπλοκ ενεργοποιεί το σήμα εξόδου $BMFin$ που ξεκινά (trigger) την διαδικασία εύρεσης θέσεων λάθους στο μπλοκ του Chien. Το σήμα αυτό αντιστοιχεί ακριβώς στο σήμα $ChienStart$ που αναμένει το μπλοκ του Chien για την εκκίνησή του. Σε εκείνο τον κύκλο τοποθετούνται και τα υπολογισμένα πολυώνυμα $\Lambda(x)$ και $\Omega(x)$ στις αντίστοιχες εξόδους τους, ώστε να χρησιμοποιηθούν στην συνέχεια στην διαδικασία εύρεσης θέσεων λάθους. Οι έξοδοι ενημερώνονται επομένως στην τιμή $2t$ του μετρητή, όπου για ένα κύκλο ενεργοποιείται και το σήμα $BMFin$. Στην τιμή $2t + 1$ του μετρητή το μπλοκ RiBM βρίσκεται σε αναμονή για είσοδο από το μπλοκ των συνδρόμων και τίθεται ανενεργό.

8.4 Μπλοκ Chien Search

Η εύρεση των θέσεων λάθους πραγματοποιείται με τον κλασσικό σειριακό αλγόριθμο Αναζήτησης Chien. Για εξοικονόμηση πόρων υπολογίζονται ξεχωριστά τα αθροίσματα των ζυγών και περιττών μεταβλητών κατάστασης $lamda$, εφόσον το $sumOdd$ ισούται με την παράγωγο $\Lambda'(\alpha^i)$ που απαιτεί το μπλοκ του Forney για τον υπολογισμό της τιμής των λαθών. Για την περίπτωση του κώδικα $RS(n, n-2t)$, για τον οποίο υλοποιήθηκε ο αποκωδικοποιητής, απαιτούνται $t+1$ καταχωρητές που θα ενεργούν ως μεταβλητές κατάστασης $lamda$. Αυτοί οι καταχωρητές αρχικοποιούνται σε 0 όταν λαμβάνεται σήμα αρχικοποίησης $reset$, ενώ λαμβάνουν τις τιμές των συντελεστών του $\Lambda(x)$ όταν ληφθεί το σήμα εκκίνησης του Chien, $ChienStart$.

Η πρώτη μεταβλητή δεν τροποποιείται καθ' όλη την διάρκεια εκτέλεσης και ισούται με τον 1ο συντελεστή του πολυωνύμου $\Lambda(x)$ και έτσι ορίζεται ως $ErrPoly0$. Στις υπόλοιπες μεταβλητές πραγματοποιείται σε κάθε κύκλο πολλαπλασιασμός επί σταθερά, που στην περίπτωση κώδικα $RS(255, 245)$ έχουν τις τιμές: 2_{hex}, 4_{hex}, 8_{hex}, 10_{hex} ή 20_{hex} αντίστοιχα, ενημερώνοντας την τιμή του κάθε $lamda$. Έτσι, απαιτούνται t πολλαπλασιαστές του τύπου μεταβλητής-σταθεράς. Μαθηματική απόδειξή τους υπάρχει στο Παράρτημα Γ.

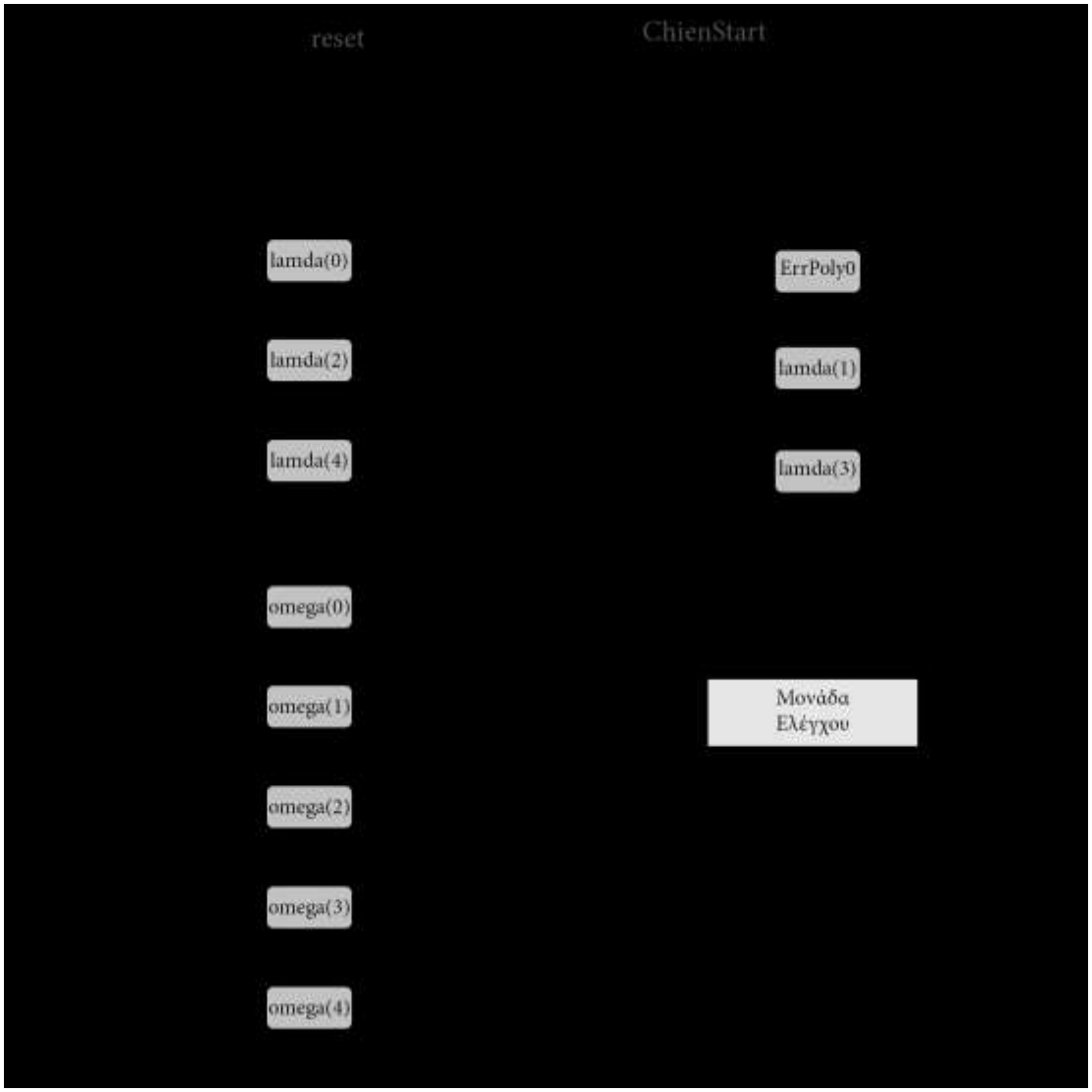
Η αξιολόγηση του πολυωνύμου $\Omega(x)$ υλοποιείται με παρόμοιο τρόπο με αυτήν του $\Lambda(x)$, προσθέτοντας τις μεταβλητές $omega$, οι οποίες ενημερώνονται σε κάθε κύκλο με χρήση πολλαπλασιαστή μεταβλητής-σταθεράς. Εφόσον, χρησιμοποιείται ο αλγόριθμος RiBM απαιτείται ολίσθηση των συντελεστών του $\Omega(x)$ κατά $2t$ και έτσι στην περίπτωση του κώδικα $RS(255, 245)$ πραγματοποιείται πολλαπλασιασμός με τις σταθερές:

$$a^{10} = 74_{hex}, \qquad a^{11} = E8_{hex}, \qquad a^{12} = CD_{hex}, \qquad a^{13} = 87_{hex}, \qquad a^{14} = 13_{hex}$$

Στο Παράρτημα Γ μπορούν να βρεθούν οι μαθηματικές αποδείξεις των πολλαπλασιαστών αυτών.

Για να γνωρίζει το μπλοκ σε ποιον εκθέτη του a, a^i, βρίσκεται απαιτείται μετρητής στο διάστημα [-1, n+1]. Στην τιμή -1 δεν έχουν ενημερωθεί ακόμα οι μεταβλητές απόφασης και έτσι δεν εκτελείται έλεγχος για θέση λάθους. Στο διάστημα $[0, n)$ κάθε φορά που βρίσκετε θέση λάθους $(errZed = 0)$ αποθηκεύεται η θέση λάθους a^i σε πίνακα καταχωρητών $ErrLocs$. Επιπρόσθετα, αποθηκεύεται και η αξιολόγηση του $\Omega(a^i)$ σε πίνακα $ErrEval$, καθώς και η παράγωγος $\Lambda'(a^i)$ σε πίνακα $sumOddCalcs$. Στην τιμή n το μπλοκ βγάζει ως έξοδο τους τρείς αυτούς πίνακες, καθώς και σηματοδοτεί την εκκίνηση του Forney μέσω του σήματος $ChienFin$ που είναι ταυτόσημο με το $ForneyStart$ που λαμβάνει το μπλοκ του Forney. Στην τιμή $n+1$ το μπλοκ βρίσκεται σε αναμονή εισόδου.

Επειδή οι κύκλοι εκτέλεσης του Chien είναι ίσοι με τους ελάχιστους απαιτούμενους κύκλους για είσοδο νέου μηνύματος, κατά την λήψη του επόμενου σήματος εισόδου $ChienStart$ ο μετρητής βρίσκεται στην τιμή $n-1$. Επομένως, δεν προλαβαίνουν να τοποθετηθούν οι υπολογισμένες τιμές στην έξοδο. Για να αποφευχθεί αυτό ορίζεται καταχωρητής $NoOutput$ με αρχική τιμή (μέσω $reset$) '1'. Όταν λαμβάνεται σήμα $ChienStart$ και το $NoOutput$ είναι '1' τότε δεν πραγματοποιείται έξοδος του μπλοκ, ενώ αν το σήμα έχει τιμή '0' τότε τοποθετούνται κανονικά οι τρείς πίνακες στην έξοδο και σηματοδοτείται η εκκίνηση του Forney ($ChienFin = '1'$). Αν το σήμα δεν ληφθεί πριν την εκτέλεση του n-οστού βήματος, που αποτελεί το βήμα εξόδου τότε στο βήμα αυτό η τιμή του $NoOutput$ αλλάζει σε '1'. Στο Σχήμα 8-4 της επόμενης σελίδας παρουσιάζεται η αρχιτεκτονική του μπλοκ του Chien για την περίπτωση κώδικα $RS(255, 245)$.

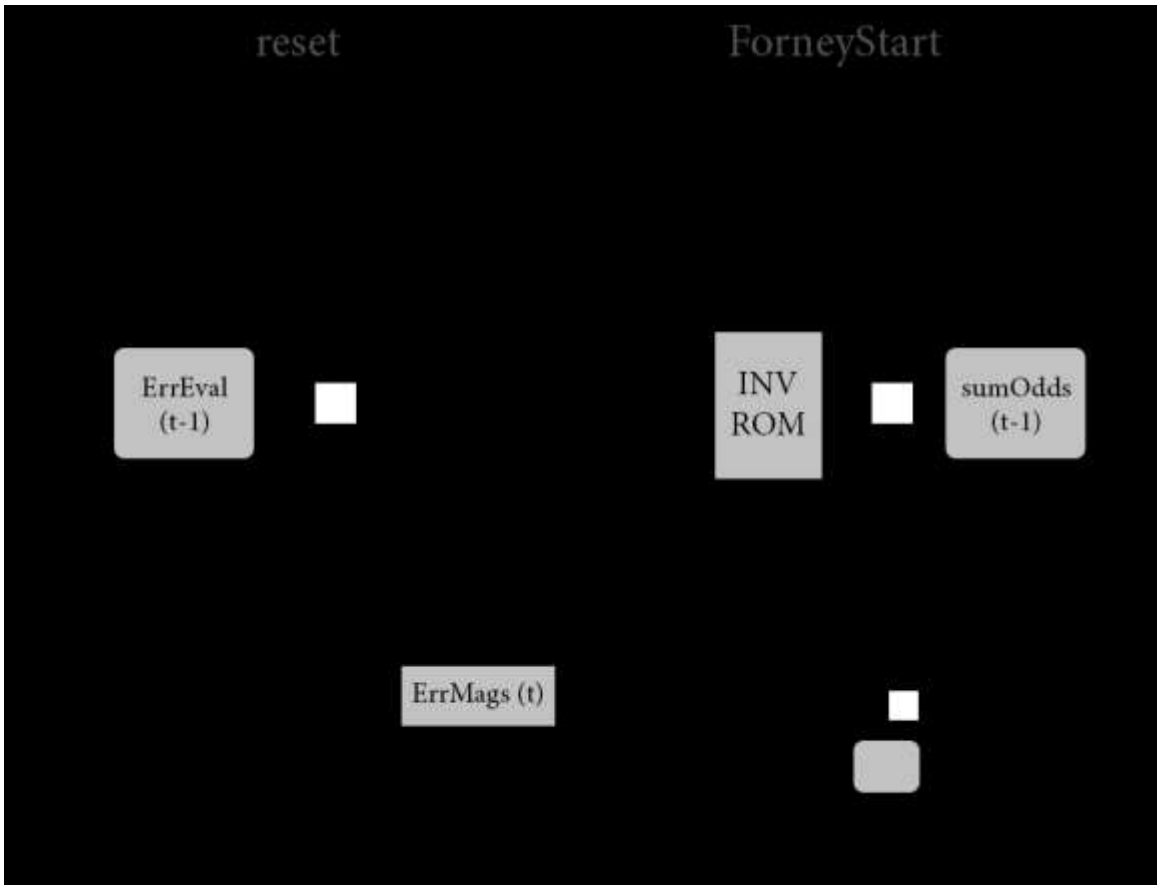


Σχήμα 8-4 - Μπλοκ εύρεσης θέσεων λάθους με Chien

8.5 Μπλοκ Forney

Η εύρεση των τιμών λάθους πραγματοποιείται με σειριακό μπλοκ του αλγορίθμου Forney που δέχεται ως είσοδο πίνακα παραγώγων $\Lambda'(a^i)$ ($sumOddIn$) και αξιολογήσεων $\Omega(a^i)$ ($ErrEvalIn$).

Για την εκτέλεση της πράξης της διαίρεσης $\Omega(a^i)/\Lambda'(a^i)$ ως πολλαπλασιασμό με τον αντίστροφο απαιτείται μνήμη ROM με τους αντιστρόφους πολλαπλασιασμού. Δίνοντας στην ROM αυτή ως είσοδο την παράγωγο $sumOdd$ μέσα σε ένα κύκλο βγαίνει ως έξοδος το inv_sumOdd που μπορεί να χρησιμοποιηθεί για την εκτέλεση του πολλαπλασιασμού. Ο πολλαπλασιασμός $omega_eval \times inv_sumOdd$ μας δίνει την τιμή λάθους για την θέση a^i, err_mag. Το σήμα $reset$ αρχικοποιεί το μπλοκ σε 0, ενώ με το σήμα εκκίνησης Forney, $ForneyStart$, αρχικοποιούνται οι πίνακες $sumOdds$, $ErrEval$ και $ErrMags$ του Forney. Μετά από $t+1$ κύκλους βγαίνουν ως έξοδο οι υπολογισμένες τιμές με την μορφή σήματος $ErrMagnitudes$, ενώ σηματοδοτείται και η εκκίνηση της διόρθωσης λαθών με σήμα εξόδου $ForneyFin$. Το μπλοκ του Forney παρουσιάζεται στο Σχήμα 8-5 της επόμενης σελίδας.



Σχήμα 8-5 - Μπλοκ εύρεσης τιμών λάθους με Forney

8.6 Μνήμη FIFO

Τα σύμβολα του λαμβανόμενου μηνύματος αποθηκεύονται κατά την λήψη τους σε μνήμη FIFO (First-In-First-Out), έτσι ώστε να είναι διαθέσιμα στην συνέχεια για την επιδιόρθωσή τους στο μπλοκ διόρθωσης λαθών με την σειρά που λήφθηκαν.

Όπως αναφέρθηκε και στην Ενότητα 8.1, το μήκος της FIFO είναι $2n + 3t + 5$, που πρακτικώς σημαίνει πως η καθυστέρηση του αποκωδικοποιητή είναι $2n + 3t + 5$ κύκλοι από την πρώτη είσοδο στην πρώτη έξοδο. Κάθε στοιχείο ή σύμβολο που αποθηκεύεται έχει μήκος 8 bit ή 1 Byte. Η FIFO αποτελεί σύγχρονο κύκλωμα (απαιτείται ρολόι clk) και αρχικοποιείται με χρήση σύγχρονου σήματος αρχικοποίησης $reset$. Η FIFO έχει μια είσοδο $DataIn$ και μια έξοδο $DataOut$. Στην FIFO εγγράφεται ότι υπάρχει στην είσοδο μόνο όταν το σήμα $writeEn$ είναι αληθές. Από την FIFO διαβάζεται το κατά προτεραιότητα πρώτο σύμβολο που εισάχθηκε μόνο όταν το σήμα $readEn$ είναι αληθές. Τέλος, για έλεγχο κατά την προσομοίωση προστίθενται και σήματα που δηλώνουν αν η FIFO είναι άδεια ($Empty$) ή γεμάτη ($Full$). Η FIFO που χρησιμοποιήθηκε στην υλοποίηση του αποκωδικοποιητή είναι της πηγής [XXVI] με μερικές τροποποιήσεις.

8.7 Μπλοκ Διόρθωσης Λαθών

Το μπλοκ αυτό διαχειρίζεται την ανάγνωση της FIFO και την έξοδο του αποκωδικοποιητή. Κατά την λήψη του σήματος εκκίνησης διαδικασίας επιδιόρθωσης $corrStart$ από τον Forney ξεκινά την διαδικασία διόρθωσης λαθών.

Ως είσοδοι του μπλοκ λαμβάνονται ο πίνακας θέσεων λάθους του Chien ($ErrLocations$), καθώς και οι τιμές των λαθών που υπολογίστηκαν στον Forney με την μορφή πίνακα τιμών λάθους ($ErrMagnitudes$). Επίσης, λαμβάνεται και η έξοδος της FIFO ως σήμα $PolyIn$. Ως έξοδο βγαίνουν το διορθωμένο σύμβολο $PolyOut$, καθώς και σήμα έγκυρης εξόδου $validOut$ και σήμα ανάγνωσης της FIFO $readEnable$.

Η έξοδος $PolyOut$ ισούται με το άθροισμα (XOR):

$$PolyOut = PolyIn \oplus ErrMag \tag{81}$$

Το $ErrMag$ έχει τιμή 0 αν η τωρινή θέση δεν αποτελεί θέση λάθους και λαμβάνει την τιμή του αντίστοιχου $ErrMag[i]$ για κάθε θέση $ErrLoc[i]$ που αποτελεί θέση λάθους. Οι τιμές των υπόλοιπων σημάτων εξόδου εξαρτώνται από την τιμή του μετρητή $PolyPos$, καθώς και τις τιμές των σημάτων αρχικοποίησης, όπως ορίζει ο Πίνακας 8-2.

	validOut	readEnable
$reset = '1'$	$'0'$	$'0'$
$corrStart = '1'$	$'0'$	$'1'$
$0 < PolyPos < n-2$	$'1'$	$'1'$
$PolyPos = n-2$	$'1'$	$'0'$
$PolyPos = n-1$	$'1'$	$'0'$
$PolyPos = n$	‘0’	‘0’

Πίνακας 8-2 - Τιμές σημάτων εξόδου του μπλοκ διόρθωσης λαθών

Στον κύκλο που το σήμα $corrStart$ έχει τιμή ‘1’ το σήμα $readEnable$ λαμβάνει τιμή ‘1’ εξωτερικά του μπλοκ διόρθωσης λαθών με χρήση πολυπλέκτη. Ο πολυπλέκτης αυτός αναθέτει στο $readEnable$ την τιμή $'1'$ όταν το σήμα $corrStart = '1'$, ενώ αλλιώς την τιμή του σήματος $readEnableEC$ από το μπλοκ διόρθωσης λαθών.

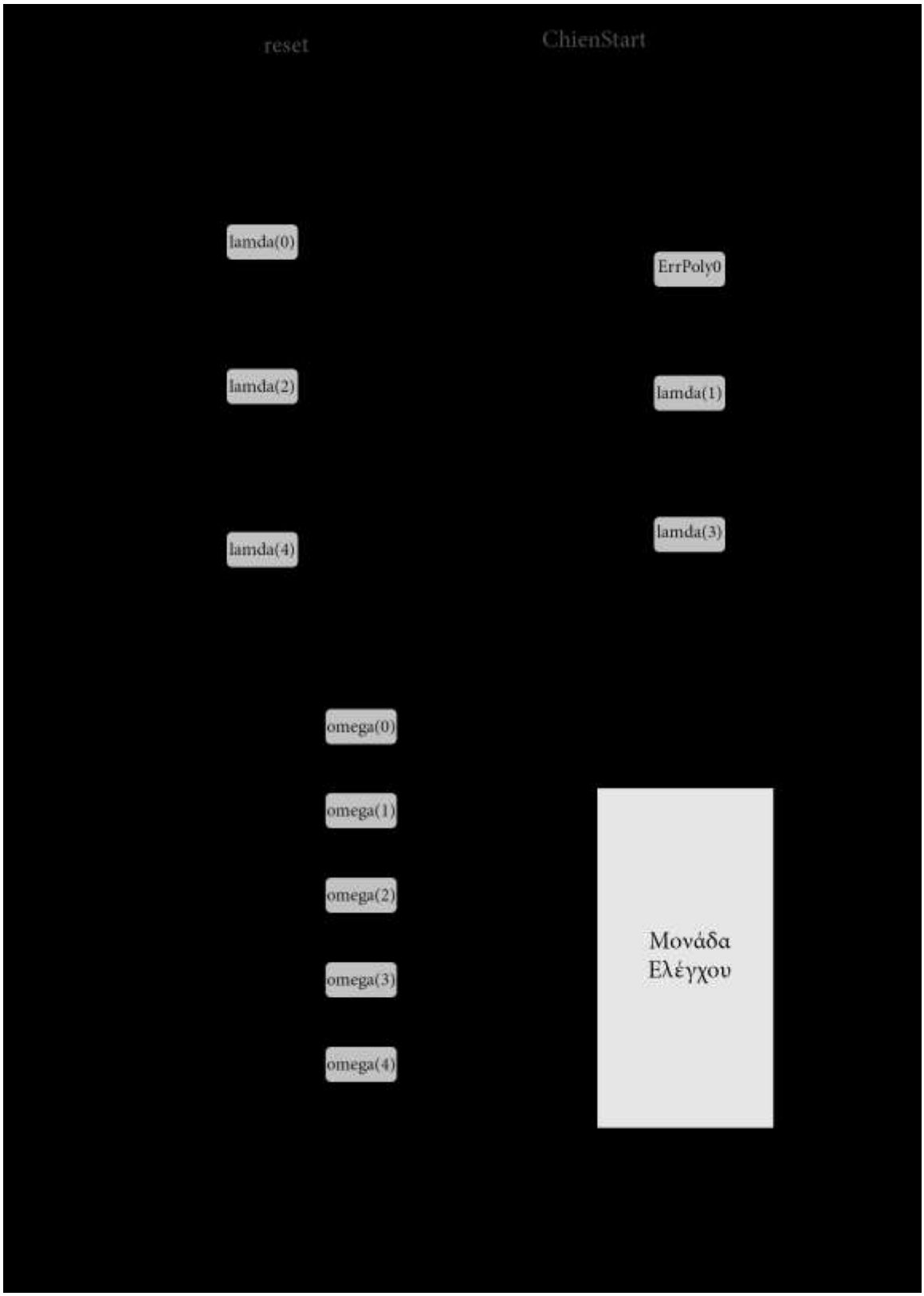
8.8 2-Παράλληλο Μπλοκ Chien

Με βελτιστοποίηση του μπλοκ του Chien μέσω παραλληλοποίησης (Ενότητα 6.3.2.2) μειώνεται αισθητά η συνολική καθυστέρηση του αποκωδικοποιητή και επιτυγχάνεται ταχύτερη επεξεργασία. Ένα κλασσικό σειριακό μπλοκ του Chien απαιτεί n κύκλους επεξεργασίας. Με 2-παραλληλοποίηση ο χρόνος αυτός μειώνεται σε:

$$cycles = \frac{n}{2} + t \tag{82}$$

όπου $\frac{n}{2}$ οι κύκλοι αναζήτησης και t οι κύκλοι συγχώνευσης των μερικών λύσεων σε μια.

Όπως αναφέρθηκε και στην αντίστοιχη Ενότητα, απαιτούνται 2 αντίγραφα του μπλοκ του Chien. Δηλαδή απαιτούνται δύο υπολογισμοί αθροισμάτων για κάθε πολυώνυμο, $\Lambda(x)$ και $\Omega(x)$. Ο ένας υπολογισμός αναφέρεται σε ζυγές θέσεις λάθους, ενώ ο άλλος σε περιττές. Για μείωση της καθυστέρησης κρίσιμου μονοπατιού στην πρώτη επανάληψη δεν υπολογίζεται θέση λάθους για περιττή θέση, γιατί απαιτούνται δύο πολλαπλασιασμοί με τις αντίστοιχες σταθερές $a^{2t}, a^{2t+1}, \dots, a^{2t+t-1}$. Έτσι, οι περιττοί υπολογισμοί ισούνται με το άθροισμα των αντίστοιχων μεταβλητών κατάστασης $lamda$ και $omega$, ενώ οι περιττοί με το πολλαπλασιασμό των αντίστοιχων τιμών με τις σταθερές. Η επόμενη τιμή των μεταβλητών κατάστασης ισούται με τον πολλαπλασιασμό επί των αντίστοιχων σταθερών 2 φορές. Τελικώς, προκύπτει το μπλοκ 2-παράλληλου Chien του σχήματος 8-6 της επόμενης σελίδας.



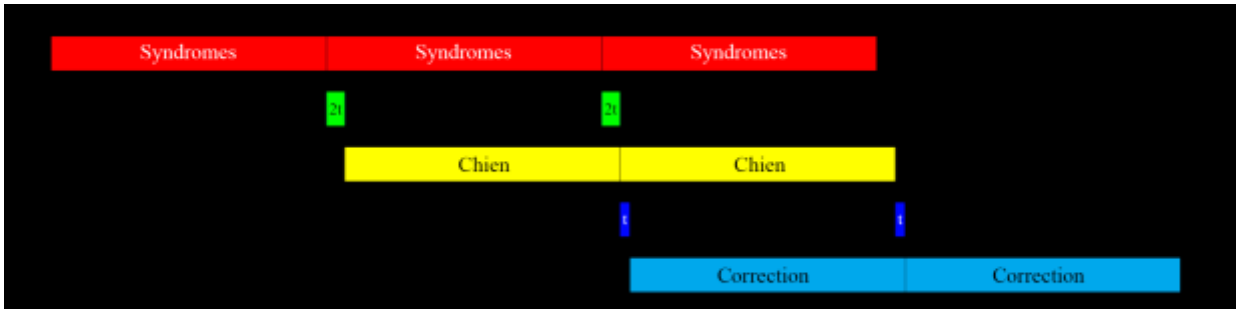
Σχήμα 8-6 – Μπλοκ εύρεσης θέσεων λάθους με 2-Παράλληλο Chien

Εφόσον το βελτιστοποιημένο μπλοκ του Chien εκτελείται σε λιγότερους κύκλους από τους ελάχιστους κύκλους που απαιτούνται για είσοδο νέου κωδικοποιημένου μηνύματος, δεν απαιτείται πλέον το σήμα ελέγχου $NoOutput$ που χρειαζόταν στο σειριακό μπλοκ του Chien.

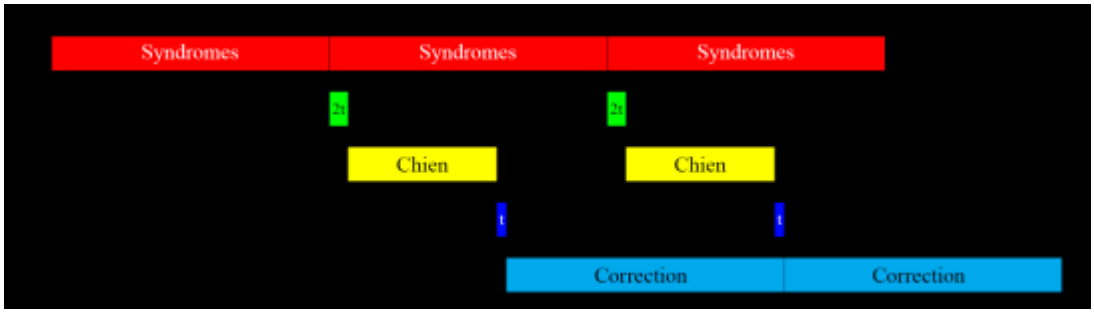
8.9 Βελτιστοποιημένη Αρχιτεκτονική

Το 2-παράλληλο μπλοκ του Chien της προηγούμενης Ενότητας μπορεί να ενσωματωθεί στην ήδη υπάρχουσα αρχιτεκτονική χωρίς επιπλέον τροποποιήσεις. Με χρήση 2-παράλληλου μπλοκ του Chien το απαιτούμενο μέγεθος της FIFO μειώνεται από $2n + 3t + 5$ σε $n + 4 * t + \frac{n+1}{2} + 6$.

Στα σχήματα 8-7 και 8-8 αντίστοιχα παρουσιάζεται ο χρονισμός των δύο αρχιτεκτονικών, δηλαδή η ακολουθία των 5 βημάτων και πώς αλληλοεπιδρούν μεταξύ τους στο πεδίο του χρόνου.



Σχήμα 8-7 Χρονισμός Σειριακής Αρχιτεκτονικής Αποκωδικοποιητή RS



Σχήμα 8-8 Χρονισμός Αρχιτεκτονικής Αποκωδικοποιητή RS με 2-παράλληλο Chien

Παρατηρείται πως η δεύτερη αρχιτεκτονική υπολογίζει σχεδόν 2 φορές πιο γρήγορα τις θέσεις και τιμές των λαθών, κάτι που επιτρέπει ταχύτερη εκκίνηση της διαδικασίας διόρθωσης λαθών. Με επιπρόσθετη μείωση των κύκλων εκτέλεσης του Chien σε $\frac{n}{p} + t$ μπορεί να επιτευχθεί και μεγαλύτερη επιτάχυνση (όπως αναφέρθηκε και στην Ενότητα 6.3.2.3), ωστόσο μια τέτοια επιτάχυνση συνεπάγεται και πολλαπλασιασμό του μεγέθους του μπλοκ κατά p τουλάχιστον. Η ακριβής αρχιτεκτονική που θα χρησιμοποιηθεί σε μια υλοποίηση εξαρτάται από την εφαρμογή στην οποία απαιτείται διόρθωση λαθών, καθώς και τους πόρους επεξεργασίας που διατίθενται. Γενικά μια σειριακή υλοποίηση εξοικονομεί πολύ πόρους, ενώ μια παράλληλη υλοποίηση προσφέρει μεγαλύτερη ταχύτητα επεξεργασίας δεδομένων.

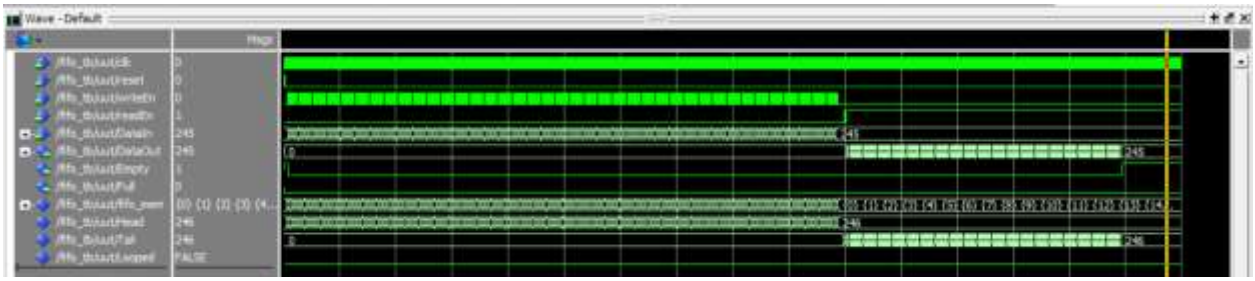
9. ΕΠΑΛΗΘΕΥΣΗ

9.1. Προσομοίωση Μνήμης FIFO

Για την επιβεβαίωση της λειτουργίας της FIFO χρησιμοποιείται Testbench που:

1. Αρχικοποιεί την FIFO με χρήση του σήματος $reset$.
2. Δίνει ως είσοδο τους αριθμούς του διαστήματος $[0, 245]$, ταλαντώνοντας το σήμα $WriteEn$ στις τιμές '1' και '0'.
3. Μετά την εγγραφή των τιμών $[0, 245]$ το σήμα $ReadEn$ αλλάζει για ένα κύκλο σε '1', σε '0' για ένα κύκλο και στην συνέχεια διατηρείται σταθερά σε '1'.

Η προσομοίωση του Testbench στο ModelSim δίνει:



Επομένως μέσω ορθής χρήσης των σημάτων $writeEn$ και $readEn$ γράφονται στην σωστή στιγμή, οι σωστές τιμές στην μνήμη FIFO, δίνοντας στην έξοδο τις τιμές που δόθηκαν στην είσοδο με προτεραιότητα FIFO.

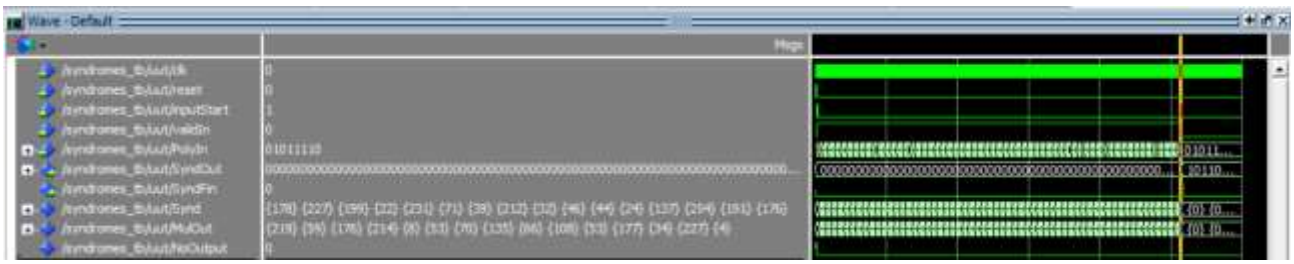
9.2 Προσομοίωση Μπλοκ Υπολογισμού Συνδρόμων

Τα σύνδρομα προσομοιώνονται με χρήση Testbench που διαβάζει το αρχείο "$input.txt$" που κατασκευάστηκε από τον κώδικα Python. Αρχικά, αρχικοποιείται το μπλοκ με χρήση του σήματος $reset$. Στην συνέχεια, λαμβάνει για ένα κύκλο το σήμα $inputStart$ τιμή '1' και δίνεται και η πρώτη γραμμή ως είσοδο. Μέσω επανάληψης δίνονται με $validIn = '1'$ και οι υπόλοιπες γραμμές στο μπλοκ για τον υπολογισμό των συνδρόμων. Όταν ολοκληρωθεί η ανάγνωση του αρχείου το σήμα $inputStart$ λαμβάνει εκ νέου την τιμή '1', ώστε να δοθούν στην έξοδο τα υπολογισμένα σύνδρομα.

Σύμφωνα με την έξοδο του λογισμικού για κωδικοποιημένο μήνυμα με συγκεκριμένα τυχαία λάθη, του οποίου τα σύμβολα γράφτηκαν στο αρχείο $input0.txt$, αναμένονται τα σύνδρομα να λάβουν τις τελικές τιμές:

$$178, 227, 199, 22, 231, 71, 39, 212, 32, 46, 44, 24, 137, 254, 191, 176$$

Η προσομοίωση του Testbench αυτού στο ModelSim δίνει:



Άρα, υπολογίζονται τα ίδια σύνδρομα και η έξοδος του μπλοκ συνδρόμων θεωρείται έγκυρη.

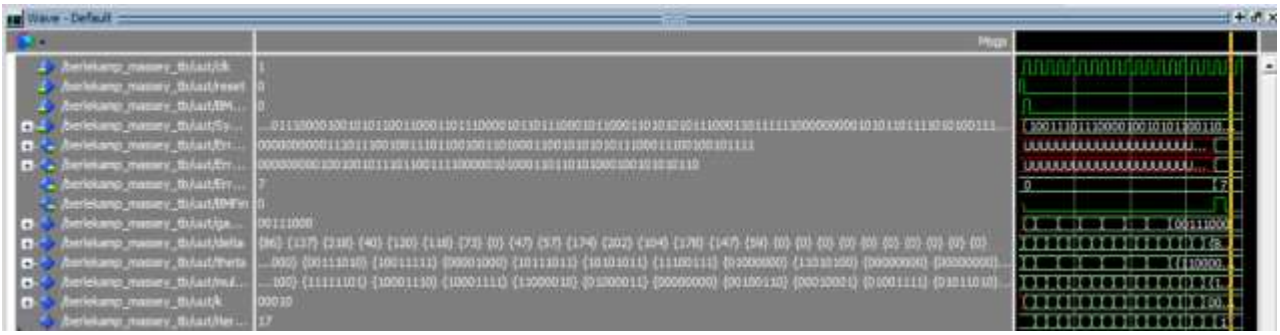
9.3 Προσομοίωση Μπλοκ RiBM

Για την επαλήθευση αυτού του μπλοκ δίνετε ως είσοδος η υπολογισμένη συμβολοσειρά 0-1 συνδρόμων που υπολογίστηκε για το ίδιο κωδικοποιημένο μήνυμα στην Ενότητα 9.2:

1001110111000010010101100110001101110000101101110001011000110101010111000110111111000000000101011011110101001110000110001010101010

Επίσης απαιτείται και κατάλληλη διαχείριση των σημάτων ελέγχου $reset$ και $BMStart$.

Η προσομοίωση στο ModelSim δίνει την εξής κυματομορφή:



Από τις τιμές των καταχωρητών $delta$ παρατηρείται πως υπολογίζονται τα πολυώνυμα:

$$\Lambda(x) = [47, 57, 174, 202, 104, 178, 147, 59, 0]$$
$$\Omega(x) = [86, 137, 218, 40, 120, 118, 73, 0]$$

Δηλαδή το μπλοκ υπολογίζει τα ίδια πολυώνυμα $\Lambda(x)$ και $\Omega(x)$ με το λογισμικό και το μπλοκ RiBM εκτελείται σωστά.

9.4 Προσομοίωση Σειριακού Μπλοκ Chien

Για την επαλήθευση του μπλοκ του Chien απαιτείται και είσοδος των δύο υπολογισμένων πολυωνύμων του RiBM σε μορφή δυαδικής συμβολοσειράς, καθώς και κατάλληλη διαχείριση των σημάτων ελέγχου $reset$ και $ChienStart$.

Σύμφωνα με την έξοδο του λογισμικού τα λάθη που προστίθενται είναι τα εξής:

```
rCode[232] ^=  206
rCode[82] ^=  66
rCode[146] ^=  204
rCode[6] ^=  27
rCode[141] ^=  231
rCode[101] ^=  119
rCode[254] ^=  97
```

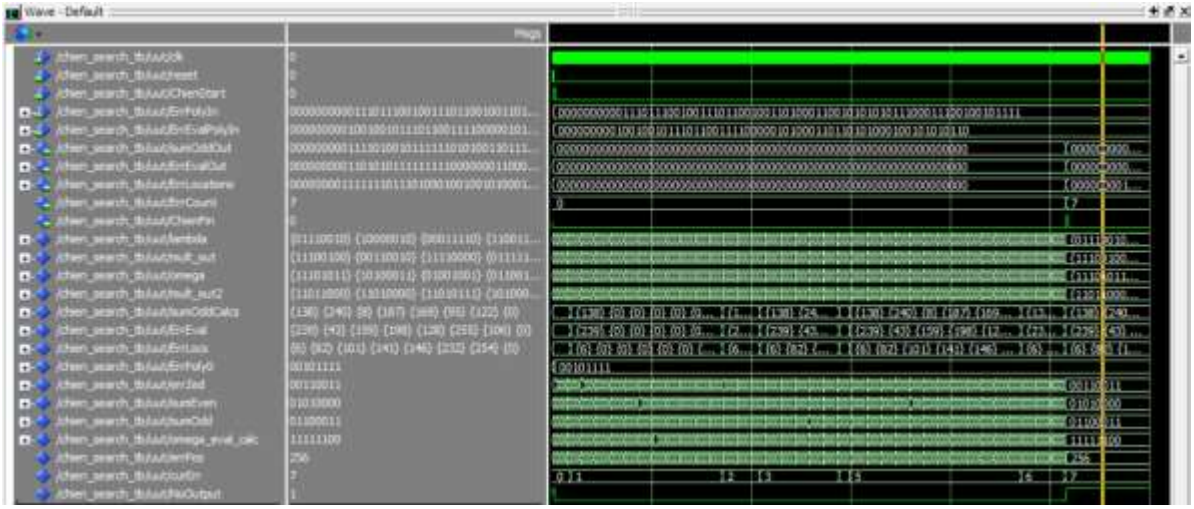
Έτσι, οι έξοδοι του Chien πρέπει να είναι ταυτόσημη με τους εξής πίνακες που υπολογίζονται για τα αντίστοιχα λάθη από το λογισμικό:

```
Error Locations: [6, 82, 101, 141, 146, 232, 254, 0]
00000000111111101110100010010010100011010110010101010010000000110

ErrPoly Derivatives: [138, 240, 8, 187, 169, 95, 122, 0]
00000000011110100101111110101001101110110000100011110000010001010

ErrEvalPoly Evaluations: [239, 43, 159, 198, 128, 255, 106, 0]
00000000011010101111111110000000110001101001111100101011111101111
```

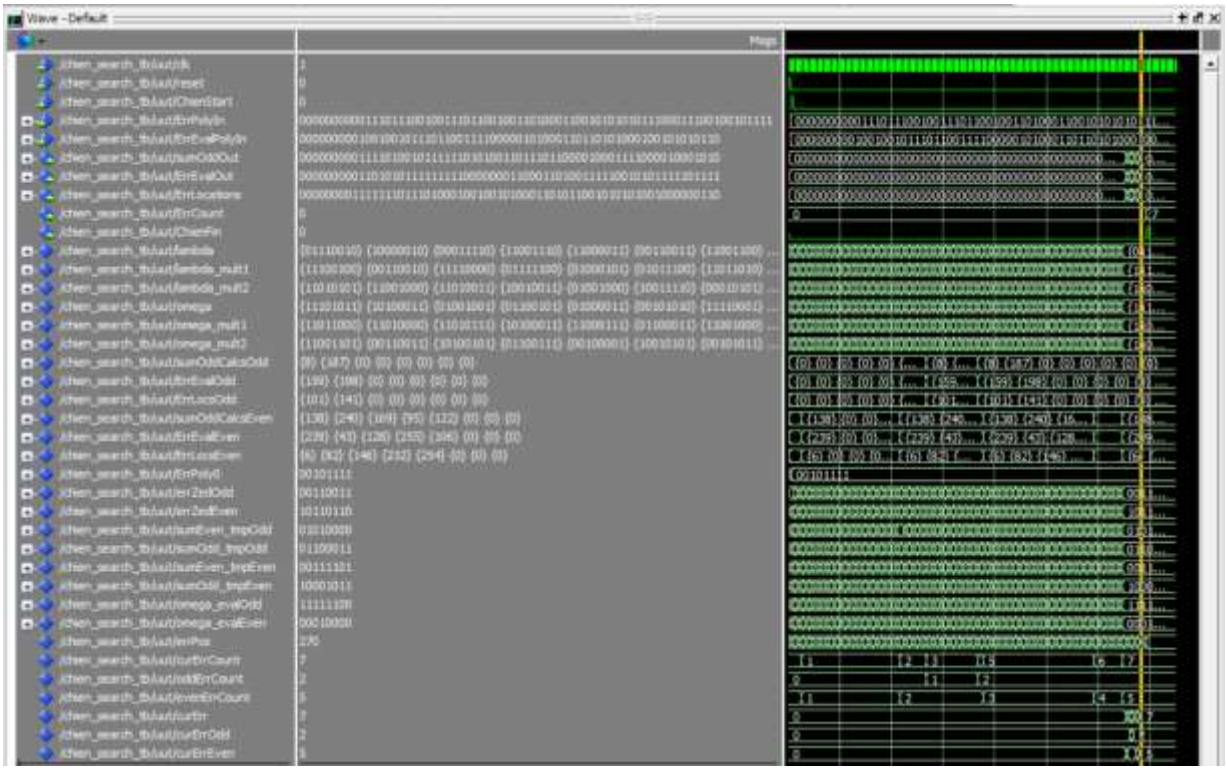
Η προσομοίωση στο ModelSim δίνει τα εξής αποτελέσματα:



Αναγνωρίζονται λάθη στα σύμβολα 6, 82, 101, 141, 146, 232 και 254 όπως αναμενόταν, καθώς και αποθηκεύονται οι παράγωγοι $\Lambda'(a^i)$ και αξιολογήσεις $\Omega(a^i)$ που αντιστοιχούν σε αυτές τις θέσεις λάθους. Μετά το πέρας της διαδικασίας αναζήτησης λαθών το μπλοκ τίθεται σε κατάσταση αναμονής για είσοδο και το σήμα $NoOutput$ λαμβάνει την τιμή '1'. Με αυτόν τον τρόπο στην επόμενη εκκίνηση της διαδικασίας εύρεσης λαθών το μπλοκ δεν θα δώσει έξοδο. Επομένως, το μπλοκ του Chien εκτελείται σωστά.

9.5 Προσομοίωση 2-Παράλληλου Μπλοκ Chien

Χρησιμοποιώντας το ίδιο ακριβώς Testbench επαληθεύεται εύκολα και η λειτουργίας του 2-παράλληλου μπλοκ. Η προσομοίωση του μπλοκ δίνει τα ακόλουθα αποτελέσματα:



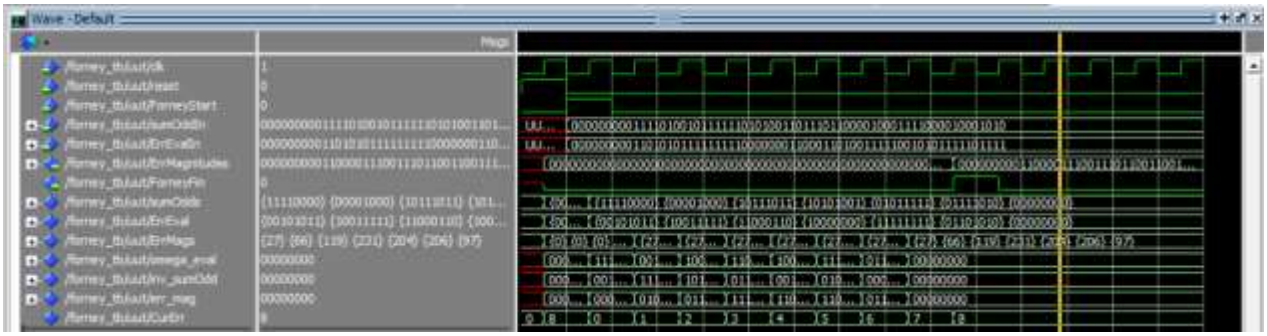
Υπολογίζονται και πάλι οι σωστές θέσεις λάθους, καθώς και οι αντίστοιχες παράγωγοι $\Lambda'(a^i)$ και αξιολογήσεις τους $\Omega(a^i)$. Επομένως το 2-παράλληλο μπλοκ εκτελείται σωστά.

9.6 Προσομοίωση Μπλοκ Forney

Η προσομοίωση του μπλοκ του Forney απαιτεί είσοδο των υπολογισμένων τιμών $\Lambda'(a^i)$ και $\Omega(a^i)$ που αντιστοιχούν στις θέσεις λάθους. Γράφοντας Testbench που λαμβάνει αυτές τις τιμές, σύμφωνα με την έξοδο του λογισμικού αναμένουμε τις εξής τιμές λάθους:

```
Error Magnitudes: [27, 66, 119, 231, 204, 206, 97, 0]
00000000011000011100111011001100111001110111011101000010000011011
```

Η προσομοίωση του μπλοκ του Forney δίνει τα εξής αποτελέσματα:

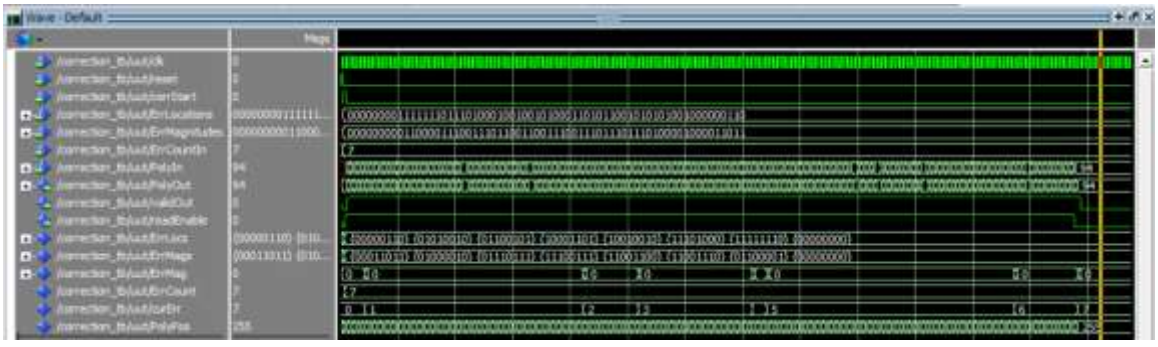


Παρατηρείται πως υπολογίζονται τα ίδια αποτελέσματα με το λογισμικό. Ως βελτιστοποίηση του Forney η τελευταία τιμή λάθους δεν εγγράφεται πρώτα σε καταχωρητή και συνδέεται απευθείας στην έξοδο. Έτσι το '0' μπορεί να βρεθεί στην τιμή του σήματος err_mag, το οποίο στον τελευταίο κύκλο έχει την τιμή "00000000". Συμπερασματικά, οι υπολογισμοί του μπλοκ του Forney εκτελούνται σωστά.

9.7 Προσομοίωση Μπλοκ Διόρθωσης Λαθών

Το μπλοκ διόρθωσης λαθών διαβάζει από την FIFO και λαμβάνει τις θέσεις λάθους από τον Chien και τις τιμές λάθους από τον Forney. Η έξοδος της FIFO προσομοιώνεται με ανάγνωση του αρχείου των συμβόλων του κωδικοποιημένου μηνύματος.

Η προσομοίωση του Testbench δίνει:

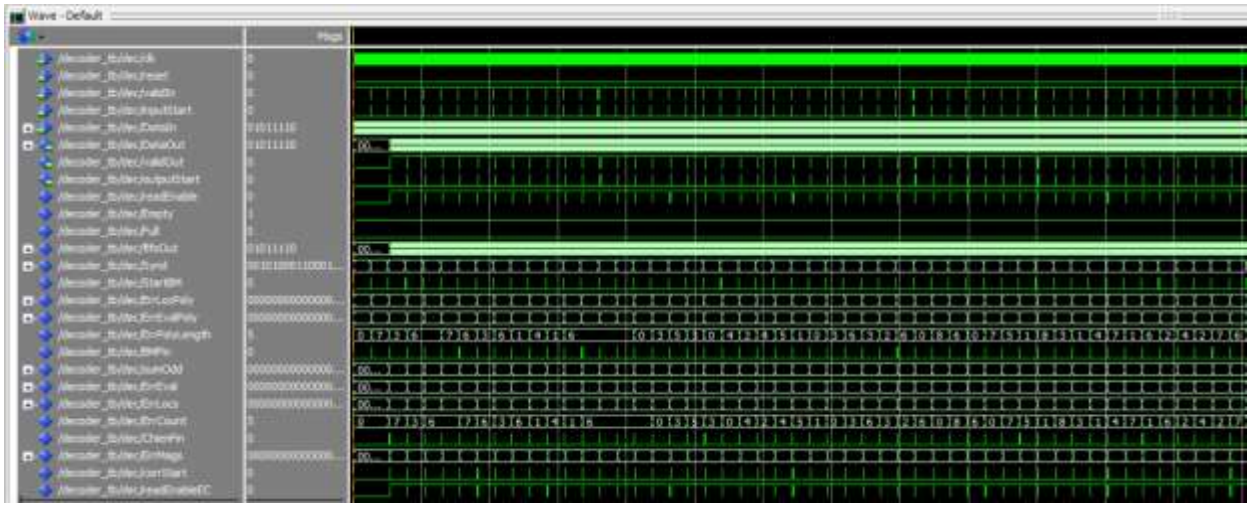


Παρατηρείται πως προστίθενται οι κατάλληλες τιμές λάθους στις αντίστοιχές τους θέσεις λάθους. Με αυτόν τον τρόπο διορθώνονται όλα τα λάθη του μηνύματος. Για παράδειγμα το τελευταίο σύμβολο (254^o) αλλάζει από 63 σε 94, διορθώνοντας το λάθος 97 που προστέθηκε.

9.8 Προσομοίωση Συνολικού Αποκωδικοποιητή με Σειριακό Chien

Η διαδικασία προσομοίωσης του συνολικού αποκωδικοποιητή παρουσιάζει μεγάλη ομοιότητα με την προσομοίωση του μπλοκ συνδρόμων. Για επιπλέον έλεγχο ελέγχονται $COUNT$ διαδοχικές αποκωδικοποιήσεις. Τα $COUNT$ αρχεία κατασκευάζονται από τον κώδικα επαλήθευσης λογισμικού που παρουσιάστηκε στο Κεφάλαιο 7.

Προσομοίωση Testbench για σύνολο $COUNT = 100$ αρχείων δίνει τα εξής αποτελέσματα:

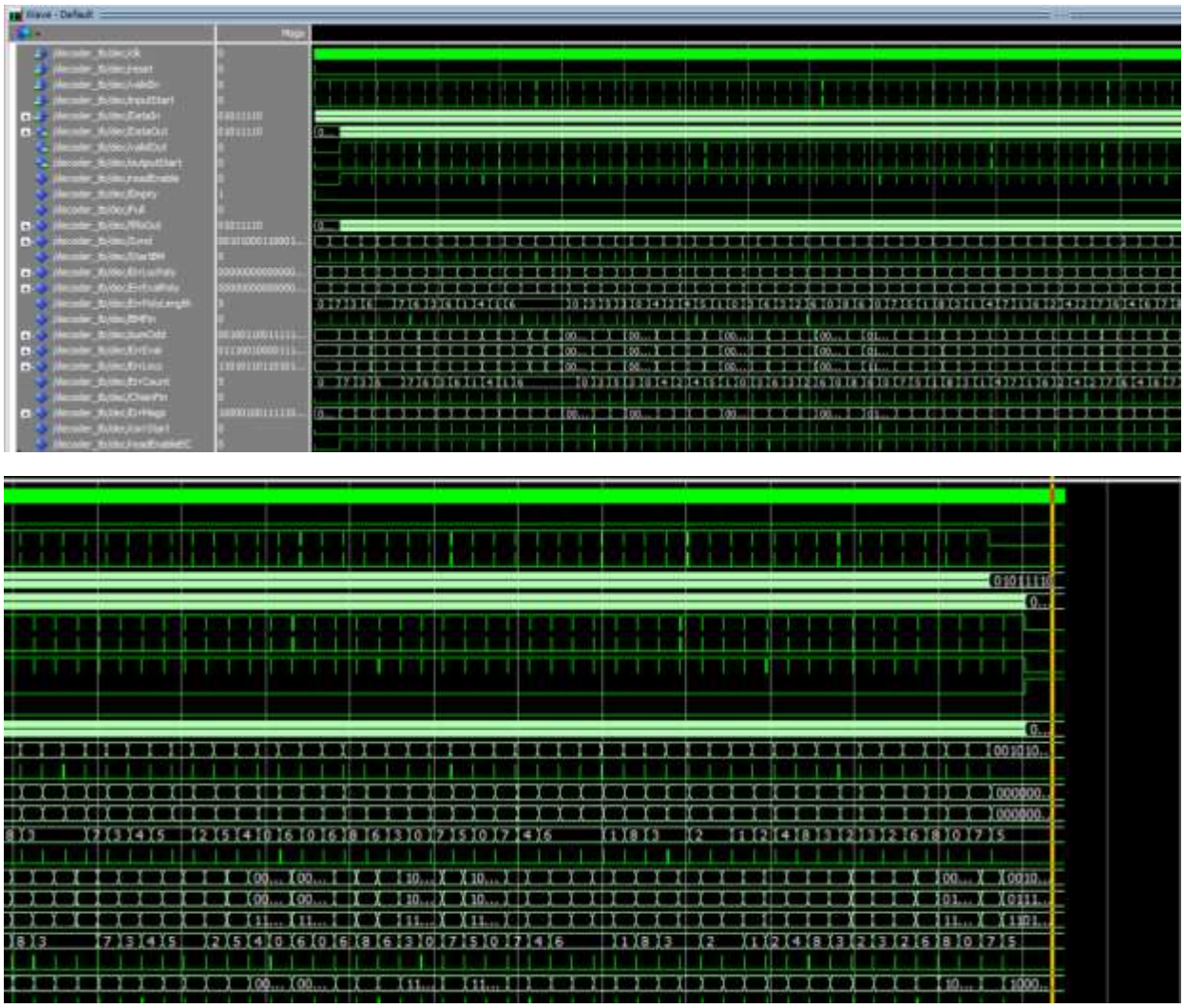


Παρατηρείται συνεχής ροή δεδομένων και υπολογίζονται τα ίδια πλήθη λάθους από τον RiBM και τον Chien. Φυσικά υπολογίζονται και τα ίδια πολυώνυμα $S(x), \Lambda(x), \Omega(x), \Lambda'(a^i), \Omega(a^i)$, καθώς και οι ίδιες θέσεις και τιμές λάθους στα μπλοκ Chien και Forney αντίστοιχα.

9.9 Προσομοίωση Συνολικού Αποκωδικοποιητή με 2-Παράλληλο Chien

Χρησιμοποιώντας το ίδιο ακριβώς Testbench μπορεί να προσομοιωθεί και η λειτουργία του αποκωδικοποιητή με 2-παράλληλο Chien. Φυσικά δεν απαιτείται το ίδιο μήκος FIFO και πραγματοποιείται αλλαγή της παραμέτρου αυτής για εξοικονόμηση πόρων.

Η προσομοίωση του αποκωδικοποιητή στο ModelSim δίνει τις εξής κυματομορφές:



Παρατηρείται και πάλι σωστός χρονισμός της επικοινωνίας των μπλοκ και υπολογίζονται τα σωστά πολυώνυμα. Κατά την διαδικασία διόρθωσης λαθών χρησιμοποιούνται οι σωστές θέσεις και τιμές λάθους και πραγματοποιείται διόρθωση των λαθών.

9.10 Σύνθεση των Διαφόρων Μπλοκ σε Xilinx FPGA

Στους παρακάτω πίνακες δίνονται τα αποτελέσματα της σύνθεσης των μπλοκ του αποκωδικοποιητή για τους κώδικες RS που αντιστοιχούν σε $1 \leq t \leq 8$ και στο FPGA Spartan3E-XC3500E-5 της εταιρίας Xilinx.

	Σειριακή Αρχιτεκτονική	2-Παράληλλη Αρχιτεκτονική
Μήκος FIFO	$2n + 3t + 5$	$n + (n + 1)/2 + 4t + 6$
Slices	26	26
Slice FFs	23	21
LUTs	48	47
Freq (MHz)	155.811	156.870

Πίνακας 9-1 - Σύνθεση FIFO στο Xilinx Spartan3E-XC3500E-5

t	RS	Slices	Slice FFs	LUTs	Freq (MHz)
1	$(255, 253)$	20	34	20	332.419
2	$(255, 251)$	38	66	37	312.282
3	$(255, 249)$	58	98	58	307.708
4	$(255, 247)$	78	130	78	311.447
5	$(255, 245)$	100	162	103	311.031
6	$(255, 243)$	121	194	125	310.616
7	$(255, 241)$	142	226	149	310.202
8	$(255, 239)$	163	258	171	309.790

Πίνακας 9-2 - Σύνθεση Μπλοκ Συνδρόμων στο Xilinx Spartan3E-XC3500E-5

t	RS	Slices	Slice FFs	LUTs	Freq (MHz)
1	$(255, 253)$	221	106	414	161.803
2	$(255, 251)$	410	171	672	168.658
3	$(255, 249)$	605	242	1127	167.958
4	$(255, 247)$	803	309	1491	173.941
5	$(255, 245)$	998	372	1852	173.259
6	$(255, 243)$	1189	438	2210	164.051
7	$(255, 241)$	1379	502	2568	164.051
8	$(255, 239)$	1573	568	2925	163.858

Πίνακας 9-3 - Σύνθεση Μπλοκ RiBM στο Xilinx Spartan3E-XC3500E-5

t	RS	Slices	Slice FFs	LUTs	Freq (MHz)
1	$(255, 253)$	62	86	57	192.793
2	$(255, 251)$	102	152	97	163.323
3	$(255, 249)$	146	216	123	139.445
4	$(255, 247)$	189	282	172	152.132
5	$(255, 245)$	247	346	221	149.881
6	$(255, 243)$	279	410	249	147.458
7	$(255, 241)$	320	474	277	145.705
8	$(255, 239)$	379	540	267	122.329

Πίνακας 9-4 - Σύνθεση Σειριακού Μπλοκ Chien στο Xilinx Spartan3E-XC3500E-5

t	RS	Slices	Slice FFs	LUTs	Freq (MHz)
1	$(255, 253)$	94	109	143	182.917
2	$(255, 251)$	154	202	254	150.132
3	$(255, 249)$	260	289	349	135.119
4	$(255, 247)$	334	380	413	136.010
5	$(255, 245)$	411	468	518	124.663
6	$(255, 243)$	506	557	620	123.682
7	$(255, 241)$	591	641	678	123.461
8	$(255, 239)$	677	737	772	122.838

Πίνακας 9-5 - Σύνθεση 2-Παράλληλου Μπλοκ Chien στο Xilinx Spartan3E-XC3500E-5

t	RS	Slices	Slice FFs	LUTs	Freq (MHz)
1	$(255, 253)$	32	18	59	148.610
2	$(255, 251)$	189	83	340	196.703
3	$(255, 249)$	202	93	384	149.626
4	$(255, 247)$	223	126	379	139.733
5	$(255, 245)$	246	159	370	146.239
6	$(255, 243)$	269	192	388	126.043
7	$(255, 241)$	291	223	406	126.975
8	$(255, 239)$	318	256	408	134.003

Πίνακας 9-6 - Σύνθεση Μπλοκ Forney στο Xilinx Spartan3E-XC3500E-5

t	RS	Slices	Slice FFs	LUTs	Freq (MHz)
1	$(255, 253)$	29	44	32	202.864
2	$(255, 251)$	53	62	64	152.031
3	$(255, 249)$	57	78	62	150.836
4	$(255, 247)$	71	96	68	147.462
5	$(255, 245)$	97	113	95	143.694
6	$(255, 243)$	103	130	97	145.680
7	$(255, 241)$	116	146	100	138.453
8	$(255, 239)$	126	164	102	138.287

Πίνακας 9-7 - Σύνθεση Μπλοκ Διόρθωσης Λαθών στο Xilinx Spartan3E-XC3500E-5

9.11 Σύνθεση Σειριακής Αρχιτεκτονικής σε Xilinx FPGA

t	RS	Slices	Slice FFs	LUTs	Freq (MHz)
1	$(255, 253)$	324	303	604	140.003
2	$(255, 251)$	728	523	1373	152.031
3	$(255, 249)$	973	731	1803	136.425
4	$(255, 247)$	1209	947	2245	129.535
5	$(255, 245)$	1460	1153	2700	130.795
6	$(255, 243)$	1712	1365	3147	126.043
7	$(255, 241)$	1939	1572	3568	126.807
8	$(255, 239)$	2180	1783	4005	122.329

Πίνακας 9-8 - Σύνθεση Σειριακής Αρχιτεκτονικής στο Xilinx Spartan3E-XC3500E-5

9.12 Σύνθεση 2-Παράλληλης Αρχιτεκτονικής σε Xilinx FPGA

t	RS	Slices	Slice FFs	LUTs	Freq (MHz)
1	$(255, 253)$	354	329	661	139.94
2	$(255, 251)$	811	570	1523	146.315
3	$(255, 249)$	1088	802	2026	135.119
4	$(255, 247)$	1338	1047	2501	127.134
5	$(255, 245)$	1629	1274	3025	124.663
6	$(255, 243)$	1898	1515	3518	123.682
7	$(255, 241)$	2128	1737	3935	123.461
8	$(255, 239)$	2400	1978	4421	122.254

Πίνακας 9-9 - Σύνθεση 2-Παράλληλης Αρχιτεκτονικής στο Xilinx Spartan3E-XC3500E-5

10.ΣΥΓΚΡΙΣΗ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ

Από τους πίνακες των αποτελεσμάτων σύνθεσης του αποκωδικοποιητή συμπεραίνεται πως η κύρια πηγή καθυστέρησης του αποκωδικοποιητή είναι το μπλοκ του Chien που υπολογίζει τις θέσεις των λαθών. Χρησιμοποιώντας 2-Παράλληλο Chien αντί του κλασσικού σειριακού μοντέλου, μειώνεται κατά μικρό ποσοστό η συχνότητα του μπλοκ λόγω της εισαγωγής επιπλέον πολλαπλασιαστών του τύπου μεταβλητής-σταθεράς. Ωστόσο, επιτυγχάνεται φυσικά και σχεδόν διπλασιασμός του ρυθμού εύρεσης των λαθών στο μπλοκ που συνεπάγεται αύξηση της διεκπεραιωτικής ικανότητας (throughput) του Chien και του συνολικού αποκωδικοποιητή.

Η μικρή αύξηση του συνολικού μεγέθους του κυκλώματος αποκωδικοποιητή από την χρήση 2-παράλληλου Chien, δεν συγκρίνεται με τους πόρους που καταλαμβάνει το μπλοκ του RiBM. Λόγω της χρήσης γενικών πολλαπλασιαστών και μεγάλου αριθμού καταχωρητών το μπλοκ του RiBM καταλαμβάνει το 50-60% του συνολικού αποκωδικοποιητή στην περίπτωση του 2-παράλληλου μοντέλου και το 60-70% στην περίπτωση του σειριακού μοντέλου. Ωστόσο, επειδή υπολογίζονται τα πολυώνυμα $\Lambda(x)$ και $\Omega(x)$ σε $2t$ αντί $3t$ κύκλων και επιτυγχάνεται μεγαλύτερη συχνότητα εκτέλεσης, ο αλγόριθμος προτιμάται.

Το μπλοκ συνδρόμων καταλαμβάνει μαζί με το μπλοκ διόρθωσης λαθών τους μικρότερους πόρους. Στο μπλοκ συνδρόμων χρησιμοποιούνται πολλαπλασιαστές μεταβλητής-σταθεράς που σε μικρές τιμές t συνεπάγονται μικρά κυκλώματα πολλαπλασιαστών. Γενικά, οι πολλαπλασιαστές σταθερών a^t καταλαμβάνουν περισσότερους πόρους όσο αυξάνεται η δύναμη t. Το μπλοκ διόρθωσης λαθών αποτελείται από 2 πίνακες καταχωρητών με συνολικό πλήθος καταχωρητών $2t$ και κυκλώματα ελέγχου που επίσης δεν συγκρίνονται με αυτά του RiBM ή και του Chien.

Το μπλοκ του Forney αποτελείται από μια ROM αντιστρόφων πολλαπλασιασμού, ένα γενικό πολλαπλασιαστή και 3 πίνακες καταχωρητών, με συνολικό πλήθος καταχωρητών $3(t-1)$ μέσω βελτιστοποιήσεων εισόδου και εξόδου. Τελικώς, καταλαμβάνονται περίπου οι ίδιοι πόροι με το σειριακό μπλοκ του Chien.

10.1 Σύγκριση με υλοποίηση της πηγής [XX]

Στην πηγή [XX] υλοποιήθηκε αποκωδικοποιητής για την περίπτωση κώδικα $RS(255, 251)$ στο FPGA Xilinx Spartan3-X3S50TQ155-5. Ο παρακάτω πίνακας συγκρίνει τα διάφορα μπλοκ από τα οποία αποτελείται ο αποκωδικοποιητή της πτυχιακής εργασίες με τα αντίστοιχα μπλοκ της υλοποίησης [XX].

		Slices	Slice FFs	LUTs	IOBs	Freq (MHz)
Υπολογισμός Συνδρόμων	Προτ.	38	66	37	45	241.250
	[XX]	56	72	104	50	180.490
Επίλυση Βασικής Εξίσωσης	Προτ.	408	165	762	78	135.825
	[XX]	662	64	1167	74	102.140
Μπλοκ Chien	Σειριακό	99	152	88	94	138.198
	2-Παράλλ.	158	203	249	94	130.750
	[XX]	60	32	105	66	184.380
Μπλοκ Forney	Προτ.	174	59	340	52	165.414
	[XX]	292	24	509	90	184.380
Μπλοκ Διόρθωσης Λαθών	Προτ.	53	62	64	55	200.138
	[XX]	28	41	41	50	183.440

Πίνακας 10-1 - Σύνθεση στο Xilinx Spartan3-X3S50TQ155-5 για κώδικα RS(255,251)

Από την σύγκριση αυτή συμπεραίνεται πως η υλοποίηση που προτείνεται στην παρούσα πτυχιακή εργασία καταλαμβάνει περισσότερο χώρο από την υλοποίηση της πηγής [XX] στα περισσότερα μπλοκ, καθώς και υστερεί σε συχνότητα. Ωστόσο, η συχνότητα δεν αποτελεί κύριο κριτήριο σύγκρισης και χρησιμοποιώντας 2-Παράλληλο Chien η υλοποίηση της πτυχιακής παρουσιάζει πολύ πιθανά μεγαλύτερη διεκπεραιωτική ικανότητας από την πηγή [XX].

10.2 Σύγκριση με υλοποίηση της πηγής [XXVIII]

Κύριο συμπέρασμα της προηγούμενης σύγκρισης αποτελεί πως η υλοποίηση που προτείνετε στην παρούσα πτυχιακή εργασία δεν παρουσιάζει μεγάλη αποδοτικότητα στην χρήση πόρων ανά μπλοκ. Η υλοποίηση της πηγής [XXVIII] για κώδικα $RS(255, 251)$ είχε ως κύριο στόχο την μείωση των πόρων. Στον παρακάτω πίνακα παρουσιάζεται σύγκριση με την υλοποίηση αυτή:

	Slices	Slice FFs	LUTs	IOBs
Σειριακό μοντέλο	728	523	1373	22
2-Παράλ. μοντέλο	811	570	1523	22
[XXVIII]	779	528	1451	19

Πίνακας 10-2 - Σύνθεση στο Xilinx Spartan3E-XC3500E-5 για κώδικα RS(255,251)

Οι υλοποιήσεις καταλαμβάνουν περίπου ίσους πόρους με το σειριακό μοντέλο να καταλαμβάνει λίγο λιγότερους πόρους από την υλοποίηση [XXVIII], ενώ το 2-παράλληλο μοντέλο ένα μικρό ποσοστό πόρων παραπάνω.

10.3 Σύγκριση με υλοποίηση της πηγής [XXIX]

Στο παρακάτω πίνακα παρουσιάζεται σύγκριση με την υλοποίηση της πηγής [XXIX] για την περίπτωση κώδικα $RS(255, 239)$:

	Slices	Slice FFs	LUTs	IOBs
Σειριακό μοντέλο	2180	1783	4005	22
2-Παράλ. μοντέλο	2400	1978	4421	22
[XXIX]	1830	1043	3408	21

Πίνακας 10-3 - Σύνθεση στο Xilinx Spartan3E-XC3500E-5 για κώδικα RS(255,239)

Η υλοποίηση που προτείνεται στην πτυχιακή καταλαμβάνει περισσότερους πόρους λόγω της χρήσης το αλγορίθμου RiBM αντί του κλασσικού BM που εφαρμόζεται στην υλοποίηση [XXIX].

10.4 Σύγκριση Chien με την υλοποίηση της πηγής [XXIV]

Στην Ενότητα 6.3.2.1 παρουσιάστηκε η βελτιστοποίηση χρήσης λογικών πυλών του αλγορίθμου Chien. Σύγκριση με την υλοποίηση Chien της πηγής [XXIV] για κώδικα $RS(255, 239)$ στο FPGA Xilinx Spartan 3E-XC3500E-5 παρουσιάζεται στο παρακάτω πίνακα:

	Slices	Slice FFs	LUTs
Σειριακός Chien	379	540	267
Υπολογισμός $\Lambda(x)$	84	89	154
[XXIV]	271	35	504

Πίνακας 10-4 - Σύνθεση Chien στο Xilinx Spartan3E-XC3500E-5 για κώδικα RS(255,239)

Το μπλοκ του Chien στην υλοποίηση της πτυχιακής υπολογίζει την αξιολόγηση του $\Lambda(x)$, καθώς και του $\Omega(x)$. Στην δεύτερη γραμμή «απενεργοποιήθηκαν» οι υπολογισμοί του $\Omega(x)$, ώστε να επικεντρωθεί η σύγκριση στον υπολογισμό του $\Lambda(x)$. Επιτυγχάνεται πολύ μεγαλύτερη βελτιστοποίηση της χρήσης πόρων του FPGA με την χρήση πολλαπλασιαστών του τύπου μεταβλητής-σταθεράς παρά με την χρήση του αλγορίθμου της πηγής [XXIV], με τον οποίο τελικώς μειώνεται μόνο ο αριθμός των απαιτούμενων πράξεων. Η υλοποίηση της πηγής [XXIV] πραγματοποιήθηκε πολύ πιθανά χρησιμοποιώντας γενικούς πολλαπλασιαστές, εφόσον η χρήση των πολλαπλασιαστών μεταβλητής-σταθεράς είναι ανέφικτη στην περίπτωση εφαρμογής αυτής της αρχιτεκτονικής.

10.5 Σύγκριση με υλοποίηση της πηγής [XXX]

Στο παρακάτω πίνακα παρουσιάζεται σύγκριση της υλοποίησης της πτυχιακής με την υλοποίηση της πηγής [XXX].

	Slices	Slice FFs	LUTs	Freq (MHz)
Σειριακό μοντέλο	2183	1783	4005	105.208
2-Παραλ. μοντέλο	2410	1978	4421	105.164
[XXX]	1830	1043	3408	100.00

Πίνακας 10-5 - Σύνθεση στο Xilinx Spartan3E-XC3500E-4 για κώδικα RS(255,239)

Με περίπου 20% αύξηση της χρήσης πόρων του FPGA επιτυγχάνεται 5% μεγαλύτερη συχνότητα εκτέλεσης του αποκωδικοποιητή στην περίπτωση κώδικα $RS(255,249)$, για τον οποίο ο αποκωδικοποιητής που προτείνετε στην παρούσα πτυχιακή παρουσιάζει την χαμηλότερη απόδοση μεταξύ όλων των t του διαστήματος $[1,8]$. Η αύξηση πόρων οφείλεται στην χρήση του RiBM που για $t = 8$ εκτελείται σε $2t = 16$ αντί $3t = 24$ κύκλων. Έτσι συνδυάζοντας τον με 2-Παράλληλο Chien μειώνονται κατά πολύ οι κύκλοι επεξεργασίας του αποκωδικοποιητή και επιτυγχάνεται μεγαλύτερη διεκπεραιωτική ικανότητα.

10.6 Σύγκριση με υλοποίηση της πηγής [XXXI]

Στο παρακάτω πίνακα παρουσιάζεται σύγκριση με την υλοποίηση της πηγής [XXXI].

	Slices	Slice FFs	LUTs	IOBs	Freq (MHz)
Σειριακό μοντ.	2108	1791	3895	22	241.348
2-Παραλ. μοντ.	2444	1974	4567	22	198.187
[XXXI]	1862	745	3484	21	152.592

Πίνακας 10-6 - Σύνθεση στο Xilinx Virtex4-XC4VLX15-12SF363 για κώδικα RS(255,239)

Η υλοποίηση της πτυχιακής περιέχει περισσότερους καταχωρητές και πολλές βελτιστοποιήσεις στην καθυστέρηση κρίσιμου μονοπατιού στα διάφορα μπλοκ του αποκωδικοποιητή. Έτσι, ακόμα και αν καταλαμβάνονται περισσότεροι πόροι επιτυγχάνεται πολύ μεγαλύτερη συχνότητα εκτέλεσης σε σχέση με την υλοποίηση της πηγής [XXXI]. Λόγω της αρκετά μεγαλύτερης συχνότητας εκτέλεσης ο αποκωδικοποιητής παρουσιάζει μεγαλύτερη ταχύτητα ήδη από το σειριακό μοντέλο. Στο 2-παράλληλο μοντέλο παρουσιάζεται αισθητή μείωση της συχνότητας που ωστόσο μέσω της μείωσης των κύκλων επεξεργασίας του μηνύματος τελικώς προσφέρει και πάλι ακόμα μεγαλύτερη διεκπεραιωτική ικανότητα.

10.7 Σύγκριση με υλοποίηση της πηγής [XXXII]

Στο παρακάτω πίνακα παρουσιάζεται σύγκριση με την υλοποίηση της πηγής [XXXII].

	Slice FFs	LUTs	IOBs	Freq (MHz)
Σειριακό μοντέλο	1783	4005	22	122.329
2-Παραλ. μοντέλο	1978	4421	22	122.254
[XXXII]	1043	3408	23	116.190

Πίνακας 10-7 - Σύνθεση στο Xilinx Spartan3E-XC3500E-5 για κώδικα RS(255,239)

Η υλοποίηση της πηγής [XXXII] καταλαμβάνει πολύ λιγότερο χώρο λόγω της εξοικονόμησης πόρων του FPGA, ωστόσο υστερεί σε συχνότητα, καθώς και ταχύτητα επεξεργασίας από την υλοποίηση της πτυχιακής. Ειδικά στην περίπτωση του 2-παράλληλου μοντέλου, το μοντέλο αυτό παρουσιάζει μεγαλύτερη διεκπεραιωτική ικανότητα.

11. ΠΗΓΕΣ ΚΑΙ ΒΙΒΛΙΟΓΡΑΦΙΑ

I. "Technology Guide 4 - Telecommunications", Arun Kumar
II. "The principle of forward error correction", Paul Nicholson, December 2016
III. "Error-correcting codes and applications", Klara Stokes, November 2017
IV. "Error detection and correction", NUS CS3235 Notes 4, Semester 2003
V. "Primer: Reed-Solomon Error Correction Codes (ECC)", AHA Application Note
VI. "Reed-Solomon error correction", C.K.P. Clarke, July 2002
VII. "Decoding algorithms of Reed-Solomon code", Szymon Czynszak, October 2011
VIII. "Construction of logarithm tables for Galois Fields", IJMEST, October 2010
IX. "General multiplication", EE 387 Lecture Notes 18, November 2015
X. "Logarithm and Anti-logarithm tables for GF(256)", Cody Planteen, August 2018
XI. "High-Speed Low-Complexity Reed-Solomon Decoder using Pipelined Berlekamp-Massey Algorithm and Its Folded Architecture", Jeong-In Park, Kihoon Lee, Chang-Seok Choi and Hanho Lee, September 2010
XII. "Bit Error Rate Analysis of Reed-Solomon Code for Efficient Communication System", Sanjeev Kumar and Ragini Gupta, IJCA Volume 30-No.12, September 2011
XIII. "Analytical Study of Reed-Solomon Error Probability", Nitin S. Sonar and R.R. Mudholkar, ISSN 0975-6469 Volume 8, Number 2 (2016), pp.297-304
XIV. "Reed-Solomon Codes", Bernard Sklar, Prentice-Hall, 2001
XV. "Decoding Reed-Solomon Codes", Bruce Maggs, October 24th 2000
XVI. "Notes #7, Handout #24", EE 387, John Gill, Stanford University
XVII. "High Throughput Reed Solomon Decoder for Ultra Wide Band", Akash Kumar, 2004/11
XVIII. "Design and Implementation of Reed-Solomon Decoder using Decomposed Inversion less Berlekamp-Massey Algorithm", Hazem Abd Elall Ahmed Elsaid, May 2010
XIX. "Implementation and Analysis of Forward Error Correction Techniques", Mohamed Elsir Ibrahim, July 2013
XX. "Design of RS (255, 251) Encoder and Decoder in FPGA", Anindya Sundar Das, Satyajit Das, Jaydeb Bhaumik, January 2013
XXI. "A High Speed Reed–Solomon decoder Chip using Inversionless Decomposed Architecture for Euclidean Algorithm", Hsie-Chia Chang, Ching-Che Chung, Chien-Ching Lin and Chen-Yi Lee, October 2002
XXII. "High-Speed Architectures for Reed-Solomon Decoders", Dilip V. Sarwate and Naresh R. Shanbhag, February 17, 2001

XXIII. “A High-Speed and Low-Latency Reed-Solomon Decoder Based on a Dual-Line Structure”, Hyeong-Ju Kang and In-Cheol Park, 2002
XXIV. “FPGA Implementation of A New Chien Search Block for Reed-Solomon Codes RS(255, 239) Used In Digital Video Broadcasting DVB-T”, El Habti El Idrissi Anas, El Gouri Rachid and Hlou Laamari, ISSN: 2248-9622, Vol.4, Issue 8 (Version 1), August 2014
XXV. “Efficient Logic Sharing in Parallel Chien Search using Algorithmic Approach”, Umair Ishaq, Jungsub Oh, Woojin Yang, Sungju Park, Multimedia Systems Lab, Dept. of CS&E, Hanyang University, South Korea, 2010
XXVI. “http://www.deathbylogic.com/2013/07/vhdl-standard-fifo/”, Daniel, 2015
XXVII. “http://twimgs.com/ddj/images/article/2013/0613/ReedSolomon.py”, 2013
XXVIII. ”Area Optimized Implementation of Reed Solomon Decoder“, Mahapatro Satish Kumar, Ganesh Kumar.P, February 2015
XXIX. “FPGA Implementation of Reed-Solomon Encoder and Decoder for Wireless Network 802.16”, Rajeev Kumar Patial, April 2013
XXX. “Design and Implementation of Reed Solomon Decoder for 802.16 Network using FPGA”, Bhawna Tiwari, Rajesh Mehra, 2012
XXXI. “Implementation of Reed-Solomon CODEC for IEEE 802.16 network using VHDL code”, Priyanka Dayal, Rajeev Kumar Patial, February 2014
XXXII. “Design High speed Reed Solomon Decoder on FPGA”, Saroj Bakale, Dhananjay Dabhade, September 2015

12. ΠΑΡΑΡΤΗΜΑΤΑ

Παράρτημα Α – Εκθετικά και Λογάριθμοι του GF(2^8) με p(x) = $(11D)_{hex}$

j	a^j	j	a^j	j	a^j	j	a^j	j	a^j	j	a^j	j	a^j	j	a^j
0	1	**42**	181	**84**	107	**126**	102	**168**	252	**210**	89	**252**	173
1	2	**43**	119	**85**	214	**127**	204	**169**	229	**211**	178	**253**	71
2	4	**44**	238	**86**	177	**128**	133	**170**	215	**212**	121	**254**	142
3	8	**45**	193	**87**	127	**129**	23	**171**	179	**213**	242	*	0
4	16	**46**	159	**88**	254	**130**	46	**172**	123	**214**	249		
5	32	**47**	35	**89**	225	**131**	92	**173**	246	**215**	239		
6	64	**48**	70	**90**	223	**132**	184	**174**	241	**216**	195		
7	128	**49**	140	**91**	163	**133**	109	**175**	255	**217**	155		
8	29	**50**	5	**92**	91	**134**	218	**176**	227	**218**	43		
9	58	**51**	10	**93**	182	**135**	169	**177**	219	**219**	86		
10	116	**52**	20	**94**	113	**136**	79	**178**	171	**220**	172		
11	232	**53**	40	**95**	226	**137**	158	**179**	75	**221**	69		
12	205	**54**	80	**96**	217	**138**	33	**180**	150	**222**	138		
13	135	**55**	160	**97**	175	**139**	66	**181**	49	**223**	9		
14	19	**56**	93	**98**	67	**140**	132	**182**	98	**224**	18		
15	38	**57**	186	**99**	134	**141**	21	**183**	196	**225**	36		
16	76	**58**	105	**100**	17	**142**	42	**184**	149	**226**	72		
17	152	**59**	210	**101**	34	**143**	84	**185**	55	**227**	144		
18	45	**60**	185	**102**	68	**144**	168	**186**	110	**228**	61		
19	90	**61**	111	**103**	136	**145**	77	**187**	220	**229**	122		
20	180	**62**	222	**104**	13	**146**	154	**188**	165	**230**	244		
21	117	**63**	161	**105**	26	**147**	41	**189**	87	**231**	245		
22	234	**64**	95	**106**	52	**148**	82	**190**	174	**232**	247		
23	201	**65**	190	**107**	104	**149**	164	**191**	65	**233**	243		
24	143	**66**	97	**108**	208	**150**	85	**192**	130	**234**	251		
25	3	**67**	194	**109**	189	**151**	170	**193**	25	**235**	235		
26	6	**68**	153	**110**	103	**152**	73	**194**	50	**236**	203		
27	12	**69**	47	**111**	206	**153**	146	**195**	100	**237**	139		
28	24	**70**	94	**112**	129	**154**	57	**196**	200	**238**	11		
29	48	**71**	188	**113**	31	**155**	114	**197**	141	**239**	22		
30	96	**72**	101	**114**	62	**156**	228	**198**	7	**240**	44		
31	192	**73**	202	**115**	124	**157**	213	**199**	14	**241**	88		
32	157	**74**	137	**116**	248	**158**	183	**200**	28	**242**	176		
33	39	**75**	15	**117**	237	**159**	115	**201**	56	**243**	125		
34	78	**76**	30	**118**	199	**160**	230	**202**	112	**244**	250		
35	156	**77**	60	**119**	147	**161**	209	**203**	224	**245**	233		
36	37	**78**	120	**120**	59	**162**	191	**204**	221	**246**	207		
37	74	**79**	240	**121**	118	**163**	99	**205**	167	**247**	131		
38	148	**80**	253	**122**	236	**164**	198	**206**	83	**248**	27		
39	53	**81**	231	**123**	197	**165**	145	**207**	166	**249**	54		
40	106	**82**	211	**124**	151	**166**	63	**208**	81	**250**	108		
41	212	**83**	187	**125**	51	**167**	126	**209**	162	**251**	216		

Πίνακας 12-1 – Εκθετικά του GF(2^8) [X]

j	a^j	j	a^j	j	a^j	j	a^j	j	a^j	j	a^j	j	a^j
0	*	**42**	142	**84**	143	**126**	167	**168**	144	**210**	59	**252**	168
1	0	**43**	218	**85**	150	**127**	87	**169**	135	**211**	82	**253**	80
2	1	**44**	240	**86**	219	**128**	7	**170**	151	**212**	41	**254**	88
3	25	**45**	18	**87**	189	**129**	112	**171**	178	**213**	157	**255**	175
4	2	**46**	130	**88**	241	**130**	192	**172**	220	**214**	85		
5	50	**47**	69	**89**	210	**131**	247	**173**	252	**215**	170		
6	26	**48**	29	**90**	19	**132**	140	**174**	190	**216**	251		
7	198	**49**	181	**91**	92	**133**	128	**175**	97	**217**	96		
8	3	**50**	194	**92**	131	**134**	99	**176**	242	**218**	134		
9	223	**51**	125	**93**	56	**135**	13	**177**	86	**219**	177		
10	51	**52**	106	**94**	70	**136**	103	**178**	211	**220**	187		
11	238	**53**	39	**95**	64	**137**	74	**179**	171	**221**	204		
12	27	**54**	249	**96**	30	**138**	222	**180**	20	**222**	62		
13	104	**55**	185	**97**	66	**139**	237	**181**	42	**223**	90		
14	199	**56**	201	**98**	182	**140**	49	**182**	91	**224**	203		
15	75	**57**	154	**99**	163	**141**	197	**183**	158	**225**	89		
16	4	**58**	9	**100**	195	**142**	254	**184**	132	**226**	95		
17	100	**59**	120	**101**	72	**143**	24	**185**	60	**227**	176		
18	224	**60**	77	**102**	126	**144**	227	**186**	57	**228**	156		
19	14	**61**	228	**103**	110	**145**	165	**187**	83	**229**	169		
20	52	**62**	114	**104**	107	**146**	153	**188**	71	**230**	160		
21	141	**63**	166	**105**	58	**147**	119	**189**	109	**231**	81		
22	239	**64**	6	**106**	40	**148**	38	**190**	65	**232**	11		
23	129	**65**	191	**107**	84	**149**	184	**191**	162	**233**	245		
24	28	**66**	139	**108**	250	**150**	180	**192**	31	**234**	22		
25	193	**67**	98	**109**	133	**151**	124	**193**	45	**235**	235		
26	105	**68**	102	**110**	186	**152**	17	**194**	67	**236**	122		
27	248	**69**	221	**111**	61	**153**	68	**195**	216	**237**	117		
28	200	**70**	49	**112**	202	**154**	146	**196**	183	**238**	44		
29	8	**71**	253	**113**	94	**155**	217	**197**	123	**239**	215		
30	76	**72**	226	**114**	155	**156**	35	**198**	164	**240**	79		
31	113	**73**	152	**115**	159	**157**	32	**199**	118	**241**	174		
32	5	**74**	37	**116**	10	**158**	137	**200**	196	**242**	213		
33	138	**75**	179	**117**	21	**159**	46	**201**	23	**243**	233		
34	101	**76**	16	**118**	121	**160**	55	**202**	73	**244**	230		
35	47	**77**	145	**119**	43	**161**	63	**203**	236	**245**	231		
36	225	**78**	34	**120**	78	**162**	209	**204**	127	**246**	173		
37	36	**79**	136	**121**	212	**163**	91	**205**	12	**247**	232		
38	15	**80**	54	**122**	229	**164**	149	**206**	111	**248**	116		
39	33	**81**	208	**123**	172	**165**	188	**207**	246	**249**	214		
40	53	**82**	148	**124**	115	**166**	207	**208**	108	**250**	244		
41	147	**83**	206	**125**	243	**167**	205	**209**	161	**251**	234		

Πίνακας 12-2 – Λογάριθμοι του GF(2^8) [X]

Παράρτημα Β – Γενικός Πολλαπλασιαστής για το GF(2^8) με p(x) = $(11D)_{hex}$

Κώδικας VHDL γενικού πολλαπλασιαστή:

```
entity gf_mul is
    port(
        a: in std_logic_vector(7 downto 0);
        b: in std_logic_vector(7 downto 0);
        c: out std_logic_vector(7 downto 0)
    );
end gf_mul;

architecture arch of gf_mul is

    -- signals to reduce gates
    signal b1_and_a7 : std_logic;
    signal b2_and_a6, b2_and_a7 : std_logic;
    signal b3_and_a5, b3_and_a6, b3_and_a7 : std_logic;
    signal b4_and_a4, b4_and_a5, b4_and_a6, b4_and_a7 : std_logic;
    signal b5_and_a3, b5_and_a4, b5_and_a5, b5_and_a6, b5_and_a7 : std_logic;
    signal b6_and_a2, b6_and_a3, b6_and_a4, b6_and_a5, b6_and_a6, b6_and_a7 : std_logic;
    signal b7_and_a1, b7_and_a2, b7_and_a3, b7_and_a4, b7_and_a5, b7_and_a6, b7_and_a7 : std_logic;

    begin

    b1_and_a7 <= b(1) and a(7);

    b2_and_a6 <= b(2) and a(6);
    b2_and_a7 <= b(2) and a(7);

    b3_and_a5 <= b(3) and a(5);
    b3_and_a6 <= b(3) and a(6);
    b3_and_a7 <= b(3) and a(7);

    b4_and_a4 <= b(4) and a(4);
    b4_and_a5 <= b(4) and a(5);
    b4_and_a6 <= b(4) and a(6);
    b4_and_a7 <= b(4) and a(7);

    b5_and_a3 <= b(5) and a(3);
    b5_and_a4 <= b(5) and a(4);
    b5_and_a5 <= b(5) and a(5);
    b5_and_a6 <= b(5) and a(6);
    b5_and_a7 <= b(5) and a(7);

    b6_and_a2 <= b(6) and a(2);
    b6_and_a3 <= b(6) and a(3);
    b6_and_a4 <= b(6) and a(4);
    b6_and_a5 <= b(6) and a(5);
    b6_and_a6 <= b(6) and a(6);
    b6_and_a7 <= b(6) and a(7);

    b7_and_a1 <= b(7) and a(1);
    b7_and_a2 <= b(7) and a(2);
    b7_and_a3 <= b(7) and a(3);
    b7_and_a4 <= b(7) and a(4);
    b7_and_a5 <= b(7) and a(5);
    b7_and_a6 <= b(7) and a(6);
    b7_and_a7 <= b(7) and a(7);
```

```
    -- 14 terms
    c(0) <= (b(0) and a(0)) xor b1_and_a7 xor b2_and_a6 xor b3_and_a5 xor
            b4_and_a4 xor b5_and_a3 xor b5_and_a7 xor b6_and_a2 xor
            b6_and_a6 xor b6_and_a7 xor b7_and_a1 xor b7_and_a5 xor
            b7_and_a6 xor b7_and_a7;

    -- 11 terms
    c(1) <= (b(0) and a(1)) xor (b(1) and a(0)) xor b2_and_a7 xor b3_and_a6 xor
            b4_and_a5 xor b5_and_a4 xor b6_and_a3 xor b6_and_a7 xor
            b7_and_a2 xor b7_and_a6 xor b7_and_a7;

    -- 20 terms
    c(2) <= (b(0) and a(2)) xor (b(1) and a(1)) xor b1_and_a7 xor (b(2) and a(0)) xor
            b2_and_a6 xor b3_and_a5 xor b3_and_a7 xor b4_and_a4 xor
            b4_and_a6 xor b5_and_a3 xor b5_and_a5 xor b5_and_a7 xor
            b6_and_a2 xor b6_and_a4 xor b6_and_a6 xor b6_and_a7 xor
            b7_and_a1 xor b7_and_a3 xor b7_and_a5 xor b7_and_a6;

    -- 24 terms
    c(3) <= (b(0) and a(3)) xor (b(1) and a(2)) xor b1_and_a7 xor (b(2) and a(1)) xor
            b2_and_a6 xor b2_and_a7 xor (b(3) and a(0)) xor b3_and_a5 xor
            b3_and_a6 xor b4_and_a4 xor b4_and_a5 xor b4_and_a7 xor
            b5_and_a3 xor b5_and_a4 xor b5_and_a6 xor b5_and_a7 xor
            b6_and_a2 xor b6_and_a3 xor b6_and_a5 xor b6_and_a6 xor
            b7_and_a1 xor b7_and_a2 xor b7_and_a4 xor b7_and_a5;

    -- 24 terms
    c(4) <= (b(0) and a(4)) xor (b(1) and a(3)) xor b1_and_a7 xor (b(2) and a(2)) xor
            b2_and_a6 xor b2_and_a7 xor (b(3) and a(1)) xor b3_and_a5 xor
            b3_and_a6 xor b3_and_a7 xor (b(4) and a(0)) xor b4_and_a4 xor
            b4_and_a5 xor b4_and_a6 xor b5_and_a3 xor b5_and_a4 xor
            b5_and_a5 xor b6_and_a2 xor b6_and_a3 xor b6_and_a4 xor
            b7_and_a1 xor b7_and_a2 xor b7_and_a3 xor b7_and_a7;

    -- 21 terms
    c(5) <= (b(0) and a(5)) xor (b(1) and a(4)) xor (b(2) and a(3)) xor b2_and_a7 xor
            (b(3) and a(2)) xor b3_and_a6 xor b3_and_a7 xor (b(4) and a(1)) xor
            b4_and_a5 xor b4_and_a6 xor b4_and_a7 xor (b(5) and a(0)) xor
            b5_and_a4 xor b5_and_a5 xor b5_and_a6 xor b6_and_a3 xor
            b6_and_a4 xor b6_and_a5 xor b7_and_a2 xor b7_and_a3 xor
            b7_and_a4;

    -- 19 terms
    c(6) <= (b(0) and a(6)) xor (b(1) and a(5)) xor (b(2) and a(4)) xor (b(3) and a(3)) xor
            b3_and_a7 xor (b(4) and a(2)) xor b4_and_a6 xor b4_and_a7 xor
            (b(5) and a(1)) xor b5_and_a5 xor b5_and_a6 xor b5_and_a7 xor
            (b(6) and a(0)) xor b6_and_a4 xor b6_and_a5 xor b6_and_a6 xor
            b7_and_a3 xor b7_and_a4 xor b7_and_a5;

    -- 17 terms
    c(7) <= (b(0) and a(7)) xor (b(1) and a(6)) xor (b(2) and a(5)) xor (b(3) and a(4)) xor
            (b(4) and a(3)) xor b4_and_a7 xor (b(5) and a(2)) xor b5_and_a6 xor
            b5_and_a7 xor (b(6) and a(1)) xor b6_and_a5 xor b6_and_a6 xor
            b6_and_a7 xor (b(7) and a(0)) xor b7_and_a4 xor b7_and_a5 xor
            b7_and_a6;

end arch;
```

Παράρτημα Γ – Πολλαπλασιαστές μεταβλητής-σταθεράς στο GF(2^8) με p(x) = $(11D)_{hex}$

Στα μπλοκ υπολογισμού συνδρόμων και εύρεσης θέσεων λάθους του κώδικα $RS(255, 239)$ απαιτούνται τα εκθετικά του $GF(2^8)$: $a, \alpha^2, \dots, \alpha^{15}$. Επειδή χρησιμοποιείται ο αλγόριθμος RiBM, κατά την αξιολόγηση του $\Omega(x)$ απαιτούνται και τα εκθετικά: $\alpha^{16}, \alpha^{17}, \dots, \alpha^{23}$, εφόσον οι συντελεστές ολισθαίνουν κατά $2t = 16$ θέσεις.

Παρακάτω παρουσιάζονται οι μαθηματικές αποδείξεις των απαιτούμενων πολλαπλασιαστών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a = 2_{hex} = [01000000]}$:

$$B = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{aligned} c_0 &= \alpha_7 \\ c_1 &= \alpha_0 \\ c_2 &= \alpha_1 \oplus \alpha_7 \\ c_3 &= \alpha_2 \oplus \alpha_7 \\ c_4 &= \alpha_3 \oplus \alpha_7 \\ c_5 &= \alpha_4 \\ c_6 &= \alpha_5 \\ c_7 &= \alpha_6 \end{aligned}$$

Απαιτούνται μόνο 3 πύλες XOR.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^2 = 4_{hex} = [00100000]}$:

$$B = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \rightarrow \begin{aligned} c_0 &= \alpha_6 \\ c_1 &= \alpha_7 \\ c_2 &= \alpha_0 \oplus \alpha_6 \\ c_3 &= \alpha_1 \oplus \alpha_6 \oplus \alpha_7 \\ c_4 &= \alpha_2 \oplus \alpha_6 \oplus \alpha_7 \\ c_5 &= \alpha_3 \oplus \alpha_7 \\ c_6 &= \alpha_4 \\ c_7 &= \alpha_5 \end{aligned}$$

Με εξάλειψη του $\alpha_6 \oplus \alpha_7$ που αποτελεί κοινή υποέκφραση των c_3 και c_4 απαιτούνται μόνο 5 αντί 6 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^3 = 8_{hex} = [00010000]}$:

$$B = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \rightarrow \begin{aligned} c_0 &= \alpha_5 \\ c_1 &= \alpha_6 \\ c_2 &= \alpha_5 \oplus \alpha_7 \\ c_3 &= \alpha_0 \oplus \alpha_5 \oplus \alpha_6 \\ c_4 &= \alpha_1 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_5 &= \alpha_2 \oplus \alpha_6 \oplus \alpha_7 \\ c_6 &= \alpha_3 \oplus \alpha_7 \\ c_7 &= \alpha_4 \end{aligned}$$

Με εξάλειψη είτε του $\alpha_5 \oplus \alpha_6$, είτε του $\alpha_6 \oplus \alpha_7$, προκύπτει αθροιστής με 8 αντί 9 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^4 = 10_{hex} = [00001000]}$:

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_4 \\ c_1 = \alpha_5 \\ c_2 = \alpha_4 \oplus \alpha_6 \\ c_3 = \alpha_4 \oplus \alpha_5 \oplus \alpha_7 \\ c_4 = \alpha_0 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \\ c_5 = \alpha_1 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_6 = \alpha_2 \oplus \alpha_6 \oplus \alpha_7 \\ c_7 = \alpha_3 \oplus \alpha_7 \end{array}$$

Με εξάλειψη των $\alpha_4 \oplus \alpha_5$ και $\alpha_6 \oplus \alpha_7$, απαιτούνται 10 αντί 12 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^5 = 20_{hex} = [00000100]}$:

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_3 \oplus \alpha_7 \\ c_1 = \alpha_4 \\ c_2 = \alpha_3 \oplus \alpha_5 \oplus \alpha_7 \\ c_3 = \alpha_3 \oplus \alpha_4 \oplus \alpha_6 \oplus \alpha_7 \\ c_4 = \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \\ c_5 = \alpha_0 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \\ c_6 = \alpha_1 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_7 = \alpha_2 \oplus \alpha_6 \oplus \alpha_7 \end{array}$$

Με εξάλειψη των $\alpha_3 \oplus \alpha_7$, $\alpha_4 \oplus \alpha_5$ και $\alpha_6 \oplus \alpha_7$, απαιτούνται 12 αντί 16 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^6 = 40_{hex} = [00000010]}$:

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_2 \oplus \alpha_6 \oplus \alpha_7 \\ c_1 = \alpha_3 \oplus \alpha_7 \\ c_2 = \alpha_2 \oplus \alpha_4 \oplus \alpha_6 \oplus \alpha_7 \\ c_3 = \alpha_2 \oplus \alpha_3 \oplus \alpha_5 \oplus \alpha_6 \\ c_4 = \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \\ c_5 = \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \\ c_6 = \alpha_0 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \\ c_7 = \alpha_1 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \end{array}$$

Με εξάλειψη των $\alpha_2 \oplus \alpha_3$, $\alpha_5 \oplus \alpha_6$ και $\alpha_6 \oplus \alpha_7$, απαιτούνται 15 αντί 19 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^7 = 80_{hex} = [00000001]}$:

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_1 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_1 = \alpha_2 \oplus \alpha_6 \oplus \alpha_7 \\ c_2 = \alpha_1 \oplus \alpha_3 \oplus \alpha_5 \oplus \alpha_6 \\ c_3 = \alpha_1 \oplus \alpha_2 \oplus \alpha_4 \oplus \alpha_5 \\ c_4 = \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_7 \\ c_5 = \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \\ c_6 = \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \\ c_7 = \alpha_0 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \end{array}$$

Με εξάλειψη των $\alpha_1 \oplus \alpha_5$, $\alpha_2 \oplus \alpha_3$, $\alpha_4 \oplus \alpha_5$ και $\alpha_6 \oplus \alpha_7$, απαιτούνται 16 αντί 21 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^8 = 1D_{hex} = [10111000]}$:

$$B = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_0 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \\ c_1 = \alpha_1 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_2 = \alpha_0 \oplus \alpha_2 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_7 \\ c_3 = \alpha_0 \oplus \alpha_1 \oplus \alpha_3 \oplus \alpha_4 \\ c_4 = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \oplus \alpha_6 \\ c_5 = \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_7 \\ c_6 = a_2 \oplus \alpha_3 \oplus \alpha_4 \\ c_7 = a_3 \oplus \alpha_4 \oplus \alpha_5 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_1$, $\alpha_0 \oplus \alpha_4$, $\alpha_2 \oplus \alpha_7$, $\alpha_3 \oplus \alpha_4$ και $\alpha_5 \oplus a_6$, απαιτούνται 17 αντί 23 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^9 = 3A_{hex} = [01011100]}$:

$$B = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \\ c_1 = \alpha_0 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \\ c_2 = \alpha_1 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_6 \oplus \alpha_7 \\ c_3 = \alpha_0 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_7 \\ c_4 = \alpha_0 \oplus \alpha_1 \oplus \alpha_5 \\ c_5 = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \oplus \alpha_6 \\ c_6 = a_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_7 \\ c_7 = a_2 \oplus \alpha_3 \oplus \alpha_4 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_1$, $\alpha_2 \oplus \alpha_3$ και $\alpha_4 \oplus \alpha_5$, απαιτούνται 18 αντί 22 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{10} = 74_{hex} = [00101110]}$:

$$B = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \\ c_1 = \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \\ c_2 = \alpha_0 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_5 \oplus \alpha_6 \\ c_3 = \alpha_1 \oplus \alpha_2 \oplus \alpha_6 \oplus \alpha_7 \\ c_4 = \alpha_0 \oplus \alpha_4 \oplus \alpha_7 \\ c_5 = \alpha_0 \oplus \alpha_1 \oplus \alpha_5 \\ c_6 = a_0 \oplus \alpha_1 \oplus \alpha_2 \oplus \alpha_6 \\ c_7 = a_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_7 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_6$, $\alpha_1 \oplus \alpha_2$ και $\alpha_2 \oplus \alpha_3$ απαιτούνται 17 αντί 21 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{11} = E8_{hex} = [00010111]}$:

$$B = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_7 \\ c_1 = \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \\ c_2 = \alpha_1 \oplus \alpha_2 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_7 \\ c_3 = \alpha_0 \oplus \alpha_1 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_4 = \alpha_3 \oplus \alpha_6 \\ c_5 = \alpha_0 \oplus \alpha_4 \oplus \alpha_7 \\ c_6 = a_0 \oplus \alpha_1 \oplus \alpha_5 \\ c_7 = a_0 \oplus \alpha_1 \oplus \alpha_2 \oplus \alpha_6 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_1$, $\alpha_1 \oplus \alpha_2$ και $\alpha_4 \oplus \alpha_7$, απαιτούνται 17 αντί 21 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{12} = CD_{hex} = [10110011]}$:

$$B = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \oplus \alpha_6 \\ c_1 = \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_7 \\ c_2 = \alpha_0 \oplus \alpha_1 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_6 \\ c_3 = \alpha_0 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_4 = \alpha_2 \oplus \alpha_5 \oplus \alpha_7 \\ c_5 = \alpha_3 \oplus \alpha_6 \\ c_6 = a_0 \oplus \alpha_4 \oplus \alpha_7 \\ c_7 = a_0 \oplus \alpha_1 \oplus \alpha_5 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_1$, $\alpha_0 \oplus \alpha_4$, $\alpha_3 \oplus \alpha_6$ και $\alpha_5 \oplus \alpha_7$, απαιτούνται 16 αντί 21 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{13} = 87_{hex} = [11100001]}$:

$$B = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_0 \oplus \alpha_1 \oplus \alpha_5 \\ c_1 = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \oplus \alpha_6 \\ c_2 = \alpha_0 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_5 \oplus \alpha_7 \\ c_3 = \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \\ c_4 = \alpha_1 \oplus \alpha_4 \oplus \alpha_6 \oplus \alpha_7 \\ c_5 = \alpha_2 \oplus \alpha_5 \oplus \alpha_7 \\ c_6 = \alpha_3 \oplus \alpha_6 \\ c_7 = a_0 \oplus \alpha_4 \oplus \alpha_7 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_1$, $\alpha_3 \oplus \alpha_6$, $\alpha_4 \oplus \alpha_7$ και $\alpha_5 \oplus \alpha_7$, απαιτούνται 16 αντί 20 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{14} = 13_{hex} = [11001000]}$:

$$B = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_0 \oplus \alpha_4 \oplus \alpha_7 \\ c_1 = \alpha_0 \oplus \alpha_1 \oplus \alpha_5 \\ c_2 = \alpha_1 \oplus \alpha_2 \oplus \alpha_4 \oplus \alpha_6 \oplus \alpha_7 \\ c_3 = \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \\ c_4 = \alpha_0 \oplus \alpha_3 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_5 = \alpha_1 \oplus \alpha_4 \oplus \alpha_6 \oplus \alpha_7 \\ c_6 = \alpha_2 \oplus \alpha_5 \oplus \alpha_7 \\ c_7 = a_3 \oplus \alpha_6 \end{array}$$

Με εξάλειψη των $\alpha_1 \oplus \alpha_6$, $\alpha_3 \oplus \alpha_6$, $\alpha_4 \oplus \alpha_7$ και $\alpha_5 \oplus \alpha_7$, απαιτούνται 16 αντί 21 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{15} = 26_{hex} = [01100100]}$:

$$B = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_3 \oplus \alpha_6 \\ c_1 = \alpha_0 \oplus \alpha_4 \oplus \alpha_7 \\ c_2 = \alpha_0 \oplus \alpha_1 \oplus \alpha_3 \oplus \alpha_5 \oplus \alpha_6 \\ c_3 = \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_7 \\ c_4 = \alpha_2 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \\ c_5 = \alpha_0 \oplus \alpha_3 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_6 = \alpha_1 \oplus \alpha_4 \oplus \alpha_6 \oplus \alpha_7 \\ c_7 = a_2 \oplus \alpha_5 \oplus \alpha_7 \end{array}$$

Με εξάλειψη των $\alpha_3 \oplus \alpha_6$, $\alpha_4 \oplus \alpha_7$ και $\alpha_5 \oplus \alpha_7$, απαιτούνται 18 αντί 23 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{16} = 4C_{hex} = [00110010]}$:

$$B = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_2 \oplus \alpha_5 \oplus \alpha_7 \\ c_1 = \alpha_3 \oplus \alpha_6 \\ c_2 = \alpha_0 \oplus \alpha_2 \oplus \alpha_4 \oplus \alpha_5 \\ c_3 = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_6 \oplus \alpha_7 \\ c_4 = \alpha_1 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \\ c_5 = \alpha_2 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \\ c_6 = \alpha_0 \oplus \alpha_3 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_7 = a_1 \oplus \alpha_4 \oplus \alpha_6 \oplus \alpha_7 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_3$, $\alpha_4 \oplus \alpha_5$ και $\alpha_6 \oplus \alpha_7$, απαιτούνται 19 αντί 24 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{17} = 98_{hex} = [00011001]}$:

$$B = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_1 \oplus \alpha_4 \oplus \alpha_6 \oplus \alpha_7 \\ c_1 = \alpha_2 \oplus \alpha_5 \oplus \alpha_7 \\ c_2 = \alpha_1 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_7 \\ c_3 = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_4 = \alpha_0 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \\ c_5 = \alpha_1 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \\ c_6 = \alpha_2 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \\ c_7 = \alpha_0 \oplus a_3 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_3$, $\alpha_1 \oplus \alpha_4$, $\alpha_2 \oplus \alpha_5$ και $\alpha_6 \oplus \alpha_7$, απαιτούνται 19 αντί 26 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{18} = 2D_{hex} = [10110100]}$:

$$B = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_0 \oplus \alpha_3 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_1 = \alpha_1 \oplus \alpha_4 \oplus \alpha_6 \oplus \alpha_7 \\ c_2 = \alpha_0 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_6 \\ c_3 = \alpha_0 \oplus \alpha_1 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \\ c_4 = \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \\ c_5 = \alpha_0 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \\ c_6 = \alpha_1 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \\ c_7 = a_2 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_3$, $\alpha_1 \oplus \alpha_4$, $\alpha_2 \oplus \alpha_4$, $\alpha_5 \oplus \alpha_7$ και $\alpha_6 \oplus \alpha_7$, απαιτούνται 18 αντί 25 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{19} = 5A_{hex} = [01011010]}$:

$$B = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_2 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \\ c_1 = \alpha_0 \oplus \alpha_3 \oplus \alpha_5 \oplus \alpha_6 \oplus \alpha_7 \\ c_2 = \alpha_1 \oplus \alpha_2 \oplus \alpha_5 \oplus \alpha_7 \\ c_3 = \alpha_0 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \\ c_4 = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \\ c_5 = \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \\ c_6 = \alpha_0 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \\ c_7 = a_1 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \end{array}$$

Με εξάλειψη των $\alpha_1 \oplus \alpha_2$, $\alpha_3 \oplus \alpha_4$ και $\alpha_5 \oplus \alpha_6$, απαιτούνται 18 αντί 23 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{20} = B4_{hex} = [00101101]}$:

$$B = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_1 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \\ c_1 = \alpha_2 \oplus \alpha_4 \oplus \alpha_5 \oplus \alpha_6 \\ c_2 = \alpha_0 \oplus \alpha_1 \oplus \alpha_4 \oplus \alpha_6 \oplus \alpha_7 \\ c_3 = \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_7 \\ c_4 = \alpha_0 \oplus \alpha_1 \\ c_5 = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \\ c_6 = \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \\ c_7 = a_0 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_1$, $\alpha_2 \oplus \alpha_3$, $\alpha_4 \oplus \alpha_5$ και $\alpha_4 \oplus \alpha_7$, απαιτούνται 15 αντί 21 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{21} = 75_{hex} = [10101110]}$:

$$B = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_0 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \\ c_1 = \alpha_1 \oplus \alpha_3 \oplus \alpha_4 \oplus \alpha_5 \\ c_2 = \alpha_0 \oplus \alpha_3 \oplus \alpha_5 \oplus \alpha_6 \\ c_3 = \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_6 \oplus \alpha_7 \\ c_4 = \alpha_0 \oplus \alpha_7 \\ c_5 = \alpha_0 \oplus \alpha_1 \\ c_6 = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \\ c_7 = a_1 \oplus \alpha_2 \oplus \alpha_3 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_1$, $\alpha_2 \oplus \alpha_3$ και $\alpha_3 \oplus \alpha_4$, απαιτούνται 16 αντί 19 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{22} = EA_{hex} = [01010111]}$:

$$B = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \\ c_1 = \alpha_0 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_4 \\ c_2 = \alpha_2 \oplus \alpha_4 \oplus \alpha_5 \\ c_3 = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \oplus \alpha_5 \oplus \alpha_6 \\ c_4 = \alpha_6 \oplus \alpha_7 \\ c_5 = \alpha_0 \oplus \alpha_7 \\ c_6 = \alpha_0 \oplus \alpha_1 \\ c_7 = a_0 \oplus \alpha_1 \oplus \alpha_2 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_1$, $\alpha_2 \oplus \alpha_3$ και $\alpha_2 \oplus \alpha_5$, απαιτούνται 12 αντί 16 πυλών.

Πολλαπλασιαστής με σταθερά $\boldsymbol{b = a^{23} = C9_{hex} = [10010011]}$:

$$B = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \rightarrow \begin{array}{l} c_0 = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \\ c_1 = \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \\ c_2 = \alpha_1 \oplus \alpha_3 \oplus \alpha_4 \\ c_3 = \alpha_0 \oplus \alpha_1 \oplus \alpha_4 \oplus \alpha_5 \\ c_4 = \alpha_5 \oplus \alpha_6 \\ c_5 = \alpha_6 \oplus \alpha_7 \\ c_6 = \alpha_0 \oplus \alpha_7 \\ c_7 = a_0 \oplus \alpha_1 \end{array}$$

Με εξάλειψη των $\alpha_0 \oplus \alpha_1$ και $\alpha_1 \oplus \alpha_2$, απαιτούνται 11 αντί 13 πυλών.