**ΑΛΕΞΑΝΔΡΕΙΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΘΕΣΣΑΛΟΝΙΚΗΣ**
**ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ**
**ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΤΕ**

# Μελέτη και υλοποίηση κωδικών LT σε Matlab και VHDL

## "Study and realization of LT codes in Matlab and VHDL"

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**
**ΤΟΥ**

**Κεφαλά Αργύρη**
**ΚΑΣ 507815**
**13132ΕΜ**

**Επιβλέποντες: Αθανάσιος Χ. Ιωσηφίδης, Επίκουρος Καθηγητής Δρ. Ηλεκ. Μηχ. & Μηχ. Υπολ.**
**Βάσσιος Βασίλειος, Ηλεκτρονικός Μηχανικός**

Θεσσαλονίκη, 18/4/2013 – 31/1/2014

*Θα ήθελα να ευχαριστήσω τους παρακάτω.*

*Ευχαριστώ την μικρή μου αδερφή, Ιωάννα Κεφαλά. Της ζητάω συγγνώμη που είχα όλη μέρα πιασμένο τον υπολογιστή και δεν την άφηνα να μπει στο facebook.*

*Ευχαριστώ τον κύριο Βάσσιο Βασίλειο, Ηλεκτρονικό Μηχανικό. Τον ευχαριστώ που αφιέρωσε από τον χρόνο του για να ασχοληθεί μαζί μου και να με βοήθησε να πραγματοποιήσω την Hardware εφαρμογή σε FPGA. Χωρίς την βοήθειά του δεν θα τα κατάφερνα.*

*Ευχαριστώ τον κύριο Σαπουνίδη Θεοδόσιο, Ηλεκτρονικό Μηχανικό, Ηλεκτρολόγο Μηχανικό & Μηχανικό Υπολογιστών. Τον ευχαριστώ για την τεχνική υποστήριξη και για όλες τις διευκολύνσεις που μου παρείχε όλο αυτό το διάστημα.*

*Ευχαριστώ τον κύριο Αθανάσιο Χ. Ιωσηφίδη, Δρ. Ηλεκτρολόγο Μηχανικό & Μηχανικό Υπολογιστών. Τον ευχαριστώ που κατά την περίοδο των σπουδών μου, μου δίδαξε τα μαθήματα των τηλεπικοινωνιών με τόσο ενδιαφέροντα τρόπο. Αυτός ήταν και ο λόγος που ασχολήθηκα και ανέλαβα αυτήν την εργασία.*

**Επικοινωνία:**

kefalas.argiris@gmail.com

**Περιεχόμενα**

## Περίληψη

*Σε αυτή την εργασία μελετάμε τον κώδικα Luby Transform (LT code). Ο κώδικας LT είναι ο πρώτος κώδικας που ανήκει στο είδος των Ψηφιακών κωδίκων Fountain (Digital Fountain codes). Οι κώδικες Fountain, είναι κώδικες καναλιού που χρησιμοποιούν την τεχνική Forward Error Correction (FEC). Η ιδιαιτερότητα που έχουν οι Fountain κώδικες σε σχέση με τους συνηθισμένους κώδικες που χρησιμοποιούν την τεχνική FEC, είναι ότι ο ρυθμός (rate) τους δεν είναι προκαθορισμένος, αλλά μπορεί να μεταβληθεί κατά την μετάδοση. Ο LT κώδικας προτάθηκε ως μία εναλλακτική λύση στο σύστημα TCP/IP που χρησιμοποιείται στο δίκτυο του internet, λόγω των μειονεκτημάτων που έχει το τελευταίο σε broadcast εφαρμογές εξαιτίας της υπερφόρτωσης του καναλιού ανάδρασης όταν η πιθανότητα διαγραφής του καναλιού είναι μεγάλη. Σε γενικές γραμμές στην παρούσα εργασία γίνεται αρχικά μία γενική εισαγωγή στην κωδικοποίηση καναλιού, μία περιγραφή του LT κώδικα και των Fountain κωδίκων γενικότερα. Στη συνέχεια γίνεται θεωρητική ανάλυση του LT κωδικοποιητή - αποκωδικοποιητή και παρουσιάζονται αποτελέσματα από την προσομοίωση του LT κωδικοποιητή - αποκωδικοποιητή που έγινε με τη βοήθεια του Matlab. Τέλος γίνεται ανάλυση της δομής της πειραματικής Hardware εφαρμογής του LT κωδικοποιητή - αποκωδικοποιητή που έγινε σε FPGA με την χρήση VHDL σε Xilinx και παρουσίαση των πειραματικών αποτελεσμάτων αυτής της εφαρμογής.*

## Abstract

*In this thesis we study the Luby Transform codes (LT codes). LT codes are the first practical realization of the Digital Fountain approach. Fountain codes are employing the Forward Error Correction technique (FEC). The trademark witch the Fountain codes have, in comparison with others usual FEC codes, is that the code rate is variable while the transmission is on air, it is not predetermined. LT codes have been proposed as an alternative option to the TCP/IP system used by the internet network, because of the disadvantages the latter have in broadcast applications due to the overload of the capacity of the channel caused by the feedback signals, when the erasure probability of the channel is high. This thesis starts with an introduction to channel coding, a description of LT codes and Fountain codes. Then, we present the theoretical analysis of the LT encoder – decoder and we show Matlab simulation results of the LT encoder - decoder. Finally, we present and analyze the architectural configuration of the FPGA implementation of the LT encoder - decoder, applied on Xilinx with VHDL, and we also give the experimental results of this implementation.*

# Κεφάλαιο 1 - Εισαγωγή

Στο Κεφάλαιο 1 γίνεται μία εισαγωγή στην κωδικοποίηση καναλιού γενικότερα. Στην συνέχεια γίνεται μία εισαγωγή στους Fountain κώδικες και στον LT κώδικα. Τέλος, γίνεται παρουσίαση κάποιων πρακτικών εφαρμογών του LT κώδικα και των Fountain κωδίκων.

## 1.1 - Κωδικοποίηση καναλιού

Η κωδικοποίηση καναλιού εφαρμόζεται στα ψηφιακά συστήματα επικοινωνίας και έχει ως στόχο την ασφαλή μετάδοση της πληροφορίας από τον πομπό στον δέκτη, μέσα από ένα αναξιόπιστο θορυβώδες κανάλι.

Υπάρχουν δύο κατηγορίες τεχνικών κωδικοποίησης καναλιού [1] :

- Κωδικοποιήσεις κυματομορφών (waveform coding)
- Κωδικοποιήσεις δομημένου πλεονασμού (structured redundancy)

Κατά τις κωδικοποιήσεις κυματομορφών γίνεται μετατροπή της κυματομορφής του σήματος πληροφορίας σε διαφορετική κυματομορφή. Οι κωδικοποιήσεις κυματομορφών έχουν ως στόχο την ποσοστιαία μείωση των λαθών κατά την διαδικασία εκτίμησης του λαμβανόμενου σήματος που γίνεται στον δέκτη.

Κατά τις κωδικοποιήσεις δομημένου πλεονασμού γίνεται εισαγωγή πρόσθετων συμβόλων, με αποτέλεσμα ο κωδικοποιητής να στέλνει στον αποκωδικοποιητή περισσότερα σύμβολα από τα σύμβολα που αποτελούν την πληροφορία. Αυτά τα πρόσθετα σύμβολα τα χρησιμοποιεί ο δέκτης για να εντοπίσει, ή, για να εντοπίσει και να διορθώσει τα λάθη που προκλήθηκαν από το θορυβώδες κανάλι.

Υπάρχουν τρεις τεχνικές ελέγχου λαθών που χρησιμοποιούνται στις κωδικοποιήσεις δομημένου πλεονασμού:

- FEC (Forward Error Correction)
- ARQ (Automatic Repeat Request)
- Hybrid ARQ (συνδυασμός των δύο παραπάνω)

Κατά την μέθοδο FEC, ο κωδικοποιητής παράγει κωδικοποιημένα σύμβολα που είναι περισσότερα από τα σύμβολα πληροφορίας. Με την βοήθεια των πρόσθετων κωδικοποιημένων συμβόλων ο δέκτης προσπαθεί να ανακτήσει τα σύμβολα πληροφορίας, χωρίς να κάνει αίτηση αναμετάδοσης

από τον πομπό. Πιο συγκεκριμένα, οι FEC κώδικες μετατρέπουν ένα αριθμό k συμβόλων πληροφορίας σε έναν μεγαλύτερο αριθμό n κωδικοποιημένων συμβόλων. Τα πρόσθετα σύμβολα που παράγει ο κωδικοποιητής προέρχονται από κάποιες πράξεις που γίνονται μεταξύ των συμβόλων πληροφορίας (πχ πράξεις XOR). Οι πράξεις αυτές γίνονται σύμφωνα με έναν προκαθορισμένο αλγόριθμο. Οι συνηθισμένοι FEC κώδικες, όπως είναι οι Block κώδικες και Convolutional κώδικες, χαρακτηρίζονται από το κλάσμα $r = k/n$ το οποίο ονομάζεται ρυθμός (rate) του κώδικα. Ο ρυθμός του κώδικα καθορίζεται πριν γίνει η μετάδοση. Δηλαδή ο ρυθμός στους block κώδικες και Colvolutional κώδικες, είναι προκαθορισμένος (fixed). [2]

Κατά την μέθοδο ARQ, ο δέκτης με την βοήθεια των πρόσθετων συμβόλων που παράγει ο κωδικοποιητής, μπορεί και εντοπίζει αν υπάρχουν λάθη σε κάποιο πακέτο συμβόλων που έλαβε. Ο δέκτης ενημερώνει τον πομπό για κάθε πακέτο που λαμβάνει. Αν δεν λάβει κάποιο πακέτο ή αν λάβει κάποιο πακέτο που έχει παραμορφωθεί από τον θόρυβο, τότε ζητάει από τον πομπό να του ξαναστείλει το συγκεκριμένο πακέτο.

## 1.2 - Ψηφιακοί Κώδικες Fountain (Digital Fountain Codes)

Οι Fountain κώδικες ή αλλιώς Rateless κώδικες διαγραφής (Rateless Erasures Codes) είναι ένας τύπος FEC κωδίκων που έχουν την ικανότητα να παράγουν θεωρητικά μία άπειρη ακολουθία από κωδικοποιημένα σύμβολα από έναν πεπερασμένο αριθμό συμβόλων πληροφορίας k [3]. Ο όρος Rateless αναφαίρετε στο γεγονός, ότι το επί τις εκατό overhead στους Fountain κώδικες δεν είναι προκαθορισμένο. Δηλαδή ο ρυθμός του κώδικα μπορεί να μεταβληθεί ενώ η μετάδοση έχει ξεκινήσει. Ο αποκωδικοποιητής μπορεί να ανακτήσει τα σύμβολα πληροφορίας από ένα οποιοδήποτε σετ (set) κωδικοποιημένων συμβόλων που παράγει ο κωδικοποιητής, με την προϋπόθεση ότι ο αριθμός των κωδικοποιημένων συμβόλων που θα χρησιμοποιήσει για να κάνει την αποκωδικοποίηση θα είναι *αρκετός*. Το πόσο *αρκετός* πρέπει να είναι ο αριθμός των κωδικοποιημένων συμβόλων που χρησιμοποιούνται για την αποκωδικοποίηση εξαρτάται από τον αριθμό των συμβόλων πληροφορίας k, καθώς και από κάποιες άλλες παραμέτρους που θα δούμε αναλυτικότερα παρακάτω. Πάντως, αν είναι μεγάλος ο αριθμός των συμβόλων πληροφορίας (k~10.000), τότε υπάρχει μεγάλη πιθανότητα να πετύχει η αποκωδικοποίηση για επί τις εκατό overhead που είναι κατά μέσω όρο μικρότερο από 10%. [4]

Ο κωδικοποιητής ενός Fountain κώδικα μπορεί να παρομοιαστεί σαν μία πηγή που παράγει άπειρες σταγόνες νερού (κωδικοποιημένα σύμβολα). Αν τα σύμβολα πληροφορίας είναι k, τότε οι δέκτες για να αποκωδικοποιήσουν την πληροφορία χρειάζονται να γεμίσουν την κούπα τους με λίγες παραπάνω από οποιεσδήποτε k σταγόνες που παράγει η πηγή. [4]

Οι κώδικες Fountain έχουν δείξει εξαιρετικά αποτελέσματα σε κανάλια διαγραφής (erasures channels). Κανάλι διαγραφής μπορεί να θεωρηθεί και το δίκτυο του internet. Ο δέκτης είτε λαμβάνει ένα πακέτο που δεν έχει υποστεί σφάλμα, είτε δεν λαμβάνει το πακέτο.

Συνήθως τα συστήματα επικοινωνίας που λειτουργούν σε κανάλια διαγραφής, όπως είναι το TCP/IP που χρησιμοποιείται στο δίκτυο του internet, χρησιμοποιούν ένα κανάλι ανάδρασης ούτος ώστε ο δέκτης να πληροφορεί τον πομπό για τα πακέτα που έλαβε και για να μπορεί να ζητάει από τον πομπό την αναμετάδοση των πακέτων που διαγράφηκαν [4]. Αν όμως η πιθανότητα διαγραφής των πακέτων είναι μεγάλη τότε θα είναι πολλά τα σήματα που θα αποστέλλονται μέσα από το κανάλι ανάδρασης. Σε αυτή την περίπτωση το κανάλι ανάδρασης θα σπαταλάει μεγάλο κομμάτι από την χωρητικότητα του καναλιού.

Ακόμα περισσότερο, αν κάναμε broadcast εφαρμογή σε ένα τέτοιο κανάλι, τότε εκτός ότι θα αυξάνονταν τα σήματα στο κανάλι ανάδρασης, επιπλέον, ο κάθε δέκτης θα ζητούσε από τον πομπό να αναμεταδώσει διαφορετικά πακέτα (επειδή δεν χάνουν όλοι οι δέκτες τα ίδια πακέτα). Όμως θα υπάρχουν πολύ δέκτες που έχουν είδη λάβει τα περισσότερα από τα πακέτα που αναμεταδίδονται.

Μία εναλλακτική λύση θα μπορούσαν να είναι οι Block κώδικες, οι οποίοι (σχεδόν) δεν απαιτούν κανάλι ανάδρασης. Το μειονέκτημα στους Block κώδικες είναι πως θα πρέπει πρώτα να γίνεται εκτίμηση της πιθανότητας διαγραφής του καναλιού, και στην συνέχεια να γίνει επιλογή του ρυθμού (rate) μετάδοσης που θα χρησιμοποιηθεί. Σε κανάλια όμως όπως αυτό του internet η πιθανότητα διαγραφής μεταβάλλεται συνεχώς, και είναι πολύ πιθανό η εκτίμηση που θα κάνουμε για το κανάλι να μην είναι έγκυρη.

Στους Fountain κώδικες δεν χρειάζεται να γίνει εκτίμηση του καναλιού, καθώς ο ρυθμός του κώδικα μπορεί να αλλάξει κατά την μετάδοση. Ο κωδικοποιητής μπορεί πάντα να παράγει κι άλλα κωδικοποιημένα σύμβολα, μέχρι να καταφέρει ο αποκωδικοποιητής να κάνει την αποκωδικοποίηση.

Ένα παράδειγμα στο οποίο φαίνεται το πλεονέκτημα που έχουν οι rateless κώδικες απέναντι σε αυτούς που έχουν προκαθορισμένο ρυθμό είναι το εξής. Έστω πως θέλουμε να στείλουμε μία πληροφορία μέσα από ένα κανάλι στο οποίο η ισχύς του θορύβου είναι μεταβαλλόμενη με έντονες διακυμάνσεις. Αν κάναμε εκτίμηση της ισχύς του θορύβου κάποιο χρονικό διάστημα που ήταν χαμηλή, τότε αν χρησιμοποιούσαμε έναν κώδικα με προκαθορισμένο ρυθμό, θα επιλέγαμε ένα μικρό επί τις εκατό overhead. Αν όμως κατά την διάρκεια της μετάδοσης, η ισχύς του θρύβου αυξανόταν, τότε ο αποκωδικοποιητής δεν θα κατάφερνε να λάβει αρκετά κωδικοποιημένα σύμβολα, και η αποκωδικοποίηση δεν θα πετύχαινε. Αν όμως χρησιμοποιούσαμε έναν Fountain κώδικα, θα

μπορούσαμε να αυξήσουμε την παράγωγή των κωδικοποιημένα συμβόλων ενώ η μετάδοση έχει είδη ξεκινήσει. Αν από την άλλη κάναμε εκτίμηση της ισχύς του θορύβου σε ένα χρονικό διάστημα που ήταν υψηλή, τότε θα επιλέγαμε ένα μεγάλο επί τις εκατό overhead στον κώδικα με προκαθορισμένο ρυθμό. Αν όμως η ισχύς του θορύβου κατά τη μετάδοση μειωνόταν, τότε θα στέλναμε πολλά αχρείαστα κωδικοποιημένα σύμβολα και θα καταλαμβάναμε μεγάλο κομμάτι της χωρητικότητα του καναλιού χωρίς λόγο.

## 1.3 - LT Κώδικας

Οι LT κώδικες εφευρέθηκαν από το Michael Luby το 1998 και παρουσιάστηκαν για πρώτη φορά από τον ίδιο το 2002 [2]. Οι LT κώδικες είναι οι πρώτοι κώδικες που ανήκουν στο είδος των Fountain κωδίκων ή αλλιώς Rateless κώδικες διαγραφής ή Universal κώδικες.

Ονομάζονται Universal κώδικες γιατί είναι κατάλληλοι για κανάλια διαγραφής με οποιοδήποτε πιθανότητα διαγραφής *p*, καθώς ο κωδικοποιητής μπορεί πάντα να παράγει κι άλλα κωδικοποιημένα σύμβολα [5]. Ο αποκωδικοποιητής απορρίπτει τα κωδικοποιημένα σύμβολα που έχουν υποστεί σφάλμα και συλλέγει τα υπόλοιπα κωδικοποιημένα σύμβολα. Όταν μαζέψει έναν ικανοποιητικό αριθμό από κωδικοποιημένα σύμβολα τότε κάνει την αποκωδικοποίηση. Ο αποκωδικοποιητής χρησιμοποιεί το κανάλι ανάδρασης μόνο όταν καταφέρει να κάνει ανάκτηση (recover) όλα τα σύμβολα πληροφορίας, ώστε να δώσει σήμα στον κωδικοποιητή για να σταματήσει να παράγει κωδικοποιημένα σύμβολα.

Η απόδοση του LT κώδικα αυξάνεται καθώς αυξάνεται το πλήθος των συμβόλων πληροφορίας k [2]. Το επί τις εκατό overhead στους LT κώδικες μειώνεται εκθετικά καθώς ο αριθμός των συμβόλων πληροφορίας k αυξάνεται.[5]

Λόγω των καλών αποτελεσμάτων που έχουν δείξει οι LT κώδικες για την μετάδοση μεγάλου όγκου πληροφορίας σε κανάλια διαγραφής, έχουν προταθεί για να εφαρμοστούν σε διάφορους τομείς ασύρματης επικοινωνίας περιλαμβανομένων των, multicast μετάδοση, broadcast εφαρμογές, δορυφορικές επικοινωνίες και αποθήκευση δεδομένων.

## 1.4 - Εφαρμογές του LT κώδικα

Σε αυτή την ενότητα γίνεται η παρουσίαση κάποιον εφαρμογών του LT κώδικα που έχουν προταθεί.

### *1.4.1 - Broadcast* [4]

Έστω πως δέκα χιλιάδες χρήστες θέλουν να κατεβάσουν μία ψηφιακή ταινία από έναν broadcaster.

Ο broadcasrer χωρίζει την ταινία σε k πακέτα και στην συνέχεια στέλνει τα πακέτα στους δέκτες. Έστω πως ο κάθε δέκτης έχει λάβει f=0.1k λιγότερα από τα k πακέτα. Αν ο broadcaster χρησιμοποιούσε μία ARQ κωδικοποίηση, τότε ο κάθε δέκτης θα ενημέρωνε τον broadcaster για τα f πακέτα που τους λείπουν. Ο broadcaster θα ξαναέστελνε στον κάθε δέκτη τα συγκεκριμένα f πακέτα που του έχουν ζητήσει. Με δέκα χιλιάδες δέκτες να ζητάνε την επανεκπομπή των χαμένων πακέτων, σημαίνει πως θα χρειαστεί να γίνει επανεκπομπή σχεδόν όλης της ταινίας. Εκτός ότι θα γίνει δύο φορές η εκπομπή της ταινίας για να την λάβουν όλοι οι δέκτες, επιπλέον κάποιοι χρήστες θα χρειαστεί να περιμένουν τον διπλάσιο χρόνο από τον προκαθορισμένο χρόνο που θα διαρκούσε η λήψη της ταινίας, γιατί τα πακέτα που τους λείπουν είναι από τα τελευταία πακέτα που στέλνει ο broadcaster.

Αν ο broadcaster χρησιμοποιούσε έναν Fountain κώδικα, τότε ο κάθε δέκτης θα έπρεπε να λάβει K οποιαδήποτε πακέτα. Όπου K πακέτα θα ήταν ελαφρώς περισσότερα από τα k. Με αυτόν το τρόπο ο Broadcaster θα έστελνε περίπου 1,1k πακέτα, και πιθανότατα όλοι οι δέκτες θα κατάφερναν να κάνουν την αποκωδικοποίηση.

### *1.4.2 - Multicast* [6]

Έστω ότι γίνεται διαθέσιμο ένα νέο software προϊόν. Τότε πολλοί χρήστες, σχεδόν μέσα στο ίδιο χρονικό διάστημα θα κάνουν αίτηση για να το λάβουν. Σε αυτή τη περίπτωση συμπεριλαμβάνονται όλα τα προβλήματα όπως αναφέρθηκαν και στην breoadcast περίπτωση. Επιπλέον, έστω ότι δεν έχουν όλοι οι χρήστες τον ίδιο download ρυθμό. Αν ο πομπός δεν χρησιμοποιούσε κωδικοποίηση τότε για να μπορέσει και ο αργότερος χρήστης να λάβει τα πακέτα, θα πρέπει ο πομπός να τα στέλνει με τον αργότερο ρυθμό. Τότε όμως οι χρήστες που έχουν μεγάλους ρυθμούς δεν θα ικανοποιούνται από την ταχύτητα της σύνδεσης. Χρησιμοποιώντας έναν Fountain κώδικα, τότε αν ο ρυθμός μετάδοσης του κωδικοποιητή ήταν προκαθορισμένος και ήταν πχ ίσος με τον ρυθμό του ταχύτερου χρήστη, τότε οι υπόλοιποι δέκτες δεν θα είχαν κανένα πρόβλημα. Δεν έχει σημασία ότι οι αργοί δέκτες θα χάνουν κάποια πακέτα λόγω ταχύτητας, γιατί όλα τα πακέτα που θα καταφέρνουν να λάβουν θα τους είναι χρήσιμα για την αποκωδικοποίηση. Στους Fountain κώδικες δεν έχει σημασία ποιά K πακέτα θα λάβει ο δέκτης. Έτσι θα ικανοποιείται και ο γρήγορος και ο αργός δέκτης. Για τον ίδιο λόγο, αν ένας χρήστης ξεκινήσει να κάνει download το software προϊόν, το οποίο ένας άλλος χρήστης είχε ξεκινήσει νωρίτερα να το κατεβάζει, τότε και ο νέος χρήστης θα μπορεί άμεσα να αρχίσει να λαμβάνει τα ίδια πακέτα που αποστέλλονται και στον παλιό χρήστη.

### *1.4.3 - Αποθήκευση Δεδομένων* [4]

Θέλουμε να αποθηκεύσουμε ένα buckup ενός μεγάλου αρχείου σε έναν αναξιόπιστο σκληρό δίσκο. Αυτός ο σκληρός δίσκος καταστρέφει περίπου $10^{-3}$ αποθηκευμένα πακέτα την ημέρα. Με ποιον τρόπο να αποθηκεύσουμε αρχεία σε αυτόν τον σκληρό δίσκο;

Θα μπορούσαμε να χρησιμοποιήσουμε έναν fountain κώδικα και να "ψεκάσουμε" τον σκληρό δίσκο με κωδικοποιημένα πακέτα. Αν το αρχικό μέγεθος του αρχείου ήταν k πακέτα, τότε για την ανάκτηση του αρχείου θα χρειαστούμε να συλλέξουμε K οποιαδήποτε κωδικοποιημένα πακέτα. Όπου το K είναι ελαφρός μεγαλύτερο από το k. Τα κατεστραμμένα κωδικοποιημένα πακέτα δεν μας απασχολούν καθόλου, τα απορρίπτουμε και ψάχνουμε για άλλα κωδικοποιημένα πακέτα.

### *1.4.4 - Δορυφορικές Επικοινωνίες* [7] [8] [4]

Μία κάτω ζεύξη (downlink) εφαρμογή από δορυφόρο, θα μπορούσε να είναι η ανανέωση των χαρτών στους ψηφιακούς οδηγούς χιλιάδων αυτοκινήτων. Δεν υπάρχει κανάλι ανάδρασης και τα αυτοκίνητα μπορούν να λαμβάνουν το σήμα μόνο όταν βρίσκονται σε ανοιχτό ουρανό. Μία συνηθισμένη μέθοδος για να γίνει αυτή η εφαρμογή είναι να στοιχίζονται τα πακέτα σε ένα carousel και να γίνεται ο κύκλος της αποστολής των πακέτων σε προκαθορισμένα χρονικά διαστήματα. Το αυτοκίνητο μπορεί να μπει σε ένα τούνελ. Για να λάβει τα πακέτα που έχασε θα πρέπει το αυτοκίνητο να βρίσκεται υπό καθαρό ουρανό την στιγμή που θα στέλνονται τα συγκεκριμένα πακέτα του carousel που προηγουμένως είχε χάσει. Το αυτοκίνητο μπορεί να πετύχει ξανά το carousel στο σωστό σημείο ίσως και μετά από μία μέρα. Με τον LT κώδικα δεν θα χρειαζόταν να γίνει αυτό, θα αρκούσε να λάβει οποιαδήποτε από τα K κωδικοποιημένα πακέτα για να κάνει την ανανέωση των χαρτών.

Οι LT κώδικες δεν απαιτούν κανάλι ανάδρασης, και για αυτό μπορούν να χρησιμοποιηθούν στην κάτω ζεύξη (Downlink) επικοινωνία δορυφόρων με τη γη. [7] Αν η ισχύ λήψης είναι πάνω από το ορισμένο κατώφλι, τότε τα πακέτα μεταχειρίζονται ως "σωστά" από τον αποκωδικοποιητή. Αν η ισχύ λήψης είναι χαμηλότερη από το ορισμένο κατώφλι, τότε τα ληφθέντα πακέτα μεταχειρίζονται ως "διαγραμμένα" από τον αποκωδικοποιητή. Ο δέκτης όταν λάβει k πακέτα τότε μπορεί να επιχειρήσει να κάνει αποκωδικοποίηση. Αν η αποκωδικοποίηση αποτύχει τότε περιμένει να λάβει περισσότερα πακέτα για να ξαναδοκιμάσει να κάνει την αποκωδικοποίηση.

Όπως φαίνεται στο [7] ένα πλεονέκτημα που έχουν οι LT κώδικες απέναντι σε άλλους που χρησιμοποιούνται σε δορυφορικές επικοινωνίες είναι πως οι LT κώδικες δεν χρειάζονται να υποστούν διάπλεξη, καθώς εκ τον προτέρων τα κωδικοποιημένα πακέτα του LT είναι ανακατεμένα.

# Κεφάλαιο 2 - Ανάλυση του LT κώδικα

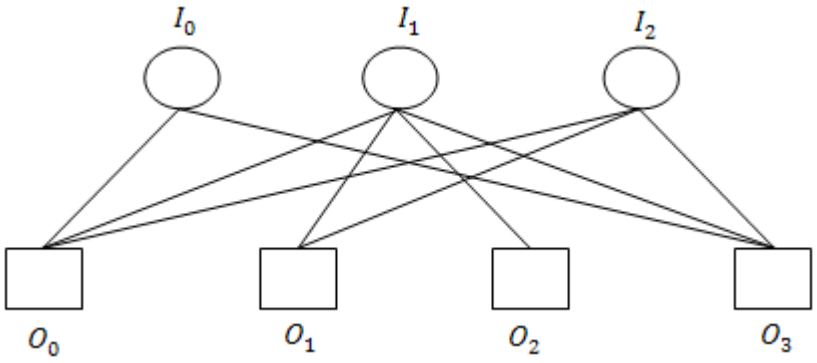
Στο κεφάλαιο 2 γίνεται ανάλυση της κωδικοποίησης και αποκωδικοποίησης LT, καθώς και της κατανομής Robust Soliton Distribution την οποία χρησιμοποιεί ο LT κώδικας

## 2.1 - LT κωδικοποίηση

Έστω ότι ο αριθμός των συμβόλων πληροφορίας είναι k και ο αριθμός των κωδικοποιημένων συμβόλων που θα παραχθούν είναι n, τότε η διαδικασία της LT κωδικοποίησης είναι η εξής:

- Σε κάθε ένα κωδικοποιημένο σύμβολο αντιστοιχεί ένας αριθμός που ονομάζεται βαθμός (degree), και ισχύει $1 \leq βαθμός \leq k$. Ο βαθμός που αντιστοιχεί σε κάθε κωδικοποιημένο σύμβολο επιλέγεται τυχαία σύμφωνα με μία κατανομή που ονομάζεται Robust Soliton distribution την οποία θα αναλύσουμε παρακάτω.
- Στην συνέχεια διαλέγει με τυχαίο τρόπο δείκτες που αντιστοιχούν στα σύμβολα πληροφορίας, όλοι οι δείκτες έχουν την ίδια πιθανότητα εμφάνισης. Διαλέγει τόσους δείκτες όσους λέει κάθε φορά ο βαθμός. Αυτά τα σύμβολα πληροφορίας ονομάζονται πλέον γειτονικά (neighbors) του κωδικοποιημένου συμβόλου. Κάνει XOR τα γειτονικά σύμβολα μεταξύ τους και το αποτέλεσμα των πράξεων αποτελεί ένα κωδικοποιημένο σύμβολο.
- Επαναλαμβάνοντας την διαδικασία n φορές παράγονται n κωδικοποιημένα σύμβολα.

Η σχέση των κωδικοποιημένων συμβόλων με τα σύμβολα πληροφορίας μπορεί να περιγραφεί με ένα bipartite graph σαν αυτό που φαίνεται στην Εικόνα 1. Επειδή οι συνδέσεις σε αυτό το γράφημα είναι αραιές ονομάζεται αραιό (sparse). Στον LT κώδικα το μέσω όρο των βαθμών είναι πάντα μικρότερο από k και για αυτό οι συνδέσεις είναι αραιές. Το συνολικό άθροισμα των βαθμών, ισούται με το πλήθος των συνδέσεων στο αραιό γράφημα του LT κώδικα.




**Εικόνα 1:** Αραιό γράφημα (Sparse graph)

Στο αραιό γράφημα της εικόνας 1, τα $I$ συμβολίζουν τα σύμβολα πληροφορίας και τα $O$ συμβολίζουν τα κωδικοποιημένα σύμβολα. Βλέποντας τις συνδέσεις που υπάρχουν μεταξύ τον συμβόλων πληροφορίας και των κωδικοποιημένων συμβόλων βγάζουμε τα εξής συμπεράσματα:

Το $O_0$ είναι βαθμού 3 γιατί έχει τρεις συνδέσεις, και τα γειτονικά του σύμβολα είναι το $I_0$, το $I_1$και το $I_2$
Το $O_1$ είναι βαθμού 2 γιατί έχει δύο συνδέσεις, τα γειτονικά του σύμβολα είναι το $I_1$και το $I_2$
Το $O_2$ είναι βαθμού 1 γιατί έχει μία μόνο σύνδεση, και το γειτονικό του σύμβολο είναι το $I_1$
Το $O_3$ είναι βαθμού 3 γιατί έχει τρεις συνδέσεις, και τα γειτονικά του σύμβολα είναι το $I_0$, το $I_1$ και το $I_2$

Αν οι τιμές των συμβόλων πληροφορίας είναι :

| Info symbols | | |
|---|---|---|
| $I_0$ | $I_1$ | $I_2$ |
| 1 | 1 | 0 |

Τότε τα κωδικοποιημένα σύμβολα παράγονται με τον εξής τρόπο:

$$O_0 = I_0\ XOR\ I_1\ XOR\ I_2$$
$$\rightarrow O_0 = 1\ XOR\ 1\ XOR\ 0 = 0$$
$$O_1 = I_1\ XOR\ I_2$$
$$\rightarrow O_1 = 1\ XOR\ 0 = 1$$

$$O_2 = \ I_1$$
$$\rightarrow O_2 = \ 1$$
$$O_3 = I_0 \ XOR \ \ I_1 \ XOR \ \ I_2$$
$$\rightarrow O_3 = 1 \ XOR \ 1 \ \ XOR \ 0 = 0$$

Οπότε συνολικά τα κωδικοποιημένα σύμβολα είναι :

| Encoded symbols | | | |
|---|---|---|---|
| $O_0$ | $O_1$ | $O_2$ | $O_3$ |
| 0 | 1 | 1 | 0 |

## 2.2 - LT αποκωδικοποίηση (αλγόριθμος belief propagation )

Έχουν χρησιμοποιηθεί και προταθεί διάφοροι αλγόριθμοι για την διαδικασία τις αποκωδικοποίησης του LT κώδικα [5][9], εμείς θα παρουσιάσουμε τον αλγόριθμο Belief Propagation που είναι αυτός που παρουσίασε ο Michael Luby και είναι ο πρώτος που χρησιμοποιήθηκε.

Ο αποκωδικοποιητής θα πρέπει να μπορεί να κάνει την αποκωδικοποίηση από ένα οποιοδήποτε υποσύνολο των n κωδικοποιημένων συμβόλων (αρκεί το υποσύνολο να είναι μεγαλύτερο του k) [5]. Ο αποκωδικοποιητής γνωρίζει ποιος είναι ο βαθμός και ποια είναι τα γειτονικά που αντιστοιχούν σε κάθε κωδικοποιημένο σύμβολο [10]. Ένας τρόπος για να γίνει αυτό είναι ο κωδικοποιητής και αποκωδικοποιητής να χρησιμοποιούν τον ίδιο σπόρο (seed) στις ψευδοτυχαίες μηχανές (Random Generator Machines) που παράγουν τους βαθμούς και τους δείκτες των γειτονικών συμβόλων.

Η ανάλυση της διαδικασίας της αποκωδικοποίησης θα γίνει με την βοήθεια του παρακάτω αραιού γραφήματος (sparse graph). Σε αυτό το γράφημα τα λαμβανόμενα σύμβολα (received symbols), δηλαδή τα κωδικοποιημένα σύμβολα που έχουν φτάσει στον αποκωδικοποιητή είναι τα *O,* και τα σύμβολα που προσπαθεί να ανακτήσει ο αποκωδικοποιητής (recovered symbols), είναι τα *I*.

- Ο αποκωδικοποιητής ψάχνει να βρει ένα λαμβανόμενο σύμβολο *O* με βαθμό 1, δηλαδή μόνο με μία σύνδεση (ένα γειτονικό).
- Αντιγράφει την τιμή αυτού του *O* στο *I* με το οποίο συνδέεται, με αυτόν τον τρόπο έχει κάνει την πρώτη ανάκτηση.

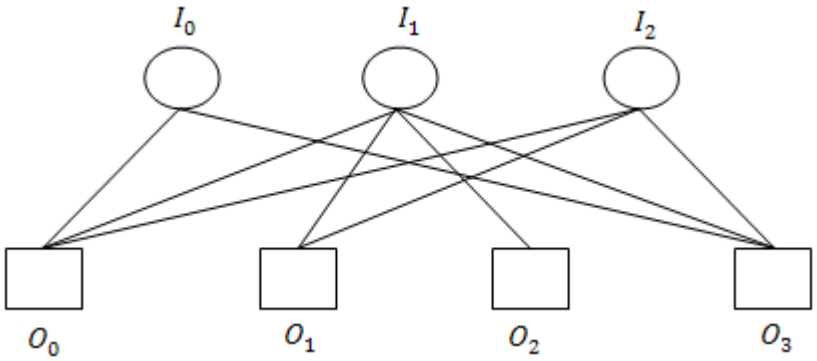
- Στην συνέχεια κάνει XOR την τιμή του *I* που μόλις ανέκτησε με κάθε ένα από τα λαμβανόμενα σύμβολα *O* που έχουν σύνδεση με το ανακτώμενο σύμβολο *I,* και το αποτέλεσμα της πράξης XOR το αναθέτει κάθε φορά στο αντίστοιχο *O* (με το οποίο έκανε XOR). Συγχρόνως κάθε φορά που κάνει XOR το *I* με ένα *O,* διαγράφει την σύνδεση μεταξύ τους. Με την πράξη XOR γίνεται ανανέωση (refresh) των λαμβανόμενων συμβόλων, και με την διαγραφή των συνδέσεων πέφτει ο βαθμός των λαμβανόμενων συμβόλων για να εμφανιστούν κι άλλα λαμβανόμενα σύμβολα με βαθμό 1.
- Τα προηγούμενα τρία βήματα επαναλαμβάνονται έως ότου δεν υπάρχει πλέον κανένα λαμβανόμενο σύμβολο με βαθμό 1, δηλαδή κανένα *O* που να έχει μόνο μία σύνδεση.

Αν ο αποκωδικοποιητής δεν μπορεί να βρει άλλο λαμβανόμενο σύμβολο *O* με βαθμό 1 και έχουν ανακτηθεί όλα τα *I* τότε η αποκωδικοποίηση πέτυχε.
Αν ο αποκωδικοποιητής δεν μπορεί να βρει άλλο λαμβανόμενο σύμβολο *O* με βαθμό 1 και δεν έχουν ανακτηθεί όλα τα *I* τότε η αποκωδικοποίηση απέτυχε. [9]

Με τον αλγόριθμο Belief Propagation οι LT κώδικες παρουσιάζουν γραμμική αύξηση των λειτουργιών (operations) του αποκωδικοποιητή [5]. Όπου ως λειτουργίες του αποκωδικοποιητή, θεωρούμε το πλήθος τον XOR πράξεων που κάνει. Από την άλλη, το μειονέκτημα που έχει ο Belief Propagation είναι πως για μικρό k απαιτεί μεγάλο επί τις εκατό overhead προκειμένου να εξασφαλίσουμε μεγάλη πιθανότητα επιτυχίας της αποκωδικοποίησης (Decoding success rate) .

Ακολουθεί ένα παράδειγμα αποκωδικοποίησης.

Ο αποκωδικοποιητής γνωρίζει τα κωδικοποιημένα σύμβολα που έλαβε, γνωρίζει τον βαθμό που έχει το κάθε κωδικοποιημένο σύμβολο, και γνωρίζει ποιοι είναι οι δείκτες των γειτονικών συμβόλων του κάθε κωδικοποιημένου συμβόλου. Με άλλα λόγια γνωρίζει το αραιό γράφημα και τα λαμβανόμενα σύμβολα.



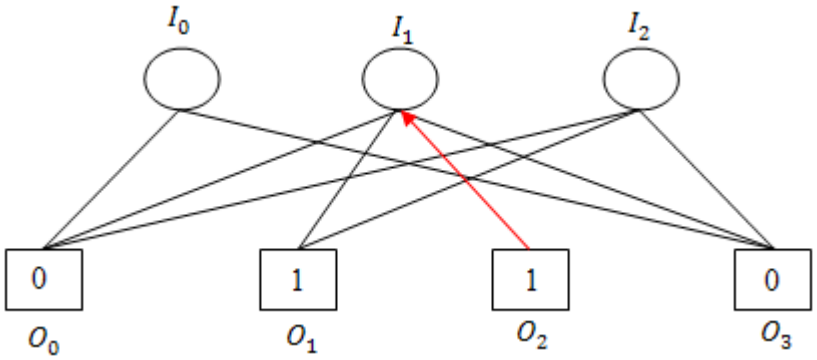

**Εικόνα 2:** Το αραιό γράφημα, πριν ξεκινήσει η αποκωδικοποίηση

Έστω πως ο αποκωδικοποιητής έλαβε τα παρακάτω σύμβολα.

| Received symbols | | | |
|---|---|---|---|
| $O_0$ | $O_1$ | $O_2$ | $O_3$ |
| 0 | 1 | 1 | 0 |

Και προσπαθεί να βρει τα $I$ του αραιού γραφήματος. Ο αποκωδικοποιητής ψάχνει να βρει κάποιο λαμβανόμενο σύμβολο με βαθμό 1.

Το $O_2$ είναι το μοναδικό λαμβανόμενο σύμβολο με βαθμό 1 και το γειτονικό του είναι το $I_1$.



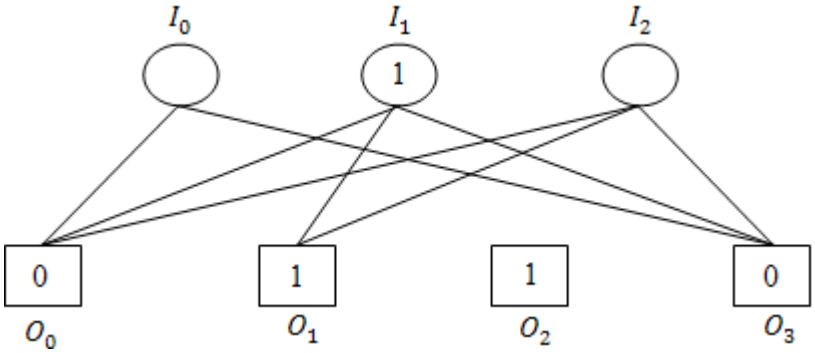

**Εικόνα 3:** $\mathbf{1^o}$ βήμα αποκωδικοποίησης

Κάνει ανάκτηση το $I_1$και διαγράφει τη σύνδεση μεταξύ τους.

Οπότε:

$$I_1 = O_2$$

$$\rightarrow I_1 = 1$$




**Εικόνα 4: $2^o$** βήμα αποκωδικοποίησης

Στη συνέχεια κάνει XOR τα λαμβανόμενα σύμβολα που είναι συνδεμένα με το $I_1$ και το αποτέλεσμα της κάθε πράξης το αναθέτει στο αντίστοιχο λαμβανόμενο σύμβολο.

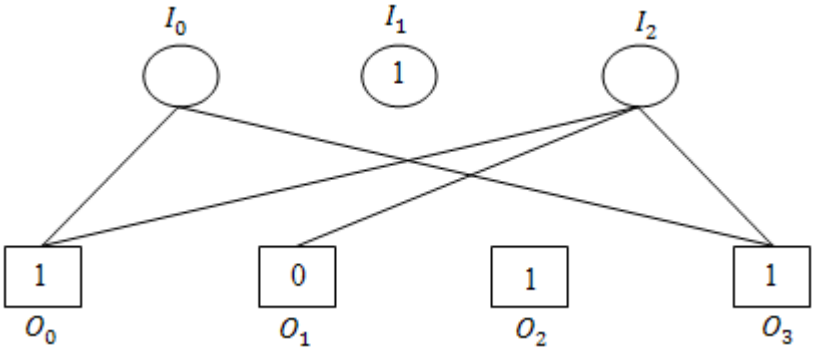
Κάθε φορά που κάνει XOR διαγράφει την σύνδεση μεταξύ του *O* και του *I*. Με αυτόν το τρόπο μειώνονται οι βαθμοί των λαμβανόμενων συμβόλων.

Οπότε:

$$O_0 = O_0 \; XOR \; I_1$$
$$\rightarrow O_0 = 0 \; XOR \; 1 = 1$$

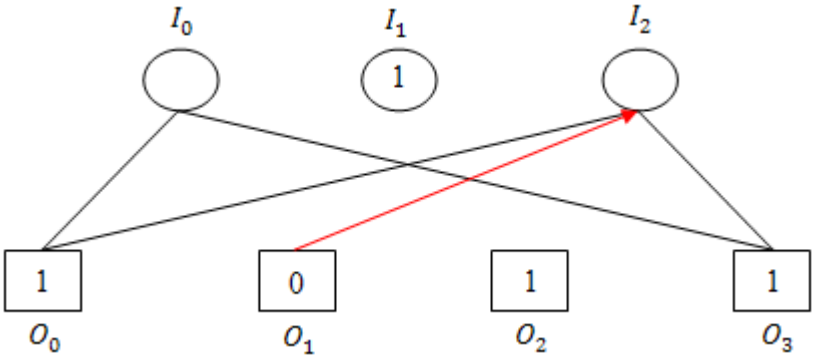
$$O_1 = O_1 \; XOR \; I_1$$
$$\rightarrow O_1 = 1 \; XOR \; 1 = 0$$

$$O_3 = O_3 \; XOR \; I_1$$
$$\rightarrow O_3 = 0 \; XOR \; 1 = 1$$




**Εικόνα 5:** $\mathbf{3^o}$ βήμα αποκωδικοποίησης

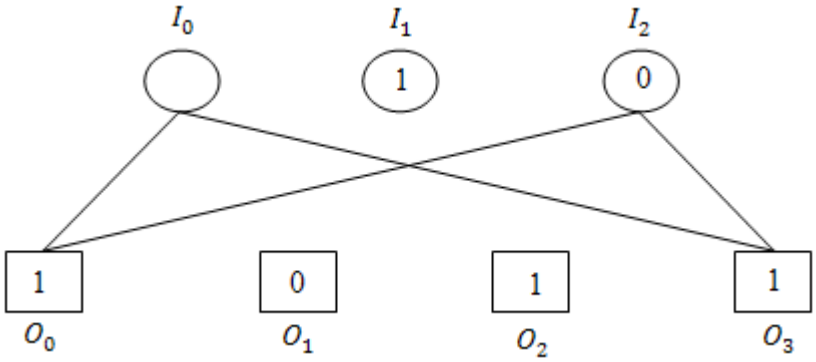
Το επόμενο λαμβανόμενο σύμβολο με βαθμό 1 που βρίσκει είναι το $O_1$,




**Εικόνα 6**: $\mathbf{4^o}$ βήμα αποκωδικοποίησης

Κάνει ανάκτηση το $I_2$.

$$I_2 = O_1$$
$$\rightarrow I_2 = 0$$



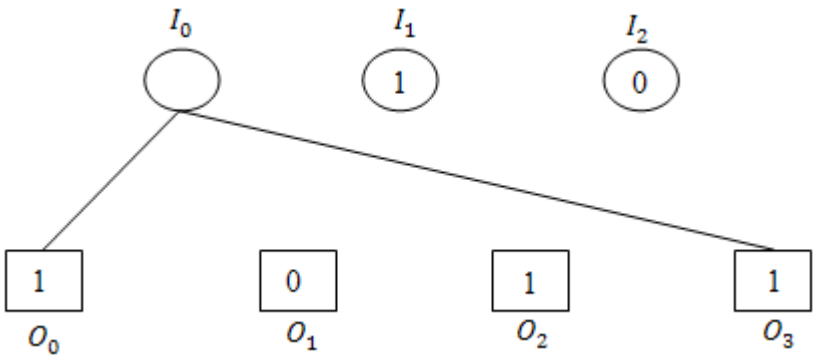

**Εικόνα 7: 5$^o$** βήμα αποκωδικοποίησης

Κάνει ανανέωση τα $O_0$ και $O_3$, και διαγράφει την σύνδεσή τους με το $I_2$.

$$O_0 = O_0 \; XOR \; I_2$$
$$\rightarrow O_0 = 1 \; XOR \; 0 = 1$$

.

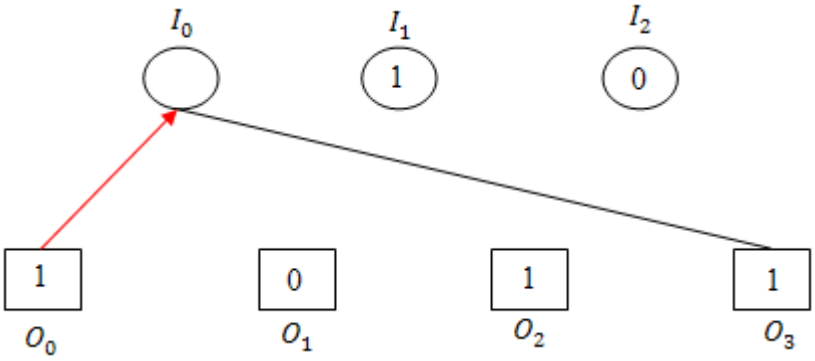
$$O_3 = O_3 \; XOR \; I_2$$
$$\rightarrow O_3 = 1 \; XOR \; 0 = 1$$




**Εικόνα 8: 6$^o$** βήμα αποκωδικοποίησης

Και τέλος, βρίσκει δύο λαμβανόμενα σύμβολα με βαθμό 1, και είναι και τα δύο συνδεδεμένα με το ίδιο γειτονικό.

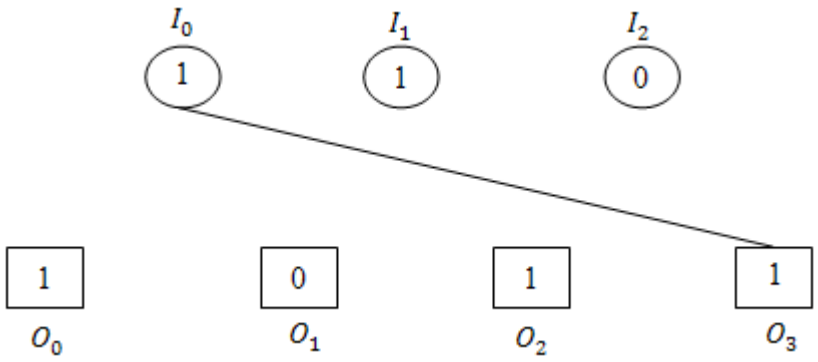
Και τα δύο λαμβανόμενα σύμβολα έχουν την ίδια τιμή. Αυτό είναι και το επιθυμητό. Αν είχαν διαφορετική τιμή, τότε σημαίνει πως έχουμε κάνει κάποιο λάθος στη διαδικασία. Μπορούμε να χρησιμοποιήσουμε όποιο λαμβανόμενο σύμβολο θέλουμε για να κάνουμε την ανάκτηση, το αποτέλεσμα θα είναι τι ίδιο. Έστω ότι διαλέγουμε το $O_0$.




**Εικόνα 9: 7$^{ο}$** βήμα αποκωδικοποίησης

Κάνει ανάκτηση το $I_0$.

$$O_0 = I_0$$
$$\rightarrow I_0 = 1$$



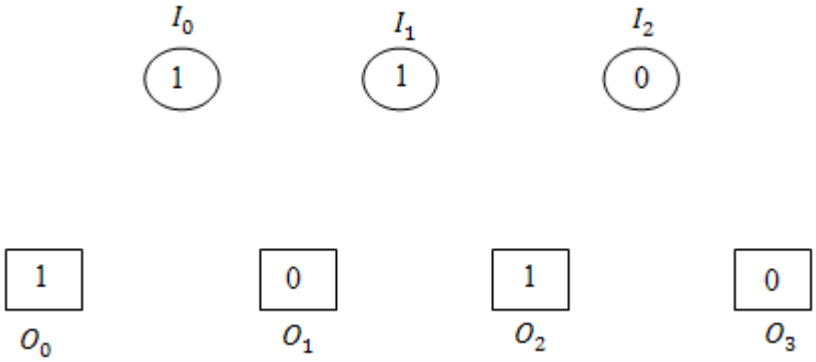

**Εικόνα 10: 8$^{ο}$** βήμα αποκωδικοποίησης

Και κάνει την τελευταία ανανέωση, η οποία δεν έχει κάποιο ουσιαστικό ρόλο καθώς έχει γίνει είδη ανάκτηση όλων των συμβόλων πληροφορίας.

$$O_3 = O_3 \; XOR \; I_0$$

$$\rightarrow O_3 = 1\ XOR\ 0 = 0$$

Τώρα πλέον δεν υπάρχει καμία σύνδεση μεταξύ των λαμβανόμενων συμβόλων και των ανακτημένων συμβόλων.




**Εικόνα 11:** $9^o$ βήμα αποκωδικοποίησης

| Recovered symbols | | |
|---|---|---|
| $I_0$ | $I_1$ | $I_2$ |
| 1 | 1 | 0 |

Η αποκωδικοποίηση αποτυγχάνει όταν ο αποκωδικοποιητής δεν καταφέρει να κάνει ανάκτηση και τα k σύμβολα πληροφορίας. Αυτό μπορεί να συμβεί είτε αν σε κάποιο στάδιο της αποκωδικοποίησης (πλην το τελευταίο) δεν υπάρξει λαμβανόμενο σύμβολο με βαθμό 1 [10], ή αν υπάρχει κάποιο σύμβολο πληροφορίας που δεν είναι εξαρχής συνδεμένο με κανένα από τα λαμβανόμενα σύμβολα.

## 2.3 - Κατανομή των Βαθμών (Degree Distribution)

Η Κατανομή των Βαθμών είναι καθοριστικό κομμάτι του LT κώδικα. Ο στόχος της Κατανομής των Βαθμών είναι να δημιουργηθούν πολλά κωδικοποιημένων συμβόλων με χαμηλό βαθμό, ώστε να μην ξεμείνει η διαδικασία της αποκωδικοποίησης από λαμβανόμενα σύμβολα με βαθμό 1, να μην καλυφθεί περισσότερες φορές από όσες χρειάζεται το ίδιο σύμβολο πληροφορίας, και συγχρόνως να έχουν καλυφθεί όλα τα σύμβολα πληροφορίας από τα κωδικοποιημένα σύμβολα ώστε να μπορέσει ο αποκωδικοποιητής να κάνει ανάκτηση όλα τα σύμβολα πληροφορίας [9][5]. Σε αυτή την παράγραφο περιγράφονται οι κατανομές που πρότεινε ο Michael Luby για την παραγωγή των βαθμών. Η Ideal Soliton Distribution είναι η ιδανική κατανομή, και δεν μπορεί να εφαρμοστεί στην πράξη. Η

ανάλυσή της όμως μας βοηθάει να καταλάβουμε καλύτερα την Robust Soliton Distribution η οποία χρησιμοποιείται σε πρακτικές εφαρμογές.

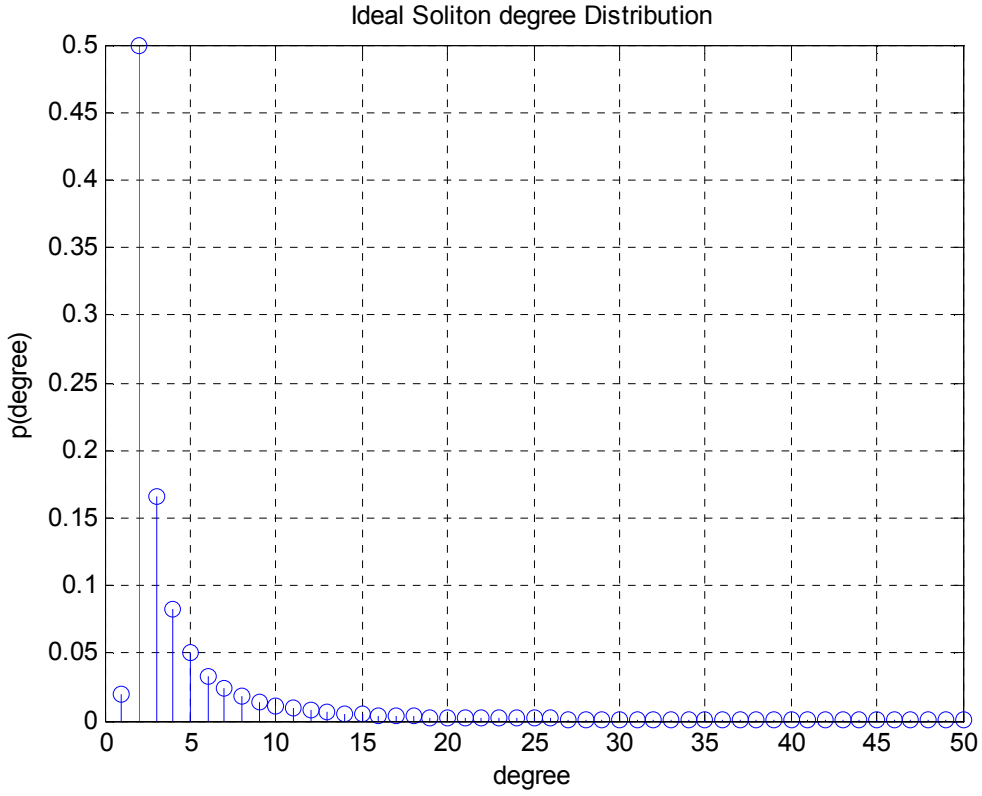
### *2.3.1 - Ideal Soliton Distribution*

Στόχος της Ideal Soliton Distribution είναι, από τα n κωδικοποιημένα σύμβολα που θα χρησιμοποιηθούν για την αποκωδικοποίηση μόνο ένα να έχει βαθμό 1, και μετά από κάθε ανανέωση (refresh) που θα κάνει ο αποκωδικοποιητής να υπάρχει πάντα μόνο ένα λαμβανόμενο σύμβολο με βαθμό 1. [4]

Η Ideal Soliton Distribution είναι πολύ εύθραυστη. Είναι πολύ εύκολο είτε αρχικά είτε μετά από κάποια ανανέωση, να μην υπάρχει πλέον κανένα λαμβανόμενο σύμβολο με βαθμό 1, και αυτό σημαίνει πως θα διακοπεί η διαδικασία της αποκωδικοποίησης.

Αν το $i$ παίρνει τιμές από 1 ως k (όπου $i$ είναι οι τιμές που παίρνει ο βαθμός) τότε η παρακάτω σχέση περιγράφει ποιές είναι οι πιθανότητες εμφάνισης που έχει ο κάθε βαθμός, σύμφωνα με την Ideal Soliton Distribution

$$\rho(i) = \begin{cases} 1/k\,, & i = 1 \\ 1/i(i-1)\,, & i = 2, \dots, k \end{cases} \quad (1) \qquad [2]$$




**Εικόνα 12:** Η Ideal Soliton Distribution για k=50

Όπως φαίνεται από τον τύπο της Ideal Soliton Distribution, αλλά και από την Εικόνα 12, το 50% των κωδικοποιημένων συμβόλων έχουν βαθμό 1.

### *2.3.2 - Robust Soliton Distribution*

Αν χρησιμοποιήσουμε μία κατανομή ώστε το μέγεθος του ripple να είναι μεγάλο τότε θα μπορούσαμε να εξασφαλίσουμε την επιτυχία της αποκωδικοποίησης, όμως παράλληλα θα παράγαμε πολλά αχρείαστα κωδικοποιημένα σύμβολα [11]. Όπου ripple είναι το μέσω όρο των κωδικοποιημένων συμβόλων με βαθμό 1 [4]. Χρειαζόμαστε μία κατανομή η οποία να διατηρεί το ripple σε τέτοιο μέγεθος ώστε να πετυχαίνει η αποκωδικοποίηση, χωρίς όμως να παράγει πολλά περιττά κωδικοποιημένα σύμβολα και να αποφεύγει την υπερκάλυψη των ίδιον συμβόλων πληροφορίας. Αυτή τη δουλεία κάνει η Robust Soliton Distribution.

Για να φτιάξουμε την Robust Soliton distribution *μ(i)* προσθέτουμε στην Ideal Soliton distribution *p(i)* την tau κατανομή *τ(i)* και το αποτέλεσμα το διαιρούμε με το *β,* όπου:

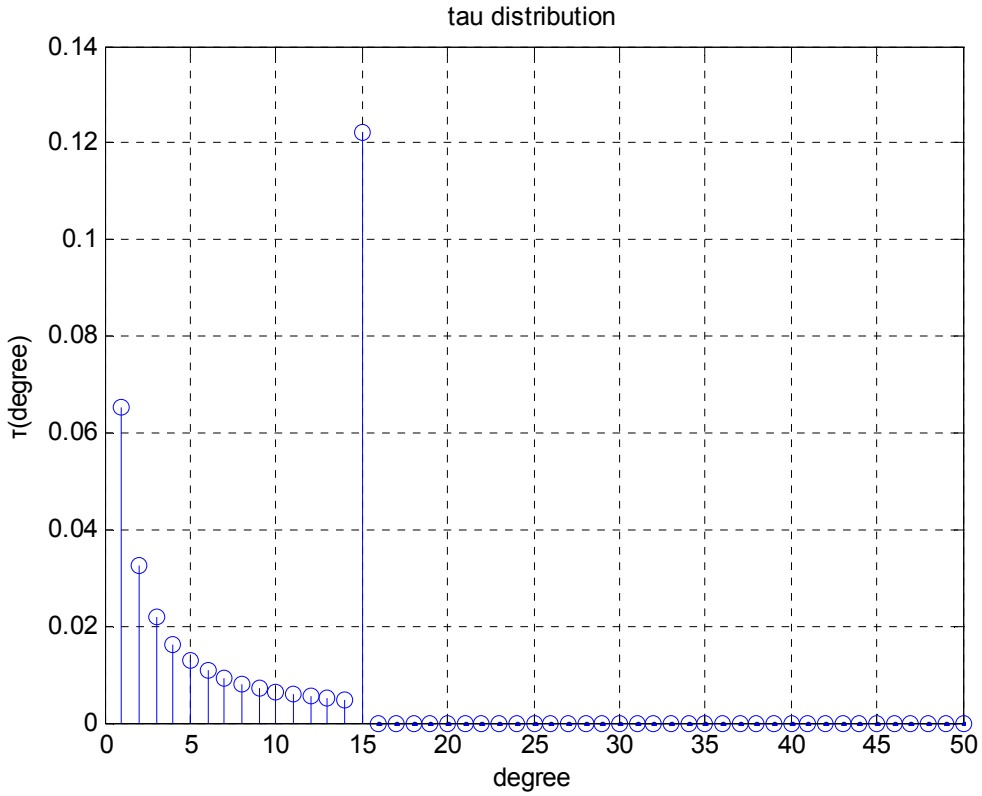
$$\mu(i) = \frac{\rho(i) + \tau(i)}{\beta}, \quad i = 1, \dots, k \quad (2)$$

$$\tau(i) = \begin{cases} R/ik\,, & i = 0, \dots, 1/R - 1 \\ R\ln(R/\delta)/k\,, & i = k/R \\ 0, & i = k/R + 1, \dots, k \end{cases} \quad (3)$$

[2]

$$R = c\sqrt{k}\ln\frac{k}{\delta} \quad (4)$$

$$\beta = \sum_{i=1}^{k} \rho(i) + \tau(i) \quad (5)$$




**Εικόνα 13:** Η tau κατανομή για k=50, δ=05 και c=0.1

Στην tau κατανομή η μεγαλύτερη ακίδα είναι για $βαθμό = k/R$ αντι για $βαθμό = 1$ που ειναι στην Ideal Soliton distribution. Αυτό έχει σαν αποτέλεσμα να έχει πιο αυξημένο το μέσω όρο των βαθμών η Robust Soliton Distribution σε σχέση με την Ideal Soliton Distribution.

Η Robust Soliton Distribution είναι σχεδιασμένη ώστε να διατηρεί το μέγεθος του ripple κατά μέσω όρο ίσο με R κατά την διαδικασία της αποκωδικοποίησης [4][2]. Όπως φαίνετε με την Robust Soliton Distribution έχουμε εισάγει δύο παραμέτρους, την $c$ και την $\delta$, των οποίων οι τιμές κυμαίνονται στα παρακάτω διαστήματα:

$$\frac{1}{k-1}\cdot\frac{\sqrt{k}}{\ln(k/\delta)} \leq c \leq \frac{1}{2}\cdot\frac{\sqrt{k}}{\ln(k/\delta)} \quad (6)$$

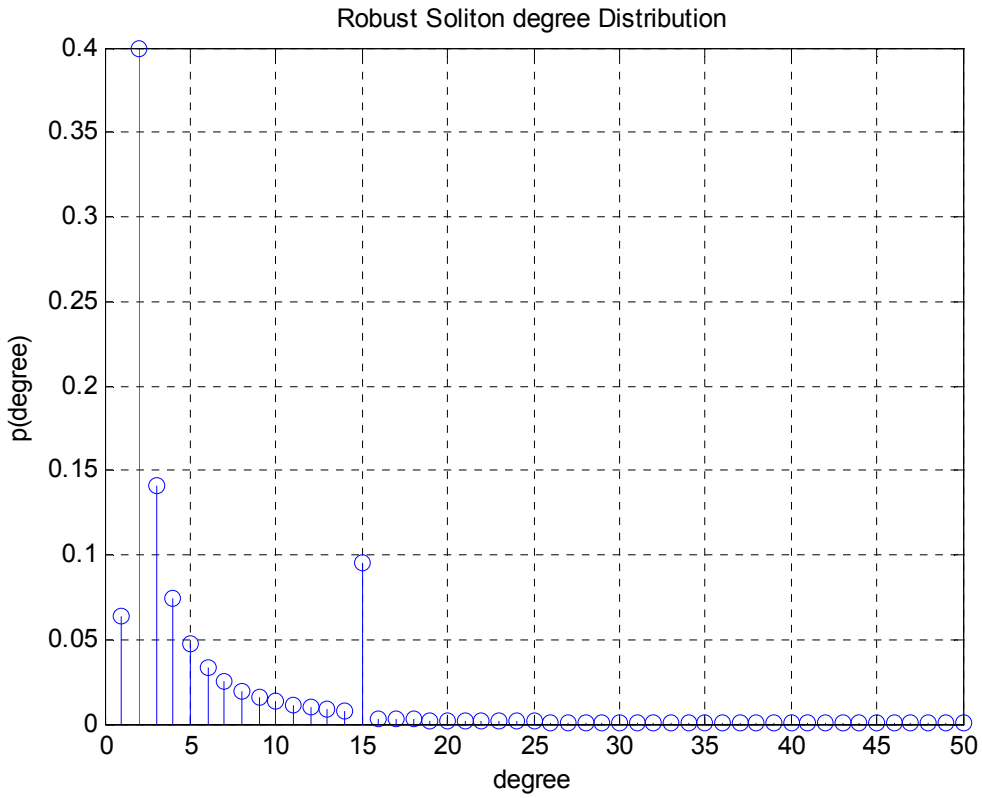
[12][13]

$$\delta \in [0,1] \quad (7)$$

Σύμφωνα με το άρθρο [2], αν ο αριθμός των κωδικοποιημένων συμβόλων που θα χρησιμοποιηθούν για την αποκωδικοποίηση είναι ίσος με $n = k \cdot \beta$, τότε το ποσοστό επιτυχίας της αποκωδικοποίησης, είναι $1 - \delta$, και το μέσο όρο των λειτουργιών (operations) του αποκωδικοποιητή είναι $O(k\ln(k/\delta))$. [2][13]

Αργότερα όμως παρατηρήθηκε πως η πιθανότητα αποτυχίας είναι πολύ μικρότερη από $\delta$. Με αποτέλεσμα να μπορούν να σχεδιαστούν LT κώδικες με μεγάλες τιμές του $\delta$ και να έχουν καλή απόδοση. [10]

Το μέσο όρο των βαθμών στην Robust Soliton Distribution είναι $D = O(\ln\frac{k}{\delta})$ [14]




**Εικόνα 14:** Η Robust Soliton Distribution για k=50, δ=05 και c=0.1

Η δεύτερη ακίδα στην Robust Soliton Distribution αυξάνει τις πιθανότητες ένα σύμβολο πληροφορίας να έχει καλυφθεί περισσότερες από μία φορά. [15]

Σύμφωνα με την μελέτη που έκανε ο M.Luby, η Robust Soliton Distribution δουλεύει πολύ καλά σε κανάλια διαγραφής (erasure channels), το μειονέκτημά της όμως είναι πως η πολυπλοκότητα του αποκωδικοποιητή αυξάνεται κατά $O(k \ln k)$ καθώς αυξάνεται το k. Στους Raptor κώδικες, οι οποίο είναι μία εξέλιξη των LT κωδίκων, η πολυπλοκότητα του αποκωδικοποιητή αυξάνεται μόνο κατά $O(k)$. [16]

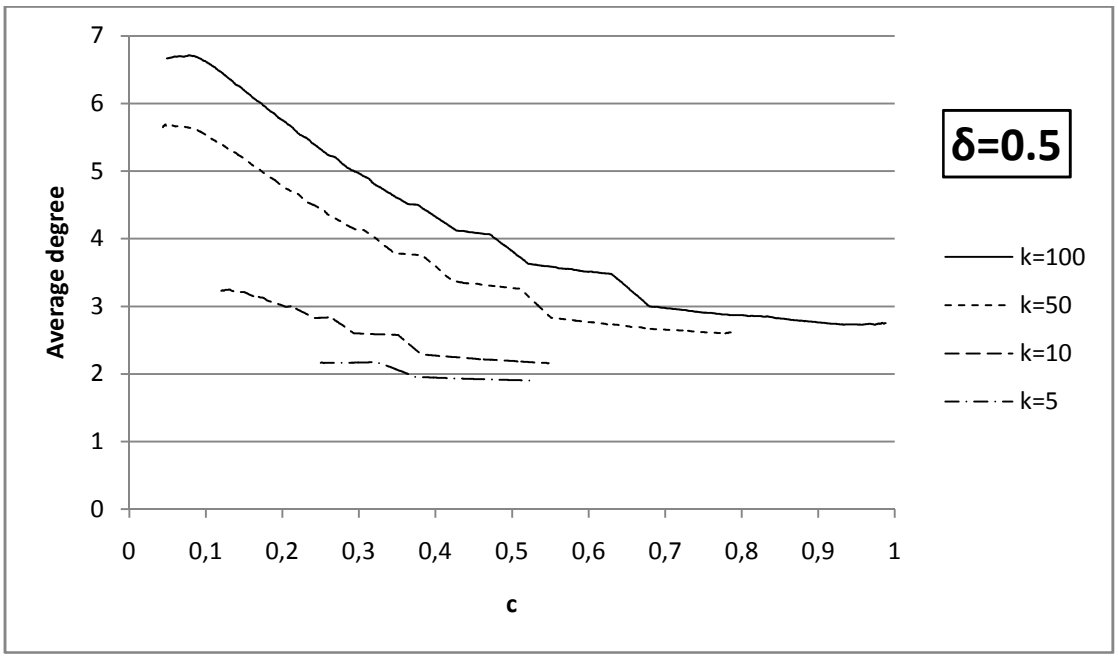
### *2.3.3 - Παράμετροι c και δ* [13]

Αυτή η παράγραφος είναι βασισμένη στο άρθρο [13].

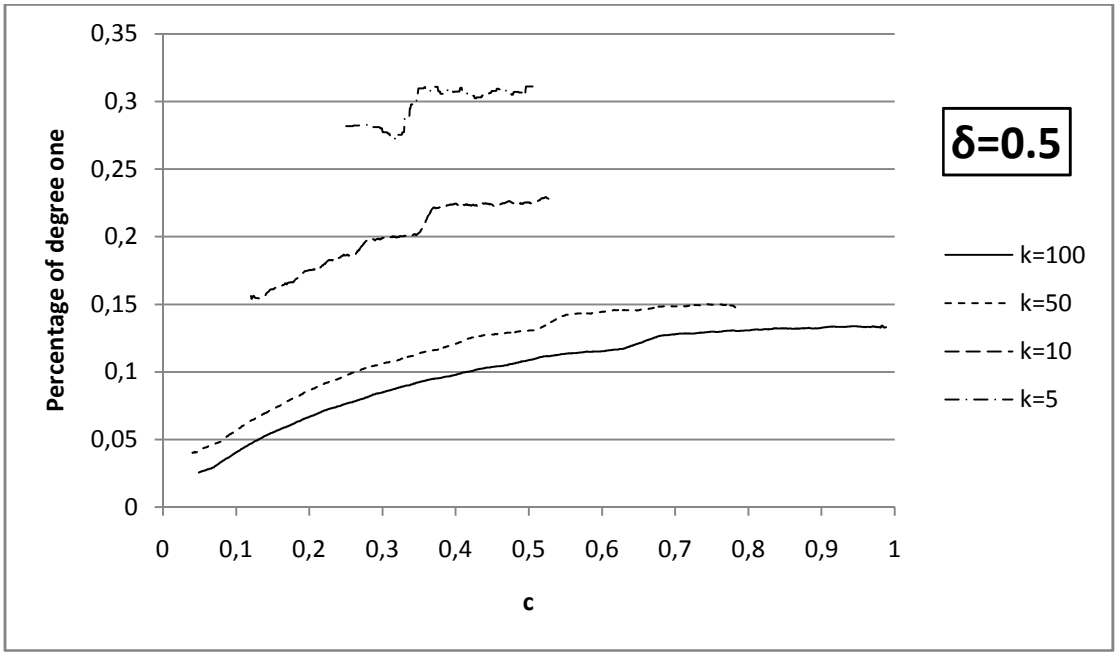
Αλλάζοντας τις τιμές των παραμέτρων *c* και *δ*, αλλάζει η μορφή της Robust Soliton Distribution.

#### ***2.3.3.1 - Αλλάζοντας το c***

Αλλάζοντας το *c* μεταβάλλεται το μέσω όρο των βαθμών και οι πιθανότητες εμφάνισης βαθμού 1. Για μικρά k, αυξάνοντας το *c* μειώνεται το μέσω όρο των βαθμών και αυξάνεται η πιθανότητα εμφάνισης του βαθμού 1, όπως φαίνεται στις εικόνες 15 και 16 αντίστοιχα.




**Εικόνα 15:** Μεταβολή του μέσου όρου του βαθμών έναντι του *c* για σταθερό δ=0.5

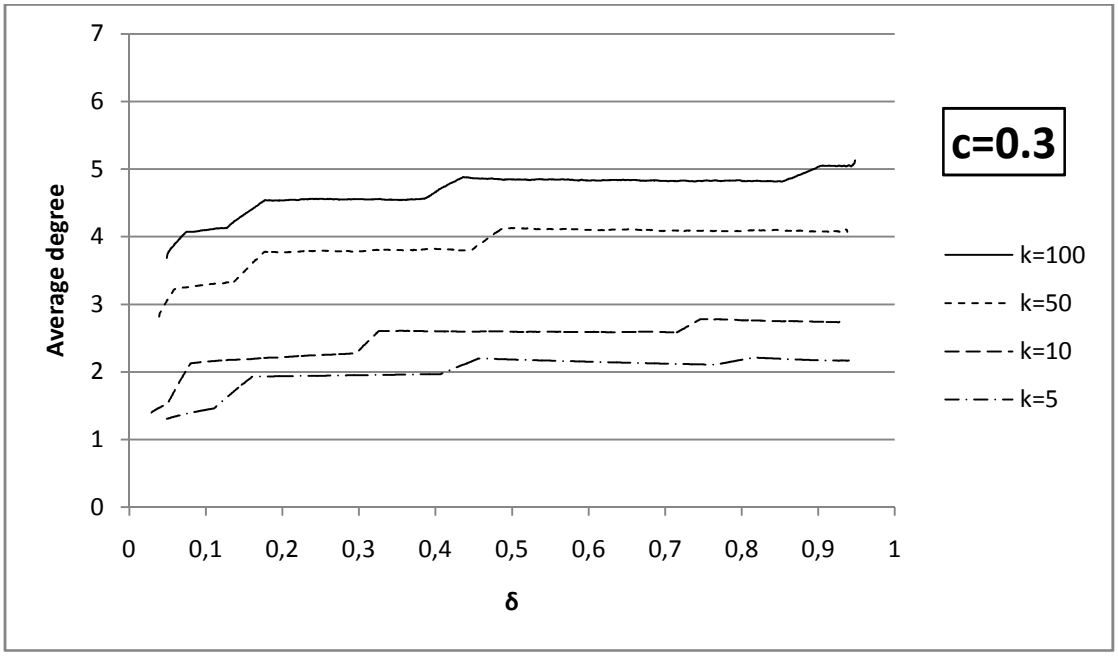



**Εικόνα 16:** Μεταβολή του ποσοστού εμφάνισης του βαθμού 1 έναντι του *c* για σταθερό δ=0.5

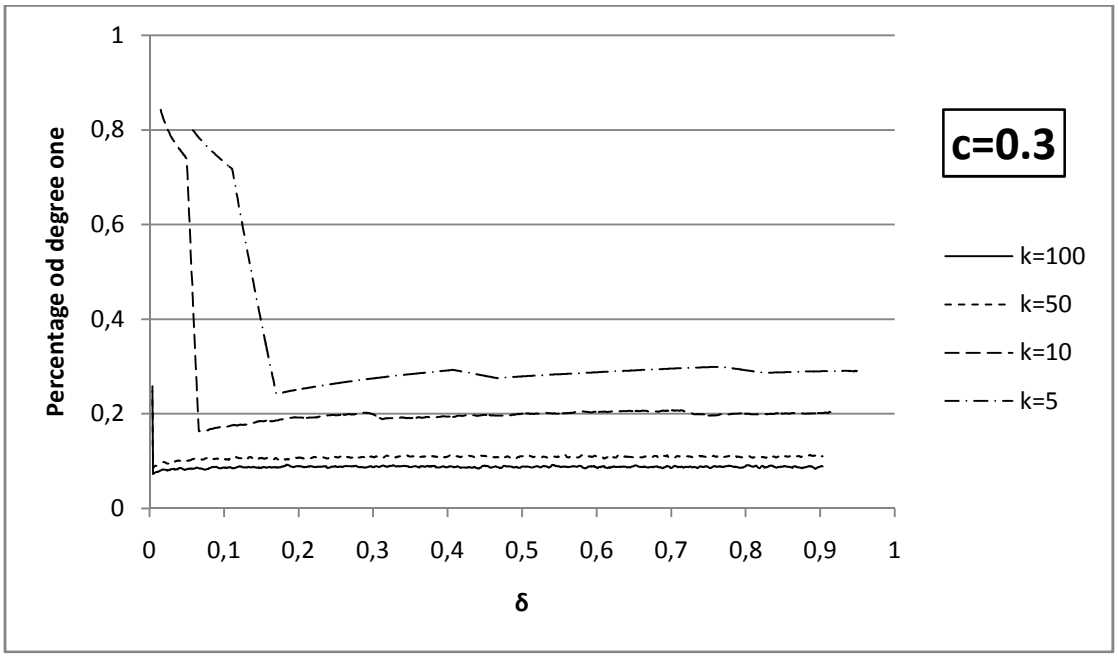
Όπως φαίνεται από τις Εικόνες 15 και 16, όσο μεγαλύτερο είναι το k τόσο μεγαλύτερο είναι και το εύρος των τιμών που μπορεί να πάρει το *c*. Όταν το k είναι πολύ μικρό (<10) τότε η μεταβολή του *c* αλλάζει ελάχιστα την μορφή της Robust Soliton Distribution.

#### ***2.3.3.2 - Αλλάζοντας το δ***

Για μεγάλα k (>1000) οι τιμές του *δ* δεν μπορούν να αλλάξουν σημαντικά την μορφή της Robust Soliton Distribution, ενώ για μικρά k μία μικρή αύξηση του *δ* μπορεί να αυξήσει λογαριθμικά το μέσο όρο των βαθμών, και αντιστοίχως να μειωθεί απότομα η εμφάνιση των βαθμών 1 [13]. Όπως φαίνεται στις εικόνες 17 και 18, η μεταβολή του *δ* (όπως και του *c*) είναι ποιο καθοριστική για k=50 και 100 από ότι για k=10 και 5

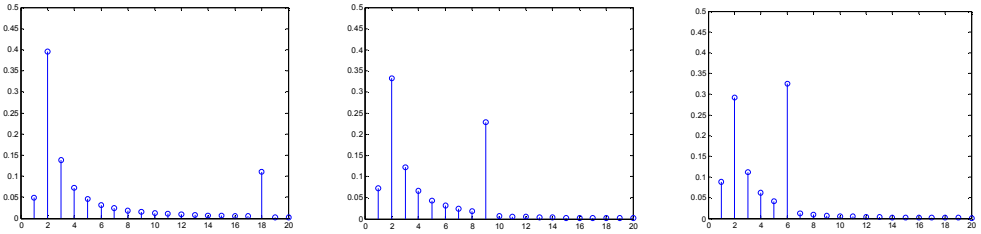



**Εικόνα 17:** Μεταβολή του μέσου όρου των βαθμών έναντι του *δ* για σταθερό c=0.3



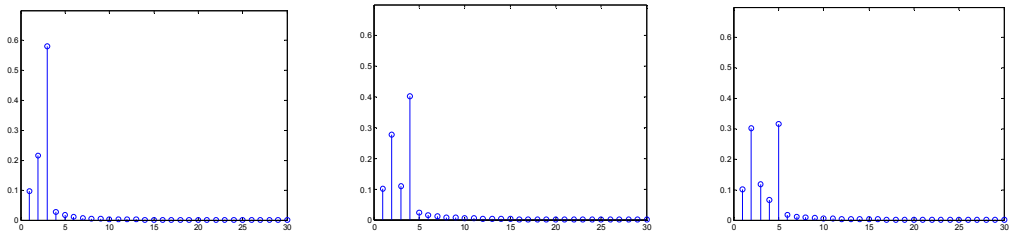

**Εικόνα 18:** Μεταβολή του ποσοστού εμφάνισης βαθμού 1 έναντι του *δ* για σταθερό c=0.3

Όπως φαίνεται στην Εικόνα 19, όταν αυξάνουμε το *c*, τότε η πρώτη ακίδα της Robust Soliton Distribution μειώνεται, ενώ η δεύτερη ακίδα αυξάνεται και έρχεται προς τα αριστερά, σε μικρότερους βαθμούς.



**Εικόνα 19:** Η Robust Soliton distribution με k=100, δ=0.4 και c=0.1, 0.2 και 0.3

Όπως φαίνεται στην Εικόνα 20, όταν αυξάνεται το *δ*, η πρώτη ακίδα της Robust Soliton Distribution αυξάνεται ενώ η δεύτερη μειώνεται.



**Εικόνα 20:** Η Robust Soliton Distribution για k=100, c=0.5 και δ=0.1, 0.5 και 0.9

**Πίνακας 1:** Χαρακτηριστικά της Robust Soliton Distribution [13]

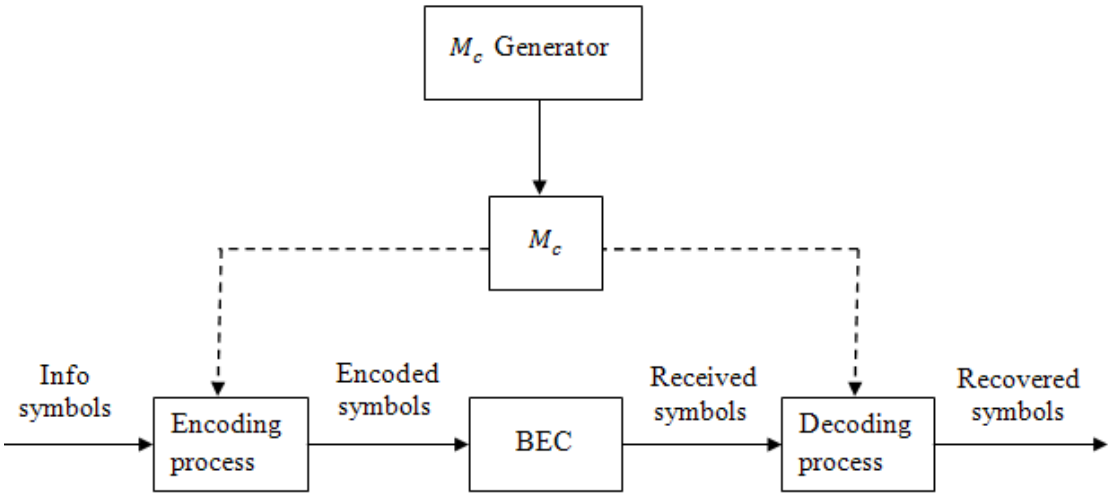
| **Χαρακτηριστικό** | Αύξηση του *c* | Αύξηση του *δ* |
|---|---|---|
| **Μέσω όρο βαθμού 1** | Μειώνεται | Αυξάνεται |
| **Πιθανότητα εμφάνισης βαθμού 1** | Αυξάνεται | Αυξάνεται (ελάχιστα) |
| **Η πρώτη ακίδα** | μειώνεται | Αυξάνεται |
| **Η δεύτερη ακίδα** | Αυξάνεται και μετακινείται προς τα αριστερά (σε μικρότερους βαθμούς) | Μειώνεται |

# Κεφάλαιο 3 - Προσομοίωση του LT κώδικα

Σε κεφάλαιο 3 γίνεται μία ανάλυση του LT κώδικα όπως πραγματοποιήθηκε στο περιβάλλον του Matlab. Γίνεται μία περιγραφή του διακριτού καναλιού διαγραφής (Binary Erasure Channel), και τέλος, παρουσιάζονται τα αποτελέσματα αυτής της προσομοίωσης.

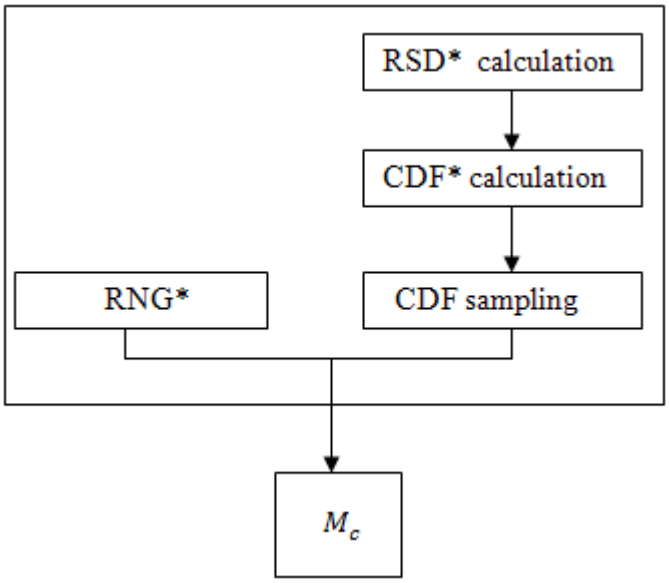
## 3.1 - Δομή του LT κώδικα στο Matlab

Η προσημείωση του LT κωδικοποιητή και αποκωδικοποιητή πραγματοποιήθηκε στο περιβάλλον του Matlab. Επιπλέον, έγινε τσεκάρισμα της λειτουργίας του συστήματος σε Binary Erasure Channel.




**Εικόνα 21:** Μπλοκ διάγραμμα του προγράμματος προσομοίωσης

Όπως φαίνεται και από το παραπάνω μπλοκ διάγραμμα του κώδικα της προσομοίωσης, ο κωδικοποιητής και ο αποκωδικοποιητής μοιράζονται τον ίδιο πίνακα Connectivity Matrix *Mc*. Ο *Mc* δίνει τους βαθμούς και τους δείκτες των γειτονικών που αντιστοιχούν σε κάθε κωδικοποιημένο-λαμβανόμενο σύμβολο.




**Εικόνα 22:** Μπλοκ διάγραμμα του $M_c$ Generator

Για την δημιουργία του *Mc*, πρώτα υπολογίζεται η Robust Soliton Distribution. Στην συνέχεια γίνεται ο υπολογισμός της Cumulative Distibution Function* της Robust Soliton Distribution*.

Αν n είναι ο αριθμός των κωδικοποιημένων συμβόλων που θα παραχθούν, τότε το πρόγραμμα παίρνει από την CDF n τυχαία δείγματα. Το κάθε ένα από τα n δείγματα που παίρνει από την CDF είναι ένας βαθμός, ο οποίος αντιστοιχεί σε ένα κωδικοποιημένο σύμβολο.

Το RNG (Random Number Generator* - γεννήτρια ψευδοτυχαίων αριθμών ) παράγει τόσους τυχαίους αριθμούς όσους λέει κάθε φορά ο βαθμός. Οι τυχαίοι αριθμοί που δίνει κάθε φορά το RNG αποτελούν τους δείκτες των γειτονικών που αντιστοιχούν σε κάθε κωδικοποιημένο σύμβολο.

Ο πίνακας *Mc* έχει k στήλες και n σειρές. Η κάθε σειρά αντιστοιχεί σε ένα κωδικοποιημένο σύμβολο. Η κάθε σειρά περιέχει τους δείκτες των γειτονικών που αντιστοιχούν στο εκάστοτε κωδικοποιημένο σύμβολο. Το σύνολο των μη μηδενικών στοιχείων που υπάρχουν σε κάθε γραμμή, ισούται με τον βαθμό που αντιστοιχεί σε κάθε κωδικοποιημένο σύμβολο.

Έστω πως k=3, n=4 και έστω ότι με κάποια τιμή του *c* και του *δ*, ο *Mc* generator δημιούργησε τον παρακάτω πίνακα.

| | | Info symbols | | |
|---|---|---|---|---|
| | | $I_1$ | $I_2$ | $I_3$ |
| Encoded symbols | $O_1$ | 1 | 0 | 0 |
| | $O_2$ | 3 | 1 | 0 |
| | $O_3$ | 2 | 1 | 3 |
| | $O_4$ | 2 | 0 | 0 |

**Εικόνα 23:** Παράδειγμα του πίνακα *Mc*

Βλέποντας αυτόν τον πίνακα βγάζουμε τα παρακάτω συμπεράσματα.

Η πρώτη γραμμή έχει ένα μη μηδενικό στοιχείο, άρα το πρώτο κωδικοποιημένο σύμβολο είναι βαθμού 1. Το μη μηδενικό στοιχείο είναι ο αριθμός 1, άρα το γειτονικό του πρώτου κωδικοποιημένου συμβόλου θα είναι το πρώτο σύμβολο πληροφορίας.

Η δεύτερη γραμμή έχει δύο μη μηδενικά στοιχεία, άρα το δεύτερο κωδικοποιημένο σύμβολο είναι βαθμού 2. Τα μη μηδενικό στοιχεία είναι οι αριθμοί 3 και 1, άρα τα γειτονικά του δεύτερου κωδικοποιημένου συμβόλου θα είναι το τρίτο και το πρώτο σύμβολο πληροφορίας.

Η τρίτη γραμμή έχει τρία μη μηδενικά στοιχεία, άρα το τρίτο κωδικοποιημένο σύμβολο είναι βαθμού 3. Τα μη μηδενικό στοιχεία είναι οι αριθμοί 2, 1 και 3, άρα τα γειτονικά του τρίτου κωδικοποιημένου συμβόλου θα είναι το δεύτερο, το πρώτο και το τρίτο σύμβολο πληροφορίας.

Η τέταρτη γραμμή έχει ένα μη μηδενικό στοιχείο, άρα το τέταρτο κωδικοποιημένο σύμβολο είναι βαθμού 1. Το μη μηδενικό στοιχείο είναι ο αριθμός 2, άρα το γειτονικό του τέταρτου κωδικοποιημένου συμβόλου θα είναι το δεύτερο σύμβολο πληροφορίας.

Στον κωδικοποιητή, είναι γνωστά τα σύμβολα πληροφορίας και ο πίνακας *Mc*. Η δουλεία του κωδικοποιητή είναι να παράγει τα κωδικοποιημένα σύμβολα.

Στον αποκωδικοποιητή, είναι γνωστά τα λαμβανόμενα σύμβολα και ο πίνακας *Mc*. Η δουλειά του αποκωδικοποιητή είναι να κάνει ανάκτηση τα k σύμβολα πληροφορίας.

Η διαδικασία κωδικοποίησης - αποκωδικοποίησης στην προσομοίωση του Matlab είναι ίδια με την θεωρητική περιγραφή που έγινε στις παραγράφους 2.1 και 2.2. Απλώς τώρα, αντί να γίνεται η

περιγραφή των συνδέσεων (των κωδικοποιημένων συμβόλων με τα σύμβολα πληροφορίας) με την βοήθεια του αραιού γραφήματος, γίνεται με την βοήθεια του πίνακα *Mc*.

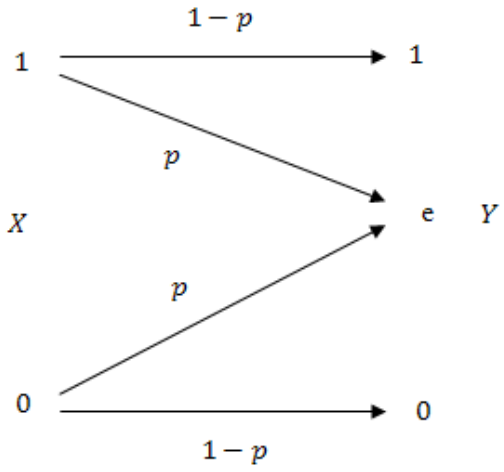
## 3.2 - Binary Erasure Channel (διακριτό κανάλι διαγραφής) [17]

Στην παράγραφο 3.3 γίνεται παρουσίαση των αποτελεσμάτων της προσομοίωσης του LT κώδικα, όπου δοκιμάζουμε την λειτουργία του LT κωδικοποιητή και αποκωδικοποιητή σε κανάλι BEC. Για αυτό το λόγο κάνουμε στην παράγραφο 3.2 μία περιγραφή των χαρακτηριστικών αυτού του καναλιού.

Το Binary Erasure Channel παρουσιάστηκε από τον Peter Elias το 1954. Το BEC είναι ένα κανάλι που χρησιμοποιείται ευρέως στην θεωρία της πληροφορίας γιατί είναι το απλούστερο κανάλι με θόρυβο για να αναλυθεί.

Στο BEC, ο πομπός μεταδίδει τις λογικές τιμές 1 ή 0, ενώ ο δέκτης λαμβάνει τις λογικές τιμές 0, 1 και *e*, όπου *e* είναι το διαγραμμένο σύμβολο. Αν το συνολικό αλφάβητο που στέλνει ο πομπός το πούμε X , και το αλφάβητο που δέχεται ο δέκτης το πούμε Y, και αν $p$ είναι η πιθανότητα να διαγραφεί ένα σύμβολο που στέλνει ο πομπός, τότε το BEC χαρακτηρίζεται από τις παρακάτω σχέσεις πιθανοτήτων.

$$P(Y=0|X=0)=1-p \qquad P(Y=e|X=0)=p$$
$$P(Y=0|X=1)=0 \qquad P(Y=1|X=0)=0$$
$$P(Y=e|X=1)=p \qquad P(Y=1|X=1)=1-p$$




.

**Εικόνα 24:** Binary Erasure Channel

Δηλαδή, αν ο δέκτης λάβει την λογική τιμή 0 ή 1, τότε είναι βέβαιο πως η τιμή που έλαβε είναι αυτή που έστειλε ο πομπός. Υπάρχει "σύγχυση" μόνο όταν ο δέκτης λάβει διαγραμμένο σύμβολο.

## 3.3 - Αποτελέσματα της προσομοίωσης

Σε αυτή την παράγραφο φαίνονται τα αποτελέσματα από την προσομοίωση του LT κώδικα σε BEC. Μελετάμε την απόδοση του LT κώδικα ως προς την πιθανότητα επιτυχίας της αποκωδικοποίησης (success rate) και τον αριθμό των λειτουργιών (operations) για διαφορετικές τιμές του k του $c$ του n και της πιθανότητας διαγραφής του καναλιού p(e).

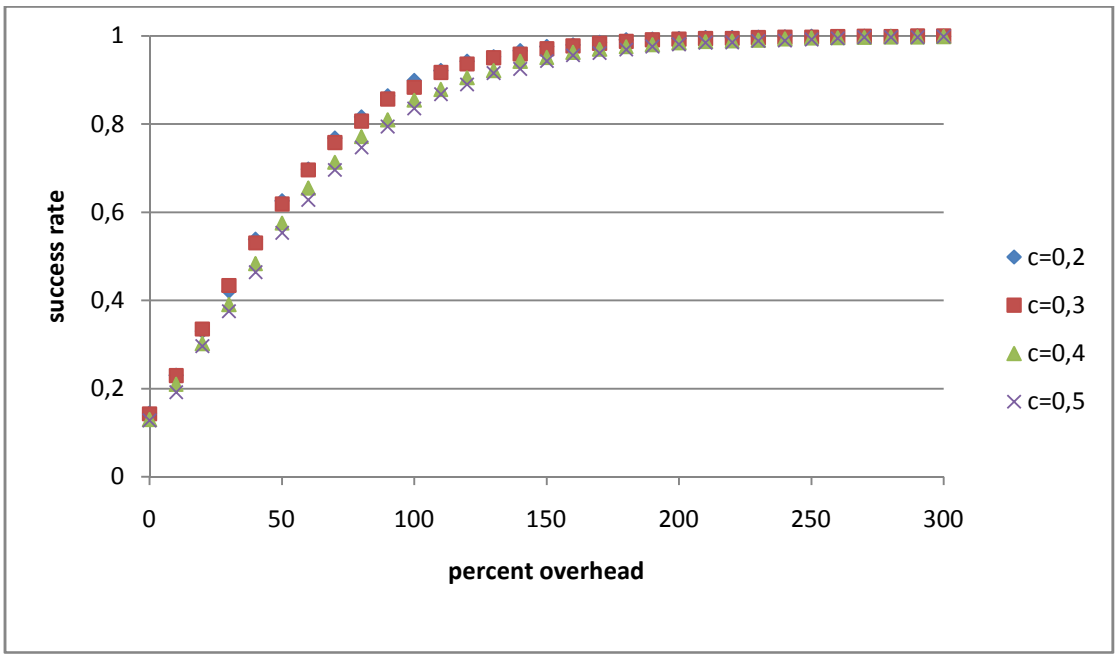
- Το επί τις εκατό overhead το υπολογίζουμε ως εξής:

$$\left(\frac{(\mu\acute{\epsilon}\sigma\omega\ \acute{o}\rho o\ \lambda\alpha\mu\beta\alpha\nu\acute{o}\mu\epsilon\nu\omega\nu\ \kappa\omega\delta\iota\kappa o\pi o\iota\eta\mu\acute{\epsilon}\nu\omega\ \sigma\upsilon\mu\beta\acute{o}\lambda\omega\nu) - k}{k}\right) \cdot 100\%$$

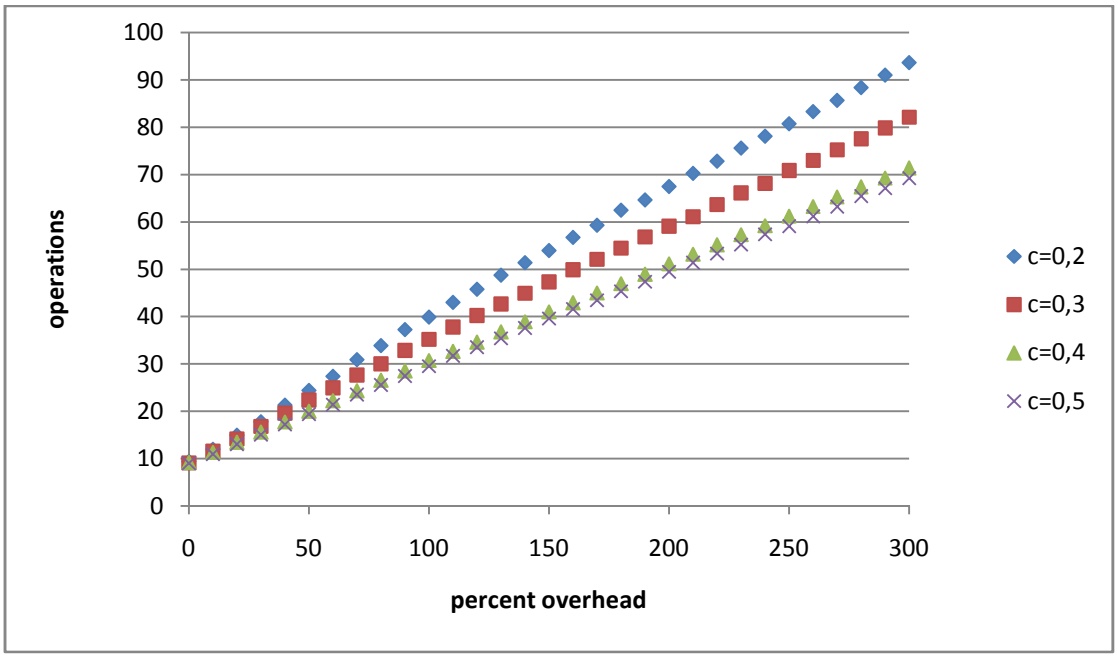
- Ως αριθμό των λειτουργιών (operations) θεωρούμε το μέσω όρο του πλήθος των πράξεων XOR που κάνει ο αποκωδικοποιητής. Με τον αριθμό των πράξεων συμβολίζεται η καθυστέρηση της αποκωδικοποίησης.

Στις εικόνες 25 και 26 φαίνονται τα αποτελέσματα για k=10, δ=0,5 και για μεταβολή του $c$ εντός των επιτρεπτών ορίων. Από την σχέση (6) το εύρος τιμών του $c$ για k=10 είναι από 0,1173 ως 0.52.

Στην Εικόνα 25 φαίνεται το ποσοστό επιτυχίας της αποκωδικοποίησης έναντι της επί τις εκατό overhead, ενώ στην Εικόνα 26 φαίνεται ο αριθμός των XOR πράξεων έναντι του επί τις εκατό overhead. Παρατηρούμε ότι για μικρό k χρειάζεται μεγάλο επί τις εκατό overhead ώστε να φτάσει η πιθανότητα επιτυχίας στο 1. Επιπλέον για διαφορετικά $c$ μεταβάλλεται ελάχιστα η πιθανότητα επιτυχίας της αποκωδικοποίησης, ενώ από την άλλη φαίνεται μία ποιο ξεκάθαρη αλλαγή στον αριθμό τον λειτουργιών. Όπως παρατηρήθηκε στο άρθρο [13], για τόσο μικρά k θα ήταν προτιμότερο να επιλέγουμε $c$ κοντά στο ψηλό όριο, καθώς η πιθανότητα επιτυχίας της αποκωδικοποίησης μειώνεται ελάχιστα ενώ ο αριθμός των λειτουργιών μειώνεται σημαντικά.



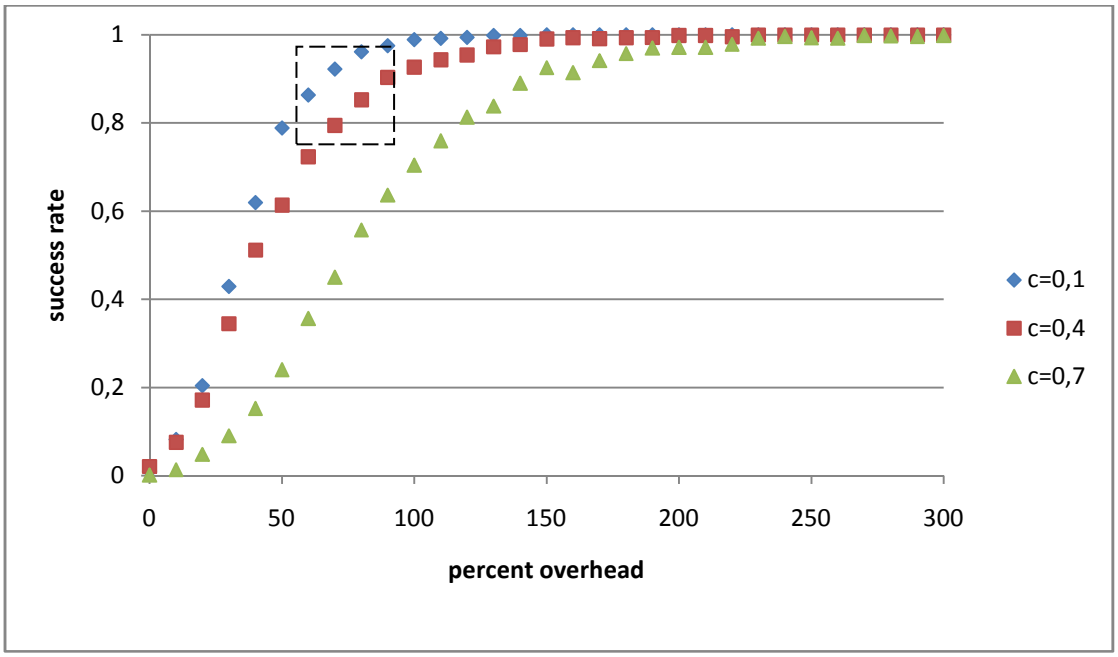

**Εικόνα 25:** Πιθανότητα επιτυχίας της αποκωδικοποίησης, έναντι του επί τις εκατό overhead για k=10, n=20:2:80 δ=0.5 και πιθανότητα διαγραφής p(e)=0.5



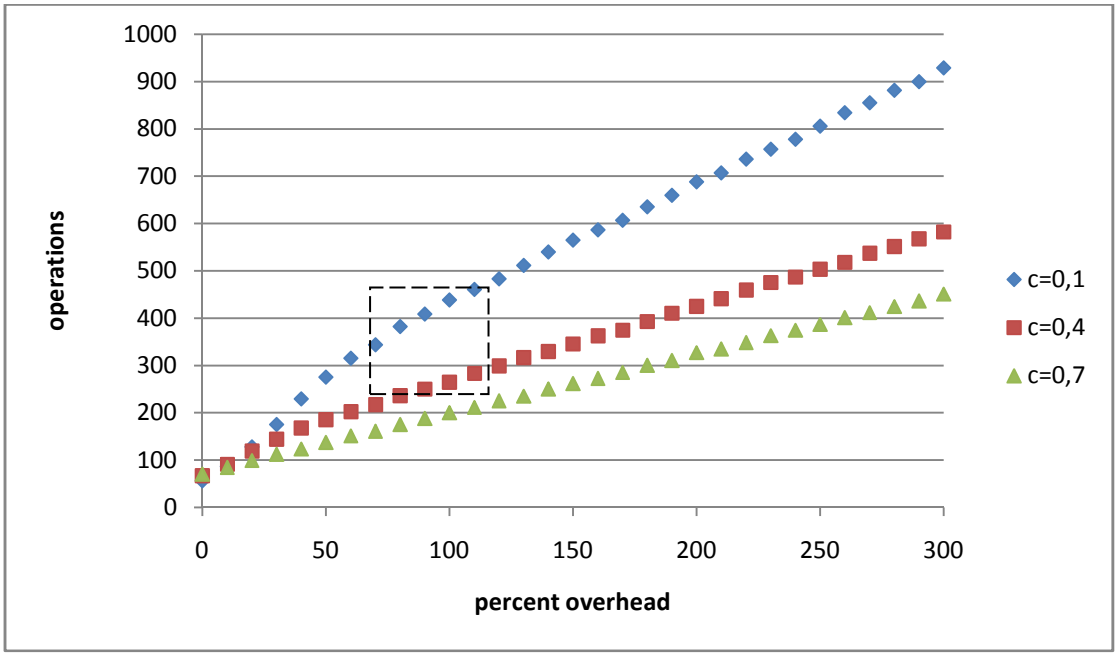

**Εικόνα 26:** Πλήθος λειτουργιών του αποκωδικοποιητή, έναντι του επί τις εκατό overhead για k=10, n=20:2:80 δ=0.5 και πιθανότητα διαγραφής p(e)=0.5

Στις Εικόνες 27 και 28, έχει γίνει η προσομοίωση για k=50, δ=05 και για κάποιες τιμές του *c* εντός των ορίων 0,0313 ως 0,7677. Τώρα που αυξήθηκε το k, έχει βελτιωθεί η πιθανότητα επιτυχίας της αποκωδικοποίησης έναντι του επί τις εκατό overhead και οι μεταβολές του *c* έχουν πλέον μεγαλύτερες επιπτώσεις. Όπως παρατηρήθηκε στο άρθρο [13], αν σε κάποια εφαρμογή μπορούσαμε

να αποδεχτούμε λίγο μειωμένη πιθανότητα επιτυχίας αποκωδικοποίησης για να μειώσουμε όμως σημαντικά το πλήθος των πράξεων θα μπορούσαμε να βάλουμε c=0,4 αντί c=0.1. (κοιτάξτε το ορθογώνιο στα Εικόνες 27 και 28).

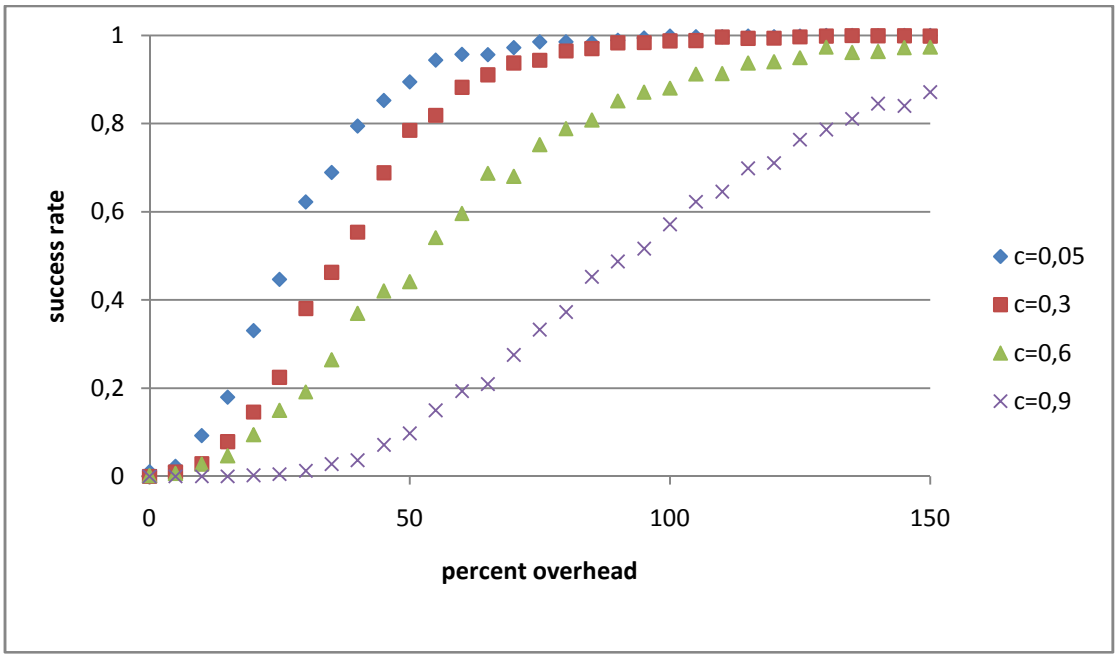



**Εικόνα 27:** Πιθανότητα επιτυχίας της αποκωδικοποίησης, έναντι του επί τις εκατό overhead για k=50, n=100:10:400 δ=0.5 και πιθανότητα διαγραφής p(e)=0.5

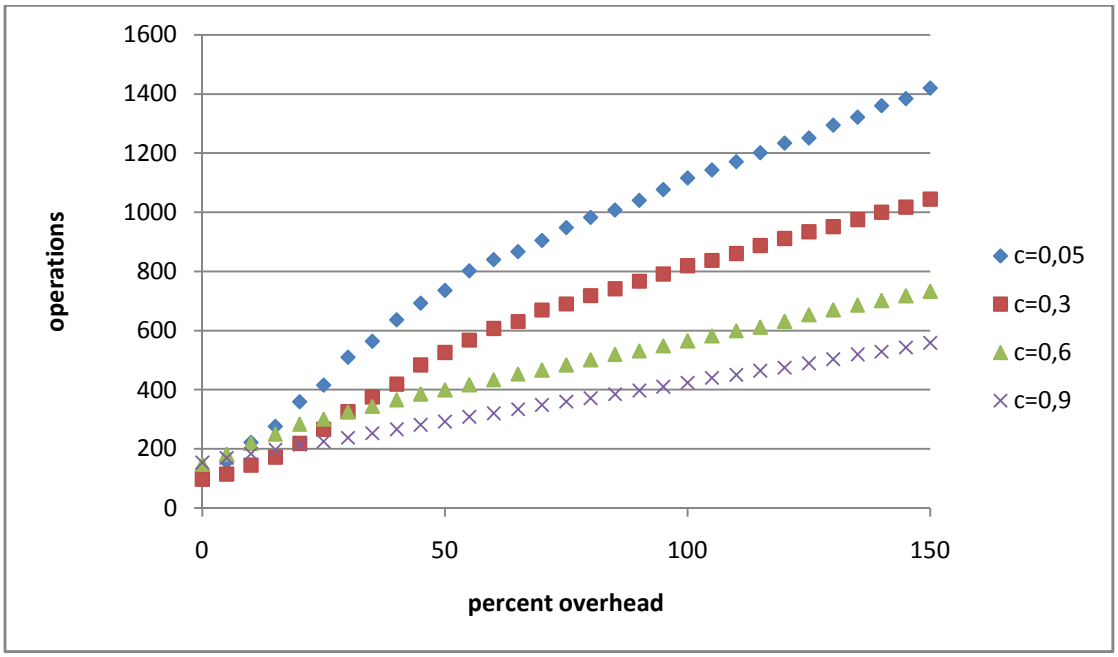



**Εικόνα 28:** Πλήθος των λειτουργιών του αποκωδικοποιητή, έναντι του επί τις εκατό overhead για k=50, n=100:10:400 δ=0.5 και πιθανότητα διαγραφής p(e)=0.5

Για k=100 βελτιώθηκε κι άλλο η απόδοση του κώδικα και οι μεταβολές του *c* γίνανε ακόμα πιο καθοριστικές. Για k=100 το εύρος των τιμών του *c* είναι από 0,0191 ως 0,9437.




**Εικόνα 29:** Πιθανότητα επιτυχίας της αποκωδικοποίησης, έναντι του επί τις εκατό overhead για k=100, n=200:10:500 δ=0.5 και πιθανότητα διαγραφής p(e)=0.5

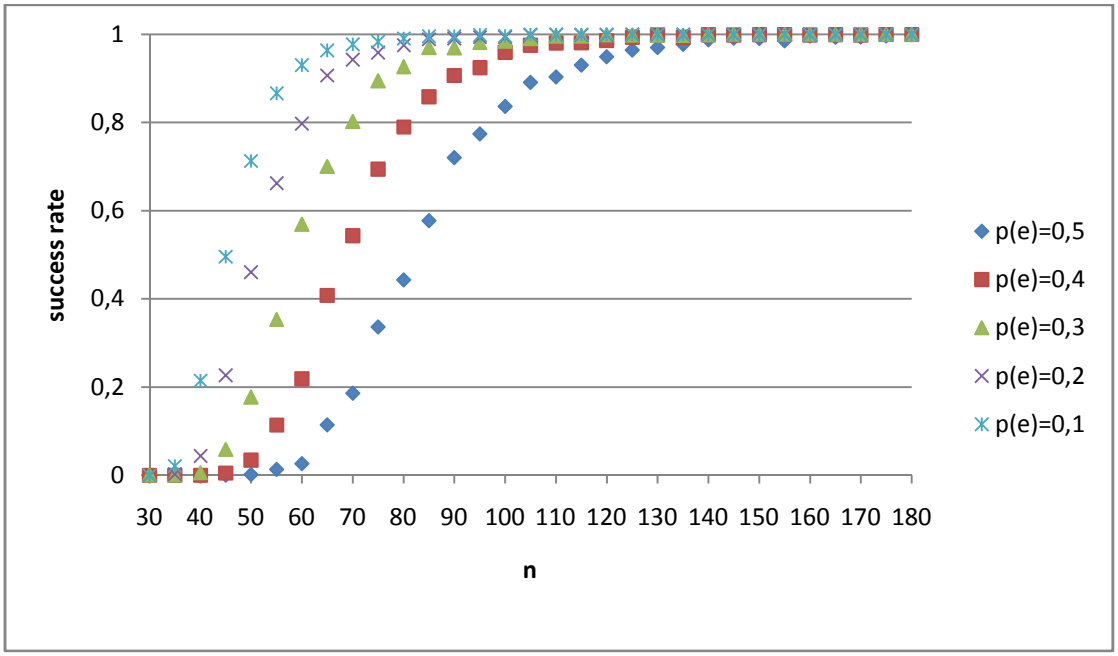



**Εικόνα 30:** Πλήθος των λειτουργιών του αποκωδικοποιητή, έναντι του επί τις εκατό overhead k=100, n=200:10:500 δ=0.5 και πιθανότητα διαγραφής p(e)=0.5

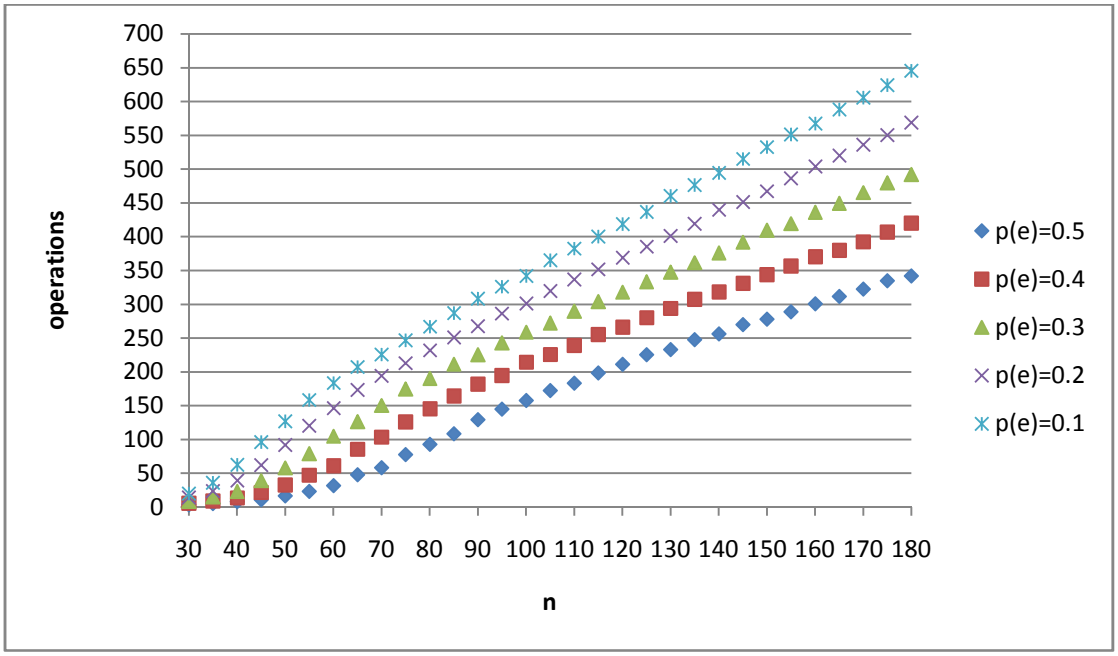
Όταν το *c* πλησιάζει στο κάτω όριο αυξάνεται η πιθανότητα επιτυχίας αποκωδικοποίησης αλλά αυξάνονται και οι πράξεις. Όπως φαίνεται το *c* έχει μεγαλύτερη επίδραση στην πιθανότητα επιτυχίας και στις πράξεις όσο το k αυξάνεται. Ακόμα, παρατηρούμε ότι η απόδοση του LT κώδικα βελτιώνεται καθώς αυξάνουμε το k

Για όλα τα k το πλήθος των πράξεων αυξάνεται γραμμικά καθώς αυξάνουμε το επί τις εκατό overhead, αυτό είναι χαρακτηριστικό της belief propagetion αποκωδικοποίησης [5].

Στις εικόνες 31 και 32 βλέπουμε την απόδοση του κώδικα LT για διάφορες πιθανότητες διαγραφής p(e) του καναλιού. Όπως είναι φυσικό, για μικρότερη πιθανότητα διαγραφής p(e) απαιτούνται να παραχθούν λιγότερα κωδικοποιημένα σύμβολα ώστε να έχουμε μεγάλες πιθανότητες πετυχημένης αποκωδικοποίησης. Στην εικόνα 32 βλέπουμε πως όσο αυξάνεται η πιθανότητα διαγραφής p(e) του καναλιού, μειώνεται το μέσω όρο των λειτουργιών του αποκωδικοποιητή. Αυτό συμβαίνει γιατί καθώς αυξάνεται το p(e), μειώνονται οι πιθανότητες επιτυχούς αποκωδικοποίησης, άρα αυξάνονται οι περιπτώσεις που η αποκωδικοποίηση διακόπτεται πριν την ολοκλήρωση της, και για αυτό μειώνεται και το μέσω όρο τον λειτουργιών που κάνει ο αποκωδικοποιητής.




**Εικόνα 31:** Πιθανότητα επιτυχίας της αποκωδικοποίησης, έναντι του επί τις εκατό overhead για k=30, n=30:5:180, δ=0.5 και c=0.05 για διάφορες πιθανότητες διαγραφής p(e)




**Εικόνα 32:** Πλήθος των λειτουργιών του αποκωδικοποιητή, έναντι του επί τις εκατό overhead για k=30, n=30:5:180, δ=0.5 και c=0.05 για διάφορες πιθανότητες διαγραφής p(e)

# Κεφάλαιο 4 - Γενικά για την HW εφαρμογή

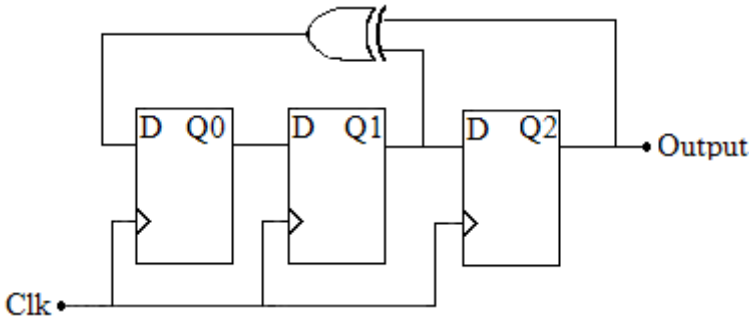
Στο κεφάλαιο 5, κατά την ανάλυση του LT κωδικοποιητή και αποκωδικοποιητή σε FPGA, θα γίνεται αναφορά στα παρακάτω στοιχεία

- LFSR Fibonacci
- Δειγματοληψία από κατανομή
- ROM
- Block RAM και Distributed RAM του FPGA

Για αυτό λόγο, στο κεφάλαιο 4 γίνεται μία περιγραφή και ανάλυση αυτών.

## 4.1 - Linear Feedback shift Register [18][19]

Linear feedback shift register είναι ένας shift register στον οποίο το bit εισόδου εξαρτάται από την προηγούμενη κατάστασή του. Σε έναν LFSR το bit εισόδου συνήθως προκύπτει μετά από πράξεις XOR μεταξύ κάποιων bits που βρίσκονται εντός του shift register. Η αρχική κατάσταση του LFSR ονομάζεται σπόρος (seed). Όλες οι μελλοντικές καταστάσεις που θα πάρει ο LFSR καθορίζονται από την παρούσα κατάστασή του. Επειδή είναι πεπερασμένος ο αριθμός τον καταστάσεων που μπορεί να πάρει ο LFSR, κάποια στιγμή αρχίζει να επαναλαμβάνεται. Αν γίνει καλός σχεδιασμός της ανάδρασης (feedback) τότε η περίοδος του LFSR γίνεται πολύ μεγάλη και η ακολουθία που παράγει μπορεί να χρησιμοποιηθεί ως γεννήτρια ψευδοτυχαίων αριθμών (RNG).




**Εικόνα 33:** 3-bit Fibonacci

Οι έξοδοι των Flip Flops που έχουν επιλεχθεί για να λειτουργούν ως ανάδραση και επηρεάζουν το επόμενο bit εισόδου ονομάζονται taps (στην συγκεκριμένη περίπτωση είναι το Q1). Γίνεται η πράξη XOR μεταξύ των taps και του bit εξόδου του shift register και το bit του αποτελέσματος γίνεται το επόμενο bit εισόδου (input bit) του shift register.

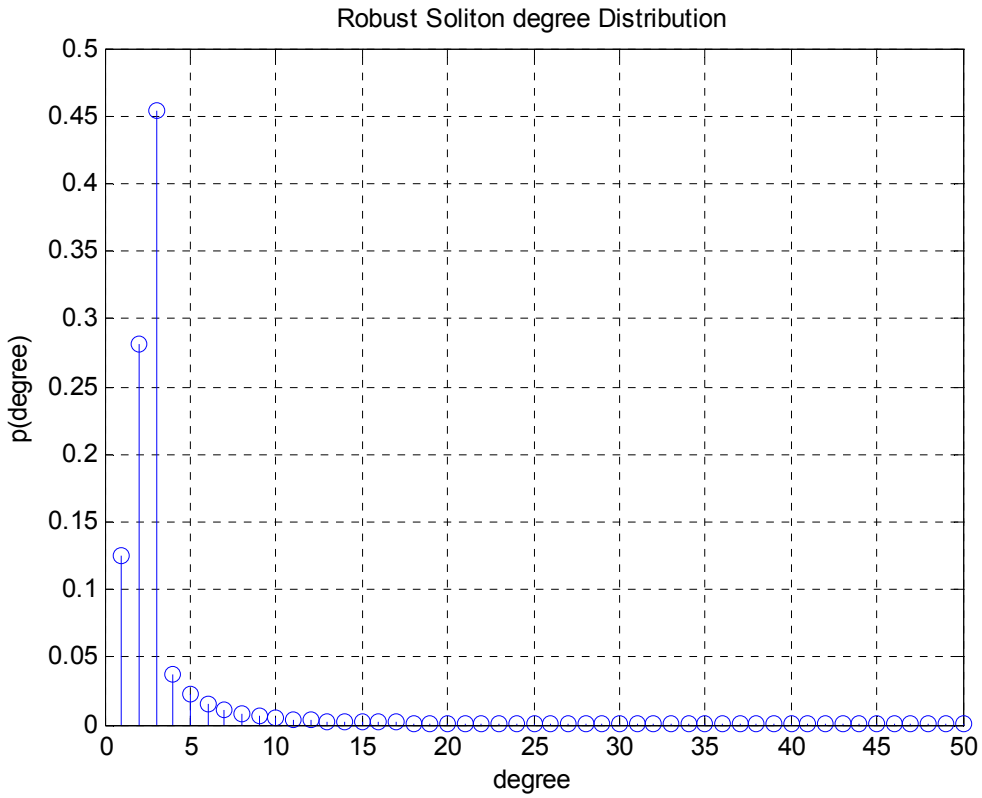
**Πίνακας 2:** Ακολουθία 3-bit Fibonacci για Seed 111

| Q2 | Q1 | Q0 |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

Αν n είναι ο αριθμός των Flip Flops που απαρτίζουν τον Fibonacci, τότε ο Fibonacci μπορεί να πάρει έως $2^n - 1$ καταστάσεις. Εξαιρείται η περίπτωση όλα τα Flip Flops να έχουν έξοδο 0. Επίσης, δεν μπορούμε να δώσουμε μόνο μηδενικά για σπόρο (seed) διότι τότε ο Fibonacci θα μείνει μόνιμα σε αυτήν την κατάσταση.

## 4.2 - Δειγματοληψία της Robust Soliton Degree Distribution

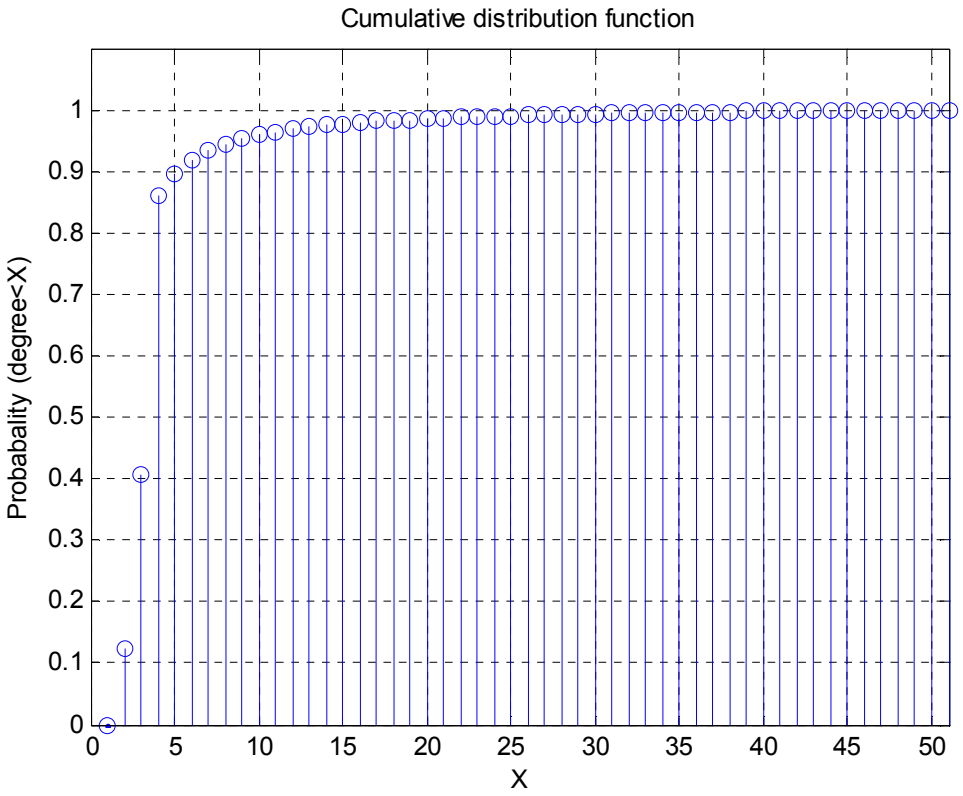
Αρχικά γίνεται υπολογισμός την Probability Robust Soliton Distribution στο Matlab με την βοήθεια των σχέσεων που φαίνονται στην παράγραφο 2.3.2.




**Εικόνα 34:** Probability distribution function της Robust Soliton Distribution για c=0.4 δ=0,5 k=50

Στον οριζόντιο άξονα της Εικόνας 34 υπάρχουν όλες οι τιμές που μπορεί να πάρει ο βαθμός. Εξαιρείται η τιμή μηδέν. Στον κάθετο άξονα βλέπουμε ποιες είναι οι πιθανότητες εμφάνισης που έχει ο κάθε ένας από τους βαθμούς.

Στην συνέχεια γίνεται υπολογισμός της Cumulative distribution function της Robust Soliton Distribution στο Matlab.




**Εικόνα 35:** Cumulative distribution function της Robust Soliton distribution για c=0.4 δ=0,5 k=50

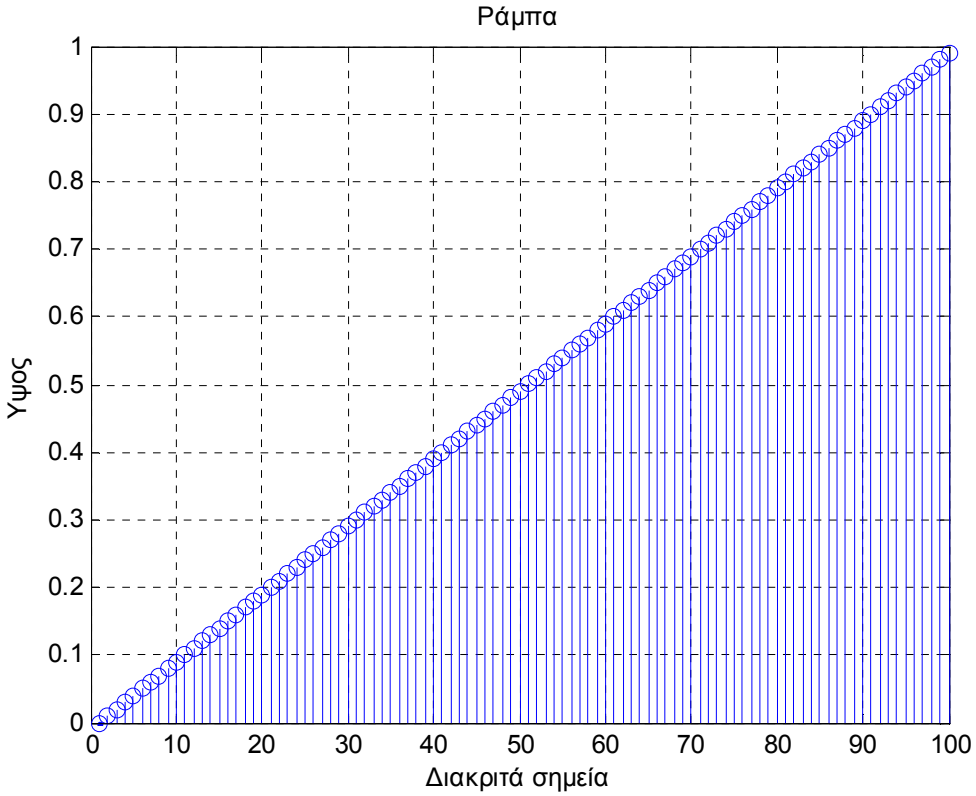
Η Cumulative Distribution διαβάζεται με τον εξής τρόπο:

Η πιθανότητα ο βαθμός που θα εμφανιστεί να είναι μικρότερος από 1 είναι 0%
Η πιθανότητα ο βαθμός που θα εμφανιστεί να είναι μικρότερος από 2 είναι περίπου 12%
Η πιθανότητα ο βαθμός που θα εμφανιστεί να είναι μικρότερο από 3 είναι περίπου 41%
...
...
κ.ο.κ.

Ο κάθετος άξονας της Cumulative distribution παίρνει τιμές από 0 ως 1. Αφού πρώτα αποφασίσουμε πόσα δείγματα θέλουμε να πάρουμε, στην συνέχεια διαιρούμε τον αριθμό 1 με τον αριθμό δειγμάτων που θέλουμε. Έστω ότι θέλουμε να πάρουμε 100 δείγματα, τότε:

$$1/100 = 0{,}1$$

Φτιάχνουμε μία ράμπα με 100 διακριτά σημεία όπου το κάθε διακριτό σημείο έχει ύψος 0.1 μονάδες μεγαλύτερο από το προηγούμενο του.

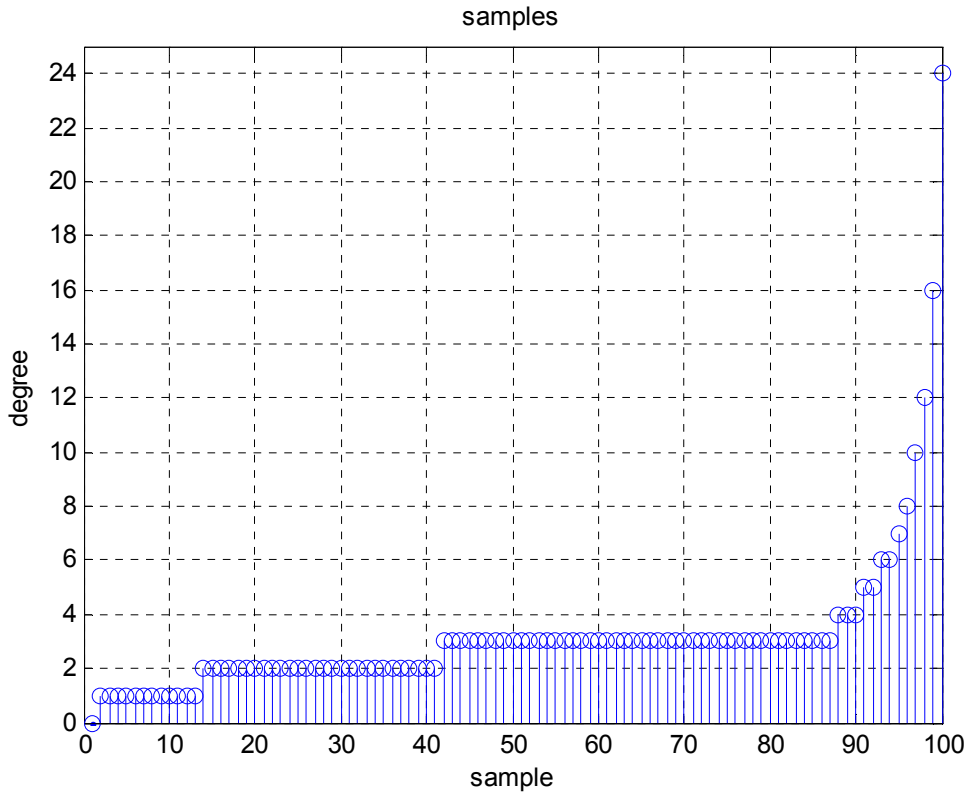



**Εικόνα 36:** 100 τιμές, μεταξύ 0 και 1, στοιχισμένες ως μία ράμπα.

Συγκρίνουμε τα στοιχεία της ράμπας με τα στοιχεία της Cummulative distribution function με τη βοήθεια της παρακάτω διαδικασίας.

```
for j=1:number_of_samples
  for i = 1:k
    if (rampa(j) > CDF(i)) && (rampa(j)<=CDF(i+1))
       samples(1,j) = i;
    end
  end
end
```

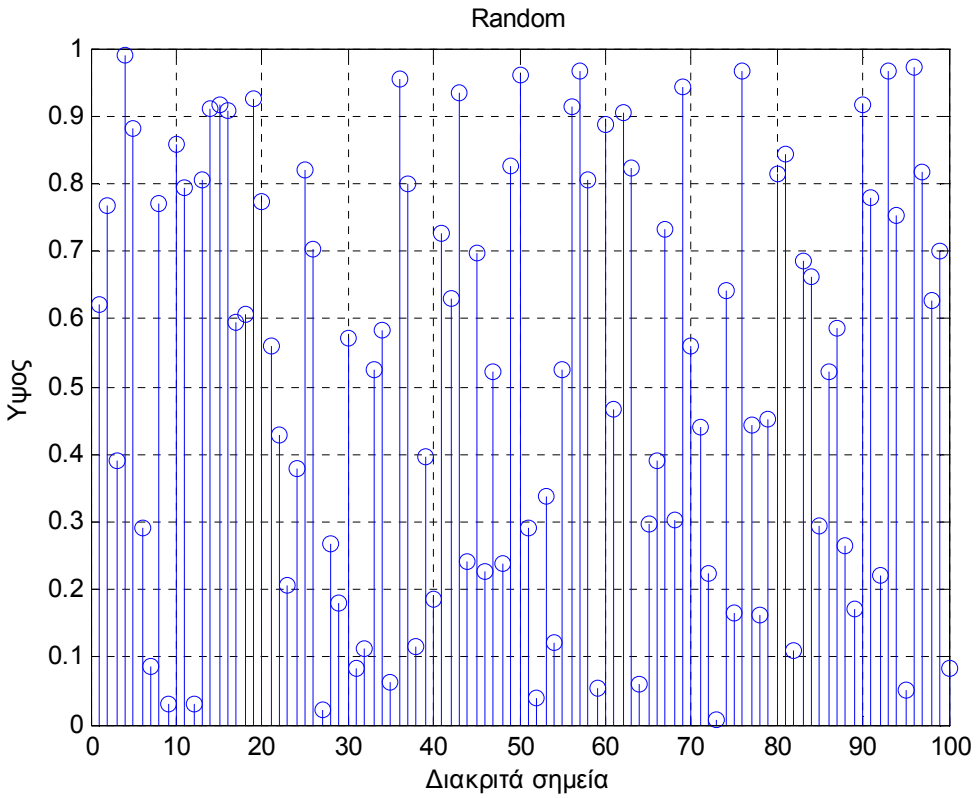
Και παίρνουμε τα δείγματα της Robust Soliton Distribution.




**Εικόνα 37:** Τα δείγματα της RSD, όπως θα τα αποθηκεύαμε στη ROM, αν τα παίρναμε με την βοήθεια της ράμπας.

Αυτά τα δείγματα τα φορτώνουμε σε μία ROM του FPGA. Διαβάζοντας τα στοιχεία της ROM με ψευδοτυχαίο τρόπο, παίρνουμε τους βαθμούς σύμφωνα με την Robust Soliton Distribution.

Για να γίνει αυτό όμως, θα πρέπει να είναι πολύ καλή η ψευδοτυχαία μηχανή που χρησιμοποιούμε στο FPGA. Ο LFSR Fibonacci, όπως φάνηκε μετά από δοκιμές που γίνανε, δεν μπορούσε να διαβάσει *αρκετά* τυχαία τα δείγματα, και αυτό ήταν μεγάλο πρόβλημα, γιατί ο LT κώδικας είναι εξαρτημένος από τις δύο RNG μηχανές που χρησιμοποιεί για να παράγει τους βαθμούς των κωδικοποιημένων συμβόλων και τους δείκτες των γειτονικών συμβόλων. Οπότε, αφού η RNG μηχανή που επιλέξαμε δεν ήταν αρκετά καλή, κάναμε το εξής. Αντί να φτιάξουμε την παραπάνω ράμπα, παίρνουμε με την βοήθεια μίας ψευδοτυχαίας εντολής του Matlab 100 διακριτά σημεία με τυχαίες τιμές μεταξύ 0 και 1.

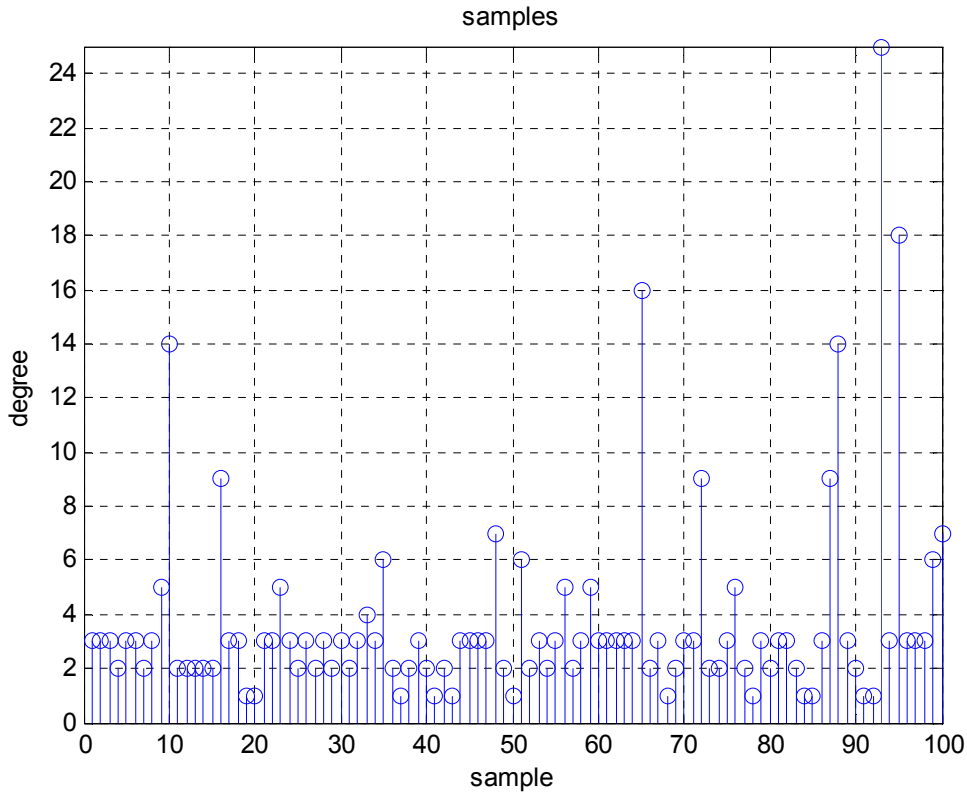



**Εικόνα 38:** 100 ψευδοτυχαίες τιμές, μεταξύ 0 και 1, όπως τις δίνει η εντολή random της Matlab.

Συγκρίνουμε τα στοιχεία που πήραμε από την ψευδοτυχαία εντολή του Matlab με τα στοιχεία της Cummulative distribution function με την βοήθεια της παρακάτω διαδικασία.

```
for j=1:number_of_samples
    for i = 1:k
     if (random(j) > CDF(i)) && (random(j)<=CDF(i+1))
        samples(1,j) = i;
     end
    end
end
```

Και παίρνουμε τα δείγματα της Robust Soliton Distribution.



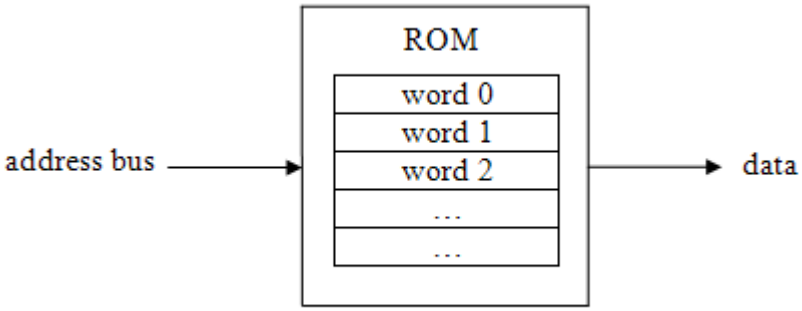

**Εικόνα 39:** Τα δείγματα της RSD, όπως θα τα αποθηκεύαμε στην ROM, αν τα παίρναμε με την βοήθεια της random εντολής του Matlab.

Τώρα τα δείγματα τα βάζουμε εξαρχής ανακατεμένα στην ROM του FPGA, οπότε το έργο του Fibonacci γίνεται πολύ πιο εύκολο.

## 4.3 - ROM ( Read-only memory) [20]

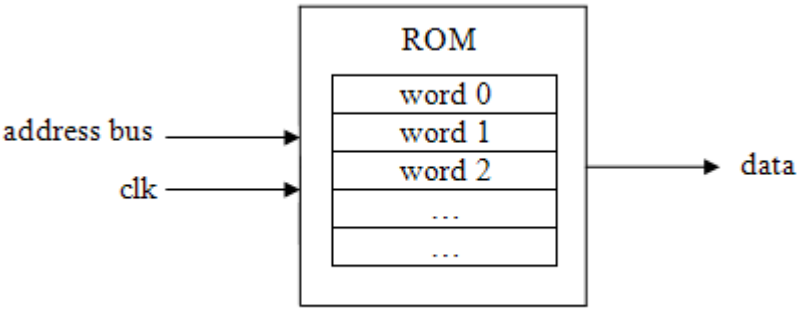
Η ROM όπως φαίνεται και από τη ονομασία της είναι μνήμη από την οποία μπορούμε μόνο να διαβάσουμε τα περιεχόμενα των στοιχείων της. Για αυτό και στο Xilinx τα περιεχόμενα της ROM τα φορτώνουμε χειροκίνητα κατά τον σχεδιασμό του κώδικα. Κατά την διάρκεια που τρέχει ο κώδικας μπορούμε μόνο να διαβάσουμε τα περιεχόμενά της.

Παρακάτω φαίνεται η γενική μορφή μίας ασύγχρονης και μίας σύγχρονης ROM.




**Εικόνα 40:** Ασύγχρονη ROM

Στην ασύγχρονη ROM, με το addres bus επιλέγουμε κάποιο στοιχείο της ROM, και αμέσως το περιεχόμενο αυτού του στοιχείο εμφανίζεται στο data bus.




**Εικόνα 41:** Σύγχρονη ROM

Στην σύγχρονη ROM θα πρέπει να αλλάξει κατάσταση το clock, για να εμφανιστεί στο data bus το περιεχόμενο του στοιχείου που επιλέξαμε με το address bus.

## 4.4 - Block RAM και Distributed RAM του FPGA [21]

Τις RAM (Random Acces Memory) τις χρησιμοποιούμε για προσωρινή αποθήκευση δεδομένων. Υπάρχουν δύο ειδών RAM που μπορούμε να χρησιμοποιήσουμε στα FPGA.

Η distributed RAM ονομάζονται έτσι γιατί είναι κατανεμημένες σε όλο το FPGA. Για να δημιουργηθεί μία distributed RAM θα πρέπει να χρησιμοποιηθούν πολλά διαφορετικά LUT's (Lookup tables ). Τα LUT's τα χρησιμοποιεί το FPGA για να δημιουργήσει και άλλα στοιχεία όπως

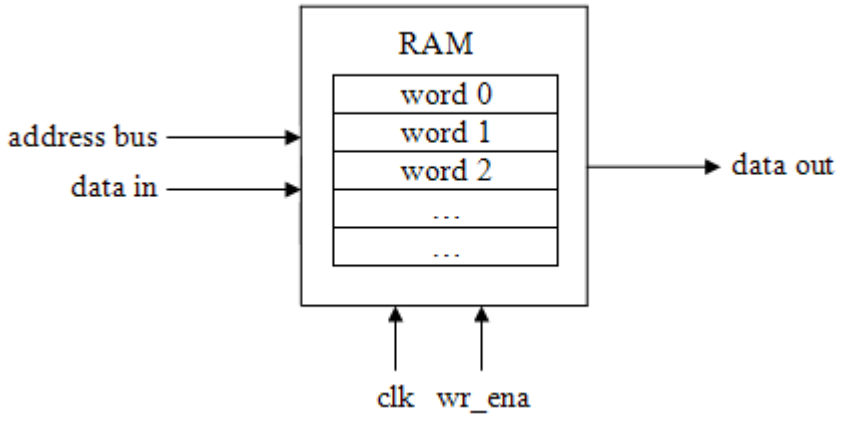
πχ. ROM's και shift register's. Για τους παραπάνω λόγους όταν θέλουμε να χρησιμοποιήσουμε μεγάλες RAM τότε οι distributed ram δεν είναι καλή λύση γιατί καθυστερεί πολύ ο σχεδιασμός τους.

Στην distributed RAM το γράψιμο γίνεται σύγχρονα και το διάβασμα ασύγχρονα. Πιο συγκεκριμένα για να γράψουμε κάτι στο στοιχείο που έχουμε επιλέξει με το addres, θα πρέπει να ενεργοποιηθεί το write enable και επιπλέον να έρθει το μέτωπο του clk. Το διάβασμα γίνεται ασύγχρονα, δηλαδή στο data out παίρνουμε πάντα την τιμή που έχει το στοιχείο που έχουμε επιλέξει με το address, ανεξάρτητα από το write enable και από το clock.

Οι block RAM's είναι τοποθετημένες σε συγκεκριμένα σημεία του FPGA και είναι αφιερωμένες στο να λειτουργούν ως RAM. Δεν μπορεί το FPGA να χρησιμοποιήσει τα αποθέματα των block RAM's για να δημιουργήσει άλλα στοιχεία. Οι block RAM's είναι ιδανικές για αν θέλουμε να χρησιμοποιήσουμε μεγάλες RAM καθώς ο χρόνος σχεδιασμού τους είναι ελάχιστος.

Στην block RAM το διάβαζα και το γράψιμο γίνεται σύγχρονα. Για να γράψουμε στην block RAM κάτι στο στοιχείο που έχουμε επιλέξει με το addres, θα πρέπει το write enable να είναι ενεργοποιημένο και επιπλέον να έρθει μέτωπο του clock. Για να διαβάσουμε το περιεχόμενοι του στοιχείου που επιλέξαμε με το address, θα πρέπει  το write enable να είναι απενεργοποιημένο και επιπλέον να έρθει μέτωπο του clock.

Παρακάτω φαίνεται ένα γενικό σχεδιάγραμμα μίας Dual port RAM. Δηλαδή μίας RAM που έχει άλλο bus για τα data in και άλλο bus για τα data out.

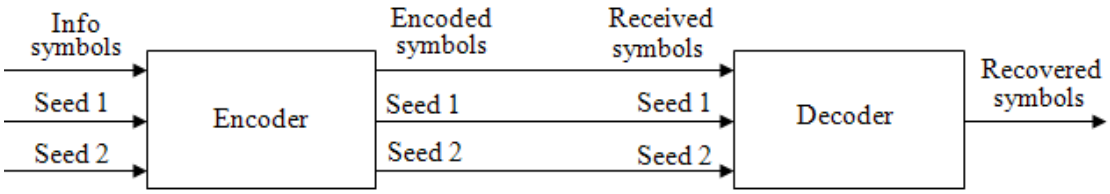



**Εικόνα 42:** Γενική μορφή της RAM

# Κεφάλαιο 5 - HW εφαρμογή του LT κώδικα

Για την δομή της Hardware εφαρμογής χρησιμοποιήθηκαν ιδέες από τις εργασίες [22] και [8]. Στις ενότητες 5.1 και 5.2 γίνεται ανάλυση των λειτουργιών του κωδικοποιητή και αποκωδικοποιητή αντίστοιχα.

Παρακάτω φαίνεται ένα γενικό μπλοκ διάγραμμα του κωδικοποιητή και του αποκωδικοποιητή, όπως εφαρμόστηκε στο FPGA.

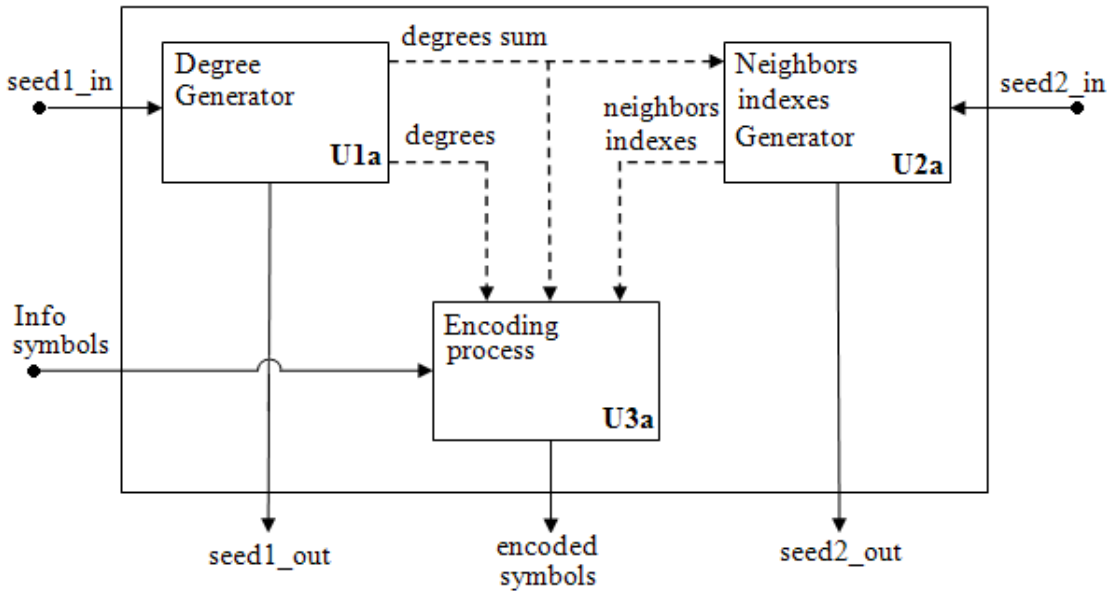



**Εικόνα 43:** Μπλοκ διάγραμμα του LT κωδικοποιητή - αποκωδικοποιητή στο FPGA

## 5.1 - Hardware εφαρμογή του LT κωδικοποιητή

Ο κωδικοποιητής αποτελείτε από τρία Units:

- U1a. Degree Generator
- U2a. Neighbors indexies Generator
- U3a. Encoding Process



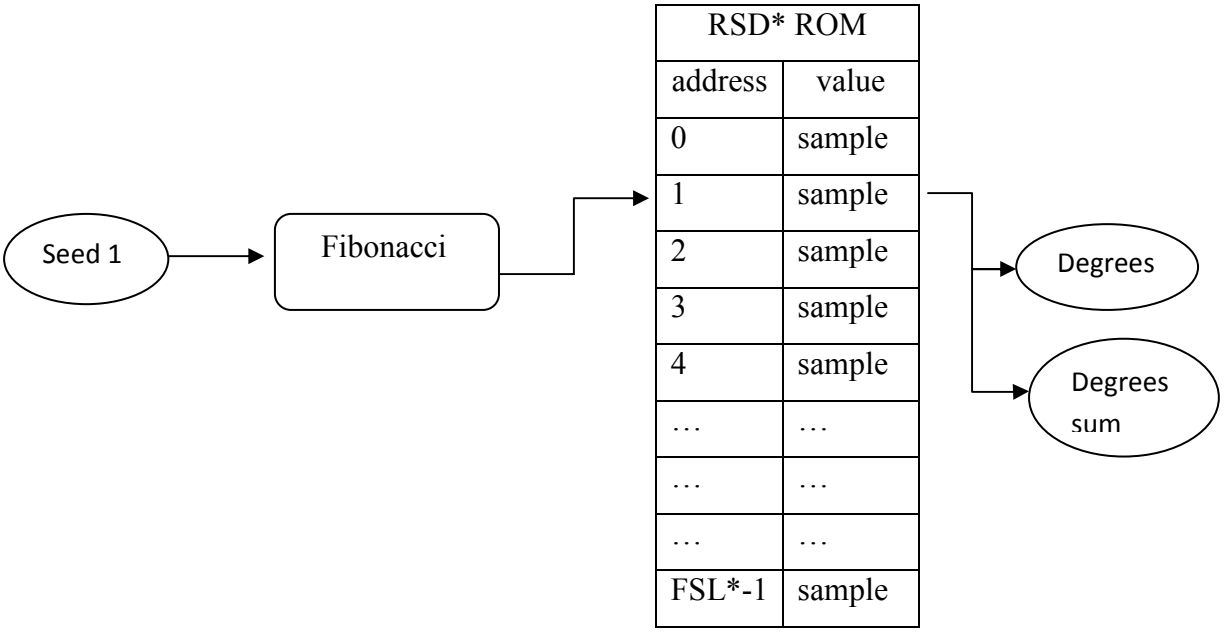

**Εικόνα 44:** Εσωτερικές λειτουργίες του κωδικοποιητή

### *5.1.1 - U1a. Degree generator*

Στην VHDL δεν μπορούμε να κάνουμε τις πολύπλοκες μαθηματικές πράξεις που χρειάζονται για την δημιουργία της Robust Soliton Distribution

Στο Degree Generator υπάρχει μία ROM (με το όνομα RSD* ROM), στην οποία έχουμε τοποθετήσει δείγματα από την Robust Soliton Distribution. Ο αριθμός τον δειγμάτων που τοποθετήσαμε στην ROM είναι περίπου $10 \cdot n$, χωρίς να κάναμε χρήση κάποιας αποδεδειγμένης μαθηματικής σχέσης. Ο αριθμός των στοιχείων που βάζουμε σε αυτή τη ROM δεν μπορεί να είναι ένας οποιοσδήποτε ακέραιος, αλλά εξαρτάται από το πλήθος των τιμών που μπορεί να μετρήσει ο Fibonacci. Το πλήθος τιμών που μπορεί να δώσει ο Fibonacci εξαρτάται από το μήκος του Fibonacci που επιλέγουμε κάθε φορά.

Σε αυτή τη ROM χρησιμοποιούμε έναν Fibonacci ως address bus. Έχουμε καθορίσει εξαρχής ποιος θα είναι ο αριθμός n των κωδικοποιημένων συμβόλων που θέλουμε να παράγουμε ώστε ο Fibonacci να δώσει n τιμές. Συνολικά το U1a παράγει n βαθμούς, ένας βαθμός αντιστοιχεί σε κάθε κωδικοποιημένο σύμβολο. Το U1a βγάζει τα στοιχεία της ROM που επιλέγει ψευδοτυχαίο ο Fibonacci στην έξοδό του, ενώ συγχρόνως υπολογίζει το άθροισμα των n βαθμών που επέλεξε.




**Εικόνα 45:** Degree generator

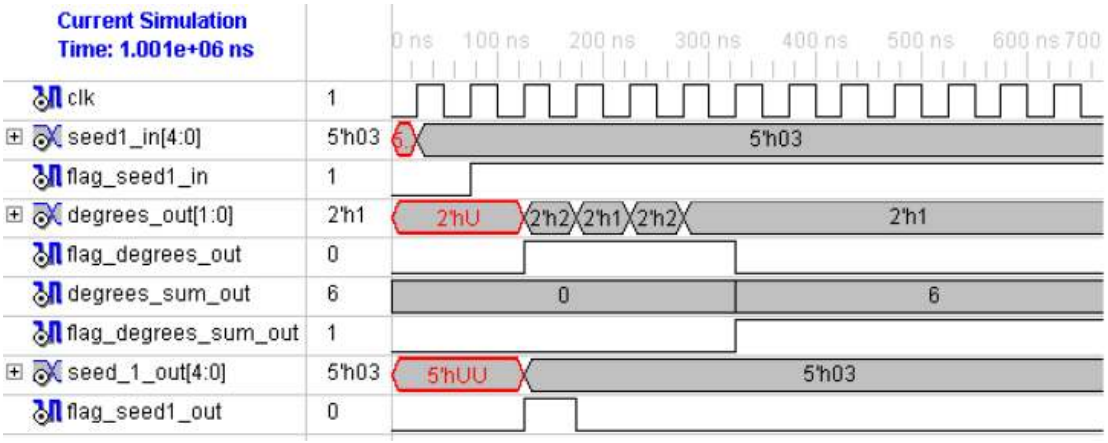
*RSD : Robust Soliton Distribution

*Fibonacci's Sequence Length: ο μέγιστος αριθμός που μπορεί να δώσει ο Fibonacci (κανονικοποιημένος με το -1) για το size του Fibonacci που έχουμε επιλέξει.

**Πίνακας 3 :** Είσοδοι και έξοδοι του U1a

| **Είσοδοι** | **Έξοδοι** |
|---|---|
| - Το seed 1 για τον Fibonacci που του το δίνει ο χειριστής. | - Το άθροισμα των βαθμών που δίνει στα U2a και U3a.<br>- Οι βαθμοί που δίνει στο U3a.<br>- Το seed 1 του Fibonacci που δίνει στο U1b. |

### *5.1.2 - Ανάλυση κυματομορφών εισόδων/εξόδων του U1a για k=3 και n=4*




**Εικόνα 46:** Κυματομορφές εισόδων/εξόδων του U1a από το Modelsim-SE VHDL του Xilinx

Όπως φαίνεται από τις κυματομορφές, τα σήματα εισόδων/εξόδων δε είναι μόνο αυτά που φαίνονται στον πιο πάνω πίνακα, αλλά συμπεριλαμβάνονται και άλλα σήματα - flags που είναι απαραίτητα για την λειτουργία του συστήματος.

Καταρχήν, το seed1_in είναι πέντε bits, και αυτό σημαίνει πως έχουμε επιλέξει 5-bit Fibonacci για να διαβάζει την RSD ROM. Άρα το πλήθος των διευθύνσεων της RSD ROM ισούται με $2^5 - 1$

Το seed που του δίνουμε εμείς είναι το “00011”( ή HEX03). Για να δεχτεί όμως το U1a το seed που του δίνουμε, θα πρέπει να μεταβεί το flag_seed1_in, από χαμηλή σε υψηλή κατάσταση. Δηλαδή το U1a δέχεται το seed που του δίνουμε στα 80ns.

Δέχεται το seed και αμέσως ο Fibonacci ξεκινά να διαβάζει την RSD ROM με ψευδοτυχαίο τρόπο. Σε κάθε θετικό μέτωπο, ο Fibonacci αλλάζει κατάσταση, δηλαδή σε κάθε θετικό μέτωπο διαβάζει ένα διαφορετικό στοιχείο της RSD ROM. Συνολικά ο Fibonacci θα διαβάσει n διαφορετικέ στοιχεία της RSD ROM. Τα στοιχεία της RSD ROM που διαβάζονταν από τον Fibonacci, αποτελούν τους βαθμούς των κωδικοποιημένων συμβόλων. Οι βαθμοί εμφανίζονται στην έξοδο degrees_out. Τους βαθμούς, τους στέλνει το U1a στο U3a. Το U3a θα τοποθετήσει τους βαθμούς σε μία RAM, για αυτό το λόγο το U1a στέλνει στο U3a το flag_degrees_out, το οποίο βρίσκεται σε υψηλή κατάσταση στο

χρονικό διάστημα από 120ns ως 320ns. Το flag_degrees_out είναι το σήμα ενεργοποίησης/απενεργοποίησης εγγραφής στην RAM του U3a.

Το flag_degrees_out μένει σε υψηλή κατάσταση για τέσσερα θετικά μέτωπα του clk, επειδή n=4. Κάθε φορά που υπάρχει θετικό μέτωπο του clk (και συγχρόνως το flag_degrees_out είναι σε υψηλή κατάσταση), σημαίνει πως γράφεται από ένας βαθμός στην RSD RAM. Όπως φαίνεται, ο Fibonacci έχει επιλέξει τους βαθμούς 2, 1, 2, 1.

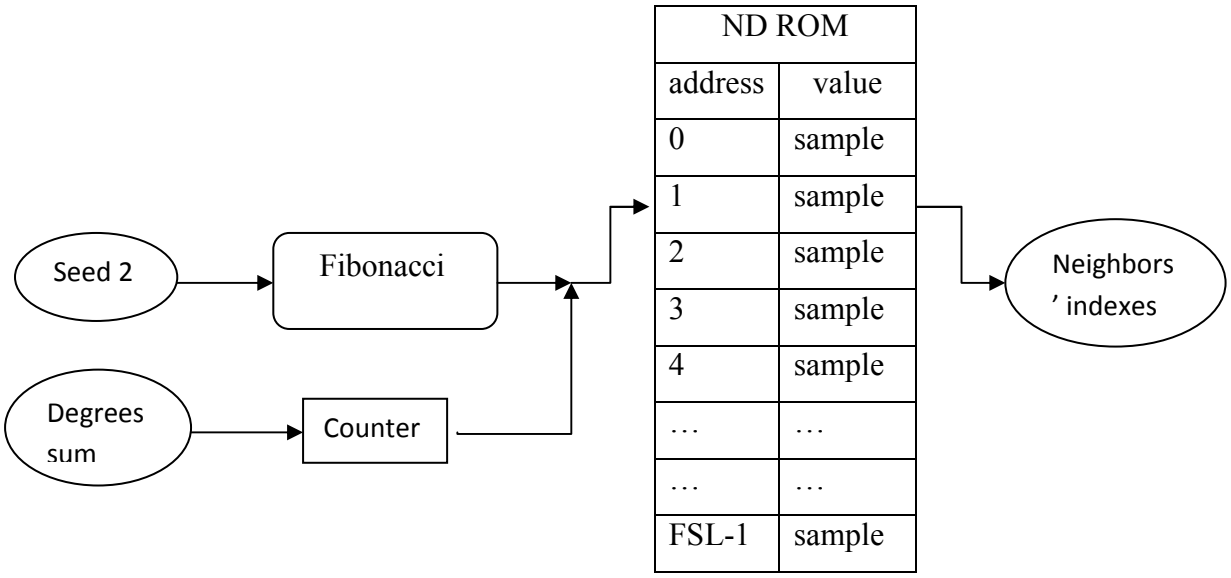
Το degrees_sum _out δίνει το άθροισμα των βαθμών στα U2a και U3b. Το degrees_sum _out δίνει την τιμή 6 (6=2+1+2+1) στα 320 ns. Αυτό γίνεται γιατί στα 320ns έχει ολοκληρωθεί η *συλλογή* των βαθμών και έχει υπολογιστεί το άθροισμά τους. Στα 320 ns αλλάζει κατάσταση το flag_degrees_sum_out, ώστε να ειδοποιηθούν τα U2a και U3a πότε να διαβάσουν το degrees_sum. Αν δεν υπήρχε το flag, τότε τα U2a και U3a θα δέχονταν degree sum=0, και αυτό είναι αδύνατον να συμβεί.

Και τέλος, το U1a δίνει το seed1 (που χρησιμοποίησε και το ίδιο) στο U1b του αποκωδικοποιητή, έτσι ώστε ο κωδικοποιητής και αποκωδικοποιητής να παράγουν τα ίδιους δείκτες των γειτονικών συμβόλων. Όπως και στις προηγούμενες περιπτώσεις, έτσι και τώρα, είναι απαραίτητο το flag, για αυτό υπάρχει το flag_seed1_out.

### *5.1.3 - U2a. Neighbors indexes Generator*

Στο U2a υπάρχει μία ROM (με το όνομα ND* ROM), η οποία περιέχει δείγματα από μία ισοκατανομή με όλα τους δείκτες των συμβόλων πληροφορίας - γειτονικών συμβόλων. Όλοι οι δείκτες έχουν τις ίδιες πιθανότητες εμφάνισης. Δηλαδή το περιεχόμενο αυτής της ROM είναι οι αριθμοί από 1 ως k από ίσες φορές το καθένα. Και εδώ δεν χρησιμοποιήθηκε κάποια αποδεδειγμένη μαθηματική σχέση για τον αριθμό των δειγμάτων, αλλά βάζουμε όσα δείγματα βάζουμε και στην ROM του U1a, δηλαδή περίπου $10 \cdot n$. Πάλη χρησιμοποιήθηκε έναν Fibonacci ως address bus για να διαβάσει με ψευδοτυχαίο τρόπο τα στοιχεία της ROM ώστε να δώσει τους δείκτες των γειτονικών πού αντιστοιχούν σε κάθε κωδικοποιημένο σύμβολο. Το σύνολο των γειτονικών ισούται με το άθροισμα των βαθμών, για αυτό και ο Fibonacci σταματάει όταν δώσει τυχαίες τιμές, ίσες με το άθροισμα των βαθμών. Το U2a βγάζει στην έξοδό του τα στοιχεία της ROM που επιλέγει ο Fibonacci.

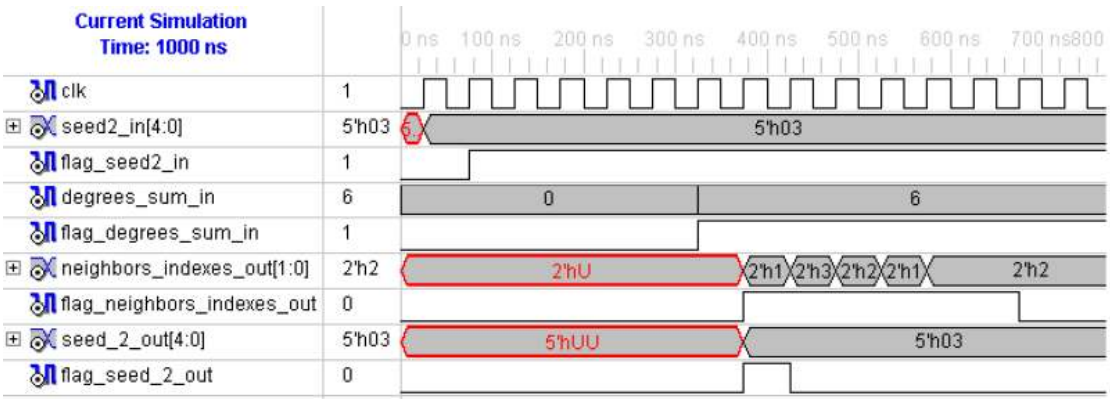
* ND : Normal Distribution (ισοκατανομή)




**Εικόνα 47:** Neighbors indexes generator

| **Πίνακας 4:** Είσοδοι και έξοδοι του U2a | |
|---|---|
| **Είσοδοι** | **Έξοδοι** |
| - Το seed 2 για τον Fibonacci που του δίνει ο χειριστής.<br>- Το άθροισμα των βαθμών που λαμβάνει από το U1a. | - Τους δείκτες των γειτονικών που δίνει στο U3a.<br>- Το seed 2 του Fibonacci στο U2b. |

### *5.1.4 - Ανάλυση κυματομορφών εισόδων/εξόδων του U2a για k=3 και n=4*




**Εικόνα48:** Κυματομορφές εισόδων/εξόδων του U1b από το Modelsim-SE VHDL του Xilinx

Στο U2a έτυχε και δώσαμε το ίδιο seed που δώσαμε και στο U2a. Άρα seed2_in=”00011”. Το flag_seed2_in έχει την ίδια λειτουργία με το flag_seed1_in του U1a.

Το U2a, παράγει τους δείκτες των γειτονικών. Δηλαδή για να ξεκινήσει αυτή η διαδικασία θα πρέπει να έχει λάβει από το U1a το άθροισμα των βαθμών, ώστε να ξέρει πόσους δείκτες να παράγει.

Το seed2 το έχει λάβει στα 80ns, και περιμένει να λάβει το άθροισμα των βαθμών. Το άθροισμα των βαθμών το λαμβάνει στα 320ns. Οπότε, από το επόμενο θετικό μέτωπο του clk και για κάθε ένα από τα επόμενα έξι θετικά μέτωπα, ο Fibonacci θα αλλάζει καταστάσεις. Με αυτόν τον τρόπο ο Fibonacci σε κάθε θετικό μέτωπο θα μας δίνει από ένα τυχαίο στοιχείο της ND ROM. Αυτά τα έξη στοιχεία της ND ROM που θα διαβάσει θα εμφανιστούν στην έξοδο neighbors_indexes_out. Οι δείκτες των γειτονικών που επέλεξε είναι τα 1 ,3, 2, 1, 2, 2. Οι δείκτες των γειτονικών θα σταλθούν στο U3a του κωδικοποιητή, το οποίο θα τα τοποθετήσει σε μία RAM. Το flag_neighbors_indexes_out ενεργοποιεί την κατάσταση εγγραφής της RAM που βρίσκεται στο U3a το χρονικό διάστημα 380ns ως 680ns.

Από το seed_2_out γίνεται αποστολή του seed2 (που χρησιμοποίησε και το ίδιο) στο U2b του αποκωδικοποιητή, ώστε κωδικοποιητής και αποκωδικοποιητής να παράγουν τους ίδιους δείκτες των γειτονικών συμβόλων.

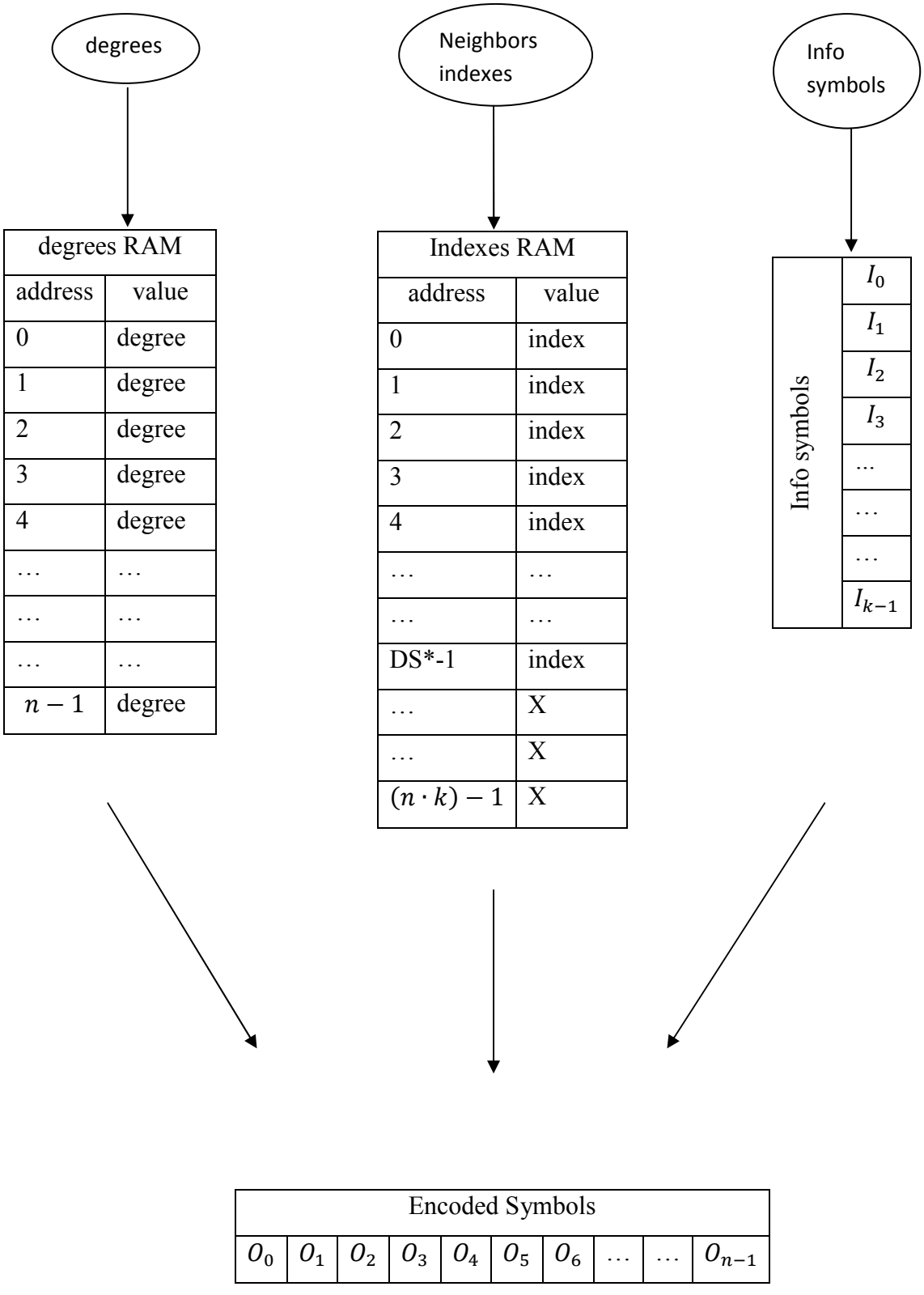
Το flag_seed2_out, λέει στο U2b του αποκωδικοποιητή πότε να λάβει το seed2.

### *5.1.5 - U3a. Encoding process*

Το U3a δέχεται τους βαθμούς και τους δείκτες και τους φορτώνει σε δύο Block RAM οι οποίες θα ονομάσουμε degrees RAM και indexes RAM. Τα σύμβολα πληροφορίας που του δίνει ο χειριστής τα φορτώνει σε registers. Τα στοιχεία που αποτελούν την degrees RAM γνωρίζουμε εξαρχής ότι είναι n, όμως δεν μπορούμε να γνωρίζουμε πόσα είναι τα στοιχεία της indexes RAM που θα χρησιμοποιούνται κάθε φορά, γιατί το άθροισμα των βαθμών μπορεί σε κάθε κωδικοποίηση να είναι διαφορετικό. Στην VHDL δεν μπορούμε να έχουμε μία RAM με μεταβλητό μέγεθος αλλά πρέπει να καθορίσουμε τα χαρακτηριστικά αυτής της RAM εξαρχής. Μία απλή λύση είναι να δηλώσουμε ότι αυτή η RAM αποτελείται από $n \cdot k$ στοιχεία. Γνωρίζουμε ότι τα κωδικοποιημένα σύμβολα θα είναι n και γνωρίζουμε ότι ο μέγιστος αριθμός των γειτονικών που μπορεί να έχει ένα κωδικοποιημένο

σύμβολο είναι k. Αυτή όμως είναι μία πρόχειρη λύση γιατί είναι αδύνατον η Robust Soliton Distribution να δώσει βαθμό k σε όλα τα κωδικοποιημένα σύμβολα, και αυτό σημαίνει ότι θα γίνεται μεγάλη σπατάλη των αποθεμάτων του FPGA. Μία πιο αποδοτική λύση είναι να υπολογίσουμε το μέσω όρο των βαθμών που παράγει η Robust Soliton Distribution και να ορίσουμε την indexes RAM με $μεσω\ όρο\ βαθμών\ \cdot k$ στοιχεία.

Για να κάνει το U3a την κωδικοποίηση θα χρειαστεί να διαβάσει τις δύο RAM και να κάνει τις XOR πράξεις μεταξύ των κωδικοποιημένων συμβόλων με τον τρόπο που περιγράφουν οι δύο αυτές RAM.




**Εικόνα 49:** Encoding process

* Degrees Sum

Η διαδικασία της κωδικοποίησης θα αναλυθεί με την βοήθεια του παρακάτω παραδείγματος.

Έστω ότι τα σύμβολα πληροφορίας είναι $k = 3$ και τα κωδικοποιημένα σύμβολα είναι $n = 4$. Και έστω πως οι τιμές στις δύο RAM και οι τιμές των συμβόλων πληροφορίας είναι οι παρακάτω.

| degrees RAM | |
|---|---|
| address | value |
| 0 | 3 |
| 1 | 1 |
| 2 | 2 |
| 3 | 2 |

| indexes RAM | |
|---|---|
| address | value |
| 0 | 1 |
| 1 | 0 |
| 2 | 2 |
| 3 | 0 |
| 4 | 1 |
| 5 | 2 |
| 6 | 1 |
| 7 | 0 |
| … | X |
| … | X |
| $(n \cdot k) - 1$ | X |

| Info symbols | | |
|---|---|---|
| $I_0$ | $I_1$ | $I_2$ |
| 1 | 1 | 0 |

**Εικόνα 50:** Παράδειγμα κωδικοποίησης

Καταρχήν, το συνολικό άθροισμα των βαθμών ισούται με 8, για αυτό και οι δείκτες των γειτονικών είναι 8.

Διαβάζοντας την degrees RAM καταλαβαίνουμε ότι το πρώτο κωδικοποιημένο σύμβολο που θα δημιουργηθεί θα είναι βαθμού 3, το δεύτερο κωδικοποιημένο σύμβολο θα είναι βαθμού 1, το τρίτο θα είναι βαθμού 2, και το τέταρτο θα είναι βαθμού 2.

Για να καταλάβουμε το περιεχόμενο της indexes RAM πρέπει να γνωρίζουμε το περιεχόμενο της degrees RAM.

Το στοιχείο 0 της degrees RAM είναι το 3, αυτό σημαίνει πως τα γειτονικά του πρώτου κωδικοποιημένου συμβόλου θα είναι αυτά που λένε τα περιεχόμενα των τριών πρώτων στοιχείων της indexes RAM.

Τα γειτονικά του πρώτου κωδικοποιημένου συμβόλου θα είναι τα σύμβολα πληροφορίας που βρίσκονται στις θέσεις 1, 0 και 2, δηλαδή όλα τα σύμβολα πληροφορίας.

Άρα

$$O_0 = I_1 \ xor \ I_0 \ xor \ I_2$$
$$\rightarrow O_0 = 1 \ xor \ 1 \ xor \ 0 = 0$$

Το στοιχείο 1 της degrees RAM είναι το 1, αυτό σημαίνει πως το μοναδικό γειτονικό του δεύτερου κωδικοποιημένου συμβόλου θα είναι αυτό που λέει το περιεχόμενο του στοιχείου 3 της indexes RAM.

Δηλαδή, το γειτονικό του πρώτου κωδικοποιημένου συμβόλου θα είναι το σύμβολο πληροφορίας που βρίσκεται στη θέση 0.

Άρα

$$O_1 = I_0$$
$$\rightarrow O_1 = 1$$

Το στοιχείο 2 της degrees RAM είναι το 2 αυτό σημαίνει πως τα γειτονικά του τρίτου κωδικοποιημένου συμβόλου θα είναι αυτά που λένε τα περιεχόμενα των δύο επόμενων στοιχείων της neighbors RAM από το σημείο που είχαμε μείνει την τελευταία φορά. Δηλαδή τώρα θα διαβάζει τα στοιχεία 4 και 5 της indexes RAM .

Τα γειτονικά του τρίτου κωδικοποιημένου συμβόλου θα είναι τα σύμβολα πληροφορίας που βρίσκονται στις θέσεις 1 και 2.

Άρα

$$O_2 = I_1 \ xor \ I_2$$
$$\rightarrow O_2 = 1 \ xor \ 0 = 1$$

Το στοιχείο 3 της degrees RAM είναι το 2, αυτό σημαίνει πως τα γειτονικά του τέταρτου κωδικοποιημένου συμβόλου θα είναι αυτά που λένε τα περιεχόμενα των στοιχείων 6 και 7 της neighbors RAM.

Τα γειτονικά του τέταρτου κωδικοποιημένου συμβόλου θα είναι τα σύμβολα πληροφορίας που βρίσκονται στις θέσεις 1 και 0.

Άρα

$$O_3 = I_1 \; xor \; I_0$$
$$\rightarrow O_3 = 1 \; xor \; 1 = 0$$

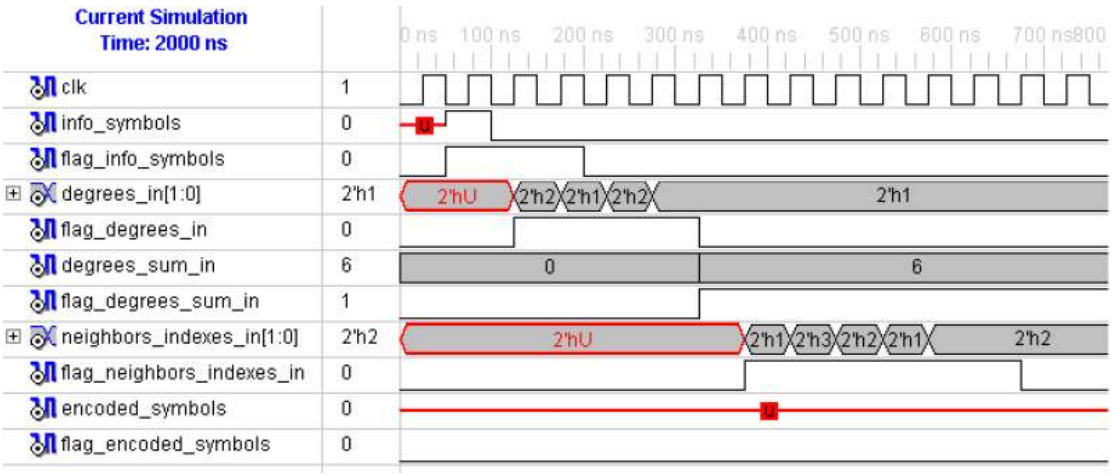
Συνολικά τα κωδικοποιημένα σύμβολα είναι :

| Encoded symbols | | | |
|---|---|---|---|
| $O_0$ | $O_1$ | $O_2$ | $O_3$ |
| 0 | 1 | 1 | 0 |

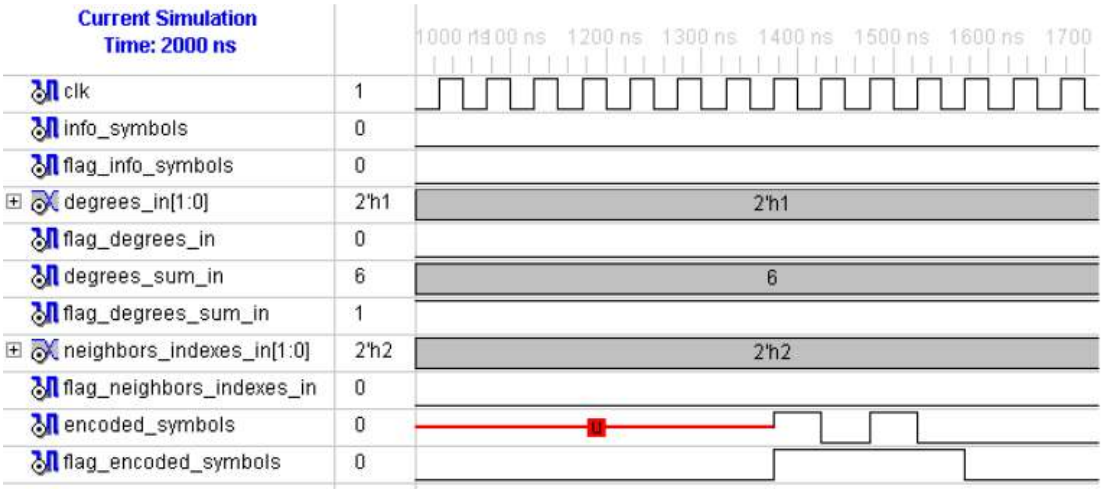
**Πίνακας 5:** Είσοδοι και έξοδοι του U3a

| **Είσοδοι** | **Έξοδοι** |
|---|---|
| - Τα σύμβολα πληροφορίας που του δίνει ο χειριστής.<br>- Το άθροισμα των βαθμών που παίρνει από το U1a.<br>- Οι βαθμοί που παίρνει από το U1a.<br>- Οι δείκτες των γειτονικών που παίρνει από το U2a. | - Τα κωδικοποιημένα σύμβολα που στέλνει στο U3b του αποκωδικοποιητή. |

### *5.1.6 - Ανάλυση κυματομορφών εισόδων/εξόδων του U3a για k=3 και n=4*




**Εικόνα 51:** Κυματομορφές εισόδων/εξόδων του U3a από το Modelsim-SE VHDL του Xilinx




**Εικόνα 52:** Κυματομορφές εισόδων/εξόδων του U3a από το Modelsim-SE VHDL του Xilinx

Από την είσοδο info_symbols, το U3a δέχεται σειριακά τα κωδικοποιημένα σύμβολα. Όσο το flag_info_symbols βρίσκεται σε θετική κατάσταση, τότε κάθε φορά που εμφανίζεται θετικό μέτωπο του clk το U3a δέχεται την κατάσταση που έχει η είσοδος info_symbols. Δηλαδή, βλέποντας το χρονικό διάστημα 60ns ως 200ns καταλαβαίνουμε ότι τα σύμβολα πληροφορίας είναι “100”.

Το flag_degrees_in ενεργοποιεί την εγγραφή στην degrees_RAM. Οπότε η degrees RAM γεμίζει με τους βαθμούς 2, 1, 2 και 1.

Από την  είσοδο degrees_sum_ in και με την βοήθεια του flag_degrees_sum δέχεται στα 320 ns το άθροισμα των βαθμών από το U1a.

Από την είσοδο του neighbors_indexes, δέχεται τις τιμές 1, 3, 2, 1, 2, 2 και τις τοποθετεί στην ND RAM.

Για να ξεκινήσει η διαδικασία τις κωδικοποίησης θα πρέπει πρώτα να έχουν γίνει αυτά που περιγράφηκαν παραπάνω.

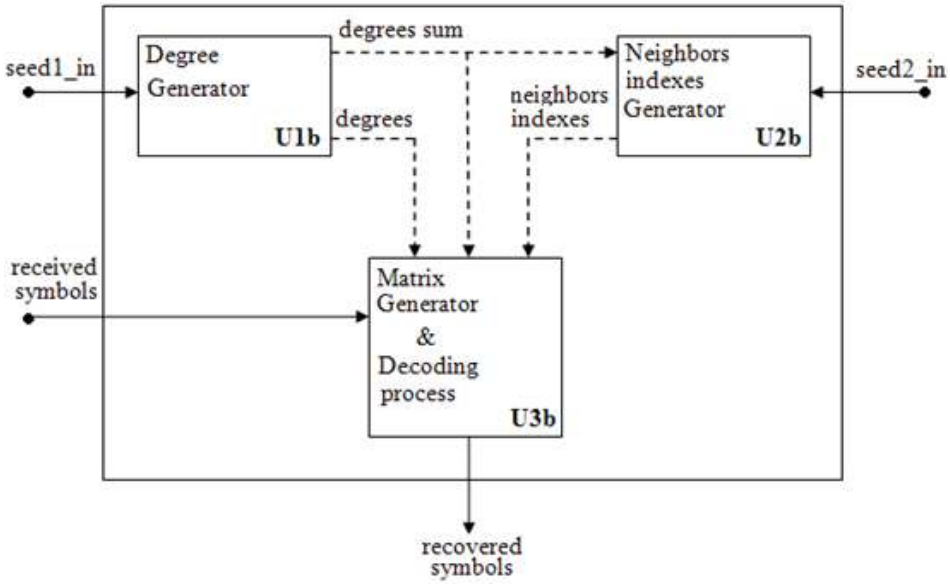
Στην συνέχεια ξεκινά να γίνεται η κωδικοποίηση. Κάθε ένα κωδικοποιημένο σύμβολο που παράγεται, αποθηκεύεται σε μία προσωρινή μνήμη. Όταν η μνήμη γεμίσει με n κωδικοποιημένα σύμβολα, τότε ξεκινά να γίνεται η μετάδοση τους στο U3b του αποκωδικοποιητή. Όπως φαίνεται από την παραπάνω κυματομορφή, ο κωδικοποιητής στέλνει και το flag_encoded_symbols για να μπορέσει να λάβει ο αποκωδικοποιητής τα encoded symbols. Η αποστολή των encoded symbols γίνεται από τα 1380 ως τα 1580ns.

Θα μπορούσε το κάθε encoded symbol να αποστέλλεται μόλις δημιουργηθεί. Δηλαδή, δεν είναι απαραίτητο τα κωδικοποιημένα σύμβολα να αποθηκεύονται σε προσωρινή μνήμη και μετά τα στέλνουμε. Με αυτό τον τρόπο θα χρησιμοποιούσαμε λιγότερη μνήμη, αλλά θα ήταν λίγο ποιο δύσκολο να γίνει η περιγραφή των κυματομορφών των encoded_symbols και flag_encoded_symbols.

## 5.2 - Hardware εφαρμογή του LT αποκωδικοποιητή

Ο αποκωδικοποιητής αποτελείτε από τρία Units:

- U1b. Degree Generator
- U2b. Neighbors indexes Generator
- U3b. Matrix generation and Decoding Process




**Εικόνα 53:** Εσωτερικές λειτουργίες του αποκωδικοποιητή

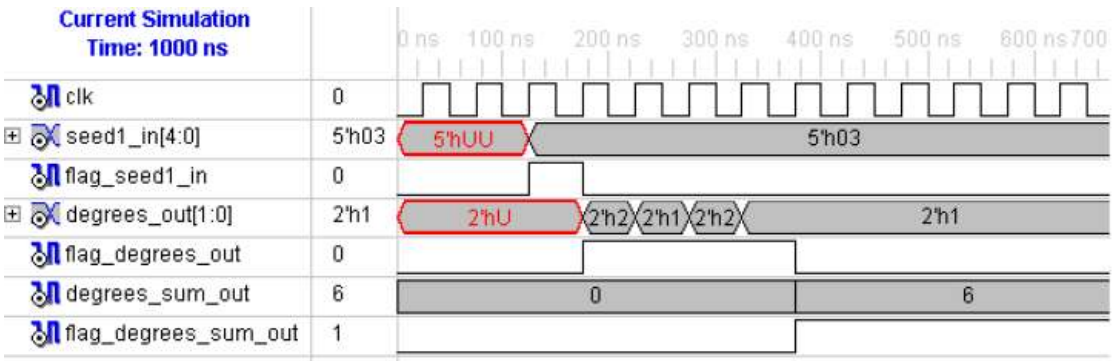
### *5.2.1 - U1b. Degree generator*

Το U1b κάνει τις ίδιες λειτουργίες με το U1a του κωδικοποιητή, με την διαφορά ότι δεν του δίνει ο χειριστής το seed του Fibonacci αλλά το δέχεται από το U1a.

**Πίνακας 6:** Είσοδοι και έξοδοι του U1b

| **Είσοδοι** | **Έξοδοι** |
|---|---|
| - Το seed 1 για τον Fibonacci που δέχεται από το U1a. | - Το άθροισμα των βαθμών στα U2b και U3b.<br>- Τους βαθμούς που δίνει στο U3b. |

### *5.2.2 - Ανάλυση κυματομορφών εισόδων/εξόδων του U1b για k=3 και n=4*




**Εικόνα 54:** Κυματομορφές εισόδων/εξόδων του U3b από το Modelsim-SE VHDL του Xilinx

Η ανάλυση του U1b του αποκωδικοποιητή είναι ουσιαστικά ίδια με την ανάλυση των κυματομορφών του U1a του κωδικοποιητή.

Η διαφορά είναι ότι στο U1b δεν του δίνει ο χειριστής to seed1 και flag seed1, αλλά τα δέχεται από το U1b του κωδικοποιητή. Επίσης, το U1b δεν στέλνει πουθενά το seed που χρησιμοποίησε.

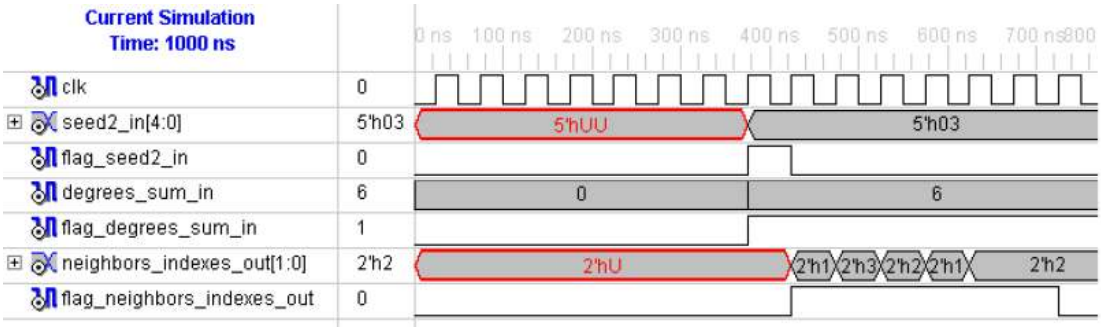
### *5.2.3 - U2b. Neighbors indexes Generator*

Το U2b κάνει τις ίδιες λειτουργίες με το U2a του κωδικοποιητή, με την διαφορά ότι δέχεται το seed του Fibonacci από το U2a.

**Πίνακας 7:** Είσοδοι και έξοδοι του U2b

| **Είσοδοι** | **Έξοδοι** |
|---|---|
| - Το seed 2 για τον Fibonacci που δέχεται από το U2a.<br>- Το άθροισμα των βαθμών που λαμβάνει από το U1b. | - Τους δείκτες των γειτονικών στο U3b. |

### *5.2.4 - Ανάλυση κυματομορφών εισόδων/εξόδων του U2b για k=3 και n=4*




**Εικόνα 55:** Κυματομορφές εισόδων/εξόδων του U2b από το Modelsim-SE VHDL του Xilinx

Η ανάλυση του U2b του αποκωδικοποιητή είναι ουσιαστικά ίδια με την ανάλυση των κυματομορφών του U2a του κωδικοποιητή.

Η διαφορά είναι ότι στο U2b δεν του δίνει ο χειριστής το seed2 και το flag seed2, αλλά το δέχεται από το U2b του κωδικοποιητή. Επίσης, το U2b δεν στέλνει πουθενά το seed που χρησιμοποίησε

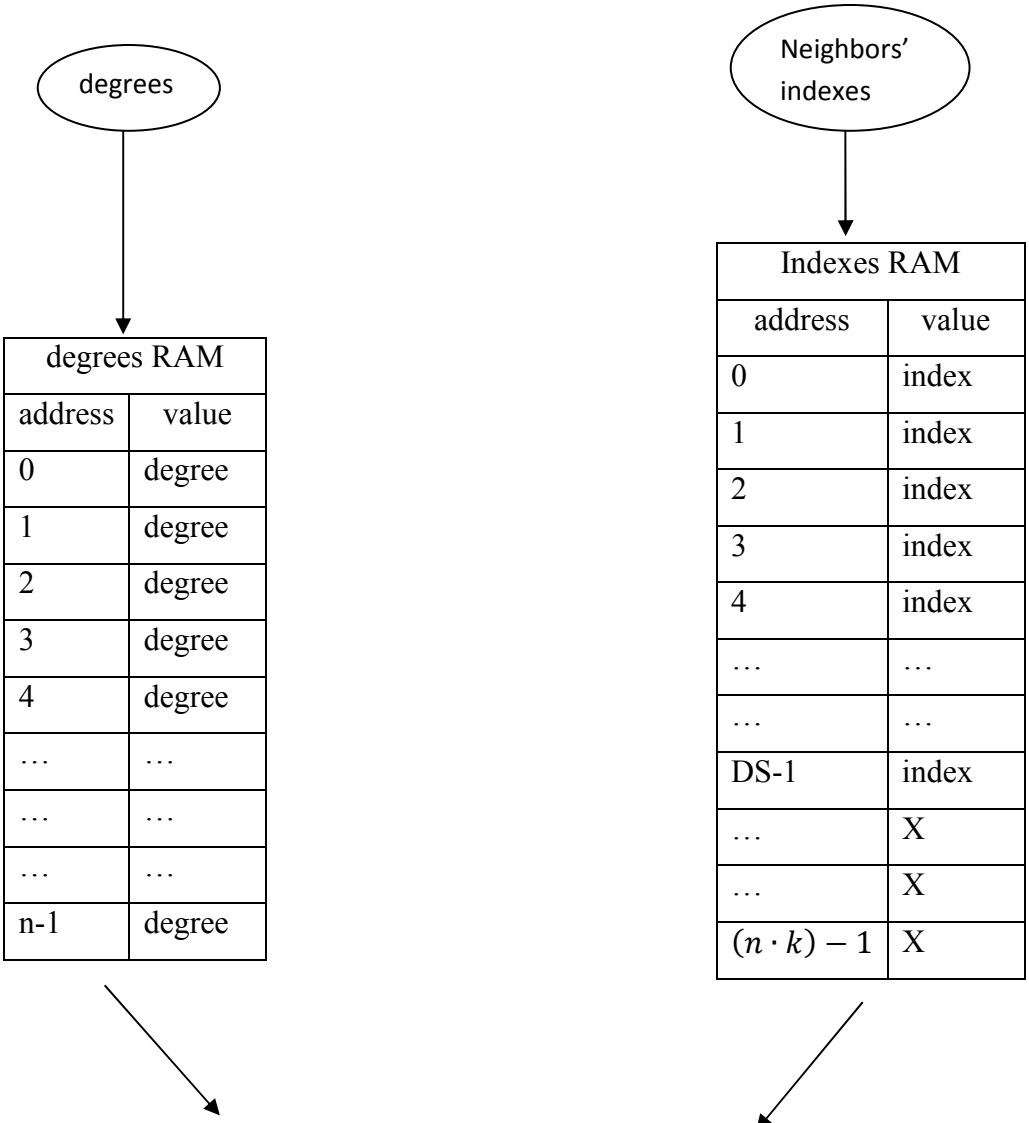
### *5.2.5 - U3b Matrix generation και Decoding process*

Το U3b αποτελείται από δύο υπό-κομμάτια:

- Matrix Generator και
- Decoding process

#### ***5.2.5.1 - Matrix Generator***

Το Matrix Generator δέχεται τους βαθμούς και τους δείκτες των γειτονικών από τα U1b και U2b αντίστοιχα, και τα φορτώνει σε δύο RAM, που ονομάζονται όπως και πριν, degrees RAM και indexes RAM. Διαβάζει αυτές τις δύο RAM, και με βάση τα περιεχόμενά τους δημιουργεί έναν πίνακα ο οποίος ονομάζεται Connectivity Matrix $M_c$. Αυτός ο πίνακας εξηγεί με απλό τρόπο ποια είναι η σύνδεση που έχουν τα κωδικοποιημένα σύμβολα με τα σύμβολα πληροφορίας.




**Εικόνα 56:** Matrix Generation

Με την βοήθεια του παρακάτω παραδείγματος, θα περιγραφεί ο τρόπος με τον οποίο γίνεται το διάβασμα των δύο RAM, και η δημιουργία του πίνακα $M_c$.

| degrees RAM | |
|---|---|
| address | value |
| 0 | 3 |
| 1 | 1 |
| 2 | 2 |
| 3 | 2 |

| indexes RAM | |
|---|---|
| address | value |
| 0 | 1 |
| 1 | 0 |
| 2 | 2 |
| 3 | 0 |
| 4 | 1 |
| 5 | 2 |
| 6 | 1 |
| 7 | 0 |
| … | X |
| … | X |
| $(n \cdot k) - 1$ | X |

**Εικόνα 57:** Παράδειγμα δημιουργίας του $M_c$

Αρχικά ο $M_c$ είναι γεμάτος μηδενικά. Βάζουμε άσους στα στοιχεία του $M_c$ που περιγράφουν οι δύο RAM.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 0 | 0 | 0 | 0 |
| $I_1$ | 0 | 0 | 0 | 0 |
| $I_2$ | 0 | 0 | 0 | 0 |

**Εικόνα 58:** Ο πίνακας $M_c$ πριν γίνουν οι συνδέσεις

Το στοιχείο 0 του degrees RAM είναι 3, αυτό σημαίνει ότι στη στήλη 0 του Connectivity Matrix θα βάλουμε τρεις άσους. Στην συνέχεια διαβάζουμε τα πρώτα τρία στοιχεία της indexes RAM για να

δούμε σε ποιες γραμμές της στήλης 0 του $M_c$ θα βάλουμε τους άσους. Τα πρώτα τρία στοιχεία της indexes RAM είναι τα 1, 0 και 2.

Άρα θα βάλουμε άσους στις γραμμές 1, 0 και 2 της στήλης 0 του $M_c$.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 1 | 0 | 0 | 0 |
| $I_1$ | 1 | 0 | 0 | 0 |
| $I_2$ | 1 | 0 | 0 | 0 |

**Εικόνα 59:** Δημιουργία των συνδέσεων του πρώτου κωδικοποιημένου συμβόλου

Το στοιχείο 1 της degrees RAM είναι το 1, αυτό σημαίνει πως στη στήλη 1 του $M_c$ θα βάλουμε έναν άσο. Διαβάζουμε το επόμενο στοιχείο από εκεί που είχαμε μείνει την τελευταία φορά στην indexes RAM, δηλαδή το στοιχείο 3. Το περιεχόμενο του στοιχείου 3 της indexes RAM είναι ο αριθμός 0.

Άρα θα βάλουμε άσο στο στοιχείο 0 της στήλης 1 του $M_c$.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 1 | 1 | 0 | 0 |
| $I_1$ | 1 | 0 | 0 | 0 |
| $I_2$ | 1 | 0 | 0 | 0 |

**Εκόνα 60:** Δημιουργία των συνδέσεων του δευτέρου κωδικοποιημένου συμβόλου

Το στοιχείο 2 της Degrees RAM είναι ο αριθμός 2, άρα στη στήλη 2 του $M_c$ θα βάλουμε δύο άσους. Τα στοιχεία 4 και 5 της indexes RAM περιέχουν τους αριθμούς 1 και 2.

Άρα θα βάλουμε άσους στις γραμμές 1 και 2 της στήλης 2 του $M_c$.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 1 | 1 | 0 | 0 |
| $I_1$ | 1 | 0 | 1 | 0 |
| $I_2$ | 1 | 0 | 1 | 0 |

**Εκόνα 61:** Δημιουργία των συνδέσεων του τρίτου κωδικοποιημένου συμβόλου

Το στοιχείο 3 της degrees RAM είναι ο αριθμός 2, άρα στη στήλη 2 του $M_c$ θα βάλουμε δύο άσους. Τα στοιχεία 6 και 7 της indexes RAM περιέχουν τους αριθμούς 1 και 0.

Άρα θα βάλουμε άσσους στις γραμμές 1 και 0 της στήλης 3 του $M_c$.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 1 | 1 | 0 | 1 |
| $I_1$ | 1 | 0 | 1 | 1 |
| $I_2$ | 1 | 0 | 1 | 0 |

**Εικόνα 62:** Δημιουργία των συνδέσεων του τέταρτου κωδικοποιημένου συμβόλου

#### ***5.2.5.2 - Decoding process***

Για να ξεκινήσει η Decoding process (διαδικασία αποκωδικοποίησης) θα πρέπει να έχει δημιουργηθεί ο $M_c$ και επιπλέον να έχει λάβει ο αποκωδικοποιητής τα κωδικοποιημένα σύμβολα.

Connectivity Matrix $M_c$

| | | Encoded Symbols | | | | | |
|---|---|---|---|---|---|---|---|
| | | $O_0$ | $O_1$ | $O_2$ | $O_3$ | … | $O_{n-1}$ |
| Info symbols | $I_0$ | 1 | 1 | 1 | 1 | … | 0 |
| | $I_1$ | 0 | 1 | 0 | 1 | … | 0 |
| | $I_2$ | 1 | 0 | 0 | 1 | … | 1 |
| | … | … | … | … | … | … | 0 |
| | $I_{k-1}$ | 0 | 1 | 1 | 1 | 0 | 0 |

| Encoded symbols – Received Symbols |
|---|
| $O_0$ |
| $O_1$ |
| $O_2$ |
| $O_3$ |
| $O_5$ |
| $O_6$ |
| $O_7$ |
| $O_8$ |
| $O_9$ |
| $O_{10}$ |
| $O_{11}$ |
| … |
| … |
| $O_{n-1}$ |

| Info symbols-Recovered symbols | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $I_0$ | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | $I_6$ | … | … | $I_{k-1}$ |

**Εικόνα 63:** Decoding process

Η διαδικασία της αποκωδικοποίησης θα αναλυθεί με την βοήθεια του παραδείγματος που ακολουθεί.

Ο αποκωδικοποιητής έχει δημιουργήσει τον παρακάτω πίνακα $M_c$ και έχει λάβει τα παρακάτω λαμβανόμενα-κωδικοποιημένα σύμβολα, και προσπαθεί να ανακτήσει τα σύμβολα πληροφορίας.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 1 | 1 | 0 | 1 |
| $I_1$ | 1 | 0 | 1 | 1 |
| $I_2$ | 1 | 0 | 1 | 0 |

| received symbols | | | |
|---|---|---|---|
| $O_0$ | $O_1$ | $O_2$ | $O_3$ |
| 0 | 1 | 1 | 0 |

| Recovered symbols | | |
|---|---|---|
| $I_0$ | $I_1$ | $I_2$ |
| | | |

**Εικόνα 64: $1^o$** βήμα αποκωδικοποίησης

Ο αποκωδικοποιητής ξεκινάει να διαβάζει τον $M_c$ στήλη-στήλη και ψάχνει να βρει μία στήλη που να έχει έναν μόνο άσο. Σε αυτό το παράδειγμα έναν άσο έχει η στήλη 1 στην γραμμή 0.

Άρα κάνει copy-paste την τιμή που έχει το κωδικοποιημένο σύμβολο $O_1$, στο ανακτώμενο σύμβολο $I_0$, και μηδενίζει τον άσο που βρήκε στο στοιχείο (0,1) του $M_c$. Με αυτόν τον τρόπο έκανε την πρώτη ανάκτηση και έσβησε την σύνδεση που είχε κωδικοποιημένο σύμβολο $O_1$, με το σύμβολο πληροφορίας $I_0$.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 1 | 0 | 0 | 1 |
| $I_1$ | 1 | 0 | 1 | 1 |
| $I_2$ | 1 | 0 | 1 | 0 |

| Received symbols | | | |
|---|---|---|---|
| $O_0$ | $O_1$ | $O_2$ | $O_3$ |
| 0 | 1 | 1 | 0 |

| Recovered symbols | | |
|---|---|---|
| $I_0$ | $I_1$ | $I_2$ |
| 1 | | |

**Εικόνα 65: $2^o$** βήμα αποκωδικοποίησης

Τώρα πρέπει να κάνει ανανέωση (refresh) τον $M_c$ και τα λαμβανόμενα σύμβολα. Δηλαδή, διαβάζει μία φορά ολόκληρη την γραμμή (στην οποία προηγούμενος έσβησε τον άσο) και κάνει το εξής. Όταν βρει άσο κάνει XOR την τιμή του συμβόλου που μόλις πριν έκανε ανάκτηση με το αντίστοιχο

λαμβανόμενο σύμβολο που λέει ο $M_c$, αναθέτει το αποτέλεσμα της πράξης στο λαμβανόμενο σύμβολο, και στην συνέχεια σβήνει την σύνδεση που μόλις διάβασε στον $M_c$.

Δηλαδή συγκεκριμένα στο παράδειγμα μας, ξεκινάει και διαβάζει την γραμμή 0 ολόκληρη. Βρίσκει άσο στη γραμμή 0 στήλη 0.

Οπότε κάνει :

$$O_0 = O_0\ XOR\ I_0$$
$$\rightarrow O_0 = 0\ XOR\ 1\ = 1$$

Και στην συνέχεια σβήνει τον άσο στο στοιχείο $(0,0)$ του $M_c$.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 0 | 0 | 0 | 1 |
| $I_1$ | 1 | 0 | 1 | 1 |
| $I_2$ | 1 | 0 | 1 | 0 |

| Received symbols | | | |
|---|---|---|---|
| $O_0$ | $O_1$ | $O_2$ | $O_3$ |
| 1 | 1 | 1 | 0 |

| Recovered symbols | | |
|---|---|---|
| $I_0$ | $I_1$ | $I_2$ |
| 1 | | |

**Εικόνα 66: $3^o$** βήμα αποκωδικοποίησης

Συνεχίζει να διαβάζει την γραμμή 0 του $M_c$ και βρίσκει άσο στο στοιχείο (0,3), οπότε κάνει

$$O_3 = O_3\ XOR\ I_0$$
$$\rightarrow O_3 = 0\ \text{XOR}\ 1\ = 1$$

Και στην συνέχεια σβήνει τον άσο που μόλις διάβασε στον $M_c$, οπότε έχουμε.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 0 | 0 | 0 | 0 |
| $I_1$ | 1 | 0 | 1 | 1 |
| $I_2$ | 1 | 0 | 1 | 0 |

| Received symbols | | | |
|---|---|---|---|
| $O_0$ | $O_1$ | $O_2$ | $O_3$ |
| 1 | 1 | 1 | 1 |

| Recovered symbols | | |
|---|---|---|
| $I_0$ | $I_1$ | $I_2$ |
| 1 | | |

**Εικόνα 67:** $\mathbf{4^o}$ βήμα αποκωδικοποίησης

Τώρα τέλειωσε η ανανέωση του $M_c$ και η ανανέωση των λαμβανόμενων συμβόλων. Ο αποκωδικοποιητής ξεκινάει πάλη από την αρχή και διαβάζει στήλη-στήλη τον $M_c$ για να βρει κάποια στήλη με μόνο έναν άσο.

Βρίσκει ότι έναν μόνο άσο έχει η στήλη 3, και συγκεκριμένα ο άσος βρίσκεται στην γραμμή 1. Οπότε κάνει copy-paste την τιμή του λαμβανόμενου συμβόλου $O_3$ στο ανακτώμενο σύμβολο $I_1$ και μηδενίζει τον άσο που βρήκε στο στοιχείο (1,3) του $M_c$.

$$I_1 = O_3$$
$$\rightarrow I_1 = 1$$

Οπότε έγινε κι άλλη ανάκτηση.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 0 | 0 | 0 | 0 |
| $I_1$ | 1 | 0 | 1 | 0 |
| $I_2$ | 1 | 0 | 1 | 0 |

| Received symbols | | | |
|---|---|---|---|
| $O_0$ | $O_1$ | $O_2$ | $O_3$ |
| 1 | 1 | 1 | 1 |

| Recovered symbols | | |
|---|---|---|
| $I_0$ | $I_1$ | $I_2$ |
| 1 | 1 | |

**Εικόνα 68:** $\mathbf{5^o}$ βήμα αποκωδικοποίησης

Τώρα που έγινε η ανάκτηση έχει σειρά η ανανέωση του $M_c$ και των λαμβανόμενων συμβόλων.

Ο κωδικοποιητής ξεκινά να διαβάζει την γραμμή 1 του $M_c$. Βρίσκει άσο στο στοιχείο (1,0) του $M_c$. Οπότε κάνει

$$O_0 = O_0\ XOR\ I_1$$
$$\rightarrow O_0 = 1\ XOR\ 1 = 0$$

Σβήνει τον άσο στο στοιχείο (1,0) του $M_c$.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 0 | 0 | 0 | 0 |
| $I_1$ | 0 | 0 | 1 | 0 |
| $I_2$ | 1 | 0 | 1 | 0 |

| Received symbols | | | |
|---|---|---|---|
| $O_0$ | $O_1$ | $O_2$ | $O_3$ |
| 0 | 1 | 1 | 1 |

| Recovered symbols | | |
|---|---|---|
| $I_0$ | $I_1$ | $I_2$ |
| 1 | 1 | |

**Εικόνα 69**: **$6^o$** βήμα αποκωδικοποίησης

Και συνεχίζει να διαβάζει την γραμμή 1 του $M_c$. Βρίσκει άσο στο στοιχείο (1,2) οπότε:

$$O_2 = O_2\ XOR\ I_1$$
$$\rightarrow O_2 = 1\ XOR\ 1 = 0$$

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 0 | 0 | 0 | 0 |
| $I_1$ | 0 | 0 | 0 | 0 |
| $I_2$ | 1 | 0 | 1 | 0 |

| Received symbols | | | |
|---|---|---|---|
| $O_0$ | $O_1$ | $O_2$ | $O_3$ |
| 0 | 1 | 0 | 1 |

| Recovered symbols | | |
|---|---|---|
| $I_0$ | $I_1$ | $I_2$ |
| 1 | 1 | |

**Εικόνα 70**: **$7^o$** βήμα αποκωδικοποίησης

Διάβασε ολόκληρη την γραμμή 1 του $M_c$ , οπότε τέλειωσε η ανανέωση και ξεκινά να κάνει την επόμενη ανάκτηση.

Διαβάζει τον $M_c$ στήλη-στήλη, και ο πρώτος άσος που βρίσκει είναι αυτός που είναι στην στήλη 0 του $M_c$. Σε αυτό το σημείο κάποιος θα μπορούσε να χρησιμοποιήσει τον άσο της στήλης 2. Το αποτέλεσμα θα είναι το ίδιο, γιατί αν προσέξουμε μετά την ανανέωση των λαμβανόμενων συμβόλων, την συγκεκριμένη στιγμή το $O_0$ είναι ίσο με το $O_2$. Αυτό είναι το επιθυμητό, αν δεν συνέβαινε τότε θα σήμαινε ότι έχουμε κάνει κάποιο λάθος στη διαδικασία.

Οπότε έχουμε,

$$I_2 = O_0$$
$$\rightarrow I_2 = 0$$

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 0 | 0 | 0 | 0 |
| $I_1$ | 0 | 0 | 0 | 0 |
| $I_2$ | 0 | 0 | 1 | 0 |

| Received symbols | | | |
|---|---|---|---|
| $O_0$ | $O_1$ | $O_2$ | $O_3$ |
| 0 | 1 | 0 | 1 |

| Recovered symbols | | |
|---|---|---|
| $I_0$ | $I_1$ | $I_2$ |
| 1 | 1 | 0 |

**Εικόνα 71**: $\mathbf{8^o}$ βήμα αποκωδικοποίησης

Στην ουσία η αποκωδικοποίηση έχει τελειώσει σε αυτό το σημείο, όμως θα συνεχίσουμε το παράδειγμα για να δείξουμε πως καταλαβαίνει το πρόγραμμα ότι τελείωσε η αποκωδικοποίηση.
Τώρα κάνει και την τελευταία ανανέωση που απέμεινε. Διαβάζει την στήλη 2 του $M_c$, και βρίσκει τον άσο στο στοιχείο (2,1)

Οπότε,

$$O_2 = O_2 \; XOR \; I_2$$
$$\rightarrow O_2 = 1 \; XOR \; 0 = 1$$

Και μηδενίζει τον τελευταίο άσο στον $M_c$.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 0 | 0 | 0 | 0 |
| $I_1$ | 0 | 0 | 0 | 0 |
| $I_2$ | 0 | 0 | 0 | 0 |

| Received symbols | | | |
|---|---|---|---|
| $O_0$ | $O_1$ | $O_2$ | $O_3$ |
| 0 | 1 | 1 | 1 |

| Recovered symbols | | |
|---|---|---|
| $I_0$ | $I_1$ | $I_2$ |
| 1 | 1 | 0 |

**Εικόνα 72**: **$9^o$** βήμα αποκωδικοποίησης

Ο αποκωδικοποιητής ξεκινά πάλι να διαβάσει τον $M_c$, για να βρει κάποια στήλη με έναν μοναδικό άσο. Όμως διαβάζει όλες τις στήλες και δεν βρίσκει κανέναν άσο. Ελέγχει πόσα σύμβολα έχει ανακτήσει. Αν δεν υπάρχει κανένας άσος στον $M_c$ και έχει κάνει ανάκτηση k σύμβολα τότε έχει γίνει επιτυχείς αποκωδικοποίηση.

Υπάρχουν δύο περιπτώσεις για τις οποίες η αποκωδικοποίηση μπορεί να αποτύχει.

*5.2.5.2.1 - Ά περίπτωση μη-αποκωδικοποιήσημου $M_c$*

Να μην υπάρχει εξαρχής κανένα λαμβανόμενο σύμβολο με βαθμό 1, ή, μετά από κάποια ανανέωση να μην υπάρχει κανένα λαμβανόμενο σύμβολο με βαθμό 1.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 1 | 0 | 1 | 1 |
| $I_1$ | 0 | 1 | 1 | 0 |
| $I_2$ | 1 | 0 | 1 | 1 |

**Εικόνα 73:** Ά περίπτωση μη-αποκωδικοποιήσημου $M_c$

Σε αυτόν τον $M_c$ μπορεί να γίνει ανάκτηση το $I_1$, αλλά από εκεί και πέρα τα λαμβανόμενα σύμβολα που απομένουν είναι όλα βαθμού 2. Δηλαδή δεν υπάρχει στήλη με έναν μόνο άσο. Ο αποκωδικοποιητής δεν θα μπορέσει να μηδενίσει ποτέ όλα τα στοιχεία του $M_c$.

*5.2.5.2.2 - Β περίπτωση μη-αποκωδικοποιήσημου $M_c$*

Να μην έχει καλυφθεί κάποιο ή κάποια σύμβολα πληροφορίας από τα κωδικοποιημένα σύμβολα.

| | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|
| $I_0$ | 1 | 0 | 1 | 1 |
| $I_1$ | 0 | 1 | 1 | 0 |
| $I_2$ | 0 | 0 | 0 | 0 |

**Εικόνα 74:** Β περίπτωση μη-αποκωδικοποιήσημου $M_c$

Το $I_2$ δεν θα μπορέσει να ανακτηθεί γιατί κανένα λαμβανόμενο σύμβολο δεν έχει σύνδεση μαζί του. Πάντως όλοι οι άσοι του πίνακα θα γίνουν μηδέν. Στην συνέχεια ο αποκωδικοποιητής θα μετρήσει πόσα σύμβολα ανάκτησε και θα μετρήσει μόλις ένα αντί k.

Στην περίπτωση που δεν υπάρχει θόρυβος στο κανάλι που συνδέει τον κωδικοποιητή-αποκωδικοποιητή, τότε για την αποτυχία της αποκωδικοποίησης ευθύνεται η Robust Soliton Distribution ή οι ψευδοτυχαίες μηχανές αριθμών (RNG) που χρησιμοποιούμε. Η δημιουργία του $M_c$ είναι εξολοκλήρου βασισμένη πάνω σε αυτά τα δύο στοιχεία.

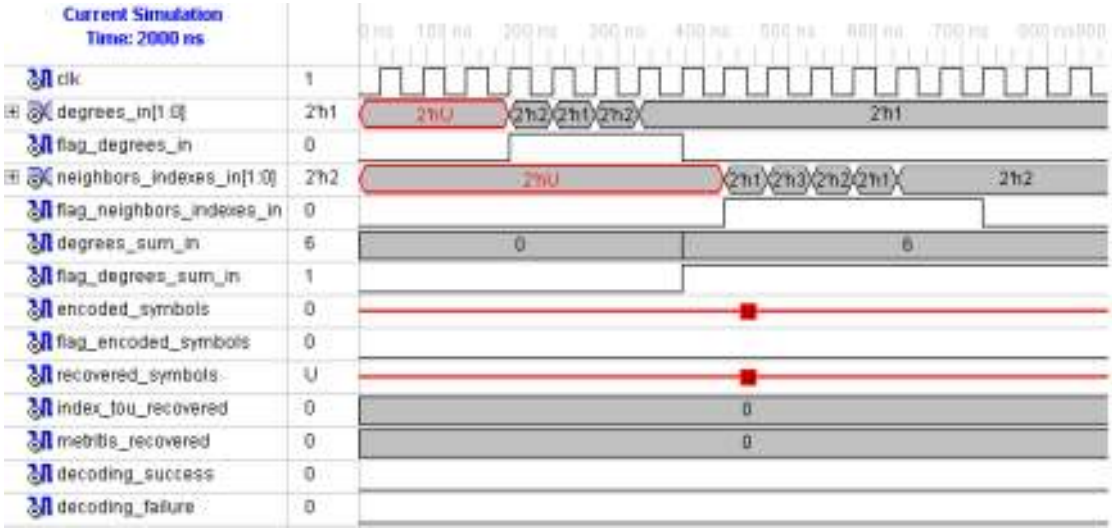
Αν υπάρχει θόρυβος τότε στους λόγους της αποτυχίας της αποκωδικοποίησης συμπεριλαμβάνονται οι περιπτώσεις όπου:

- σβήστηκαν τα κωδικοποιημένα σύμβολα που ήταν εξαρχής βαθμού 1,
- ή σβήστηκαν κωδικοποιημένα σύμβολα που θα γίνονταν βαθμού 1 μετά από κάποια ανανέωση,
- ή σβήστηκαν τα κωδικοποιημένα σύμβολα που ήταν συνδεδεμένα με ένα ή κάποια σύμβολα πληροφορίας και κανένα από τα κωδικοποιημένα σύμβολα που έλαβε ο αποκωδικοποιητής δεν καλύπτει αυτό ή αυτά τα σύμβολα πληροφορίας.

**Πίνακας 8:** Είσοδοι και έξοδοι του U3b

| **Είσοδοι** | **Έξοδοι** |
|---|---|
| - Τα κωδικοποιημένα σύμβολα που δέχεται από το U3a του κωδικοποιητή.<br>- Το άθροισμα των βαθμών που παίρνει από το U1b.<br>- Οι βαθμοί που παίρνει από το U1b.<br>- Οι δείκτες των γειτονικών που παίρνει από το U2b. | - Τα ανακτημένα σύμβολα **.** |

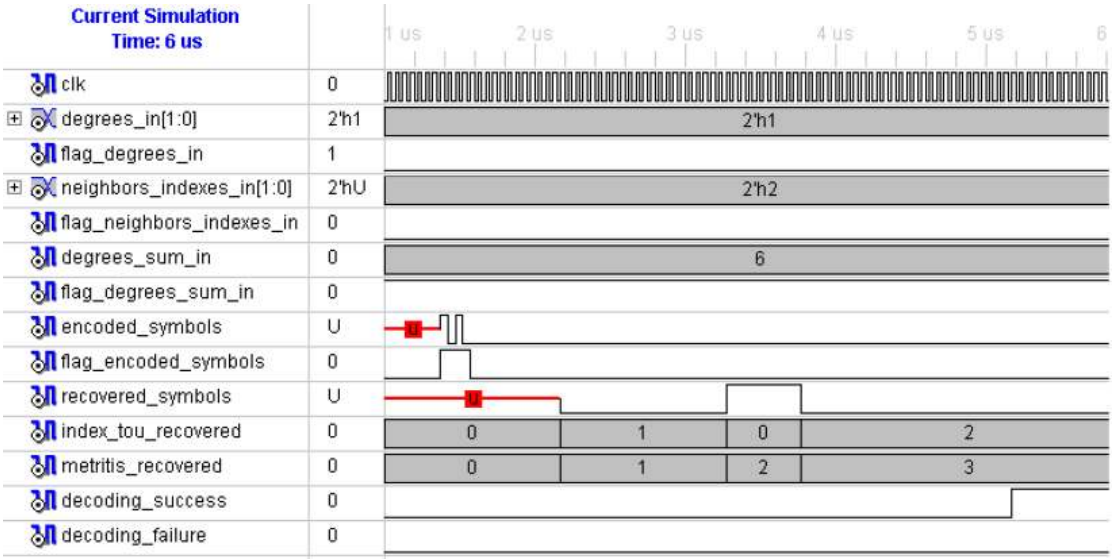
### *5.2.6 - Ανάλυση κυματομορφών εισόδων/εξόδων του U3b για k=3 και n=4*




**Εικόνα 75:** Κυματομορφές εισόδων/εξόδων του U3b από το Modelsim-SE VHDL του Xilinx

Τα σήματα εισόδου του U3b degrees_in, flag_degrees_in, degrees_sum_in, flag_degrees_sum_in, neighbors_indexes_in και flag_neighbors_indexes_in κάνουν ακριβώς τις ίδιες λειτουργίες με τις αντίστοιχες εισόδους του U3a του κωδικοποιητή.

Η διαφορά είναι ότι αυτά τα σήματα εισόδου, το U3a τα δέχεται από τα άλλα δύο Units του κωδικοποιητή, ενώ τις αντίστοιχες εισόδους το U3b τις δέχεται από τα άλλα δύο Units του αποκωδικοποιητή.




**Εικόνα 76:** Κυματομορφές εισόδων/εξόδων του U3b από το Modelsim-SE VHDL του Xilinx

Από την είσοδο encoded_symbols και με την βοήθεια του flag_info_symbols, ο αποκωδικοποιητής λαμβάνει τα κωδικοποιημένα σύμβολα από τον κωδικοποιητή, και τα αποθηκεύει σε μία προσωρινή μνήμη. Αυτό συμβαίνει στο χρονικό διάστημα 1,3 μs ως 1,6 μs περίπου.

Το σήμα recovered_symbols μας δίνει την τιμή του παρόντος ανακτημένου συμβόλου.

Το σήμα index_tou_recovered, μας λέει σε ποιο σύμβολο πληροφορίας αντιστοιχεί η τιμή που διαβάζουμε στο σήμα recovered_symbols.

Δηλαδή, διαβάζοντας τις κυματομορφές recovered_symbols και index_tou_recovered_symbol από την στιγμή 2,2 μs και ύστερα, βλέπουμε πως η ανάκτηση των συμβόλων πληροφορίας γίνεται με την παρακάτω σειρά:

$$I_1=0\ ,\ I_0=1 \text{ και } I_2=0$$

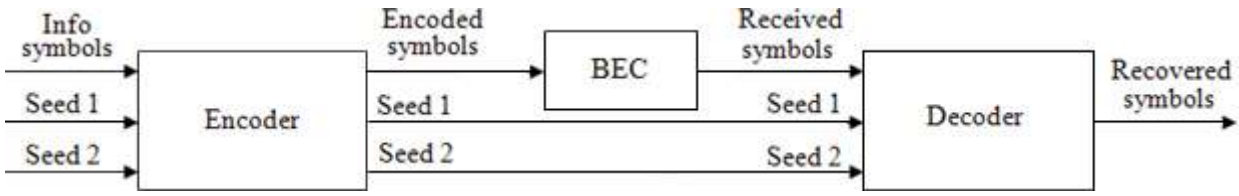
Το σήμα metritis_recovered αυξάνεται κατά μία μονάδα κάθε φορά που γίνεται μία νέα ανάκτηση.

Συνολικά ο metritis_recovered μέτρησε μέχρι το 3 και αφού k=3, βλέπουμε πως το decoding_success έγινε 1, ενώ το decoding_failure παρέμεινε στο 0. Δηλαδή η αποκωδικοποίηση πέτυχε.

### 5.3 - Αποτελέσματα της Hardware Εφαρμογής

Σε αυτήν την παράγραφο παρουσιάζουμε τα αποτελέσματα από την Hardware εφαρμογή σε FPGA. Ελέγξαμε την λειτουργία του κωδικοποιητή-αποκωδικοποιητή σε Binary Erasure Channel με πιθανότητα διαγραφής p(e)=0.5 και p(e)=0.2. Τα αποτελέσματα της HW εφαρμογής τα συγκρίνουμε με τα αντίστοιχα αποτελέσματα που πήραμε από την προσομοίωση του Matlab. Επίσης, παρουσιάζουμε κάποια στατιστικά στοιχεία που αφορούν την αξιοποίηση των αποθεμάτων του FPGA, όπως καταγράφονται από το Xilinx.

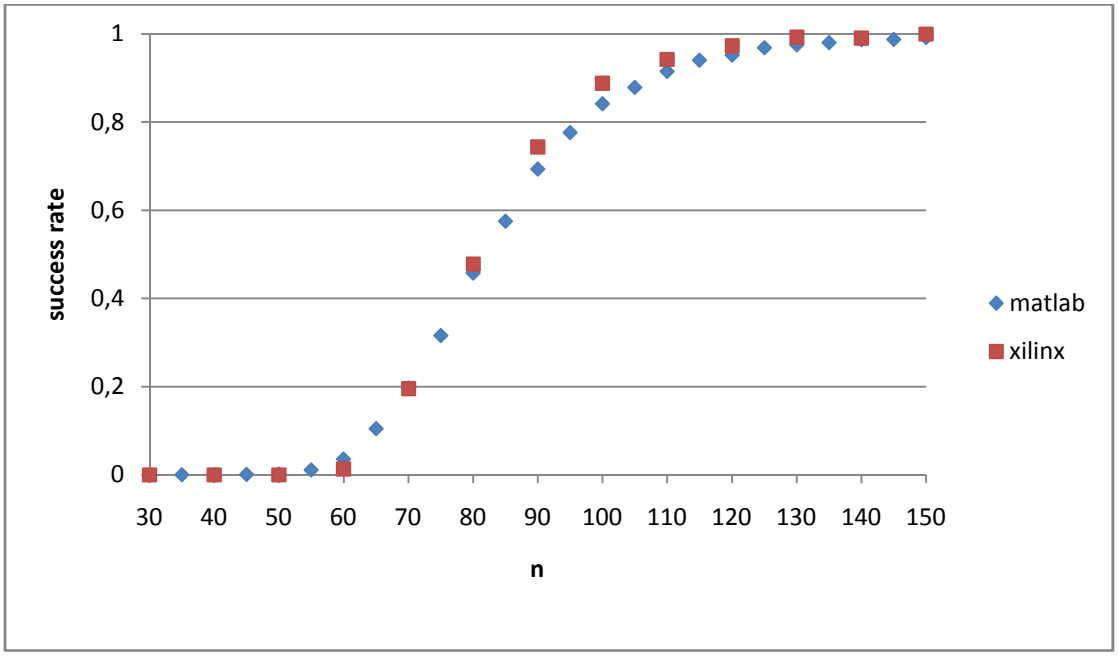
Στην εικόνα 77 δείχνουμε το μπλοκ διάγραμμα του συστήματος με το οποίο βγάλαμε τα αποτελέσματα που θα δούμε παρακάτω. Αυτό που έχει σημασία να παρατηρήσουμε σε αυτό το μπλοκ διάγραμμα είναι ότι τα κωδικοποιημένα σύμβολα περνάνε από το BEC, όχι όμως και τα seed που στέλνει ο κωδικοποιητής στον αποκωδικοποιητή. Σε ένα πραγματικό σύστημα εννοείτε πως και τα seed θα περνούσαν από το ίδιο θορυβώδες κανάλι που περνάνε και τα κωδικοποιημένα σύμβολα. Άρα, σε ένα πραγματικό σύστημα θα έπρεπε να έχουν και τα seed κάποια προστασία από τον θόρυβο. Αν δεν καταφέρει ο αποκωδικοποιητής να λάβει τα seed ή λάβει λανθασμένα seed τότε είναι αδύνατον να γίνει η αποκωδικοποίηση LT.




**Εικόνα 77:** Μπλοκ διάγραμμα του συστήματος με το οποίο κάναμε το πείραμα

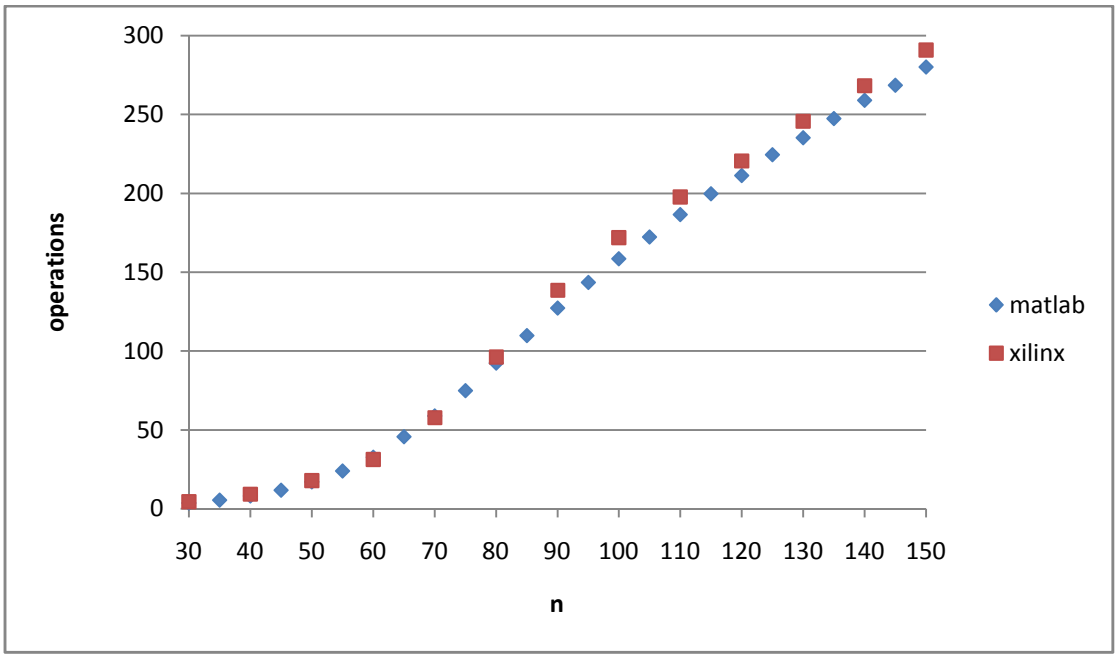
Για όλα τα διαγράμματα αυτής της παραγράφου ισχύουν τα εξής:

- Για κάθε σημείο του διαγράμματος που βγάλαμε με το Matlab, τρέξαμε το πρόγραμμα 10.000 φορές και το αποτέλεσμα είναι το μέσω όρο από αυτά τα τρεξίματα.
- Για κάθε σημείο του διαγράμματος που βγάλαμε από το πρόγραμμα VHDL, τρέξαμε το πρόγραμμα 400 φορές και το αποτέλεσμα είναι το μέσω όρο από αυτά τα τρεξίματα.
- Οι ROM του προγράμματος VHDL περιέχουν 2047 δείγματα, δηλαδή χρησιμοποιήθηκαν Fibonacci των 11 bits.

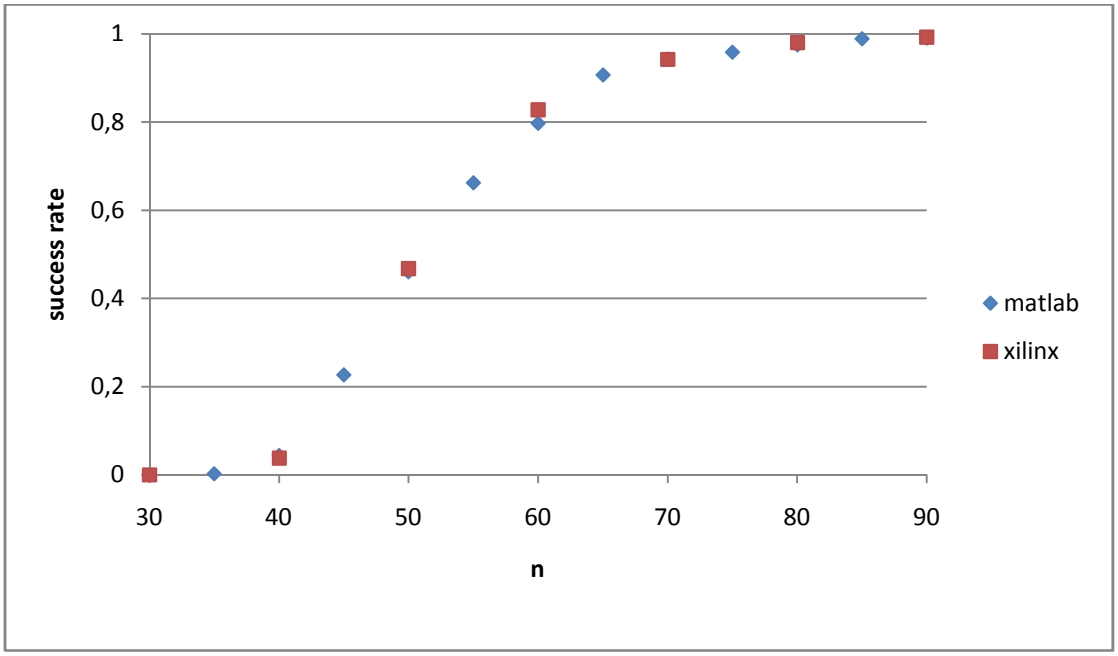



**Εικόνα 78:** Πιθανότητα επιτυχίας της αποκωδικοποίησης, έναντι του n για, k=30, δ=0.5, c=0.05 και p(e)=0.5

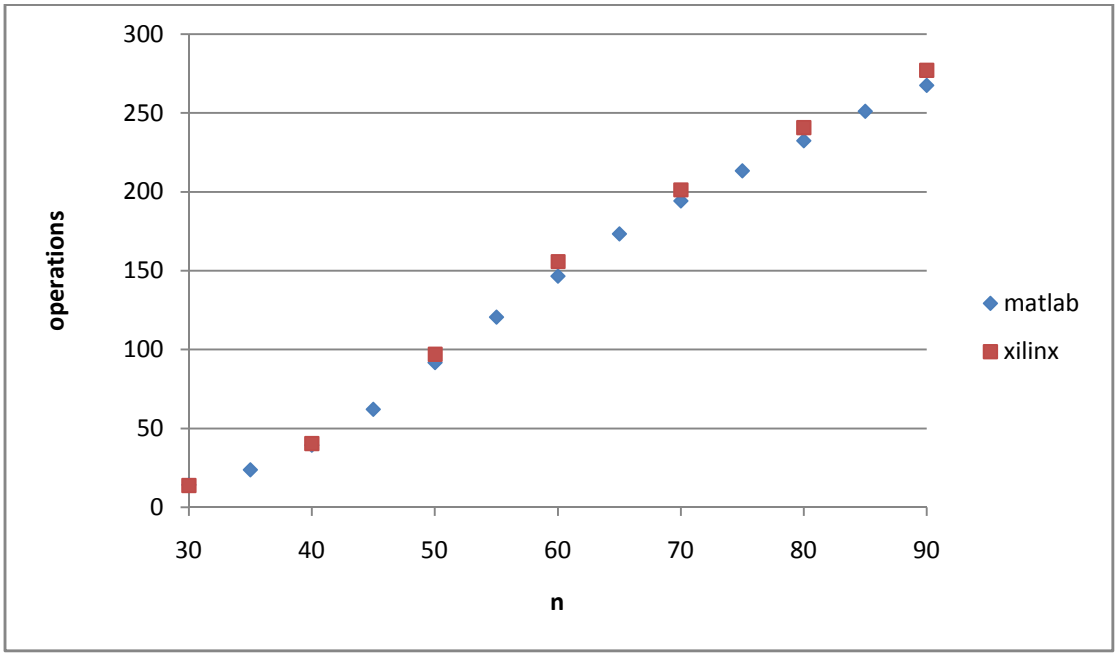
Από τις εικόνες 78 και 79 φαίνεται πως υπάρχει κάποια διαφορά μεταξύ της προγράμματος του Matlab και του προγράμματος στην VHDL. Το πιθανότερο είναι αυτή η διαφορά να ευθύνεται στο γεγονός ότι γίνεται χρήση διαφορετικών RNG. Για την προσομοίωση του Matlab χρησιμοποιούνται οι δικές της ενσωματωμένες random εντολές, ενώ στην VHDL έχει γίνει χρήση του LFSR Fibonacci.




**Εικόνα 79:** Πλήθος λειτουργιών του αποκωδικοποιητή έναντι του n για k=30, δ=0.5, c=0.05 και p(e)=0.5




**Εικόνα 80:** Πιθανότητα επιτυχίας της αποκωδικοποίησης, έναντι του n για, k=30, δ=0.5, c=0.05 και p(e)=0.2




**Εικόνα 81:** Πλήθος λειτουργιών του αποκωδικοποιητή έναντι του n για k=30, δ=0.5, c=0.05 και p(e)=0.2

Για p(e)=0.2 απαιτούνται λιγότερα κωδικοποιημένα σύμβολά από ότι για p(e)=0.5, προκειμένου να έχουμε μεγάλες πιθανότητες επιτυχούς αποκωδικοποίησης.

Παρακάτω μελετάμε και συγκρίνουμε την απόδοση του κωδικοποιητή και του αποκωδικοποιητή με την βοήθεια κάποιων τιμών και στατιστικών στοιχείων που μας δίνει το Xilinx. Τα παρακάτω στοιχεία αφορούν την εφαρμογή του κωδικοποιητή και αποκωδικοποιητή LT για k=50 και n=100 στο Virtex 5, και πιο συγκεκριμένα στο μοντέλο XC5VLX110T-2FF1136.

**Πίνακας 9:** Αξιοποίηση του Virtex 5 από τον LT κωδικοποιητή για k=50 και n=100

| **Χρήση των Slice Logic** | **Χρησιμοποιούνται** | **Διαθέσιμα** | **Ποσοστό που χρησιμοποιείται** |
|---|---|---|---|
| Ως Slice Registers | 307 | 69.120 | 1% |
| Ως Slice LUT's | 1.585 | 69.120 | 2% |

**Πίνακας 10:** Αξιοποίηση του Virtex 5 από τον LT αποκωδικοποιητή για k=50 και n=100

| **Χρήση των Slice Logic** | **Χρησιμοποιούνται** | **Διαθέσιμα** | **Ποσοστό που χρησιμοποιείται** |
|---|---|---|---|
| Ως Slice Registers | 5.313 | 69.120 | 7% |
| Ως Slice LUT's | 12.234 | 69.120 | 17% |

Συγκρίνοντας τον πίνακα 9 με τον πίνακα 10 καταλαβαίνουμε ότι ο αποκωδικοποιητής που έχουμε φτιάξει είναι πολύ πιο απαιτητικός από των κωδικοποιητή. Αυτό έχει σαν αποτέλεσμα να μην μας επιτρέπει να δουλέψουμε το σύστημα για μεγάλα n και k.

**Πίνακας 11:** Διάρκεια επεξεργασίας (CPU time – process time) για k=50 και n=100

| | **LT Κωδικοποιητής** | **LT αποκωδικοποιητής** |
|---|---|---|
| **Διάρκεια για το Synthesis** | 736.58 secs | 4867.94 secs |
| **Διάρκεια για το Map** | 1 mins 7 secs | 9 mins 49 secs |
| **Διάρκεια για το Place and Route** | 22 secs | 31 mins 44 secs |

Επιπλέον, παρατηρώντας τον πίνακα 11 καταλαβαίνουμε ότι το implement design του αποκωδικοποιητή διαρκεί πολύ. Και αυτό κάνει άβολη την χρήση του.

Ο λόγος που συμβαίνουν τα παραπάνω είναι εξαιτίας της δημιουργίας του πίνακα $M_c$ ο οποίος για την παραγωγή του απαιτεί $n \cdot k$ registers, αλλά και εξαιτίας της διαδικασίας της αποκωδικοποίησης η οποία απαιτεί $2 \cdot n$ πολυπλέκτες (k προς 1) του ενός bit, και τέσσερις πολυπλέκτες (n προς 1) του ενός bit.

# Κεφάλαιο 6 - Συμπεράσματα

**6.1 - Σύνοψη**

Στόχοι αυτής της εργασίας ήταν:

1. Να κατανοήσουμε τη θεωρία των LT κωδίκων και το πεδίο εφαρμογής τους.
2. Να εφαρμόσουμε την κωδικοποίηση - αποκωδικοποίηση LT κωδίκων σε Matlab και να εξετάσουμε τις διαφοροποιήσεις που προκαλούν στην επίδοση του συστήματος οι διάφορες παράμετροι.
3. Να αναπτύξουμε μια πειραματική εφαρμογή ενός συστήματος με LT κώδικες σε VHDL.
4. Να συγκρίνουμε τα αποτελέσματα της προσομοίωσης σε Matlab με τα αποτελέσματα που παίρνουμε σε FPGA.
5. Να διερευνήσουμε τα προβλήματα και τις απαιτήσεις της ανάπτυξης σε VHDL έτσι ώστε να καταστεί δυνατή η υλοποίηση ενός πραγματικού κωδικοποιητή - αποκωδικοποιητή σε FPGA.

Στην εισαγωγή της εργασίας γίνεται μία περιγραφή των LT κωδίκων και των Fountain κωδίκων γενικότερα καθώς και κάποιον πιθανών εμπορικών εφαρμογών τους, ώστε να αποκτήσουμε μία γενική ιδέα για τις ιδιαιτερότητές τους, την χρήση τους και τον χώρο εφαρμογής τους. Στην συνέχεια κάνουμε μία θεωρητική ανάλυση της LT κωδικοποίησης – αποκωδικοποίησης, και για την καλύτερη κατανόησή τους λύνουμε ένα παράδειγμα. Κάνουμε ανάλύση της κατανομής Robust Soliton Distribution, η οποία παίζει καθοριστικό ρόλο στον LT κώδικα, και των παραμέτρων της c και δ. Βλέπουμε πως μεταβάλλοντας το c και το δ καταφέρνουμε να μεταβάλουμε την μορφή της Robust Soliton Distribution. Παρουσιάζουμε και αναλύουμε το μπλοκ διάγραμμα του προγράμματος του LT κωδικοποιητή - αποκωδικοποιητή που γράψαμε στο Matlab και με το οποίο βγάλαμε τα αποτελέσματα της προσομοίωσης. Κάναμε προσομοίωση για διάφορες τιμές του c και του δ και είδαμε ότι μεταβάλλοντας αυτές τις παραμέτρους μπορούμε να μεταβάλουμε το μέσο όρο της πιθανότητας πετυχημένης αποκωδικοποίησης και το μέσω όρο των λειτουργιών του αποκωδικοποιητή. Δοκιμάσαμε διάφορες τιμές για τον αριθμό των συμβόλων πληροφορίας k και είδαμε αυτό που περιγράψαμε και θεωρητικά, δηλαδή ότι όσο αυξάνουμε το k αυξάνεται η απόδοση του LT κώδικα. Το k όπως είπαμε είναι ο αριθμός τον συμβόλων πληροφορίας, δηλαδή σε μία πραγματική εφαρμογή εφόσον θα γνωρίζαμε το μέσω όρο του k, θα επιλέγαμε στην συνέχεια τις

τιμές του c και δ ώστε να πετύχουμε τον αποδοτικότερο συνδυασμό. Στο τελευταίο μέρος της εργασίας αναλύουμε την πειραματική Hardware εφαρμογή που πραγματοποιήσαμε σε FPGA με την γλώσσα VHDL στο Xilinx. Αναλύουμε την αρχιτεκτονική δομή του LT κωδικοποιητή - αποκωδικοποιητή και των Units από τα οποία αποτελούνται. Με την βοήθεια παραδείγματος περιγράφουμε την διαδικασία της κωδικοποίησης - αποκωδικοποίησης για να γίνει καλύτερα κατανοητή. Παρουσιάζουμε τις κυματομορφές όπως τις παίρνουμε από το Modelsim-SE VHDL του Xilinx ώστε να δείξουμε τον τρόπο επικοινωνίας που υπάρχει μεταξύ των εσωτερικών Units που αποτελούν τον κωδικοποιητή - αποκωδικοποιητή, αλλά και συνολικά τα σήματα εισόδου - εξόδου ολόκληρου του συστήματος. Κάνουμε πείραμα αυτής της εφαρμογής σε BEC με πιθανότητα διαγραφής 0.5 και 0.2. Το ίδιο πείραμα κάναμε και με την προσομοίωση στο Matlab. Συγκρίνουμε τα αποτελέσματα και καταλήγουμε πως οι καμπύλες των αποτελεσμάτων έχουν κάποια διαφορά η οποία ευθύνεται στη χρήση διαφορετικών ψευδοτυχαίων μηχανών που γίνεται. Τέλος, αναφέρουμε και τεκμηριώνουμε πως η υλοποίηση του αποκωδικοποιητή είναι πολύ πιο απαιτητική και πολύπλοκη από αυτήν του κωδικοποιητή.

### 6.2 - Προτάσεις για περεταίρω εργασία

Σε αυτή την εργασία κάναμε μία πειραματική Hardware εφαρμογή σε FPGA του LT κώδικα. Για την υλοποίηση όμως μίας εμπορικής εφαρμογής σε FPGA θα πρέπει να διερευνηθούν κι άλλα ζητήματα, κάποια από τα οποία αναφέρουμε παρακάτω:

- Θα πρέπει να γραφεί μία λιγότερο απαιτητική διαδικασία αποκωδικοποίησης ώστε να υπάρχει δυνατότητα να δουλέψει το σύστημα για μεγάλο αριθμό συμβόλων πληροφορίας k (>1.000). Δηλαδή, θα πρέπει να αποφευχθεί ο δισδιάστατος γεννήτορας πίνακας που υπάρχει στον αποκωδικοποιητή και να αντικατασταθεί με RAM's.
- Η πειραματική εφαρμογή που κάναμε δεν είναι πραγματικά Rateless. Για n μπορούμε να βάλουμε έναν οποιοδήποτε αριθμό αλλά θα πρέπει να τον καθορίσουμε εξαρχής. Δηλαδή πριν γίνει το implement design. Αυτό σημαίνει πως για διαφορετικές τιμές του n απαιτείται και νέο implement design. Σε μία εμπορική υλοποίηση σε FPGA θα προκαθορίζαμε μόνο το k και τα στοιχεία των ROM. Στην συνέχεια θα χρειαζόταν να γίνει μόνο μία φορά το implement design. Από τον κωδικοποιητή θα είχαμε την δυνατότητα να ελέγχουμε άμεσα την παραγωγή των κωδικοποιημένων συμβόλων n. Δηλαδή θα μπορούσαμε να στείλουμε πχ 10 κωδικοποιημένα σύμβολα και να δοκιμάζαμε αν μπορεί να γίνει αποκωδικοποίηση. Αν δεν μπορούσε, τότε θα είχαμε την δυνατότητα να στείλουμε άλλο ένα κωδικοποιημένο σύμβολο και να ξανά-επιχειρήσουμε να κάνουμε αποκωδικοποίηση κ.ο.κ.

- Το σύστημα θα πρέπει να δουλεύει σε κανάλια με πραγματικό θόρυβο, όχι μόνο σε BEC. Δηλαδή, θα πρέπει ο αποκωδικοποιητής να συγκρίνει την ισχύ των λαμβανόμενων συμβόλων με κάποιο προκαθορισμένο κατώφλι και στην συνέχεια να αποφασίζει αν το σύμβολο που έλαβε είναι διαγραμμένο ή όχι.

**Επίλογος**

*Μέσα από αυτήν εργασία είδα ποιες είναι οι δυσκολίες που εμφανίζονται όταν θέλουμε να κάνουμε Hardware εφαρμογή ενός συστήματος που περιγράφεται θεωρητικά. Υπάρχουν πολλοί περιορισμοί τους οποίους συνήθως δεν αντιλαμβανόμαστε ή δεν μας απασχολούν όταν κάνουμε θεωρητική ανάλυση ή προσομοίωση. Η VHDL είναι μία δύσκολη γλώσσα, και σίγουρα έξι μήνες δεν αρκούν για να την μάθει κανείς. Όμως νομίζω πως έκανα μία καλή αρχή.*

## ΒΙΒΛΙΟΓΡΑΦΙΑ - ΑΝΑΦΟΡΕΣ


[1] Γεώργιος – Ελευθέριος Χ. Λαμπρινάκος, Διπλωματική εργασία "Μελέτη και Προσομοίωση Τεχνικών Κωδικοποίησης Καναλιού για Συστήματα Ασυρμάτων Επικοινωνιών Νέας Γενιάς", Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα Οκτώβριος 2007.

[2] Michael Luby "LT Codes" Proceedings of the ACM Symposium on the Foundations of Computer Science (FOCS), November 2002.

[3] Wikipedia. (2012,Oct.). *Luby Transform code*. [Online]. Available:

http:// en.wikipedia.org/wiki/luby_transform_code

[4] D. J. MacKay, "*Information Theory, Inference, and Lerning Algorithms*", Cambridge University Press, 2003, ch. 50.

[5] Marta Alvarez Guede M.Sc. Thesis "Optimization of the Belief Propagation algorithm for Luby Transform decoding over the Binary Erasure Channel" 2011.

[6] Michael Mitzenmacher "Digital Fountains: A Survey and Look Forward" Harvard University Division of Engineering and Applied Sciences 2004.

[7] Hiroaki Inoue, Takuma Kyo, Eiji Okamoto, Yozo Shoji, Yoshihisa Takayama, and Morio Toyoshima " A LT-Based Multi-Rate Transmission Scheme with Fast Decoding for Satellite Laser Communication" Proc. International Conference on Space Optical Systems and Applications (ICSOS) 2012, 10-3, Ajaccio, Corsica, France, October 9-12 (2012).

[8] Han Wang, MSc THESIS "Hardware Designs for LT Coding CAS-MS-2006-02".

[9] Hanady Hussien, Khaled A. Shehata, Salwa El Ramly, and Nihal M. S. Tawfik, "Design of a Merged Algorithm for Luby Transform Decoder" International Journal of Computer and Communication Engineering, Vol. 1, No. 3, September 2012.

[10] Srinath Puducheri, Jörg Kliewer, Senior Member, IEEE, and Thomas E. Fuja, Fellow, IEEE "The Design and Performance of Distributed LT Codes" IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 53, NO. 10, OCTOBER 2007.

[11] Chris Harrelson, Lawrence Ip, Wei Wang, "Limited Randomness LT Codes".

[12] P. Cataldi, M. Shatarski, M. Grangetto, and E. Magli, “Implementation and performance evaluation of LT and Raptor codes for multimedia applications,” IEEE International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), Pasadena, USA, December, 2006.

[13] E. A. Bodine and M. K. Cheng, “Characterization of luby transform codes with small message size for low-latency decoding,” in Conference on Communication, IEEE, 2008.

[14] Jaco du Toit and Riaan Wolhuter Department of Electrical and Electronic Engineering Stellenbosch University Stellenbosch, South Africa “A Practical Implementation of Fountain Codes over WiMAX Networks withan Optimised Probabilistic Degree Distribution”, ICSNC 2011 : The Sixth International Conference on Systems and Networks Communications.

[15] Christopher John Botha “Designing Luby Transform codes as an application layer reliability mechanism for media streaming over IP networks” Electrical and Electronic Engineering Sciences at the University of Johannesburg, November 2008.

[16] Ravi Palanki and Jonathan S. Yedidia “Rateless Codes on Noisy Channels” MITSUBISHI ELECTRIC RESEARCH LABORATORIES TR-2003-124 April 2004.

[17] Wikipedia. (2013,Mar.). *Binary Erasure Channel*. [Online]. Available: http://en.wikipedia.org/wiki/Binary_erasure_channel

[18] Wikipedia. (2014,Jan.). *Linear feedback shift*. [Online]. Available: http://en.wikipedia.org/wiki/Linear_feedback_shift_register

[19] Xilinx Logicore Linear Feedback Shift Register v3.0 Product Specification DS257 (v1.0) March 28, 2003.

[20] Volnei A. Pedroni, *Σχεδιασμός κυκλωμάτων με τη VHDL*, Αθήνα, Εκδώσεις Κλειδάριθμος, 2007, σελ. 248-254.

[21] VHDL coding tips and tricks. (2011 Jan.). *Block and distributed RAM's on Xilinx FPGA's* [Online]. Available: http://vhdlguru.blogspot.gr/2011/01/block-and-distributed-rams-on-xilinx.html

[22] Muhammad Usman Karim Khan “FPGA Implementation Framework for LT CODEC” Department of Electronics Engineering Politecnico di Torino, Italy Anno accademico 2010-2011.