

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ: Ηλεκτρονικής και Υπολογιστών
ΕΡΓΑΣΤΗΡΙΟ VLSI

Διπλωματική Εργασία
του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και
Τεχνολογίας Υπολογιστών της Πολυτεχνικής Σχολής του
Πανεπιστημίου Πατρών

Ισλάμ Αλέξανδρος Ελ-Κάντι

Αριθμός Μητρώου: 227498

Θέμα

**«Αλγόριθμοι, αρχιτεκτονικές υλικού και υλοποίηση σε
FPGA συστημάτων διόρθωσης λαθών βασισμένων σε
Polar Codes»**

Επιβλέπων
Βασίλειος Παλιουράς

Αριθμός Διπλωματικής Εργασίας: . . .

Πάτρα, Ιούλιος 2016

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η Διπλωματική Εργασία με θέμα

**«Αλγόριθμοι, αρχιτεκτονικές υλικού και υλοποίηση σε
FPGA συστημάτων διόρθωσης λαθών βασισμένων σε
Polar Codes»**

Του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας
Υπολογιστών

Αλέξανδρος Ελ-Κάντι του Μωχάμεντ

Αριθμός Μητρώου: 227498

Παρουσιάστηκε δημόσια και εξετάστηκε στο Τμήμα Ηλεκτρολόγων
Μηχανικών και Τεχνολογίας Υπολογιστών στις
...../...../.....

Ο Επιβλέπων

Ο Διευθυντής του Τομέα

Αριθμός Διπλωματικής Εργασίας: . . .

Θέμα: «Αλγόριθμοι, αρχιτεκτονικές υλικού και υλοποίηση σε FPGA συστημάτων διόρθωσης λαθών, βασισμένων σε Polar Codes »

Φοιτητής: Αλέξανδρος Ελ-Κάντι

Επιβλέπων: Βασίλειος Παλιουράς

Περίληψη

Η παρούσα διπλωματική εργασία ασχολείται με την μελέτη συστημάτων FEC (Forward Error Correction) βασισμένα στους Polar Codes καθώς και με την υλοποίηση τους σε FPGA. Στα πρώτα δύο κεφάλαια αναλύονται βασικές έννοιες καθώς και η βασική ιδέα του σχήματος κωδικοποίησης των Polar Codes. Στο τρίτο κεφάλαιο γίνεται ανάλυση των επιμέρους τμημάτων που τους αποτελούν, καθώς και σπηλαίωση στην βελτιστοποίηση τους. Στο κεφάλαιο τέσσερα γίνεται αναφορά στην υλοποίηση του συστήματος σε FPGA καθώς και παρουσίαση των αποτελεσμάτων που πάρθηκαν. Τέλος στο κεφάλαιο πέντε καταγράφονται τα συμπεράσματα που προέκυψαν από την μελέτη, την εξομοίωση και την υλοποίηση καθώς και μελλοντικοί στόχοι.

Περιεχόμενα

Κεφάλαιο 1. Εισαγωγή.....	9
1.1 Το επικοινωνιακό μοντέλο	9
1.2 Forward Error Correction (FEC) Κώδικες.....	9
1.3 Χωρητικότητα Καναλιού.....	11
Κεφάλαιο 2. Polar Codes	12
2.1 Η Βασική Ιδέα.....	13
2.2 Πόλωση Καναλιού (Channel Polarization).....	13
2.3 Κωδικοποίηση (Encoding)	18
2.4 Decoding Algorithms (Αλγόριθμοι Αποκωδικοποίησης)	19
2.4.1 Successive-Cancellation Decoding Algorithm (SC Αλγόριθμος).....	19
2.4.2 Successive-Cancellation List Decoding Algorithm (SCLD Αλγόριθμος)	22
2.4.3 CRC Aided SCLD (CA-SCLD Αλγόριθμος)	24
2.5 Αποτελέσματα Προσομοιώσεων.....	27
Κεφάλαιο 3. Σχεδιασμός Αποκωδικοποιητή	28
3.1 Συναρτήσεις f και g	28
3.2 Αναπαράσταση LLR Αποκωδικοποιητή (Two's Complement)	29
3.3 Υλοποίηση Στοιχείων Επεξεργασίας f και g	29
3.4 Μερικά αθροίσματα \hat{s}	31
Κεφάλαιο 4. Υλοποίηση FPGA και επαλήθευση	32
4.1 Κωδικοποιητής (Encoder)	33
4.2 Κανάλι (Channel)	33
4.3 Αποκωδικοποιητής (Decoder)	33
4.3.1 Scheduling	33
4.3.2 Παραγωγή Μερικών Αθροισμάτων	36
4.4 Finite State Machine (FSM)	36
4.5 Παραμετρική Ροή Σχεδιασμού	37
4.6 Αποτελέσματα Υλοποίησης	42
4.6.1 Encoder.....	42

ΠΕΡΙΕΧΟΜΕΝΑ

4.6.2 Decoder	42
4.6.3 Encoder-Decoder με το Κανάλι	48
4.6.4 Αποτελέσματα BER	53
Κεφάλαιο 5. Συμπεράσματα	61

Κεφάλαιο 1. Εισαγωγή

1.1 Το επικοινωνιακό μοντέλο

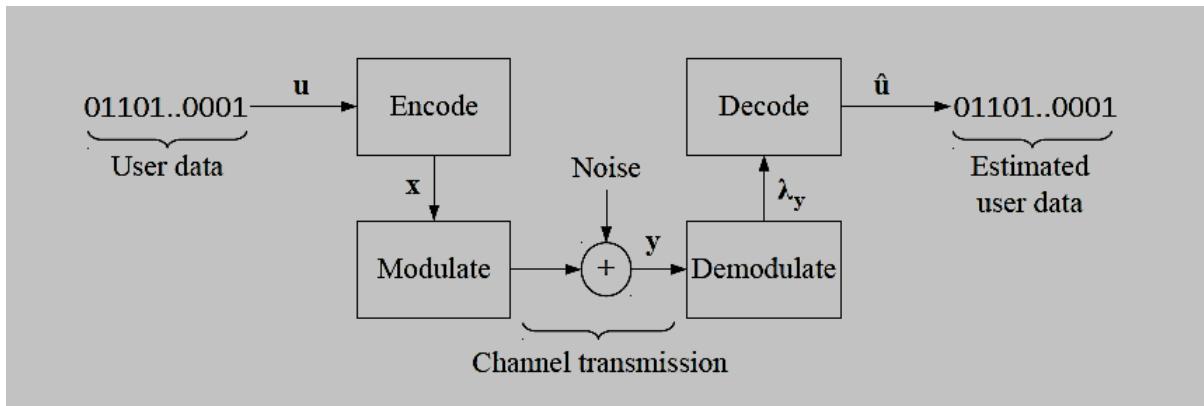
Το επικοινωνιακό μοντέλο χρησιμοποιείται από έναν πομπό και έναν δέκτη οι οποίοι επιθυμούν να επικοινωνήσουν. Αποτελείται από έξι βασικά μέρη:

- 1) Την πληροφορία: Κάθε που μειώνει την αβεβαιότητα
- 2) Το κανάλι: Τα φυσικά μέσα τα οποία μετασχηματίζουν τα σήματα/σημεία
- 3) Τον θόρυβο: Οποδήποτε προστίθεται στο σήμα μεταξύ του μετασχηματισμού και της λήψης του μηνύματος και δεν προέρχεται από την πηγή
- 4) Το μέσο: Τα τεχνικά ή φυσικά μέσα μετατροπής των μηνυμάτων σε σήματα που μπορούν να μεταδοθούν. Οποδήποτε παράγει, διακινεί και επεξεργάζεται σήματα.
- 5) Ο κώδικας: Κώδικας είναι ένα σύστημα κανόνων, σημάτων και συμβάσεων γνωστών και στον πομπό και στον δέκτη που ορίζουν πώς αυτά τα σήματα χρησιμοποιούνται και συνδυάζονται
- 6) Ανάδραση (feedback) : Η μεταφορά της αντίδρασης του δέκτη πίσω στον πομπό.

1.2 Forward Error Correction (FEC) Κώδικες

Ένα σχήμα διόρθωσης λαθών μπορεί να χρησιμοποιηθεί για την αποφυγή σφαλμάτων, λόγω θορύβου, πάνω σε ένα κανάλι μετάδοσης. Όταν ο δέκτης πρέπει να διορθώσει αυτά τα σφάλματα δίχως να επικοινωνήσει ξανά με τον πομπό, το σχήμα αυτό αναφέρεται ως *forward error-correction scheme*. Ένα απλοποιημένο τέτοιου είδους κανάλι βλέπουμε στο Σχήμα 1.1. Το σχήμα αυτό εισάγει διάφορες ένοιες που θα χρησιμοποιηθούν από εδώ και πέρα. Στο μοντέλο αυτό ο χρήστης προσπαθεί να μεταδώσει ένα δυαδικό μήνυμα, το διάνυσμα \mathbf{u} . Αυτό το K-bit μήνυμα κωδικοποιείται κατάλληλα στον κωδικοποιητή (Encoder) μέσω ενός κώδικα ανίχνευσης λαθών (error correction code) και παράγει ένα δυαδικό μήνυμα μήκους N-bit, το διάνυσμα-κωδική λέξη \mathbf{x} . Στη συνέχεια αυτή η πλέον κωδική λέξη περνάει από ένα κύκλωμα διαμόρφωσης (Modulator) και αποστέλλεται στο κανάλι μας (Channel) με την μορφή αναλογικού σήματος. Το κανάλι πλέον προσθέτει στο αναλογικό σήμα θόρυβο και φτάνει στο δέκτη, διάνυσμα \mathbf{y} . Ο δέκτης από την μεριά του, δέχεται το αναλογικό σήμα με τον θόρυβο και το εισάγει στον αποδιαμορφωτή (Demodulator), το οποίο το μετατρέπει σε ένα πιθανό διάνυσμα \mathbf{ly} (likelihood vector). Στη συνέχεια το εισάγει στον αποκωδικοποιητή (Decoder). Ο αποκωδικοποιητής ανακτά το αρχικό δυαδικό μήνυμα

μέσω του πιθανού διανύσματος λ_y και παράγει την εκπιμώμενη κωδική λέξη \hat{u} . Ο λόγος $R = K/N$ αναφέρεται ως ρυθμός κώδικα (code rate) και αντικατοπτρίζει την περιπή πληροφορία που εισάγεται από τον εκάστοτε κώδικα. Στην περίπτωση μας θα θεωρήσουμε $R=0.5$.



Σχήμα 1.1: Στάδια επικοινωνιακού μοντέλου

Το κανάλι μας ανήκει στην κατηγορία του Δυαδικού Διακριτού Χωρίς Μνήμη Καναλιού (Binary Discrete Memoryless Channel - BDMC) και θα αναφέρεται από εδώ και πέρα ως W με πιθανότητα μετάδοσης $W(Y|X)$.

Αρκετά μοντέλα υπάρχουν για την αναπαράσταση του θορύβου πάνω σε ένα κανάλι επικοινωνίας. Στην παρούσα διπλωματική γίνεται χρήση του μοντέλου του προσθετικού λευκού Γκαουσιανού Θορύβου (additive white Gaussian Noise AWGN). Στο οποίο ο θόρυβος ακολουθεί μια κανονική κατανομή $N(0, \sigma^2)$ με μέση τιμή 0 και διακύμανση σ^2 .

Επίσης χρησιμοποιούμε τη δυαδική παλμοδιαμόρφωση φάσης (binary phase-shift keying BPSK). Αυτή η δύο σταδίων διαμόρφωση επιτρέπει την αποστολή ενός bit πληροφορίας ανά χρησιμοποίηση του καναλιού. Το εκάστοτε σύμβολο καθορίζεται από την ακόλουθη αντιστοίχηση, όπου το x_i μετατρέπεται σε ένα αναλογικό σήμα πλάτους 1 ή -1.

$$\text{BPSK}(x_i) \triangleq \begin{cases} 1.0 & \text{when } x_i = 0, \\ -1.0 & \text{otherwise.} \end{cases}$$

Ως εκ τούτου η i-οστή λαμβανόμενη τιμή y_i , αναφερόμενη στο αποστελλόμενο bit x_i , μπορεί να μοντελοποιηθεί ως

$$y_i = \text{BPSK}(x_i) + n_i$$

,όπου n_i , δείγμα από την κατανομή $N(0, \sigma^2)$.

Θεωρώντας την παραπάνω εξίσωση, το σύμβολο y_i που δέχεται ο δέκτης μπορεί να αναπαρασταθεί ως πηλίκο πιθανοτήτων (likelihood-ratio LR)

$$L_{y_i} \triangleq \frac{\Pr(y_i|x_i = 0)}{\Pr(y_i|x_i = 1)} = e^{\frac{2y_i}{\sigma^2}},$$

όπου $\Pr(y_i|x_i)$ είναι η δεσμευμένη πιθανότητα να παρατηρηθεί η έξοδος y_i δεδομένου ότι στάλθηκε η πημή x_i .

1.3 Χωρητικότητα Καναλιού

Μια μεταβλητή για την αξιολόγηση ενός καναλιού είναι η κοινή πληροφορία. Η κοινή πληροφορία ενός BDMC καναλιού W με είσοδο ένα αλφάριθμο $X = \{0,1\}$ ορίζεται ως

$$I(W) \triangleq \frac{1}{2} \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} W(y|x) \log \frac{W(y|x)}{\frac{1}{2}W(y|0) + \frac{1}{2}W(y|1)}$$

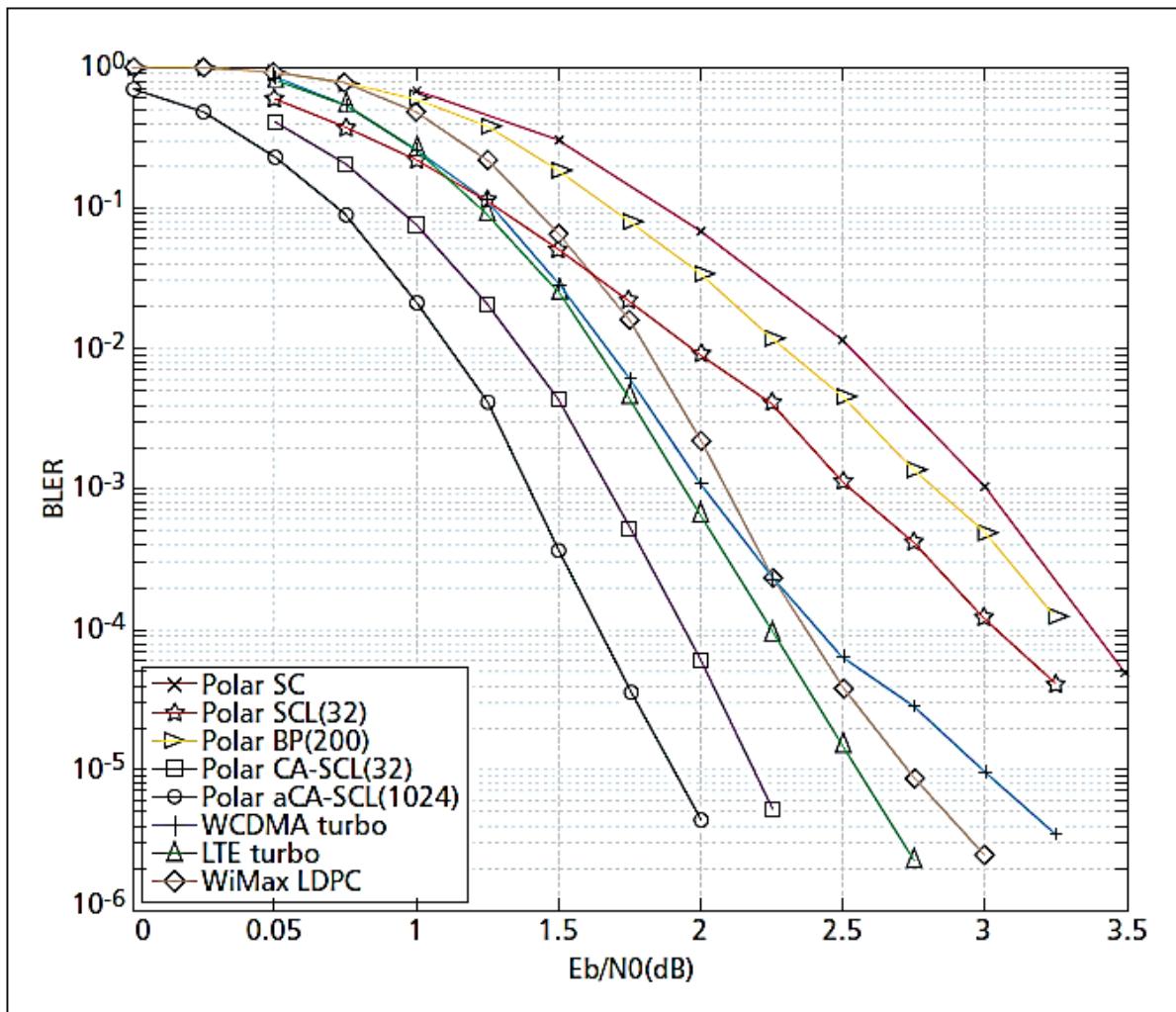
Αξίζει να σημειωθεί ότι η χωρητικότητα $C(W)$ ενός συμμετρικού BDMC ισούται με την κοινή πληροφορία ανάμεσα στις εισόδους και στις εξόδους του καναλιού. Χρησιμοποιώντας λογαρίθμους βάσεως του 2 ισχύει ότι

$$0 \leq C(W) \leq 1$$

Έτσι θα χρησιμοποιήσουμε τη χωρητικότητα $C(W)$ για την αξιολόγηση των καναλιών μας.

Κεφάλαιο 2. Polar Codes

Στις μέρες μας αρκετές υλοποιήσεις και αρχιτεκτονικές έχουν ξεπεράσει τους κώδικες LDPC σε απόδοση διόρθωσης λαθών καθώς και ρυθμού μετάδοσης (throughput rate). Οι αλγόριθμοι των περισσότερων περιγράφονται παρακάτω στην παράγραφο 2.4, οι υπόλοιποι είναι απλώς υβριδικές υλοποιήσεις με συνδυασμό των προηγουμένων.



Σχήμα 2.1: Σύγκρισης αποδόσεων μεταξύ Polar και LDPC κωδικών [2]

Τέλος έχουν προταθεί concatenated υλοποιήσεις με χρήση των polar codes εξωτερικά και εσωτερικά χρήση των LDPC που παρουσιάζουν αρκετό ενδιαφέρον, βλέπε [8]. Το μεγαλύτερο πρόβλημα σε όλα τα παραπάνω είναι ο χώρος που καταλαμβάνουν οι υλοποιήσεις, πράγμα που θα αναφερθεί και θα αναλυθεί στο κεφάλαιο 4.

2.1 Η Βασική Ιδέα

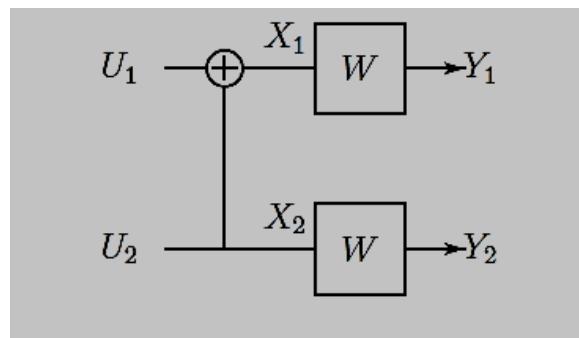
Οι Polar Codes, ανήκουν σε μια οικογένεια κωδικών FEC γνωστής ως γραμμικοί μπλοκ κώδικες (linear block codes), οι οποίοι αποτελούνται από κωδικές λέξεις που προκύπτουν με την εφαρμογή γραμμικών πράξεων πάνω στο αρχικό διάνυσμα πληροφορίας μας. Υλοποιούνται χρησιμοποιώντας την τεχνική της πόλωσης των καναλιών.

Η βασική ιδέα είναι η μετατροπή του αρχικού μας καναλιού επικοινωνίας σε έναν αριθμό από υπό-κανάλια που είναι ή τέλεια ή άχρηστα, μέσω της τεχνικής της πόλωσης η οποία θα εξηγηθεί στην συνέχεια. Αξιοποιώντας το παραπάνω επιλέγουμε τα τέλεια κανάλια για μετάδοση πληροφορίας και “παγώνουμε” τα αντίστοιχα άχρηστα κανάλια σε μια σταθερή πυρή που γνωρίζουν και ο πομπός και ο δέκτης. Αυτό σημαίνει ότι θεωρούμε τη μετάδοση σταθερής γνωστής πυρής μέσω των καναλιών αυτών, άρα δεν υπάρχει πιθανότητα λάθους. Έτσι μπορούν να δημιουργηθούν κώδικες με χαμηλό ποσοστό σφάλματος (error rate).

2.2 Πόλωση Καναλιού (Channel Polarization)

Η πόλωση καναλιού είναι μια διαδικασία που παράγει N κανάλια από N ανεξάρτητα αντίγραφα ενός BDMC W των οποίων η χωρητικότητα $C(W)$ είναι είτε κοντά στο 1 (τέλειο κανάλι) είτε κοντά στο 0 (άχρηστο κανάλι) καθώς το μήκος τους N αυξάνεται. Η πόλωση του καναλιού χωρίζεται σε δύο στάδια, όπως έχει προταθεί από τον Arikan [1]:

- 1) Συνδυασμός καναλιών (Channel combining): Σε αυτήν τη φάση, τα N αντίγραφα των BDMC συνδυάζονται με έναν αναδρομικό τρόπο και μέσω μίας 1-1 αντιστοίχησης G_N σε N βήματα για να φτιάξουνε ένα διάνυσμα καναλιού W_N , όπου $N = 2^n$. Αξίζει να σημειωθεί ότι ο μετασχηματισμός G_N μπορεί να είναι οποιοσδήποτε, αλλά από απόψεως υλοποίησης και χαμηλής πολυπλοκότητας, προτείνεται ο παρακάτω μετασχηματισμός [1]:



Σχήμα 2.2: Συνδυασμός δύο καναλιών W

Όπως μπορεί να δει κάποιος, δύο κανάλια W συνδυάζονται για να δημιουργήσουν ένα νέο διάνυσμα καναλιού $\mathbf{W}_2 : \{0,1\}^2 \rightarrow \mathbb{Y}^2$. Εφόσον ο γραμμικός μετασχηματισμός \mathbf{G}_N μεταξύ των (U_1, U_2) και (X_1, X_2) είναι 1-1 αντιστοίχηση έχουμε:

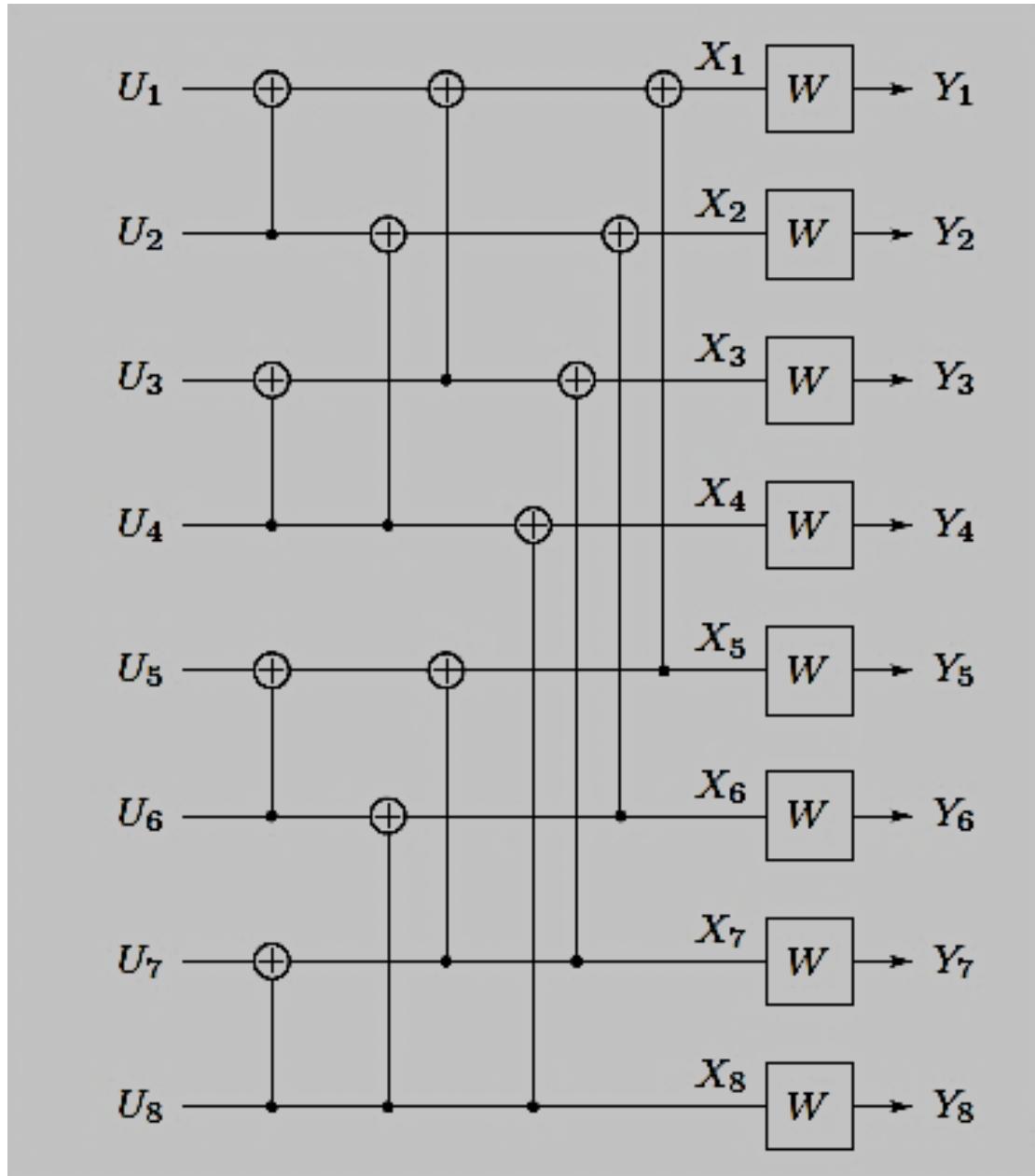
$$W_2(y_1, y_2 | u_1, u_2) = W(y_1 | u_1 \oplus u_2)W(y_2 | u_2)$$

$$I(U_1, U_2; Y_1, Y_2) = I(X_1, X_2; Y_1, Y_2) = 2I(W)$$

Ο συνδυασμός καναλιών εν γένει για $N=2^n$ γίνεται αναδρομικά με τον ακόλουθο τρόπο και παράγεται ένα διάνυσμα καναλιών \mathbf{W}_N :

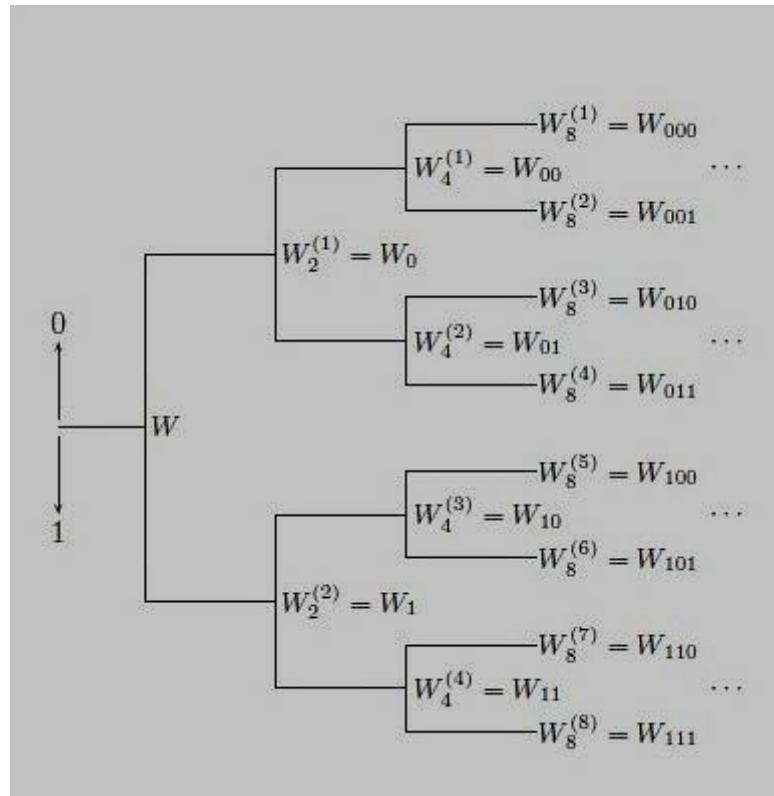
$$W_N(\mathbf{Y}^N | \mathbf{U}^N) = W_{N/2}(\mathbf{Y}^{N/2} | \mathbf{U}_o^N \oplus \mathbf{U}_e^N)W_{N/2}(\mathbf{Y}_{N/2+1}^N | \mathbf{U}_e^N),$$

Για παράδειγμα για $N=8$ έχουμε το διάνυσμα καναλιών \mathbf{W}_8 , αποτελούμενο από οχτώ ανεξάρτητα κανάλια W , τα οποία απεικονίζονται στο Σχήμα 2.3.



Σχήμα 2.3: Διάνυσμα καναλιού W_8

Μία σχηματική αναπαράσταση της αναδρομικότητας για τη δημιουργία των καναλιών διανυσμάτων φαίνεται στο Σχήμα 2.4



Σχήμα 2.4: Αναδρομική Δομή Καναλιών

- 2) Διαχωρισμός Καναλιών (Channel Splitting): Σε αυτήν τη δεύτερη φάση, το διάνυσμα καναλιών \mathbf{W}_N , που δημιουργήθηκε προηγουμένως, διαχωρίζεται πίσω σε N κανάλια $\mathbf{W}_N^{(i)}: \{0,1\}^N \rightarrow Y^N \times \{0,1\}^{N-i}$ όπου $1 \leq i \leq N$.

Αναφορικά με την κοινή πληροφορία και χρησιμοποιώντας τον κανόνα της αλυσίδας, έχουμε για το Σχήμα 2.2 :

$$I(U_1, U_2; Y_1, Y_2) = I(U_1; Y_1, Y_2) + I(U_2; Y_1, Y_2, U_1).$$

Βλέπουμε ότι το πρώτο μέρος του δεξιού μέλους είναι η κοινή πληροφορία μεταξύ του U_1 και $Y_1 Y_2$. Ας θεωρήσουμε ότι αυτό το κανάλι είναι το $W: \{0,1\}^2 \rightarrow Y^2$. Το δεύτερο μέρος του δεξιού μέλους είναι η κοινή πληροφορία μεταξύ του U_2 και της εξόδου δεδομένου ότι το U_1 είναι γνωστό. Ας θεωρήσουμε ότι αυτό το κανάλι

είναι το W^+ . Έτσι οι πιθανότητες μετάδοσης των δύο αυτών καναλιών είναι σύμφωνα με τον Arikan [1]:

$$W^-(y_1, y_2|u_1) = \frac{1}{2} \sum_{u_2 \in \{0,1\}} W(y_1|u_1 \oplus u_2)W(y_2|u_2)$$

$$W^+(y_1, y_2, u_1|u_2) = \frac{1}{2}W(y_1|u_1 \oplus u_2)W(y_2|u_2)$$

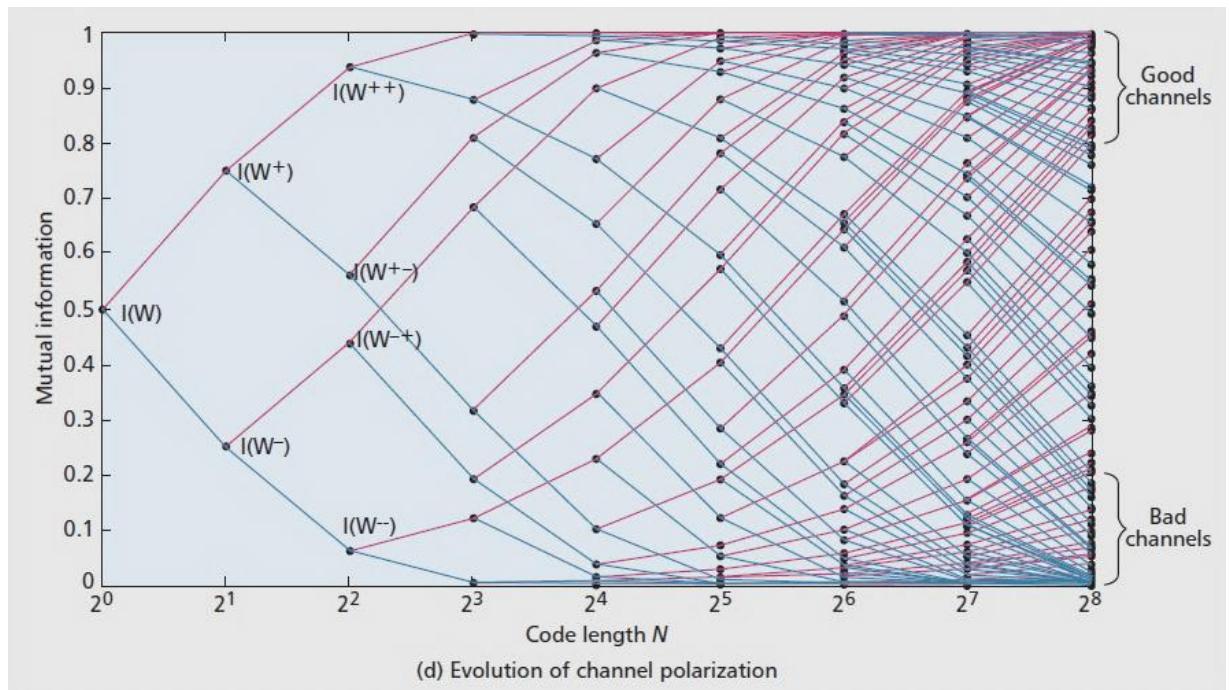
Πιο απλά το διάνυσμα καναλιού \mathbf{W}_2 διαχωρίζεται πίσω σε δύο κανάλια W^- και W^+ . Η ακόλουθη ιδιότητα ισχύει για την κοινή πληροφορία των δύο νέων καναλιών:

$$I(W^+) + I(W^-) = 2I(W)$$

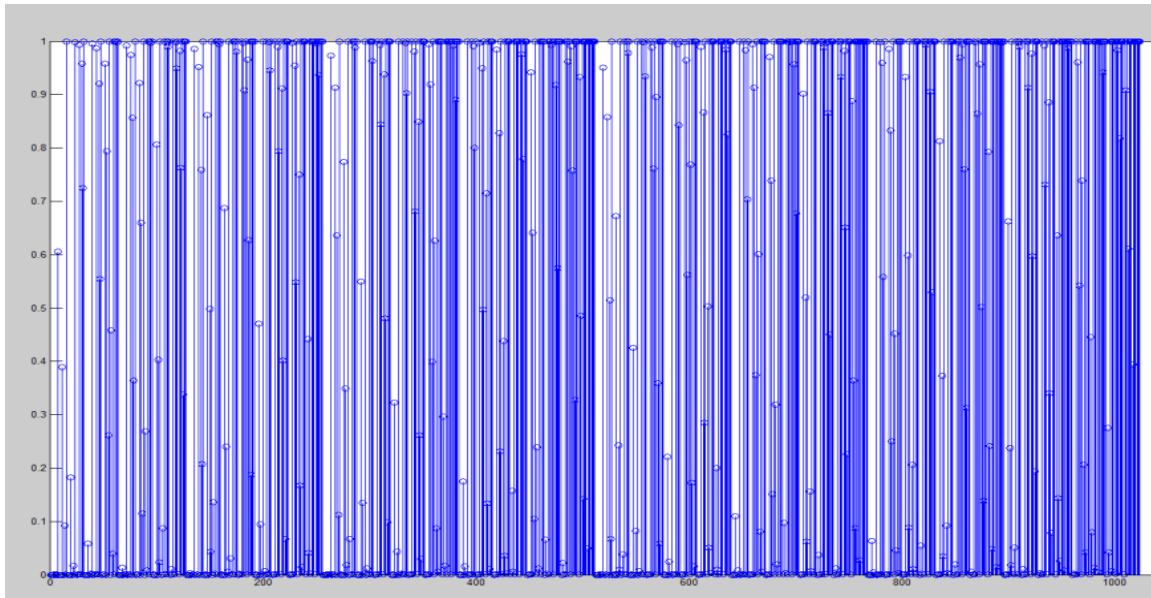
Αξίζει να σημειωθεί ότι $I(W^+) \geq I(W) \geq I(W^-)$.

Βλέπουμε ότι το κανάλι W^+ πηγαίνει την κοινή πληροφορία του προς το 1 και το αντίστοιχο W^- προς το 0, ενώ το άθροισμα τους συνεχίζει να είναι $2I(W)$.

Σκεπτόμενοι αναδρομικά το παραπάνω σενάριο και για κοινή πληροφορία $I(W)=0.5$ έχουμε τα σχήματα 2.5 και 2.6.



Σχήμα 2.5 : Η Πόλωση καναλιών συναρτήσει της χωρητικότητάς τους [2]



Σχήμα 2.6: Οι χωρητικές για N=1024 κανάλια όπως προέκυψαν στην Matlab

Βλέπουμε λοιπόν ότι για μόλις $N=2^8$ ήδη τα αρχικά μας κανάλια έχουν πολωθεί σε έναν ικανοποιητικό βαθμό. Συνεχίζοντας κάποιος για μεγαλύτερα N μπορεί να επιτύχει καλύτερη πόλωση, αναλόγως πάντα και με το code rate επιλογής.

2.3 Κωδικοποίηση (Encoding)

Γενικότερα μια κωδική λέξη στους Polar Codes παρέχεται όπως προαναφέραμε μέσω του μετασχηματισμού :

$$\mathbf{x} \triangleq \mathbf{u} \cdot \mathbf{G}.$$

Παραδοσιακά οι Polar Codes δομούνται με μήκος δυνάμεως του δύο, $N=2^n$. Η γεννήτρια μήτρα τους \mathbf{G}_N , σύμφωνα με τον Arikān [1], δημιουργείται χρησιμοποιώντας την n-οστή δύναμη του Kronecker πάνω στη βασική μήτρα \mathbf{F}_2 :

$$\mathbf{F}_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Για παράδειγμα η γεννήτρια μήτρα για N=8 ορίζεται ως :

$$\mathbf{G} = \mathbf{F}_8 \triangleq (\mathbf{F}_2)^{\otimes 3} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes 3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

Σχήμα 2.7: Γεννήτρια μήτρα \mathbf{G}_8

Όπου \otimes δηλώνει την εκάστοτε δύναμη του Kronecker.

Αυτή η μέθοδος δημιουργεί, όπως αναφέρθηκε στην παράγραφο 2.2, ένα διάνυσμα καναλιών μήκους N όπου τα K μόνο χρησιμοποιούνται για μετάδοση πληροφορίας. Το σύνολο των ιδανικών καναλιών που επιλέγονται για μετάδοση αναφέρεται ως \mathbf{A} (information bits) και τα υπόλοιπα σταθερής πιμής ως \mathbf{Ac} (frozen bits). Έτσι το αρχικό μας διάνυσμα \mathbf{x} επεκτείνεται σε ένα διάνυσμα N θέσεων \mathbf{x} , τοποθετώντας την πληροφορία στις θέσεις i που ανήκουν στο σύνολο \mathbf{A} και παγώνοντας τις υπόλοιπες θέσεις των i, που ανήκουν στο σύνολο \mathbf{Ac} , με μια σταθερή πιμή, συνήθως 0.

2.4 Decoding Algorithms (Αλγόριθμοι Αποκωδικοποίησης)

Υπάρχουν διάφοροι αλγόριθμοι, άλλοι με χαμηλή πολυπλοκότητα και χαμηλότερες επιδόσεις σε ber/fer και άλλοι με μεγαλύτερη πολυπλοκότητα και υψηλότερες αποδόσεις σε ber/fer. Παρακάτω αναφέρονται οι τρεις βασικοί αλγόριθμοι που υπάρχουν. Οι υπόλοιποι αλγόριθμοι που θα συναντήσει κανείς συνδυάζουν τους παρακάτω σε διάφορες υβριδικές μορφές.

2.4.1 Successive-Cancellation Decoding Algorithm (SC Αλγόριθμος)

Στο παράγραφο 2.2 δείξαμε πώς να δημιουργήσουμε πολωμένα κανάλια που είναι είτε θορυβώδη (κακά κανάλια) ή μη θορυβώδη (καλά κανάλια). Έτσι προκύπτει εκ φύσεως το σχήμα κωδικοποίησης, χρησιμοποιώντας τα καλά κανάλια για τη μετάδοση της πληροφορίας μας και τα κακά κανάλια να τα παγώσουμε με μία πιμή, συνήθως 0, και να πούμε στον δέκτη ποια είναι αυτά. Επειδή το εκάστοτε διάνυσμα κανάλι $\mathbf{W}_N^{(i)}$ εξαρτάται από τις πιμές των εξόδων των προηγούμενων καναλιών \mathbf{W} κάπι τέτοιο μας βολεύει. Έτσι προκύπτει ο λεγόμενος αποκωδικοποιητής Successive Cancellation (SC Decoder), ο

οποίος αποκωδικοποιεί τα bits $\mathbf{U}_1, \mathbf{U}_2, \dots, \mathbf{U}_N$ με τη σειρά, διότι κάθε έξοδος χρειάζεται τις αποφάσεις των προηγουμένων $\hat{\mathbf{u}}_i$.

Ο αλγόριθμος αυτός αποκωδικοποίησης ανήκει στην κατηγορία Soft-input/Soft-output (SISO), καθώς καταναλώνει και παράγει πηλίκα πιθανοτήτων LR. Το SC έγκειται στις παρακάτω τρεις τελικές αποφάσεις εξόδου:

$$\hat{u}_i \triangleq \begin{cases} 0, & \text{if } i \in \mathcal{A}_C, \\ 0, & \text{if } \frac{\Pr(\mathbf{y}, \hat{u}_0^{i-1} | u_i=0)}{\Pr(\mathbf{y}, \hat{u}_0^{i-1} | u_i=1)} \geq 1, \\ 1, & \text{otherwise,} \end{cases}$$

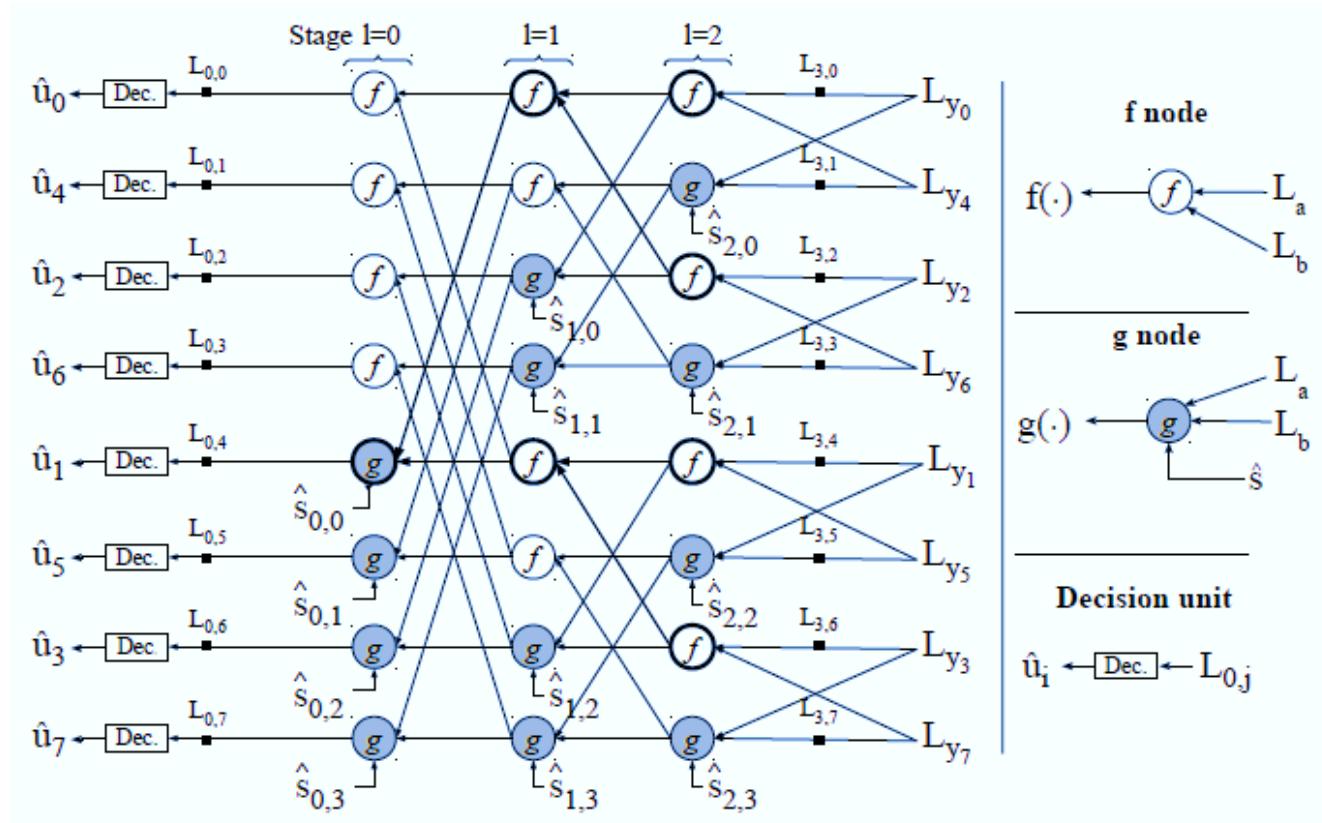
όπου $\frac{\Pr(\mathbf{y}, \hat{u}_0^{i-1} | u_i=0)}{\Pr(\mathbf{y}, \hat{u}_0^{i-1} | u_i=1)}$ είναι το πηλίκο πιθανοτήτων που παράγεται από τον

αποδιαμορφωτή για το bit u_i δεδομένου ότι έχει ληφθεί το διάνυσμα \mathbf{y} και όλα τα προηγούμενα αποκωδικοποιημένα bits $\{\hat{u}_0, \hat{u}_1, \dots, \hat{u}_{i-1}\}$.

Αυτή η αποκωδικοποίηση γίνεται χρησιμοποιώντας την αναδρομική δομή

$$L_{l,i} \triangleq \begin{cases} f(L_{l+1,i}; L_{l+1,i+2^{n-l-1}}) & \text{if } \frac{i}{2^l} \text{ is even,} \\ g(\hat{s}_{l,z}; L_{l+1,i-2^{n-l-1}}; L_{l+1,i}) & \text{otherwise,} \end{cases}$$

και παρουσιάζεται σχηματικά στο Σχήμα 2.8 για N=8.



Σχήμα 2.8: Η δομή γράφου του αποκωδικοποιητή για $N=8$ (Polar Code Decoder) [5][10]

Σε αυτήν τη δομή το $\hat{s}_{l,z}$ είναι το z-οστό μερικό άθροισμα του σταδίου l και $L_{l,i}$ είναι το πηλίκο πιθανότητας LR που παράγεται από τον i-οστό κόμβο του σταδίου l . Για το στάδιο n , τα LR ταυτίζονται με τις εισόδους από το κανάλι μας και χρησιμοποιούνται ως είσοδοι του σταδίου $n-1$. Οι πιές που δεχόμαστε από το κανάλι αναπαριστώνται σε bit reverse order (ανάστροφη αναπαράσταση των bit), με τον παρακάτω τύπο:

$$\text{bitreverse}([a_{m-1} \dots a_1 a_0]) \triangleq [a_0 a_1 \dots a_{m-1}],$$

Όπου $a = [a_m \dots a_1 a_0]$ είναι ένας m-bit δυαδικός δείκτης και a_j είναι το j-οστό λιγότερο σημαντικό bit (LSB).

Τέλος τα bit \hat{u}_i υπολογίζονται εκ του αποτελέσματος του σταδίου 0, δηλαδή μέσω των $L_{0,j}$, όπου $i = \text{bitreverse}(j)$.

Η παραπάνω δομή επαναλαμβάνεται για $0 \leq l < \log_2 N$ και $0 \leq i < N$.

Οι συναρτήσεις f και g ορίζονται ως

$$f(L_a, L_b) \triangleq \frac{L_a \cdot L_b + 1}{L_a + L_b},$$

$$g(\hat{s}, L_a, L_b) \triangleq (L_a)^{1-2\hat{s}} \cdot L_b,$$

όπου το \hat{s} αναπαριστά ένα μερικό άθροισμα των προηγούμενων αποκωδικοποιημένων bit \hat{u}_i . Αυτό το μερικό άθροισμα αντιστοιχίζεται στην διάδοση των προηγούμενων αποκωδικοποιημένων bit από τα αριστερά προς τα δεξιά στον γράφο και θα αναλυθεί περαιτέρω παρακάτω.

Η διαδικασία από τα δεξιά προς τα αριστερά αναφέρεται ως η μεταφορά soft information. Η αντίστοιχη δε μεταφορά των αποκωδικοποιημένων bits \hat{u}_i από τα αριστερά στα δεξιά, για τον υπολογισμό των μερικών αθροισμάτων, αναφέρεται ως η μεταφορά hard information και χρησιμοποιούνται για τον υπολογισμό των μερικών αθροισμάτων.

Έτσι για κάθε αποκωδικοποιημένο bit έχουμε $I = \log_2 N$ στάδια υπολογισμών, οπότε η πολυπλοκότητα του SC για N bits αποκωδικοποίησης είναι $O(N \log_2 N)$.

Αξίζει να σημειωθεί ότι για τα όρια του σφάλματος πιθανότητας των μπλοκ (Block Error Probability) για τους Polar Codes, ισχύει:

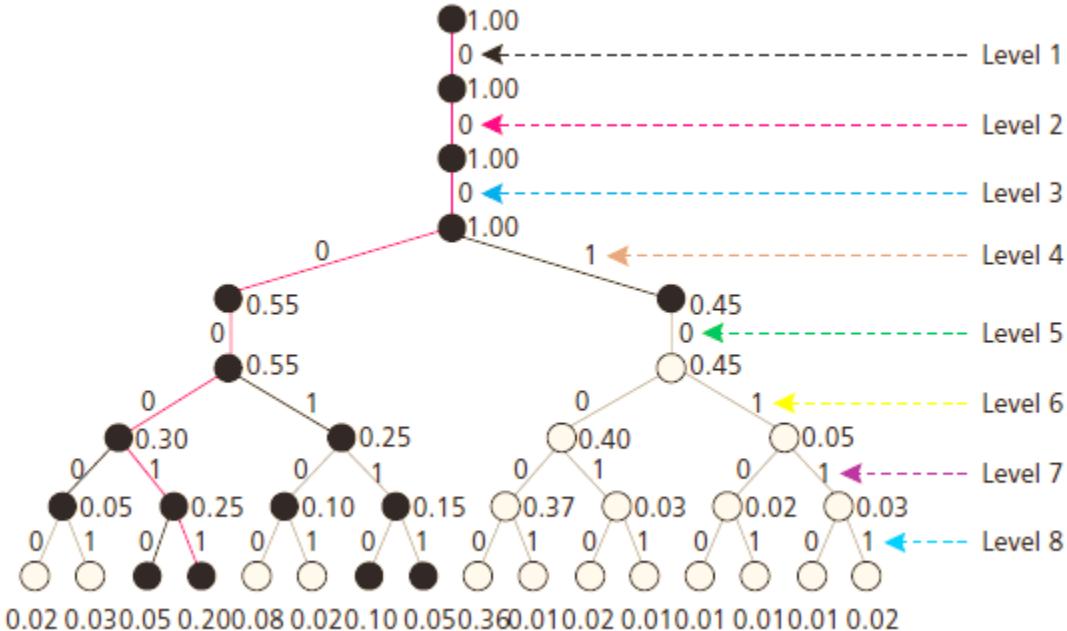
$$\max_{i \in F^c} \frac{1}{2} \left(1 - \sqrt{1 - Z(W_N^{(i)})^2} \right) \leq P_{SC} \leq \sum_{i \in F^c} Z(W_N^{(i)}),$$

Όπου F είναι το σύνολο των δεικτών των παγωμένων καναλιών και P_{SC} είναι ο μέσος όρων των σφαλμάτων πιθανότητας των μπλοκ υπό την αποκωδικοποίηση SC. Το οποίο έχει αποδειχθεί ότι τείνει στο 0, $\mathcal{O}(2^{-\sqrt{N}})$ ασυμπτωτικά [1].

2.4.2 Successive-Cancellation List Decoding Algorithm (SCLD Αλγόριθμος)

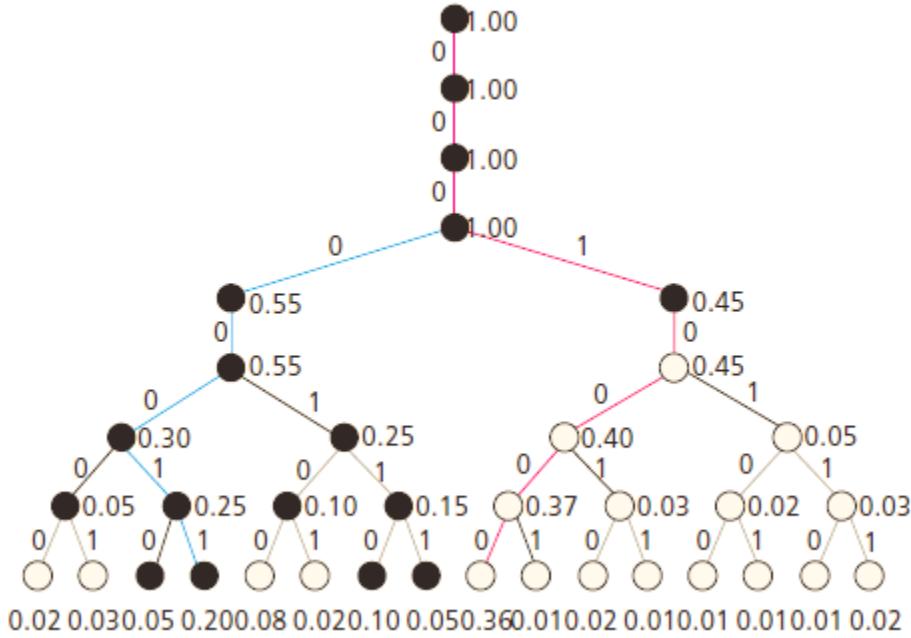
Η SC αποκωδικοποίηση των Polar codes μπορεί να θεωρηθεί ως ένας άπληστος αλγόριθμος πάνω στο δέντρο αποφάσεων. Ανάμεσα σε δύο κλάδους, που αναπαριστούν πληροφορία ενός bit σε ένα συγκεκριμένο επίπεδο του δέντρου, μόνο

αυτό με τη μεγαλύτερη πιθανότητα επιλέγεται για περαιτέρω επεξεργασία. Εάν ένα bit καθοριστεί λάθος, είναι αδύνατο σε μελλοντική αποκωδικοποίηση αυτό το bit να αλλάξει σε σωστή πιμή. Τα έντονα κόκκινα κλαδιά στο διάγραμμα 2.7α αναπαριστούν μία ακολουθία SC αποκρυπτογράφησης "00000011". Προφανώς αυτή η ακολουθία δεν είναι ιδανική λόγω της στρατηγικής απόφασης ανά επίπεδο:



Σχήμα 2.7α: SC decoding tree [2]

Ως μία βελτιωμένη έκδοση της SC, έχει προταθεί η αποκωδικοποίηση SCL όπου η στρατηγική απόφαση είναι επίσης ανά στάδιο και είναι περίπου σαν του SC. Παρόλα αυτά, αντίθετα από τον αλγόριθμο SC όπου μόνο μία ακολουθία επιλέγεται μετά την επεξεργασία σε κάθε επίπεδο, ο SCL έχει μεγαλύτερο εύρος στις αναζητήσεις του και επιτρέπει έναν μέγιστο αριθμό L υποψήφιων ακολουθιών που μπορούν να διερευνηθούν περαιτέρω. Σε κάθε επίπεδο ενός bit πληροφορίας, ο SCL διπλασιάζει τον αριθμό των υποψήφιων ακολουθιών προσαρτώντας ένα bit 0 ή 1 σε κάθε μία από αυτές. Έπειτα επιλέγει έναν μέγιστο αριθμό L από αυτές με κατάλληλο κριτήριο, όπως η μέγιστη πιμή μιας μετρικής (path metrics) και πις αποθηκεύει σε μία λίστα. Το Σχήμα 2.7β δείχνει το παράδειγμα της SCL μεθόδου με μέγιστο αριθμό $L=2$. Σε κάθε επίπεδο αποκωδικοποίησης, δύο υποψήφιες ακολουθίες (αναπαρίστανται ως κόκκινες και μπλε έντονες ακμές) αποθηκεύονται, και έτσι βρίσκεται η πιο αξιόπιστη «000100000»:

Σχήμα 2.7β: SCL decoding tree για $L=2$ [2]

Για την υλοποίηση του SCLD προτείνεται στο [4] μία δομή που εξοικονομεί χώρο για να εκτελέσει τον SC αποκωδικοποιητή, ενώ η πολυπλοκότητα χρόνου είναι $O(N \log_2 N)$ και η πολυπλοκότητα χώρου $O(N)$. Μία απευθείας εκτέλεση του αποκωδικοποιητή SCL απαιτεί υπολογισμούς $O(LN^2)$. Μία αποκαλούμενη «lazy copy» τεχνική, βασισμένη στο πρωτόκολλο διαμοιρασμού της μνήμης μεταξύ των υποψήφιων ακολουθιών, εισάγεται ώστε να ελαχιστοποιήσει τις περιπτές εργασίες ανπιγραφής. Επομένως ο αποκωδικοποιητής SCL μπορεί να εκτελεστεί σε χρόνο τάξης $O(LN \log_2 N)$.

2.4.3 CRC Aided SCLD (CA-SCLD Αλγόριθμος)

Έχουν προταθεί για περαιτέρω βελτίωση της απόδοσης των polar codes υποβοηθούμενα CRC(Cyclic Redundancy Check)-aided SCL σχήματα αποκωδικοποίησης όπως το CA-SCL. Σε αυτά τα σχήματα ο αποκωδικοποιητής περνά τα υποψήφια μονοπάτια από έναν CRC detector, ο οποίος υποβοηθά το σύστημα στην απόφαση της σωστής κωδικής λέξης.

Αυτό που γίνεται στην πράξη είναι ότι ο δέκτης και ο αποστολέας έχουν προσυμφωνήσει σε μία λέξη κλειδί συγκεκριμένου μήκους Z που λέγεται και CRC. Στη συνέχεια ο αποστολέας προσθέτει στο τέλος της λέξης που θέλει να αποστείλει το πηλίκο της με το CRC, μήκους $Z-1$. Έτσι η καινούργια λέξη αποστολής περιέχει επιπλέον επιβάρυνση.

Τέλος ο δέκτης λαμβάνει τη λέξη και την αποκωδικοποιεί. Αν το πηλίκο της λέξης αυτής με το CRC ισούται με μηδέν, τότε αυτή η λέξη περνάει τον έλεγχο του CRC detector.

Έστω ότι θέλουμε να αποστείλουμε τη λέξη 11010011101100 και έχουμε ένα CRC=1011. Ο αποστολέας διαιρεί αρχικά, όπως φαίνεται στο σχήμα 2.8 , την λέξη με το CRC και παράγει το υπόλοιπο 100.

```

11010011101100 000
1011
01100011101100 000
1011
00111011101100 000
1011
00010111101100 000
1011
00000001101100 000
...
1011
00000000110100 000
1011
00000000011000 000
1011
00000000001110 000
1011
00000000000101 000
101 1
-----
00000000000000 100

```

Σχήμα 2.8: Διαιρεση της λέξης για εύρεση υπολοίπου στον αποστολέα

Και στέλνει την αρχική λέξη με το υπόλοιπο στο τέλος. Δηλαδή 11010011101100 100. Στην άλλη πλευρά μετά την αποκωδικοποίηση ο CRC detector διαιρεί με το CRC, όπως φαίνεται στο Σχήμα 2.9 (έστω ότι δεν έχουμε λάθος αποκωδικοποίησης).

```

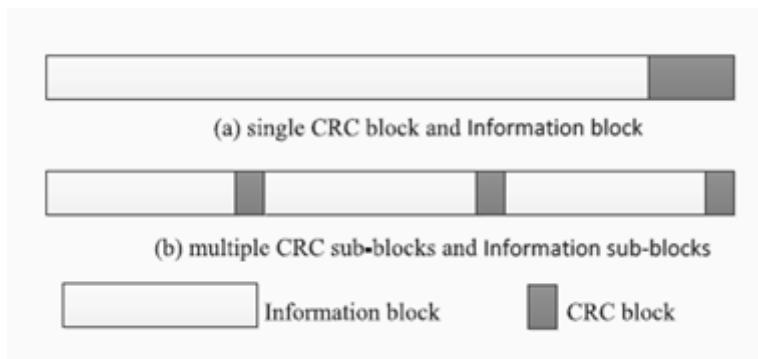
11010011101100 100
1011
01100011101100 100
1011
00111011101100 100

.....
00000000001110 100
1011
00000000000101 100
101 1
-----
0

```

Σχήμα 2.9: CRC Detector, Διαίρεση με το CRC και σύγκριση υπολοίπου με το 0

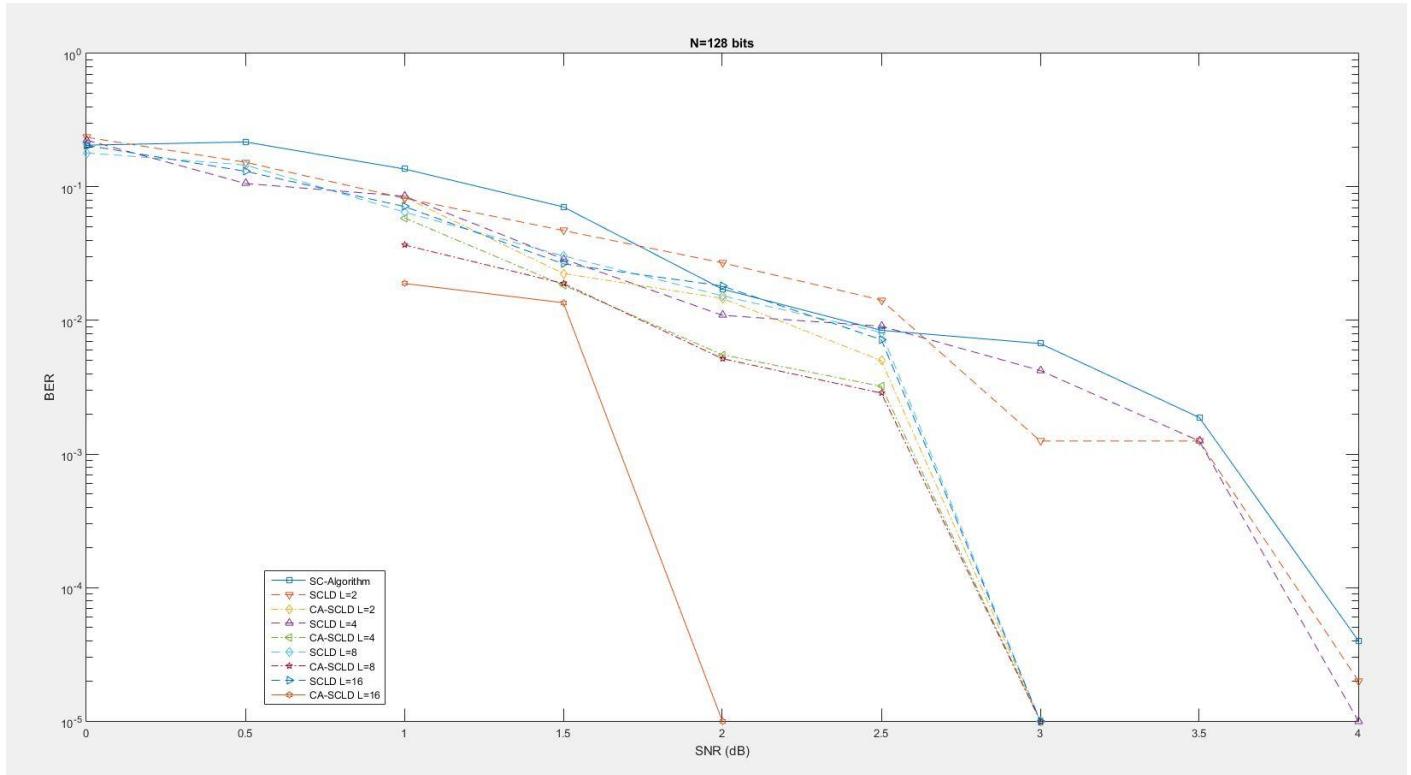
Εφόσον το υπόλοιπο είναι μηδέν, την δεχόμαστε. Σε διαφορετική περίπτωση την απορρίπτουμε, εφόσον υπάρχει κάποια άλλη λέξη που περνά τον έλεγχο. Αξιοσημείωτο εδώ είναι ότι η διάσπαση του CRC σε block από CRC μπορεί να αυξήσει το throughput από άποψη FER (Frame Error Rate) όπως αναφέρεται στο [10]. Αυτό συμβαίνει διότι η διαίρεση της κωδικής λέξης σε block επιφέρει μικρότερη πιθανότητα συνολικού σφάλματος από το αρχικό.



Σχήμα 2.10: Τα δύο είδη CRC

2.5 Αποτελέσματα Προσομοιώσεων

Οι αλγόριθμοι που παρουσιάστηκαν στις προηγούμενες παραγράφους υλοποιήθηκαν σε matlab και προσομοιώθηκε η συμπεριφορά του σε κανάλι προσθετικού λευκού Γκαουσιανού θορύβου. Τα αποτελέσματα με τη μορφή διαγραμμάτων BER (Bit Error Rate) vs θόρυβος παρουσιάζονται στο Σχήμα 2.11.



Σχήμα 2.11: Σύγκριση αλγορίθμων για μήκος κώδικα N=128

Κεφάλαιο 3. Σχεδιασμός Αποκωδικοποιητή

3.1 Συναρτήσεις f και g

Όσον αφορά την υλοποίηση σε υλικό, έχουμε το αρχικό πρόβλημα όπι και οι δύο διεργασίες επιπελούν πράξεις πολλαπλασιασμού και διαίρεσης που είναι αρκετά πολύπλοκες στο hardware. Έτσι χρησιμοποιούμε τον μετασχηματισμό στο λογαριθμικό πεδίο των LR σε LLR (logarithmic likelihood ratio). Έτσι οι διεργασίες f και g μετατρέπονται στις αντίστοιχες SPA (sum-product-algorithm) εξισώσεις

$$\begin{aligned}\lambda_f(\lambda_a, \lambda_b) &\triangleq 2 \tanh^{-1} \left(\tanh\left(\frac{\lambda_a}{2}\right) \cdot \tanh\left(\frac{\lambda_b}{2}\right) \right), \\ \lambda_g(\hat{s}, \lambda_a, \lambda_b) &\triangleq \lambda_a(-1)^{\hat{s}} + \lambda_b,\end{aligned}$$

όπου πλέον τα LLR ορίζονται ως $\lambda_i = \ln(L_i)$. Χρησιμοποιώντας τον ορισμό των LLR για διαμόρφωση BPSK και κανάλι AWGN,

$$\lambda_{y_i} \triangleq \frac{2y_i}{\sigma^2}.$$

Εν τέλει χρησιμοποιώντας μια δημοφιλή προσέγγιση οι διεργασίες f και g μετατρέπονται στις ακόλουθες MSA (min-sum algorithm) εξισώσεις, σύμφωνα με το [10]

$$\begin{aligned}\lambda_f(\lambda_a, \lambda_b) &\approx \psi^*(\lambda_a)\psi^*(\lambda_b) \min(|\lambda_a|, |\lambda_b|), \\ \lambda_g(\hat{s}, \lambda_a, \lambda_b) &\triangleq \lambda_a(-1)^{\hat{s}} + \lambda_b,\end{aligned}$$

όπου το $|\lambda_i|$ αναπαριστά το μέτρο του λ_i και το $\psi^*(\lambda_i)$ το πρόσημο του και

$$\psi^*(\lambda_i) \triangleq \begin{cases} 1 & \text{when } \lambda_i \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

Βλέπουμε τελικά ότι οι πράξεις μειώθηκαν σε πολυπλοκότητα και η υλοποίηση σε hardware είναι αρκετά πιο εύκολη πλέον και λιγότερο χρονοβόρα, καθώς χρησιμοποιούνται απλής πολυπλοκότητας αθροιστές και συγκριτές.

3.2 Αναπαράσταση LLR Αποκωδικοποιητή (Two's Complement)

Το συμπλήρωμα βάσης του δύο χρησιμοποιείται εισάγοντας τη βιβλιοθήκη signed της VHDL (Hardware Description Language), όπου είναι κομμάτι του NUMERIC_STD package της IEEE.

Σε κάθε κύκλο αποκωδικοποίησης, από τη φύση της g , μπορεί να έχουμε υπερχείλιση αν δεν προσέξουμε το μήκος αναπαράστασης των LLR. Μπορεί να παρατηρήσει κάποιος, όπι σε έναν κύκλο αποκωδικοποίησης (1 σταδίων) η λέξη μας μπορεί να αυξηθεί κατά 1 bits, όσα και τα στάδια αποκωδικοποίησης. Αυτό συμβαίνει γιατί η συνάρτηση g μπορεί να αυξήσει κατά 1 το μήκος λέξης.

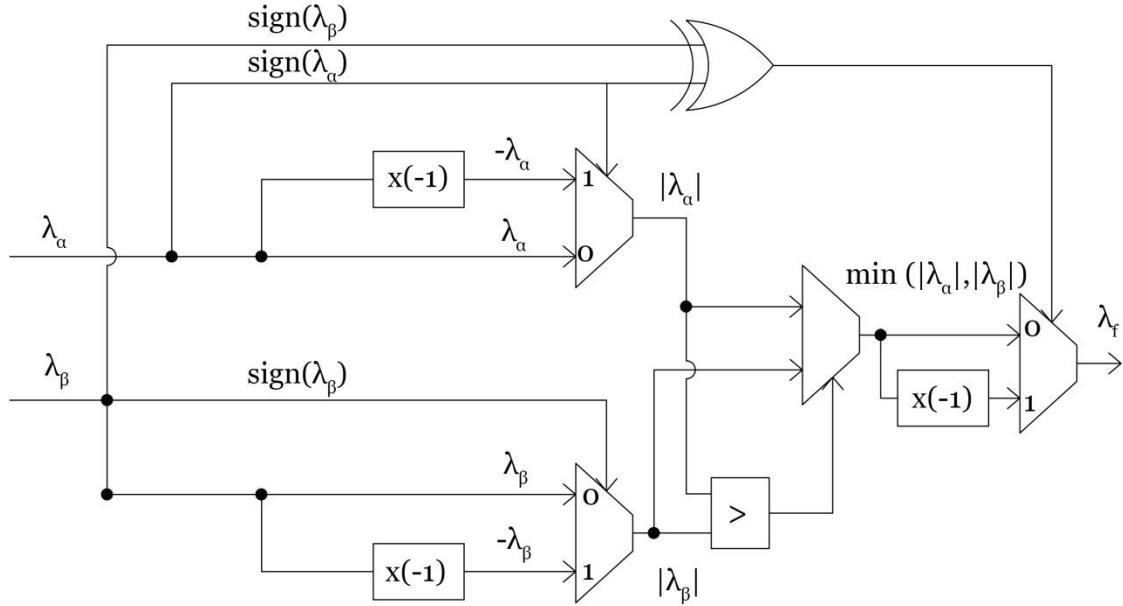
Σύμφωνα με τη μελέτη αναπαράστασης των LLR καναλιού στο [5] αποδεικνύεται ότι βέλτιστη είναι η αναπαράσταση 6-2 (6 bit για το ακέραιο μέρος και 2 bit για το δεκαδικό μέρος), η οποία δεν επιφέρει επιπτώσεις στην απόδοση του συστήματος. Εμείς θα επιλέξουμε εδώ την αναπαράσταση 5-1 καθώς οι στάθμες θορύβου είναι αρκετά μικρές για μικρά N και οδηγεί σε απλούστερο hardware.

Έτσι έχοντας μια συγκεκριμένη αναπαράσταση για τα LLR καναλιού 5-1 προσθέτουμε 1 bits. Έτσι για παράδειγμα για $N=8$ έχουμε 6 bit τα LLR καναλιού και 9 (6+3) bit τα LLR αποκωδικοποιητή.

Αξίζει να σημειωθεί ότι η χρησιμοποίηση ενός saturating adder (κορεσμένου αθροιστή) είναι καταλληλότερη για μεγαλύτερα N όπως γίνεται στα [5],[10]. Αυτό συμβαίνει διότι η αναπαράσταση μεγέθους των LLR πρέπει να ισούται με το μέγεθος των LLR που έρχονται από το κανάλι επαυξημένο λόγω των ανίστοιχων σταδίων, όπως προαναφέρθηκε, με αποτέλεσμα τεράστιου μεγέθους αναπαράσταση. Σπηλική μας υλοποίηση παραλείπεται κάπι τέτοιο, αλλά υπάρχει ως μελλοντικό στάδιο ανάπτυξης του συστήματος.

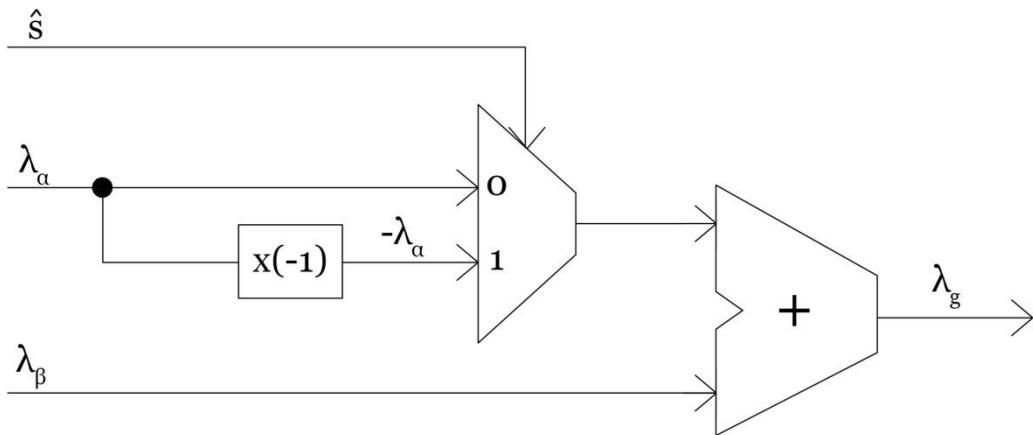
3.3 Υλοποίηση Στοιχείων Επεξεργασίας f και g

Σύμφωνα με τα παραπάνω οι συναρτήσεις λ_f και λ_g είναι εύκολα υλοποίησιμες χρησιμοποιώντας απλές πράξεις όπως φαίνεται στο σχηματικό για την f , που παρουσιάζεται στο σχήμα 3.1.



Σχήμα 3.1: Processing Element f

Το αντίστοιχο σχηματικό για την g παρουσιάζεται στο σχήμα 3.2.



Σχήμα 3.2: Processing Element g

Η συνάρτηση $\psi^*(\lambda_i)$ απλοποιείται διότι το MSB (περισσότερο σημαντικό ψηφίο) αναπαριστά πάντοτε το πρόσημο του αντίστοιχου LR, 1 για αρνητικούς και 0 για θετικούς. Δηλαδή :

$$\psi(\lambda_i) \triangleq \begin{cases} 0 & \text{when } \lambda_i \geq 0, \\ 1 & \text{otherwise.} \end{cases}$$

Ο πολλαπλασιασμός για την εύρεση του αντιθέτου αντιστοιχίζεται πλέον σε αντιστροφή όλων των bit και πρόσθεση του 1, λόγω του συμπληρώματος του δύο. Επίσης η απόλυτη πιμή υλοποιείται με έναν πολυτπλέκτη που παίρνει ως είσοδο την πιμή λ και το αντίστοιχο συμπλήρωμα της και επιλέγει τη θετική, βάση του πρόσημου του λ. Τέλος χρησιμοποιείται ένας αθροιστής μεγέθους λ ίσου με το μέγεθος των LLR του αποκωδικοποιητή.

3.4 Μερικά αθροίσματα \hat{s}

Όπως προαναφέρθηκε το μερικό άθροισμα αντιστοιχίζεται στη διάδοση των προηγούμενων αποκωδικοποιημένων bit από τα αριστερά προς τα δεξιά στον γράφο. Η υλοποίηση τους ξεκινά από το λιγότερο σημαντικό ψηφίο LSB αποκωδικοποίησης και συνεχίζεται προς το περισσότερο σημαντικό ψηφίο MSB.

Έτσι για το bit \hat{u}_0 βλέπει κανείς ότι δεν χρησιμοποιούνται μερικά αθροίσματα. Στην συνέχεια για το bit \hat{u}_1 χρησιμοποιείται ένα μερικό άθροισμα το οποίο περιέχει μόνο το \hat{u}_0 κ.ο.κ. Αν σε κάποιο μερικό άθροισμα χρησιμοποιείται ήδη ένα bit \hat{u}_i από προηγούμενη επαναχρησιμοποίηση δικού του μερικού αθροίσματος, δεν προστίθεται σε άλλο μερικό άθροισμα.

Έτσι εν τέλει στην διαδρομή για την αποκωδικοποίηση ενός bit \hat{u}_i , τα μερικά αθροίσματα που χρησιμοποιούνται από τις διεργασίες g περιέχουν όλα τα προηγούμενα αποκωδικοποιημένα bits.

Έτσι προκύπτει ο παρακάτω εμπειρικός τύπος, όπως αναφέρεται στα [5],[10], ο οποίος περιέχει δυαδικές πράξεις

$$I(l, i, z) \triangleq \overline{B(l, i)} \cdot \prod_{v=l}^{n-2} \left(\overline{B(n-v-2, z)} \oplus B(v+1, i) \right) \cdot \prod_{w=0}^{l-1} \left(\overline{B(n-w-2, z)} + B(w, i) \right),$$

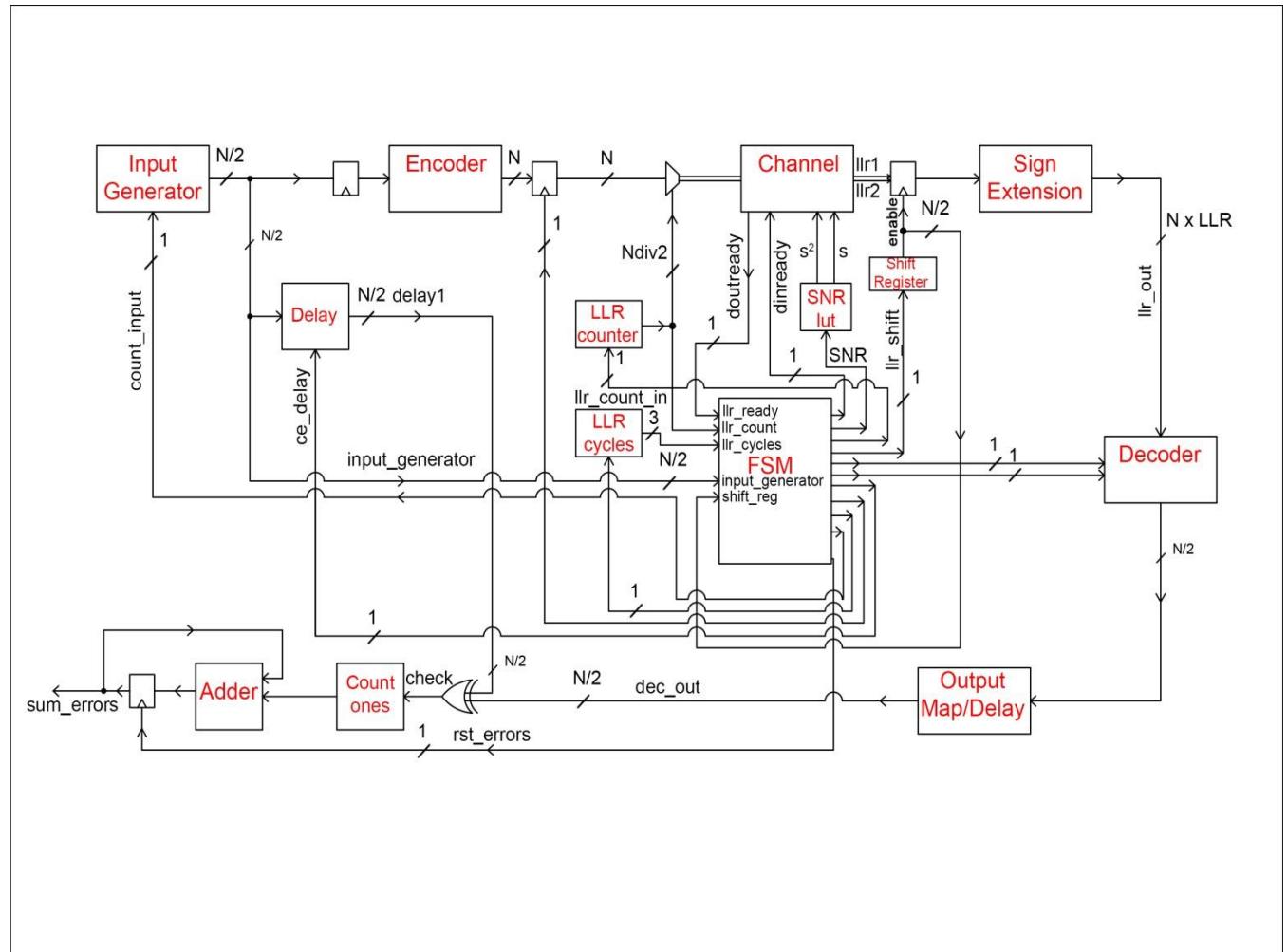
όπου z είναι ο δείκτης του μερικού αθροίσματος $\hat{s}_{l,z}$. Οι πράξεις \cdot και Π αναφέρονται στην δυαδική πράξη AND. Οι πράξη $+$ αναφέρεται στην δυαδική πράξη OR και το \bar{A} αναφέρεται στην πράξη NOT πάνω στο bit A. Τέλος η συνάρτηση B ορίζεται ως

$$B(a, b) \triangleq \frac{b}{2^a} \mod 2$$

Κεφάλαιο 4. Υλοποίηση FPGA και επαλήθευση

Το σύστημα που υλοποιήθηκε μετράει τα BER για τα διάφορα μήκη λέξεων N . Αρχικά ένα σύστημα παράγει πις διάφορες λέξεις οι οποίες κωδικοποιούνται στον Encoder, ύστερα περνούν από το Channel (κανάλι), φτάνουν στον Decoder όπου και αποκωδικοποιούνται και τελικά μετρώνται τα λάθη, αν έγιναν, ως προς την αρχική λέξη που εστάλη. Η παραπάνω διαδικασία επαναλαμβάνεται για διάφορες λέξεις εισόδους καθώς και διάφορα στάδια θορύβου SNR.

Στο σχήμα 4.1 φαίνεται ένα πλήρες σχεδιάγραμμα του συστήματος.



Σχήμα 4.1: Σχεδιάγραμμα του Συστήματος

4.1 Κωδικοποιητής (Encoder)

Αρχικά υλοποιήθηκε ο κωδικοποιητής μέσω της γεννήτριας G_N . Τα ανίστοιχα 1 στην γεννήτρια σηματοδοτούν την ύπαρξη xor στα ανίστοιχα bit εισόδου του διανύσματος \mathbf{x} . Έτσι ο κωδικοποιητής παίρνει ένα $\text{std_logic_vector}(N/2-1 \text{ downto } 0)$ και παράγει ένα $\text{std_logic_vector}(N-1 \text{ downto } 0)$ υλοποιώντας τις κατάλληλες προσθέσεις μεταξύ των bit εισόδου.

4.2 Κανάλι (Channel)

Το κανάλι παραγωγής των LLR και πρόσθεσης θορύβου πάρθηκε έτοιμο από προηγούμενη διπλωματική του τμήματος μας, όπως αναφέρεται στο [7]. Το οποίο παίρνει ως εισόδους δύο bit και την κλίμακα του θορύβου SNR (Signal-to-noise-Ratio) και παράγει με καθυστέρηση δύο LLR 5-1 μετά από την πρόσθεση λευκού Γκαουσιανού θορύβου.

4.3 Αποκωδικοποιητής (Decoder)

Υλοποιήθηκε η αρχιτεκτονική ενός πλήρως παράλληλου αποκωδικοποιητή δίχως pipeline αρχικά και στην συνέχεια βελτιώθηκε με την προσθήκη pipeline. Για να εξεταστεί η υλοποίηση με pipeline πρέπει αρχικά να δούμε το scheduling του αποκωδικοποιητή.

4.3.1 Scheduling

To scheduling ενός πλήρως παράλληλου για $N=8$ αποκωδικοποιητή φαίνεται στο Σχήμα 4.2.

Stage / i	0	1	2	3	4	5	6	7
$l = 2$	f	f	f	f	g	g	g	g
$l = 1$	f	f	g	g	f	f	g	g
$l = 0$	f	g	f	g	f	g	f	g

Σχήμα 4.2: To schedule ενός πλήρως παράλληλου SC αποκωδικοποιητή

Βλέπουμε ότι για την αποκωδικοποίηση N bits όπως έχει προαναφερθεί χρειάζονται N κύκλοι των l σταδίων, όπου σε κάθε στάδιο υλοποιούνται διεργασίες της f ή της g. Μπορεί κάποιος να παρατηρήσει ότι σε κάθε στάδιο επιτελείται ένα μόνο είδος διεργασίας, είτε f είτε g. Επίσης από τις είκοσι τέσσερις διεργασίες (τρία στάδια επτί οχτώ

διεργασίες) που επιτελούνται, μόνο οι 14 παράγουν νέα πληροφορία. Αυτές αναπαριστώνται με bold. Οι μη bold διεργασίες παράγουν πληροφορία που έχει ήδη παραχθεί στα προηγούμενα στάδια, αν και καμία δεν μπορεί να παραλειφθεί γιατί την χρειάζονται τα επόμενα στάδια, έτσι πρέπει να αποθηκευτεί η πληροφορία τους σε κάποιον καταχωρητή (register). Τελικά με αυτήν την παραπόρηση μπορούμε να μειώσουμε τον φόρτο εργασίας ενός πλήρους κύκλου αποκωδικοποίησης.

Για να μιλήσουμε για ένα πιο ρεαλιστικό κύκλωμα, θα θέλαμε το κύκλωμα μας να είναι πλήρως παράλληλο και pipelined, έτσι ώστε να αυξήσουμε το throughput rate και παράλληλα να μειώσουμε και το ρολόι λειτουργίας προσθέτοντας στα κατάλληλα σημεία registers.

Αρχικά για το ρολόι λειτουργίας στην αρχή κάθε σταδίου τοποθετούμε registers για τα LLR, καθώς και στην έξοδο του decoder. Έτσι το ρολόι λειτουργίας ακολουθεί τον χρόνο λειτουργίας του critical path και εν τέλει μειώνεται η συχνότητα λειτουργίας του σημαντικά.

Στη συνέχεια αναλύουμε το scheduling μας ανά κύκλο ρολογιού με την παρουσία των registers ανάμεσα στα στάδια. Πλέον σε κάθε κύκλο ρολογιού εκτελούνται οι διεργασίες κάθε σταδίου, που είναι πλέον αποκομμένες μεταξύ τους, με την ύπαρξη των register.

Περιέχονται μόνο οι bold πράξεις πλέον, εφόσον έχουμε αποθηκευμένη την πληροφορία σε registers. Οι ανίστοιχες έξοδοι, για $N=8$, σύμφωνα με τους κύκλους ρολογιού παρουσιάζονται στο Σχήμα 4.3.

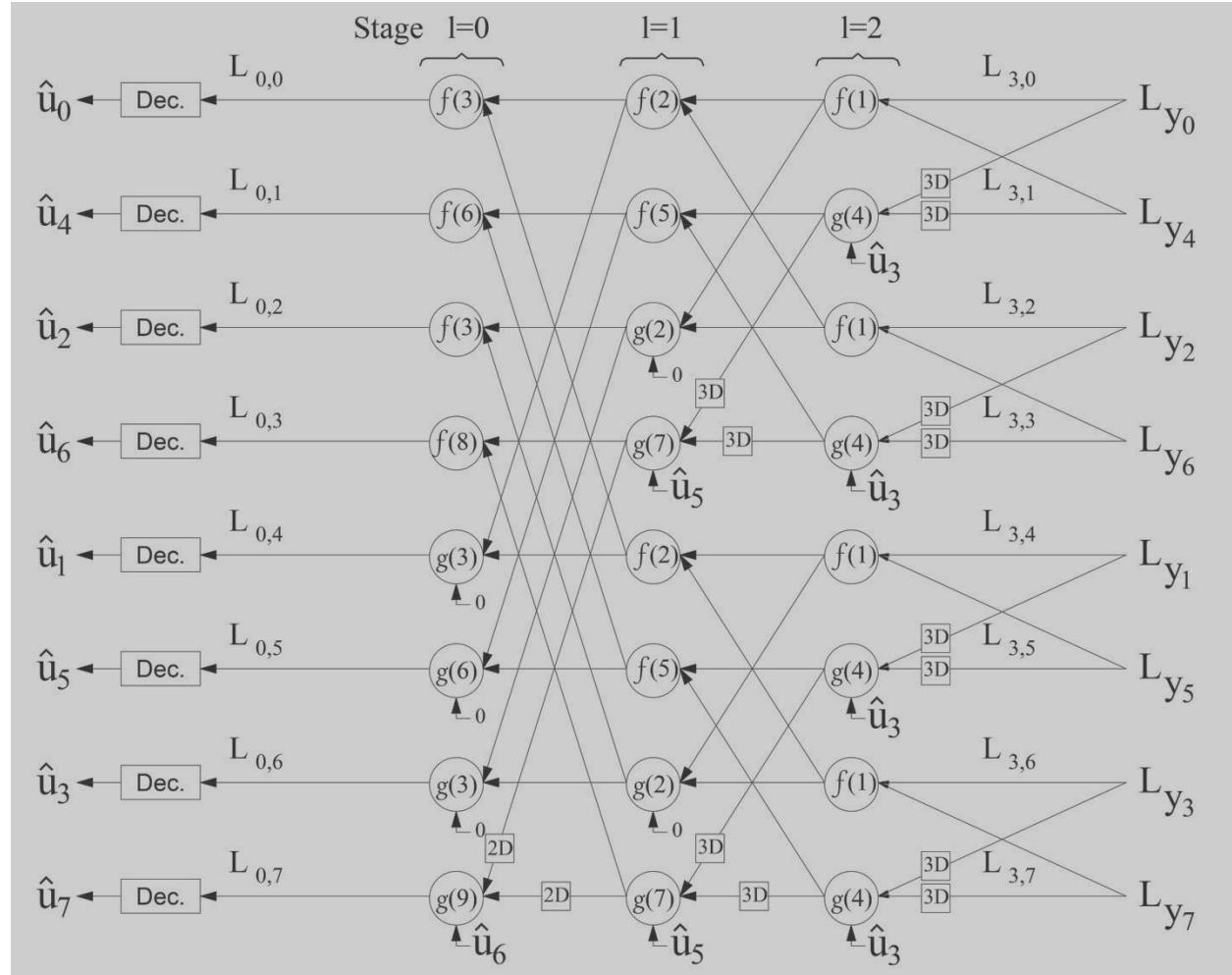
Stage / CC	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$l = 2$	<i>f</i>						<i>g</i>							
$l = 1$		<i>f</i>		<i>g</i>				<i>f</i>			<i>g</i>			
$l = 0$			<i>f</i>	<i>g</i>	<i>f</i>	<i>g</i>			<i>f</i>	<i>g</i>		<i>f</i>	<i>g</i>	
Output			\hat{u}_0	\hat{u}_1	\hat{u}_2	\hat{u}_3			\hat{u}_4	\hat{u}_5		\hat{u}_6	\hat{u}_7	

Σχήμα 4.3: To schedule ενός pipelined SC αποκωδικοποιητή

Βλέπουμε λοιπόν ότι όντως το κύκλωμά μας με τα κατάλληλα delays θα μπορούσε να δεχτεί pipeline. Προσοχή βέβαια πρέπει να δοθεί στα μερικά αθροίσματα και στο πότε είναι αυτά διαθέσιμα, καθώς χρησιμοποιούν τις εξόδους του σταδίου 0 για το εκάστοτε bit \hat{u}_i .

Στην είσοδο και την έξοδο του αποκωδικοποιητή υπάρχουν registers εισόδου και εξόδου για τα LLR και τα αποκωδικοποιημένα bit \hat{u}_i ανίστοιχα.

Το pipelined σχεδιάγραμμα για $N=8$ παρουσιάζεται στο σχήμα 4.4.



Σχήμα 4.4: Pipelined Σχεδιάγραμμα του κυκλώματος του SC Decoder

Στο Σχήμα 4.4 ο αριθμός μέσα στις f και g αντιπροσωπεύει τον κύκλο ρολογιού που θα είναι έτοιμο το αποτέλεσμα και τα κουπιά εισάγουνε καθυστέρηση ίση με τον αριθμό που περιέχουν.

Οι πιμές των L_{yi} είναι διαθέσιμες στην έξοδο των registers στο πρώτο clock. Το αποτέλεσμα του εκάστοτε bit \hat{u}_i είναι διαθέσιμο ένα clock αργότερα από όταν η πληροφορία είναι διαθέσιμη στο στάδιο 0, εφόσον πρέπει να περάσουν από τους registers εξόδου.

4.3.2 Παραγωγή Μερικών Αθροισμάτων

Η παραγωγή των μερικών αθροισμάτων όπως προαναφέρθηκε γίνεται μέσω του τύπου

$$I(l, i, z) \triangleq \overline{B(l, i)} \cdot \prod_{v=l}^{n-2} \left(\overline{B(n-v-2, z)} \oplus B(v+1, i) \right) \cdot \prod_{w=0}^{l-1} \left(\overline{B(n-w-2, z)} + B(w, i) \right)$$

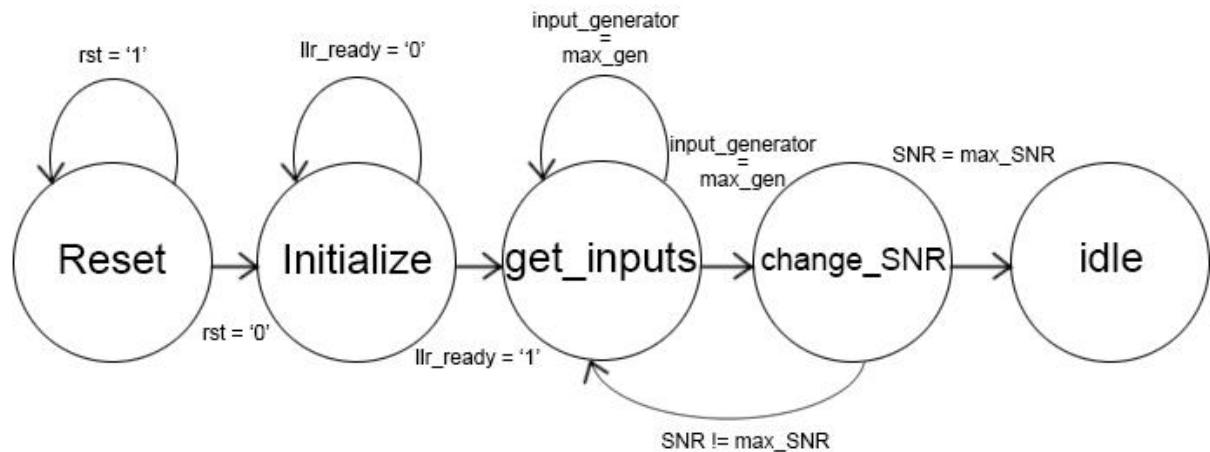
Μετά τις πράξεις έχουμε πίνακες που αναπαριστάνε τα μερικά αθροίσματα. Έτσι όπως και στον encoder καταλήγουμε σε αντίστοιχες προσθέσεις ενός bit xor των αντίστοιχων bit που αναφέρονται στον πίνακα.

Τέλος δημιουργούμε ένα entity που δέχεται τα αποκωδικοποιημένα bits \hat{u}_i και παράγει τα μερικά αθροίσματα κάθε διεργασίας g με τις αντίστοιχες καθυστερήσεις όπου χρειάζονται.

4.4 Finite State Machine (FSM)

Ο έλεγχος και η λογική όλων των παραπάνω κυκλωμάτων, καθώς και ο συγχρονισμός τους υλοποιείται από μία μηχανή διακριτών καταστάσεων FSM.

Δύο είναι τα βασικά είδη FSM, Moore και Mealy. Για την υλοποίηση μας επιλέχτηκε το μοντέλο του Moore. Το σχεδιάγραμμα των καταστάσεων της FSM παρουσιάζεται στο Σχήμα 4.5.



Σχήμα 4.5: Finite State Machine του συστήματος για την μέτρηση λαθών

Στο Σχήμα 4.5 έχουμε τις παρακάτω καταστάσεις:

- Reset: Η κατάσταση αυτή υπάρχει όσο το κύκλωμα μας δέχεται reset. Δηλαδή είναι πατημένο το κουμπί rst. Αυτό συμβαίνει είτε στην εκκίνηση του κυκλώματος, είτε για επανεκκίνηση της λειτουργίας του.
- Initialize: Η κατάσταση αυτή υπάρχει κατά την ώρα που αρχικοποιείται το κανάλι μας. Δηλαδή όσο το σήμα llr_out = 0. Αυτό συμβαίνει διότι το κανάλι μας χρειάζεται κάποιο χρόνο για να αρχικοποιηθεί (latency) και να παράγει τον θόρυβο που χρειάζεται.
- get_inputs: Η κατάσταση αυτή υπάρχει κατά την ώρα του encoding/decoding και υπολογισμού λαθών για ένα συγκεκριμένο στάδιο θορύβου SNR. Δηλαδή όσο το σήμα input_generator != ones(N-1 downto 0).
- change_SNR: Η κατάσταση αυτή υπάρχει για την αλλαγή της στάθμης θορύβου SNR.
- Idle: Εάν έχουν περάσει όλες οι διαθέσιμες καταστάσεις SNR και έχουν υπολογισθεί όλα τα σφάλματα για τα παραπάνω, το κύκλωμα μπαίνει σε κατάσταση idle και δεν υλοποιεί καμία διαδικασία από τις παραπάνω.

4.5 Παραμετρική Ροή Σχεδιασμού

Παρατηρήθηκε ότι αρκετά κομμάτια του συστήματος επαναχρησιμοποιούνται και έτσι προέκυψε η παραμετρική γραφή κώδικα για την παραγωγή διαφόρων μηκών λέξεων N. Ο κώδικας για τα διάφορα μήκη κωδικών λέξεων N, γράφτηκε παραμετρικά σε Python αξιοποιώντας αποτελέσματα διαφόρων αρχείων κειμένου (.txt) που παράγει η Matlab. Στην συνέχεια αυτός ο κώδικας εισάγεται στο πρόγραμμα σχεδιασμού ISE όπου και γίνεται η σύνθεση του συστήματος για το συγκεκριμένο μήκος N και στην συνέχεια η υλοποίηση του στο FPGA. Αυτό έχει σαν αποτέλεσμα την ταχεία παραγωγή κώδικα ακόμα και για μεγάλου μήκους συστημάτων π.χ N=4096.

Ως παράδειγμα στα σχήματα 4.9,4.10 φαίνεται το script στην python που παράγει τον κώδικα της οντότητας που ασχολείται με τα μερικά αθροίσματα. Το οποίο δέχεται τρία αρχεία που παράγονται μέσω της Matlab, ένα για τα bits τα οποία είναι "παγωμένα", ένα για τους χρονισμούς σε όπι αφορά το scheduling και τέλος ένα αρχείο με τα μερικά αθροίσματα. Στο Σχήμα 4.6 φαίνεται ο κώδικας για το πρώτο αρχείο, στο Σχήμα 4.7 ο κώδικας για το δεύτερο αρχείο και στο Σχήμα 4.8 ο κώδικας για το τρίτο αρχείο.

Τέλος ένα κομμάτι του κώδικα που παράχθηκε φαίνεται στο Σχήμα 4.11.

Αντίστοιχα έχουνε γραφεί όλα οι οντότητες που επηρεάζονται από το μήκος κώδικα N , όπως ο encoder, ο decoder, το δέντρο των αθροιστών για την μέτρηση λαθών και η εισαγωγή καθυστέρησης στην έξοδο του decoder.

```

function frozen_bits = initialize_frozen_bits( N, K, capacity )
    %inputs N codelength, K code keyword length, capacity of the channel
    %outputs a N-length array with 0,1. If 0 then the channel is frozen
    %if 1 not frozen.
    capacity_array = bitrevorder(capacities(N, capacity));
    [capacity_array,sortIndex] = sort(capacity_array(:));
    frozen_bits = ones(1,N);
    for i=1:1:N-K
        frozen_bits(sortIndex(i)) = 0;
    end
end

```

Σχήμα 4.6: Κώδικας σε Matlab που παράγει τις θέσεις στις οποίες είναι τα “παγωμένα” bits

```

function outputs = partial_sums_initialize(N)           %!!!! care outputs
outputs = zeros(N/2,N,log2(N));                      %outputs(z,i,l) -- 1 stage ,bit
for l=0:1:log2(N)-1
    for z=0:1:N/2-1
        for i=0:1:N-1
            templ = 1;
            for u=1:1:log2(N)-2
                templ=and(not(xor(b(log2(N)-u-2,z),b(u+1,i))),temp1);
            end
            temp2=1;
            for w=0:1:l-1
                temp2=and(or(not(b(log2(N)-w-2,z)),b(w,i)),temp2);
            end
            outputs(z+1,i+1,l+1) = templ*temp2* not(b(l,i));
        end
    end
end

```

Σχήμα 4.7: Κώδικας σε Matlab που παράγει τον πίνακα με τα μερικά αθροίσματα

```

file = fopen('pipeline.txt','w');
for data = [8];
N = power(2,data);
K = N/2;
capacity = 0.5;
reverse_order = bitrevorder(1:1:N);
frozen_bits = initialize_frozen_bits(N,K,capacity);
partial_sum_adders = partial_sums_initialize(N);
sc_array = sc_array_initialize(N,zeros(1,N));
for bit=1:1:N
    [cycles,sc_array] = rec_c(1,reverse_order(bit),sc_array,N);
end
for bit=1:1:N %Arikan 0:1:N-1
    if(frozen_bits(bit) == 1)
        [cycles,sc_array] = rec_c(1,reverse_order(bit),sc_array,N);
        for l= log2(N):-1:1 %Arikan log2(N)-1:-1:0
            z = 0;
            for i = 1:1:N
                if(l ~= log2(N))
                    tmp = max(sc_array(i+sc_array(i,l,3),l+1,1),sc_array(i,l+1,1))+1;
                    if( tmp > sc_array(i,l,1))
                        sc_array(i,l,1) = tmp;
                    end
                end
                if(sc_array(i,l,2) == 1)
                    z = z+1;      %the number of adder
                    if(partial_sum_adders(z,bit,l) == 1)
                        sc_array(i,l,1) = 1 + cycles;      %getting values from l+1 state
                    end
                end
            end
        end
    end
end
sc_array(:,:,1)
end

```

Σχήμα 4.8: Κώδικας σε Matlab που παράγει τους χρονισμούς του scheduling για N=8

```

from math import *
N = 128
stages = int(log(N,2))
print '\n'
print '\n'
file = open(str(N)+"bitfrozen.txt","r")
frozen = file.readline()
frozen = frozen.replace(' ', '')
frozen = frozen.replace('\t', '')
bit_rev_order = range(0,N)
my_format = '{:0'+str(int(log(N,2)))+'}b'
for j in range(0,N):
    bit_rev_order[j] = int(my_format.format(j)[::-1], 2)
string = "u <="
k = -1;
reverse_frozen = frozen[::-1]
reverse_outputs = range(0,N/2)
for i in range(N/2-1,-1,-1):
    k = reverse_frozen.index('1',k+1)
    reverse_outputs[i]= bit_rev_order.index(N-1-k)
    string += "FF_O(0) ("+str(i)+") (sign_bit)&"
string += "\n"
print string
file.close()

#Initialize sc_array
file = open(str(N)+"pipeline.txt","r")
i = 0
sc_array = [[0 for x in range(stages+1)] for y in range(N)]
for line in file:
    line = [int(s) for s in line.split() if s.isdigit()]
    stage = 0
    for number in line:
        sc_array[i][stage] = number
        stage += 1
    i += 1

#Initialize g position in each stage
g_map = [[0 for x in range(N/2)] for y in range(stages)]
stage_temp = stages-1
for i in range(0,int(log(N,2))):
    counter = 0
    for j in range(0,N):
        if(int(((j)/(pow(2,i)))%2) == 1):
            g_map[stage_temp][counter] = j
            counter += 1
    stage_temp -= 1

file = open(str(N)+"bitpartials.txt","r")
stage = 0
output = 0
counter = 0
string = ""
string1 = ""
temp_c = 0
for line in file:
    if(counter % (N/2) == 0 and string!=""):
        stage += 1
        output = 0
    string += "\noutputs("+ str(stage)+") ("+ str(output) +" ) <= "
    line = line.replace(' ', '')
    line = line.replace('\t', '')
    for i in range(N):
        string += "-----"
        string1 += "-----"
        temp_c += 1

```

Σχήμα 4.9: Κώδικας σε Python που παράγει τον κώδικα σε VHDL για τα μερικά αθροίσματα

```

for i in range(N):
    if(i!=N-1 and line[i]=="1" and frozen[i]=="1"):
        if(sc_array[bit_rev_order[i]][0]+1 < sc_array[g_map[stage][output]][stage]):
            temp = ':entity Delay_FF_1bit generic map(cycles => '+str(sc_array[g_map[stage][output]][stage]-sc_array[bit_rev_order[i]][0]-1)+') port map(c
            check = string1.find(temp)
            if(check == -1):
                string1 += '\nU_Delay_FF_1bit'+str(temp_c)+':entity Delay_FF_1bit generic map(cycles => '+str(sc_array[g_map[stage][output]][stage]-sc_array[bit_rev_order[i]][0]-1)+') port map(c
                string += "temp("+str(temp_c)+" xor "
                temp_c += 1
            else:
                check1 = string1.find(")",check+len(temp)+1);
                string += "temp("+str(string1[check+len(temp):check1])+") xor "
        else:
            string += "estimated("+str(reverse_outputs.index(int(my_format.format(i)[::-1], 2)))+") xor "
    elif(i==N-1 and line[i]=="1" and frozen[i]=="1"):
        if(sc_array[bit_rev_order[i]][0]+1 < sc_array[g_map[stage][output]][stage]):
            temp = ':entity Delay_FF_1bit generic map(cycles => '+str(sc_array[g_map[stage][output]][stage]-sc_array[bit_rev_order[i]][0]-1)+') port map(c
            check = string1.find(temp)
            if(check == -1):
                string1 += '\nU_Delay_FF_1bit'+str(temp_c)+':entity Delay_FF_1bit generic map(cycles => '+str(sc_array[g_map[stage][output]][stage]-sc_array[bit_rev_order[i]][0]-1)+') port map(c
                string += "temp("+str(temp_c)+" xor "
                temp_c += 1
            else:
                string += "temp("+str(string1[check+len(temp):])+")"
        else:
            string += "estimated("+str(reverse_outputs.index(int(my_format.format(i)[::-1], 2)))+")"
    counter += 1
    output += 1
string = string.replace("<= \n", "<= '0';\n")
print string.replace(" xor \n", ',';\n').replace(")\n", ');\n')
print string1
#don't forget to fix last line!
file.close()

```

Σχήμα 4.10: Κώδικας σε Python που παράγει τον κώδικα σε VHDL για τα μερικά αθροίσματα

```

outputs(6)(32) <= temp(84) xor temp(85) xor temp(87) xor temp(88) xor temp(89) xor temp(91) xor temp(92) xor temp(94) xor temp(96) xor estimated(14);
outputs(6)(33) <= temp(85) xor temp(87) xor temp(88) xor temp(89) xor temp(91) xor temp(92) xor temp(94) xor temp(96) xor estimated(14);
outputs(6)(34) <= temp(84) xor temp(88) xor temp(89) xor temp(91) xor temp(92) xor temp(94) xor temp(96) xor estimated(14);
outputs(6)(35) <= temp(88) xor temp(89) xor temp(91) xor temp(92) xor temp(94) xor temp(96) xor estimated(14);
outputs(6)(36) <= temp(84) xor temp(85) xor temp(87) xor temp(92) xor temp(94) xor temp(96) xor estimated(14);
outputs(6)(37) <= temp(85) xor temp(87) xor temp(92) xor temp(94) xor temp(96) xor estimated(14);
outputs(6)(38) <= temp(84) xor temp(92) xor temp(94) xor temp(96) xor estimated(14);
outputs(6)(39) <= temp(92) xor temp(94) xor temp(96) xor estimated(14);
outputs(6)(40) <= temp(84) xor temp(85) xor temp(87) xor temp(89) xor temp(91) xor temp(96) xor estimated(14);
outputs(6)(41) <= temp(85) xor temp(87) xor temp(89) xor temp(91) xor temp(96) xor estimated(14);
outputs(6)(42) <= temp(84) xor temp(89) xor temp(91) xor temp(96) xor estimated(14);
outputs(6)(43) <= temp(89) xor temp(91) xor temp(96) xor estimated(14);
outputs(6)(44) <= temp(84) xor temp(85) xor temp(87) xor temp(96) xor estimated(14);
outputs(6)(45) <= temp(85) xor temp(87) xor temp(96) xor estimated(14);
outputs(6)(46) <= temp(84) xor temp(96) xor estimated(14);
outputs(6)(47) <= temp(96) xor estimated(14);
outputs(6)(48) <= temp(84) xor temp(87) xor temp(88) xor temp(91) xor temp(94) xor estimated(14);
outputs(6)(49) <= temp(87) xor temp(88) xor temp(91) xor temp(94) xor estimated(14);
outputs(6)(50) <= temp(84) xor temp(88) xor temp(91) xor temp(94) xor estimated(14);
outputs(6)(51) <= temp(88) xor temp(91) xor temp(94) xor estimated(14);
outputs(6)(52) <= temp(84) xor temp(87) xor temp(94) xor estimated(14);
outputs(6)(53) <= temp(87) xor temp(94) xor estimated(14);
outputs(6)(54) <= temp(84) xor temp(94) xor estimated(14);
outputs(6)(55) <= temp(94) xor estimated(14);
outputs(6)(56) <= temp(84) xor temp(87) xor temp(91) xor estimated(14);
outputs(6)(57) <= temp(87) xor temp(91) xor estimated(14);
outputs(6)(58) <= temp(84) xor temp(91) xor estimated(14);
outputs(6)(59) <= temp(91) xor estimated(14);
outputs(6)(60) <= temp(84) xor temp(87) xor estimated(14);
outputs(6)(61) <= temp(87) xor estimated(14);
outputs(6)(62) <= temp(84) xor estimated(14);
outputs(6)(63) <= estimated(14) xor

U_Delay_FF_1bit0:entity Delay_FF_1bit generic map(cycles => 1) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(48), q => temp(0));
U_Delay_FF_1bit1:entity Delay_FF_1bit generic map(cycles => 1) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(40), q => temp(1));
U_Delay_FF_1bit2:entity Delay_FF_1bit generic map(cycles => 1) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(25), q => temp(2));
U_Delay_FF_1bit3:entity Delay_FF_1bit generic map(cycles => 1) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(56), q => temp(3));
U_Delay_FF_1bit4:entity Delay_FF_1bit generic map(cycles => 1) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(36), q => temp(4));
U_Delay_FF_1bit5:entity Delay_FF_1bit generic map(cycles => 1) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(21), q => temp(5));
U_Delay_FF_1bit6:entity Delay_FF_1bit generic map(cycles => 1) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(52), q => temp(6));
U_Delay_FF_1bit7:entity Delay_FF_1bit generic map(cycles => 1) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(44), q => temp(7));
U_Delay_FF_1bit8:entity Delay_FF_1bit generic map(cycles => 1) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(29), q => temp(8));
U_Delay_FF_1bit9:entity Delay_FF_1bit generic map(cycles => 1) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(11), q => temp(9));
U_Delay_FF_1bit10:entity Delay_FF_1bit generic map(cycles => 1) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(60), q => temp(10));
U_Delay_FF_1bit11:entity Delay_FF_1bit generic map(cycles => 3) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(33), q => temp(11));
U_Delay_FF_1bit12:entity Delay_FF_1bit generic map(cycles => 1) port map(clk => clk, rst=> rst, ce=> ce_inputs, d=> estimated(34), q => temp(12));

```

Σχήμα 4.11: Κώδικας VHDL που παράγθηκε από την Python

4.6 Αποτελέσματα Υλοποίησης

Τρία είναι τα βασικά κομμάτια που μας απασχόλησαν στο σύστημα μας. Ο encoder, ο decoder και το ολοκληρωμένο σύστημα μέτρησης λαθών BER.

4.6.1 Encoder

Ο Encoder, εφόσον αποτελείται από ένα κύκλωμα xor ανάμεσα στα bits εισόδου και εξόδου, δεν παίζει καθοριστικό ρόλο στο ρολόι του κυκλώματος, καθώς το critical path του μπορεί να αποτελείται μερικές πύλες xor, άρα δεν θα ασχοληθούμε ιδιαίτερα με αυτό. Παρουσιάζεται για $N=256$ που επικαλύπτει την κατανάλωση των πόρων για μικρότερα N .

Οπότε για $N=256$ τα resources στην συσκευή Virtex-7 συγκεντρώνονται στο σχήμα 4.12.

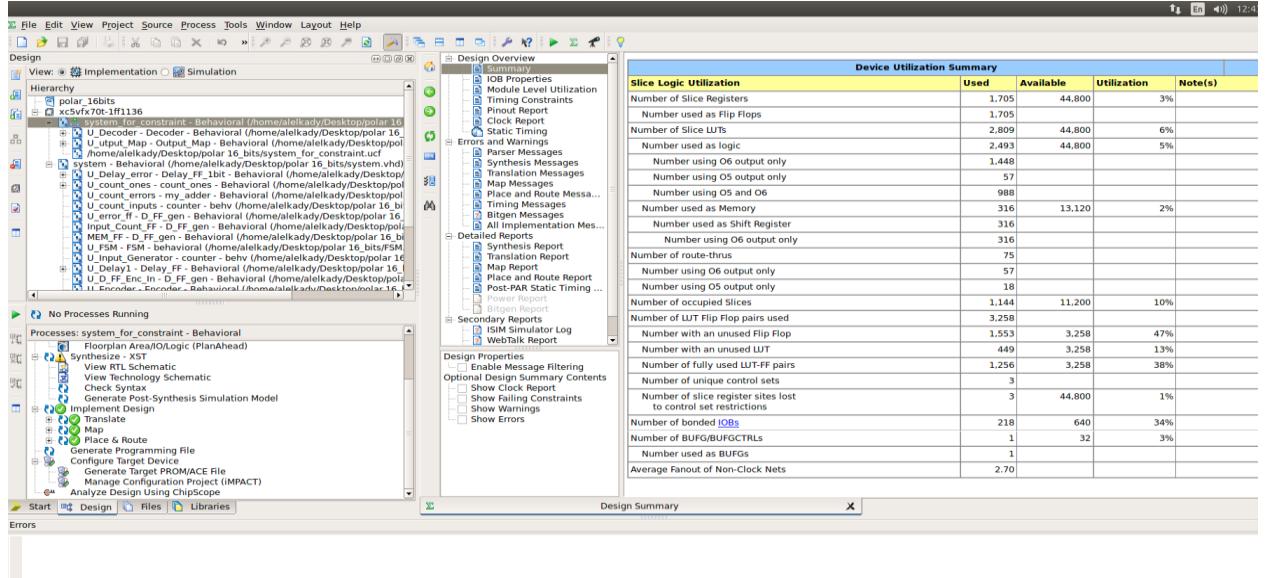
Device Utilization Summary (estimated values)				[]
Logic Utilization	Used	Available	Utilization	
Number of Slice LUTs	538	303600	0%	
Number of fully used LUT-FF pairs	0	538	0%	
Number of bonded IOBs	384	700	54%	

Σχήμα 4.12: Resources of Encoder $N=256$ System, on Virtex 7

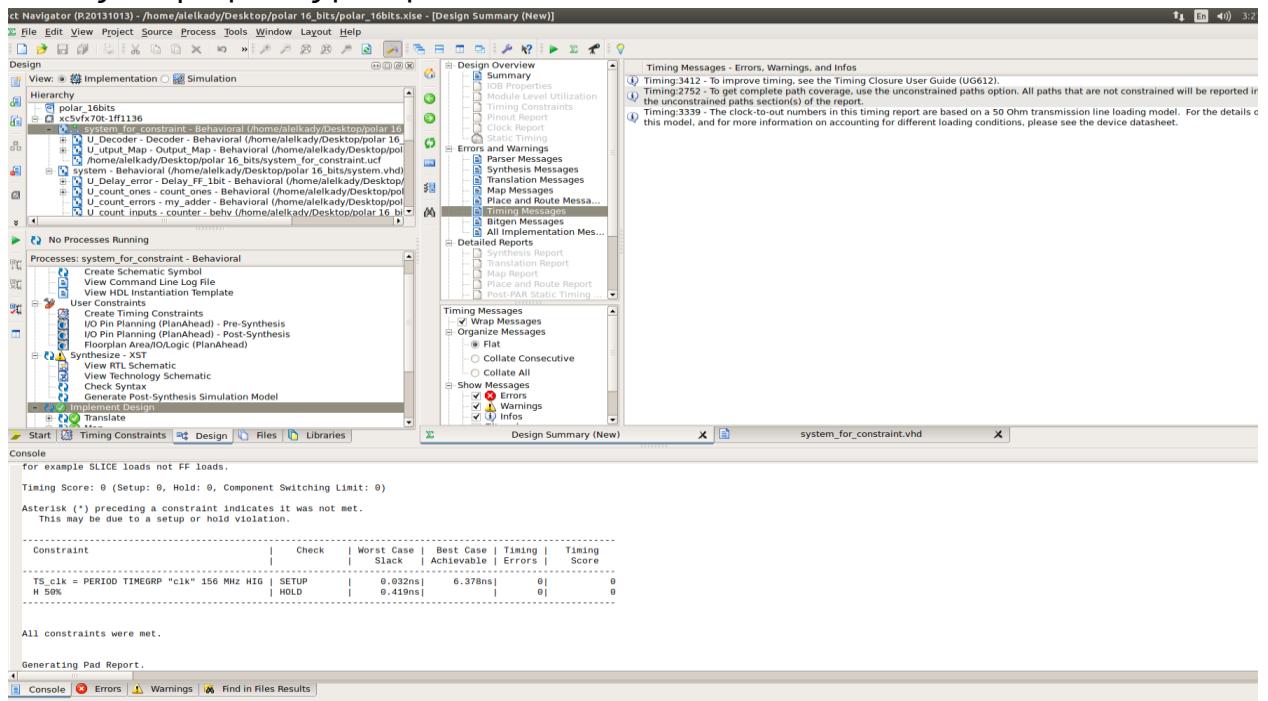
4.6.2 Decoder

Ο Decoder αποτελεί το βασικό κομμάτι του συστήματος και αυτός ορίζει και το ρολόι, όπως αναφέρθηκε παραπάνω.

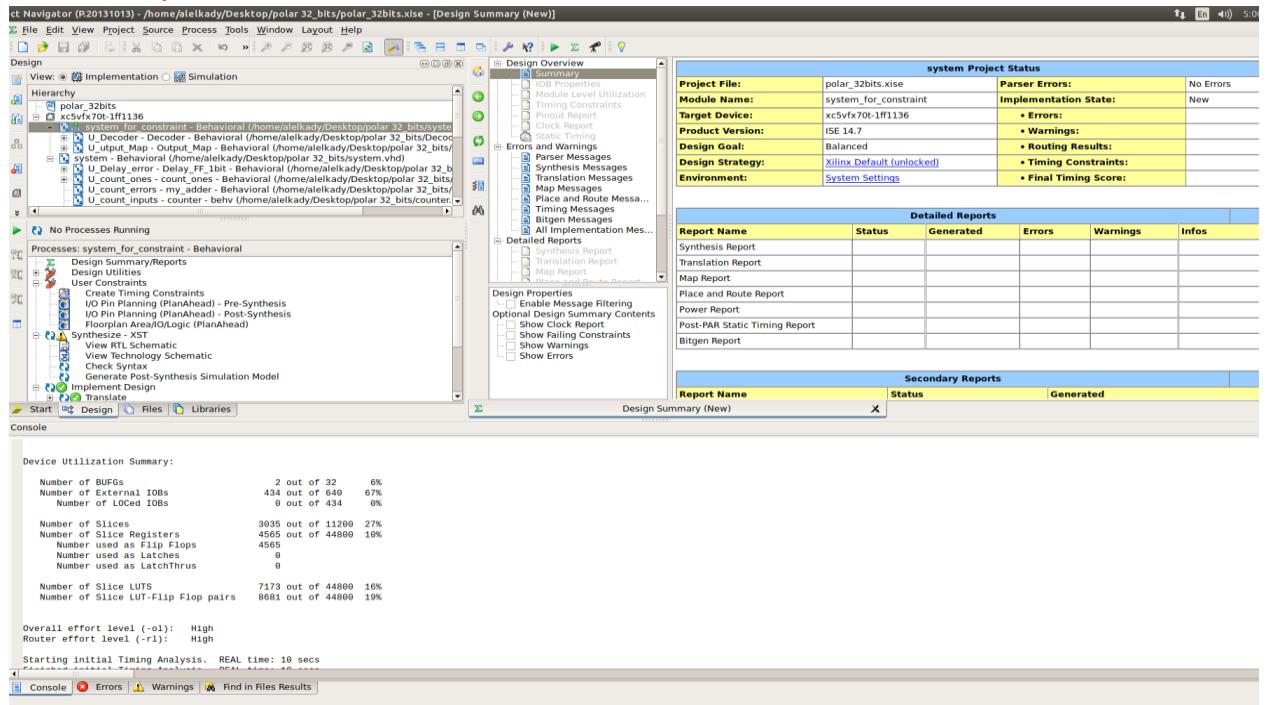
- Τα resources που χρησιμοποιούνται για $N=16$, στην συσκευή Virtex-5 vfx70t, είναι τα παρακάτω:



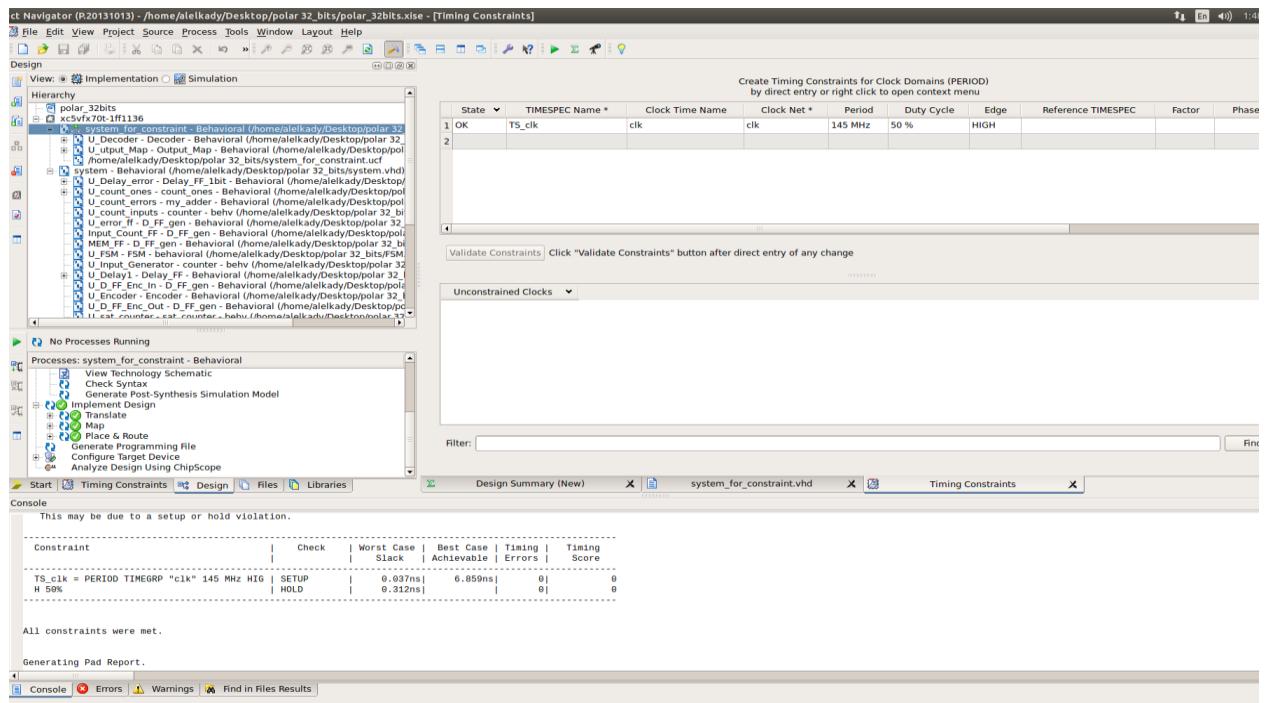
Καθώς και η περίοδος ρολογιού 156 MHz:



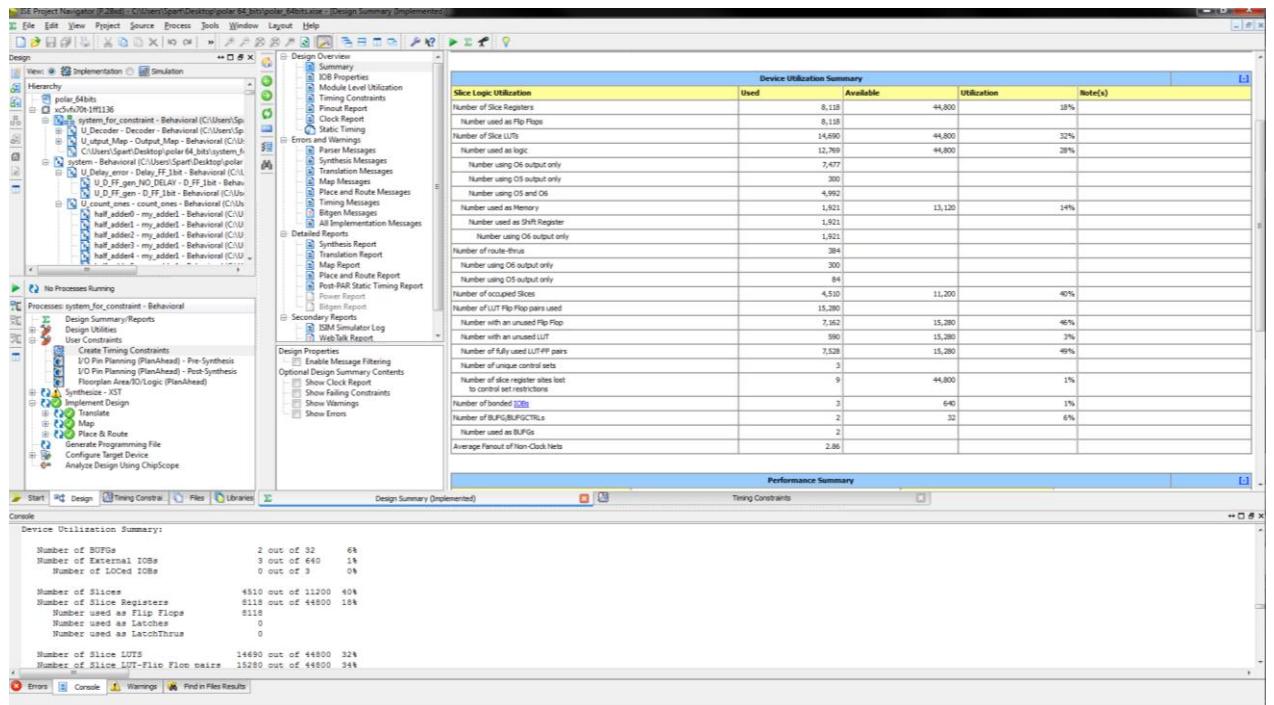
2. Τα resources που χρησιμοποιούνται για $N=32$, στην συσκευή Virtex-5 vfx70t, είναι τα παρακάτω:



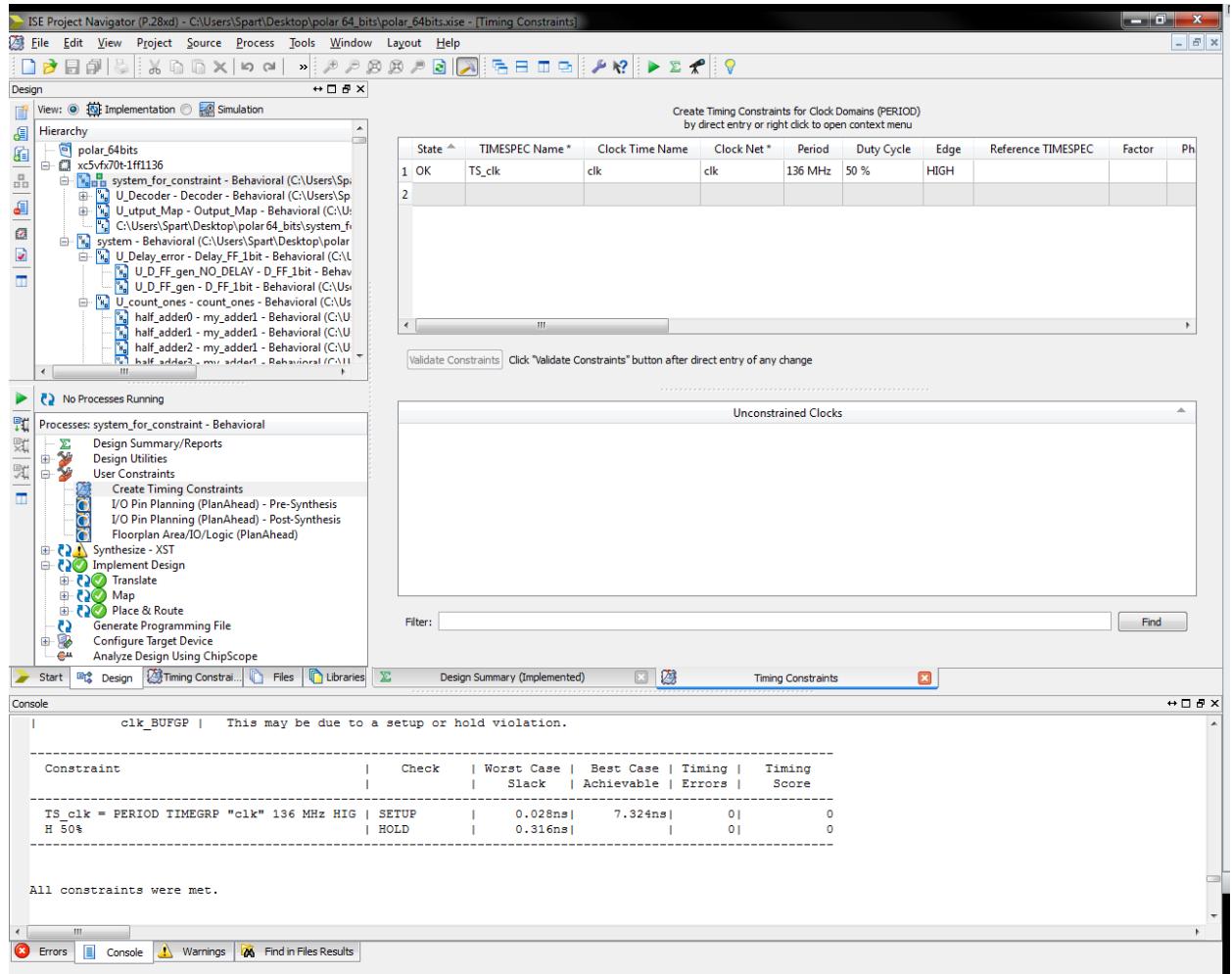
Καθώς και η περίοδος ρολογιού 145 MHz:



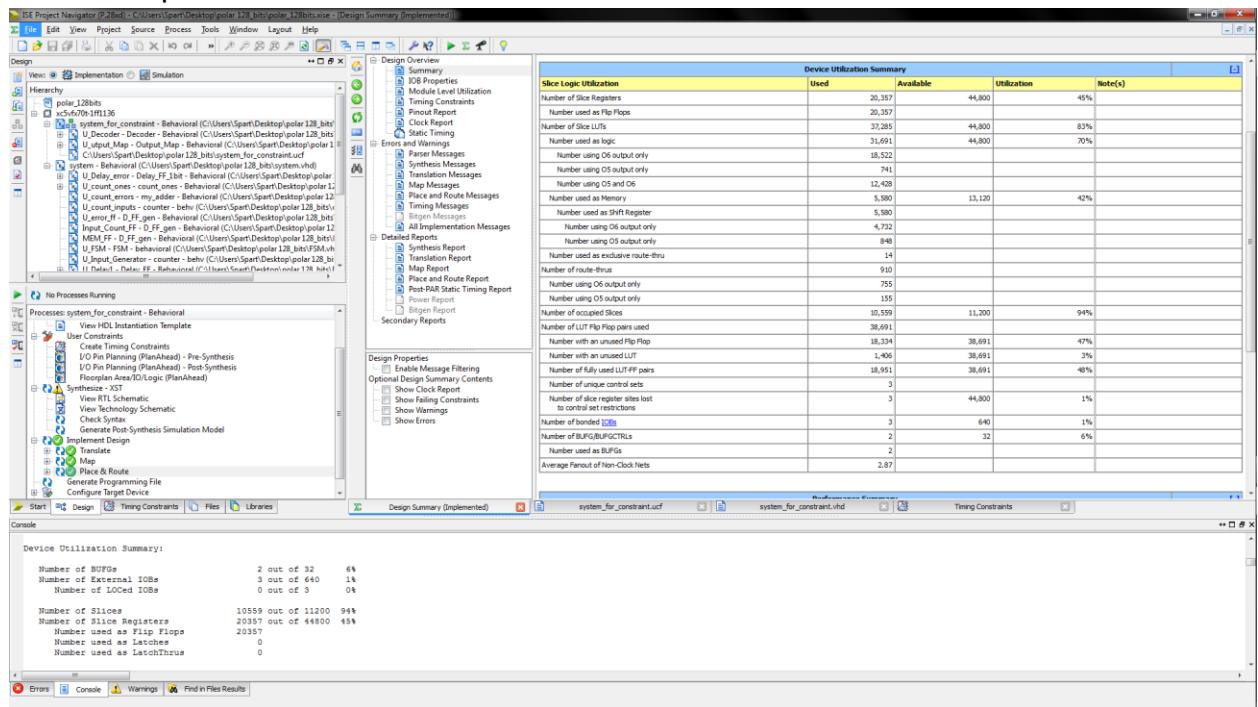
3. Τα resources που χρησιμοποιούνται για **N=64**, στην συσκευή **Virtex-5 vfx70t**, είναι τα παρακάτω:



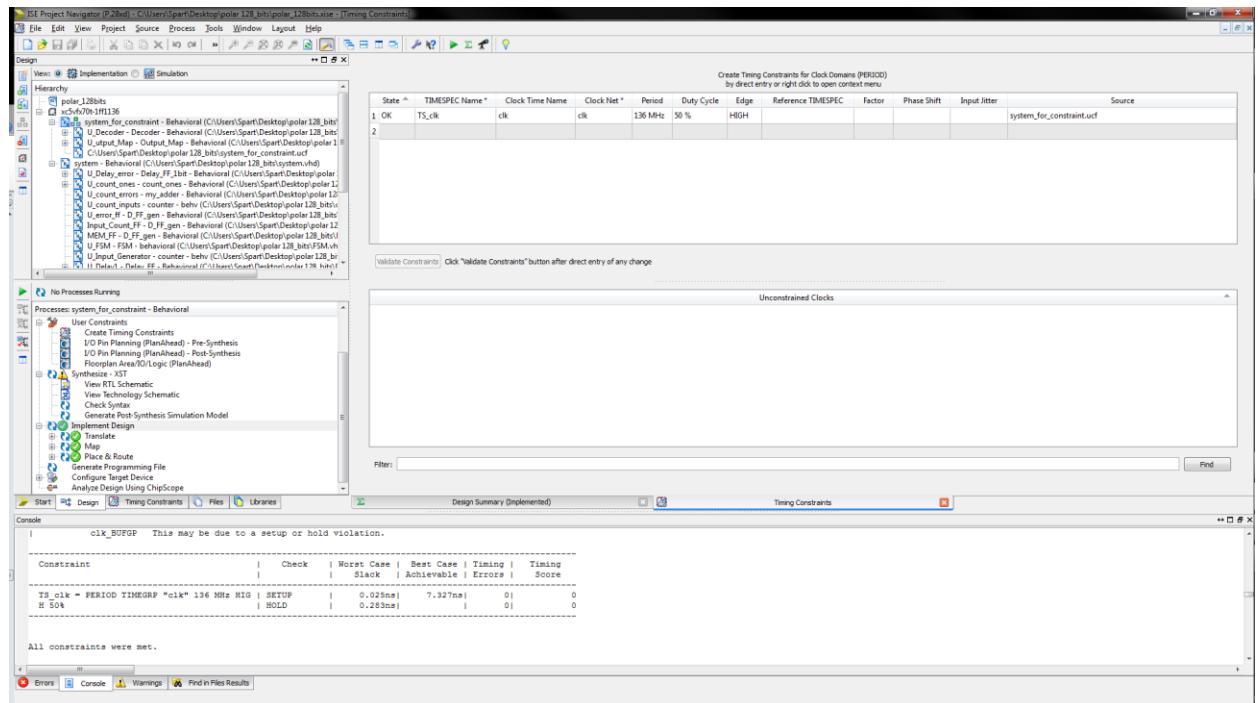
Καθώς και η περίοδος ρολογιού 136 MHz:



4. Τα resources που χρησιμοποιούνται για $N=128$, στην συσκευή Virtex-6 vlx240t, είναι τα παρακάτω:



Καθώς και η περίοδος ρολογιού 136 MHz:



Βλέπουμε ότι η συχνότητα του ρολογιού έχει μειωθεί αρκετά, και αυτό διότι πρώτον αυξάνεται η διάδοση του ρολογιού σε καλώδια και δεύτερον το critical path αυξάνεται καθώς αυξάνονται και τα μερικά αθροίσματα που έχουν να υπολογισθούν.

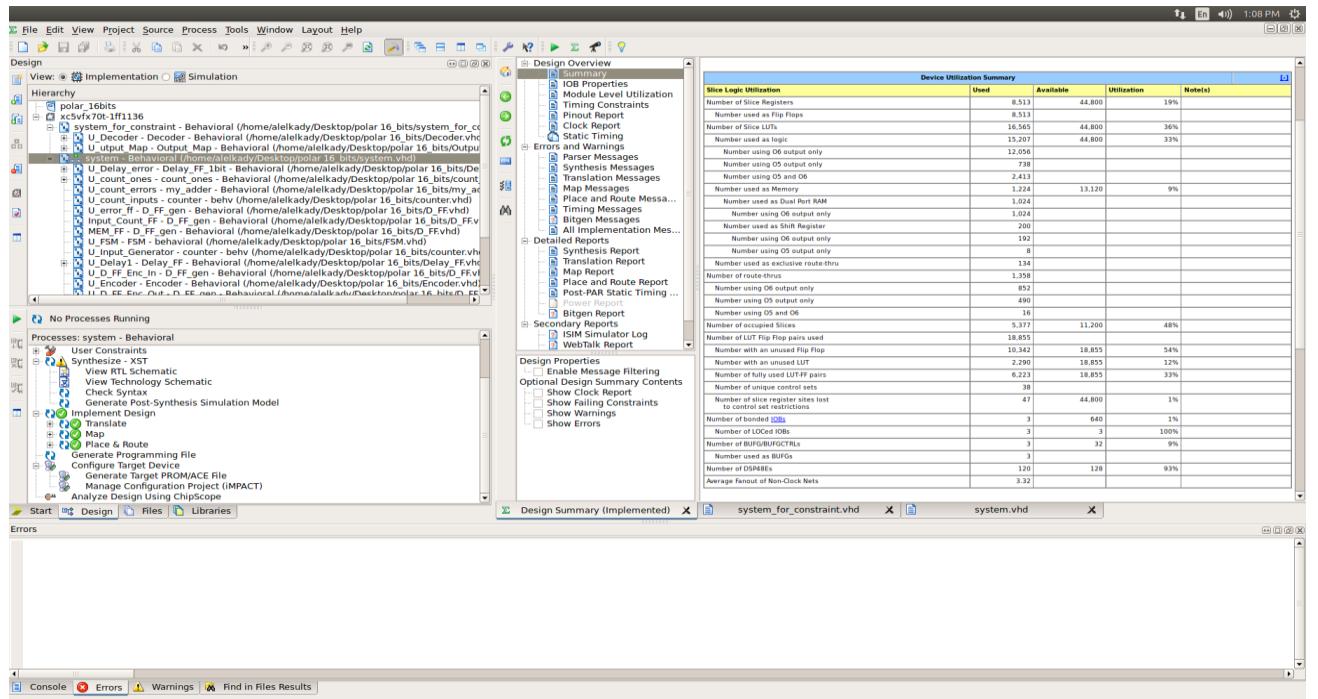
Μετά από όλα αυτά υπολογίζουμε το throughput rate των κυκλωμάτων πολλαπλασιάζοντας τη συχνότητα με την παραλληλία τους, έτσι έχουμε:

<i>N</i>	<i>Information bits</i>	<i>Device</i>	<i>Clock Frequency (MHz)</i>	<i>Throughput Rate (Gbps)</i>
16	8	Virtex 5	156	1,248
32	16	Virtex 5	145	2,320
64	32	Virtex 5	136	4,352
128	64	Virtex 6	136	8,704

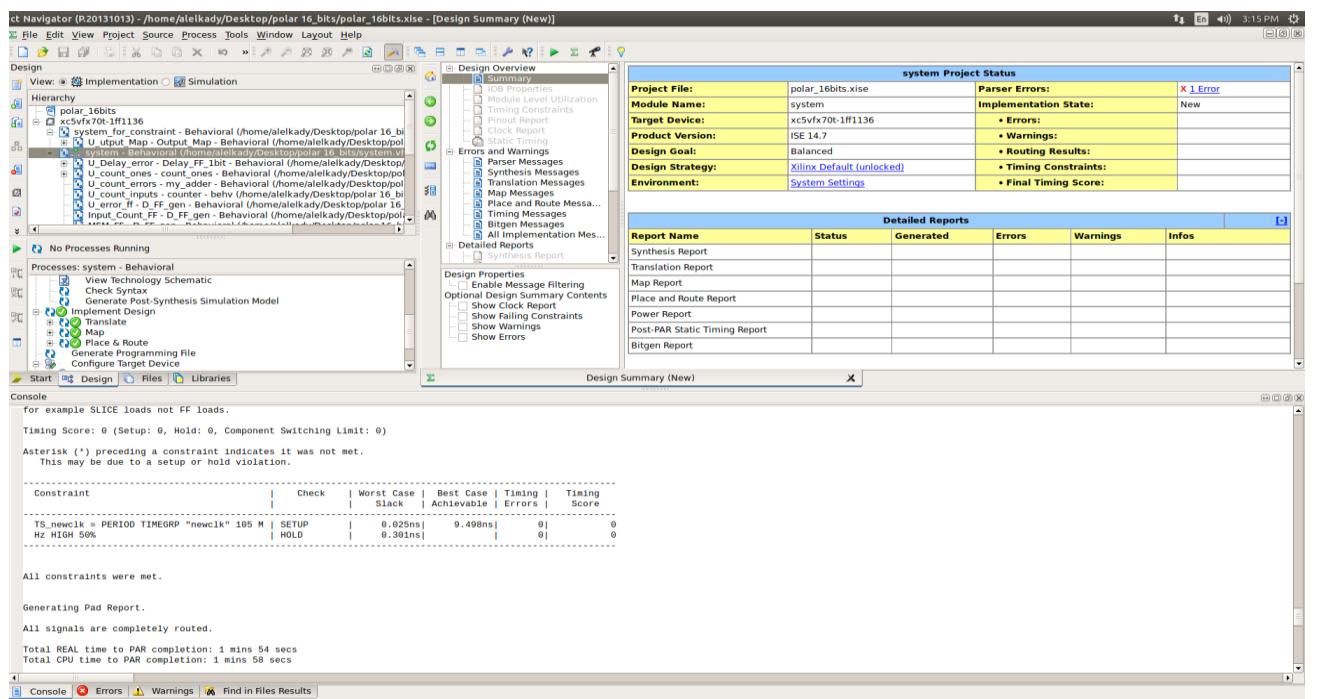
4.6.3 Encoder-Decoder με το Κανάλι

Το σύστημα αυτό αποτελεί το τελικό για την μέτρηση λαθών BER και σύγκριση των αποτελεσμάτων του hardware με αυτών του software (Matlab). Το σχηματικό του φαίνεται στο σχήμα 4.1. Αποτελείται από τον κωδικοποιητή, τον αποκωδικοποιητή και το κανάλι. Επιπροσθέτως έχει υλοποιηθεί και μία FSM η οποία έχει αναλυθεί παραπάνω για τον έλεγχο του συστήματος.

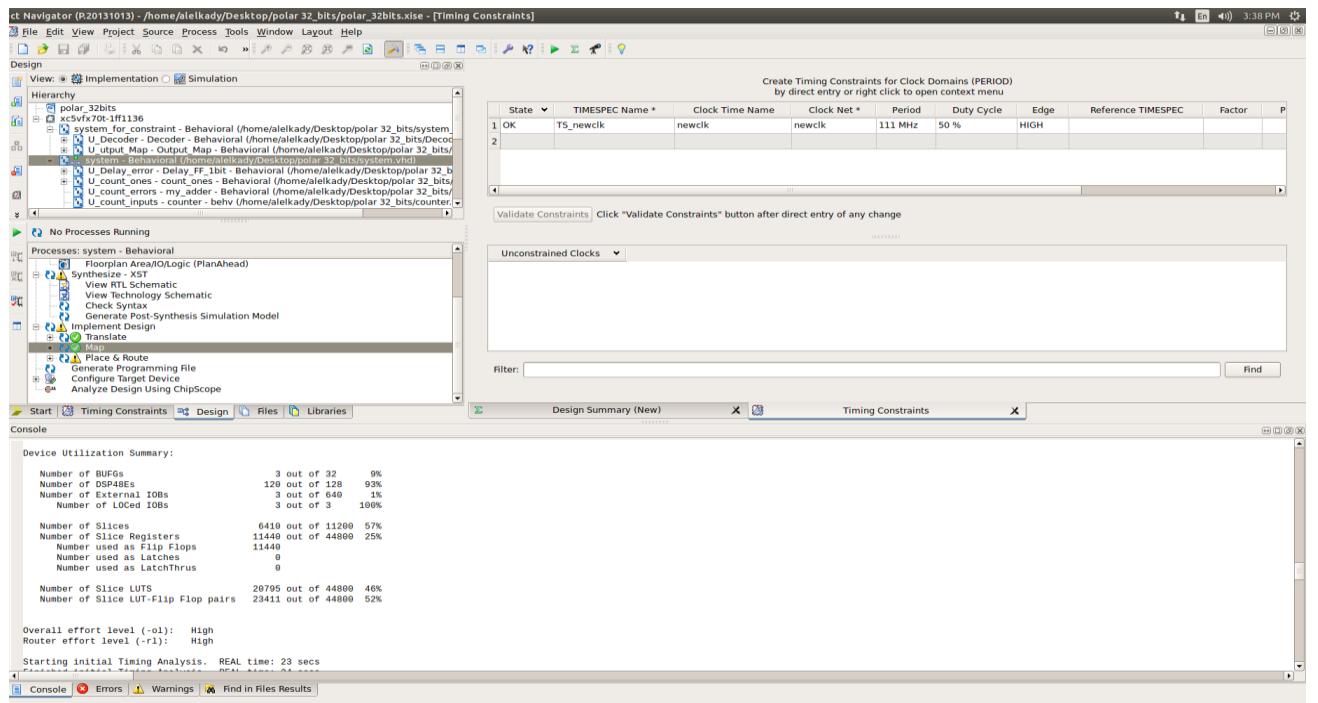
1. Τα resources που χρησιμοποιούνται για $N=16$, στην συσκευή Virtex-5 vlx70t, είναι τα παρακάτω:



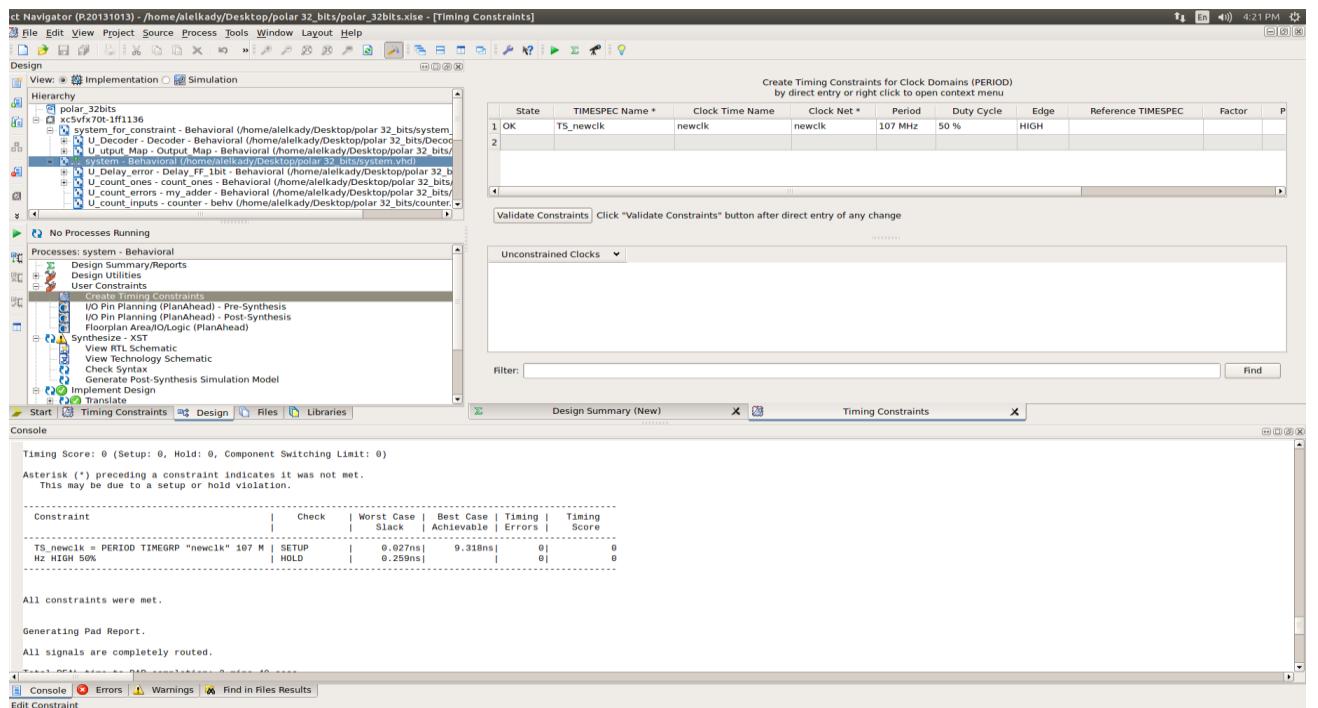
Και το ρολόι 105 MHz:



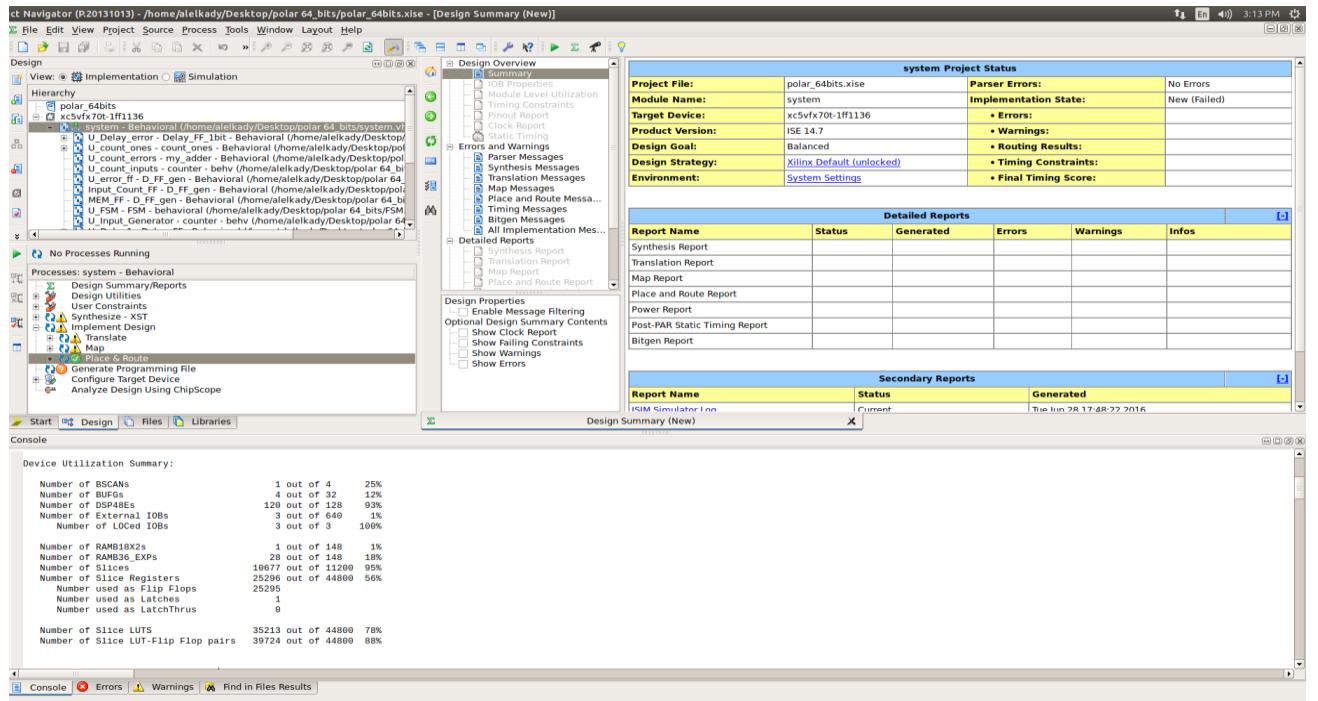
2. Τα resources που χρησιμοποιούνται για $N=32$, στην συσκευή Virtex-5 vlx70t, είναι τα παρακάτω:



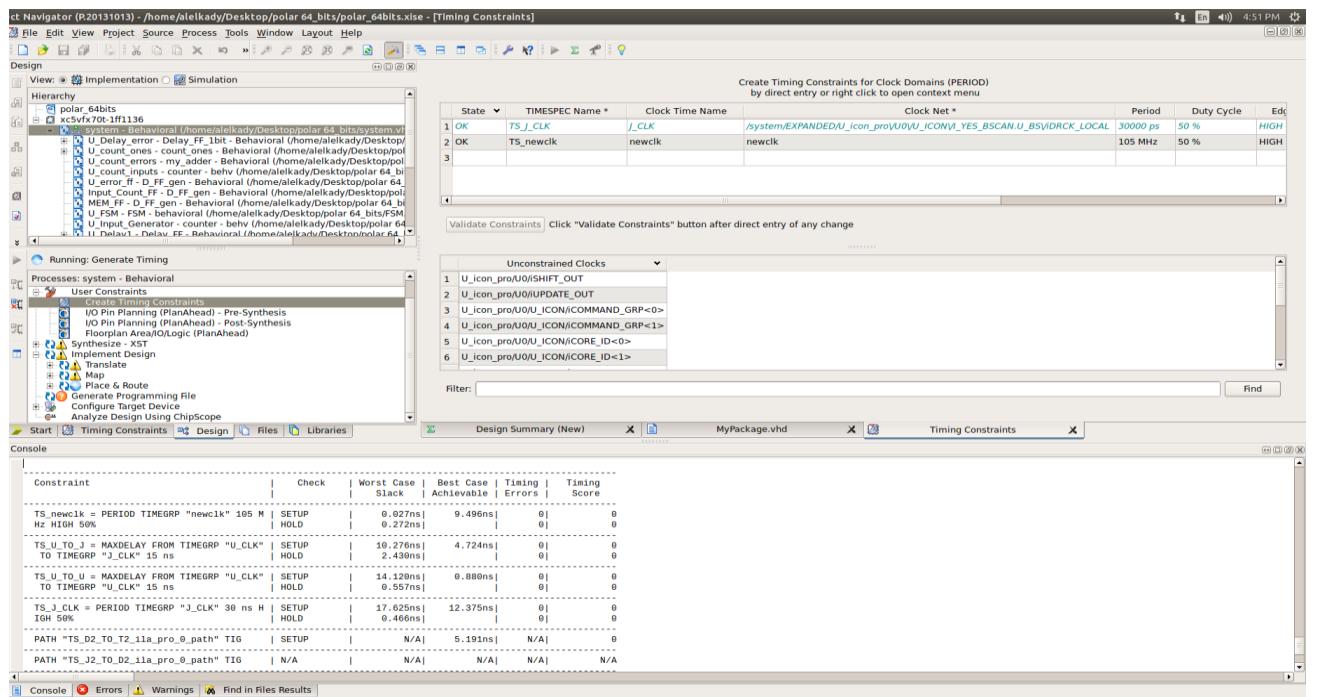
Και το ρολόι 107 MHz:



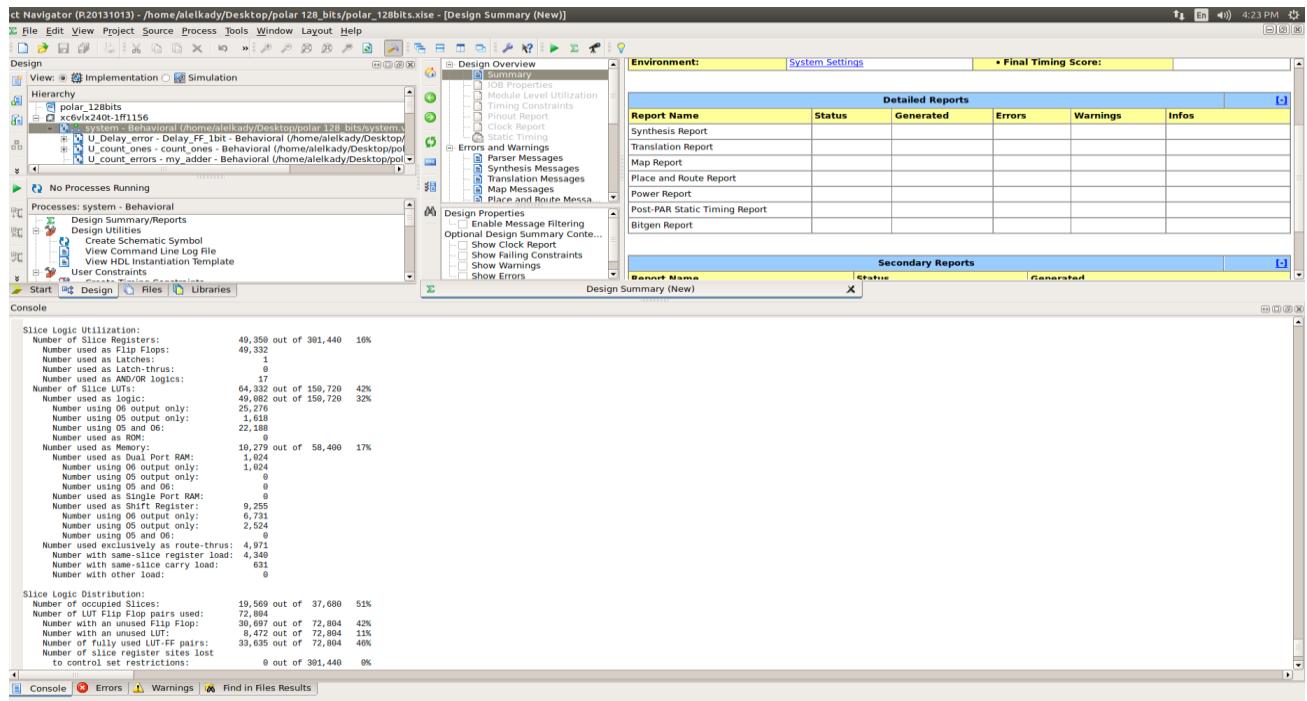
3. Τα resources που χρησιμοποιούνται για $N=64$, στην συσκευή Virtex-5 vlx70t, είναι τα παρακάτω:



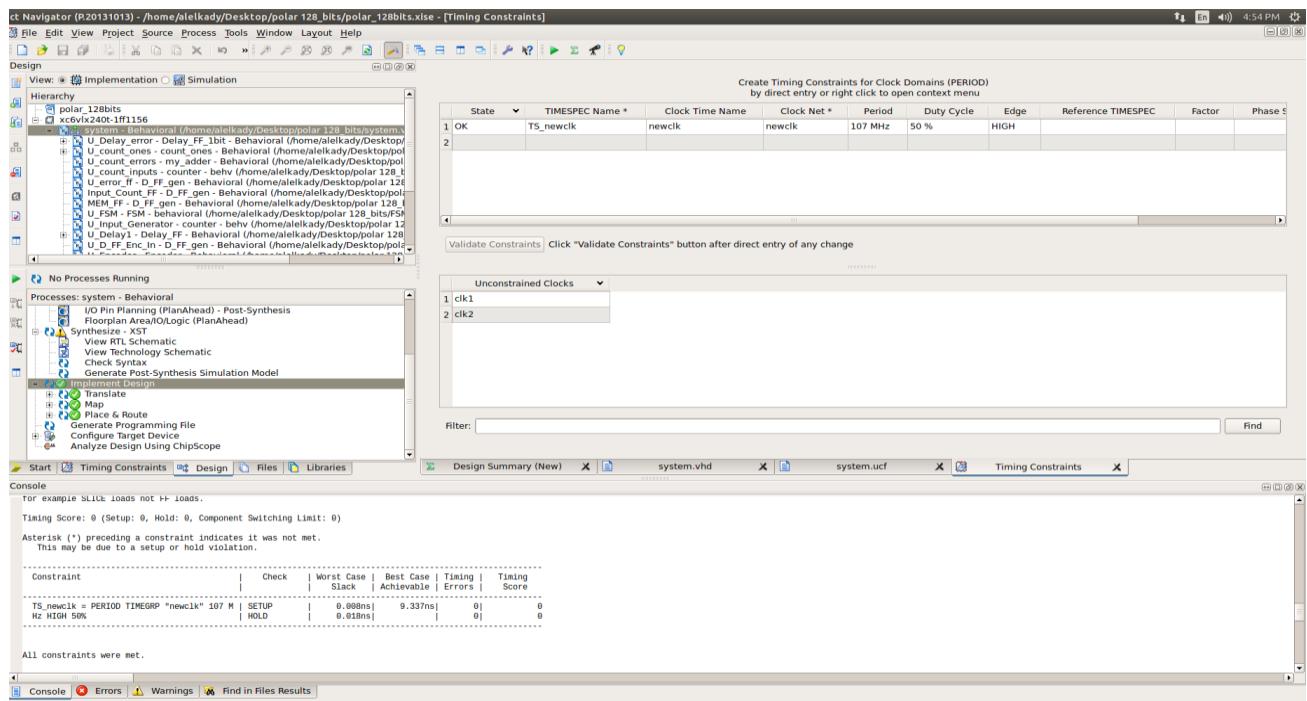
Και το ρολόι 105 MHz:



4. Τα resources που χρησιμοποιούνται για $N=128$, στην συσκευή Virtex-6 vlx240t, είναι τα παρακάτω:



Και το ρολόι 107 MHz:



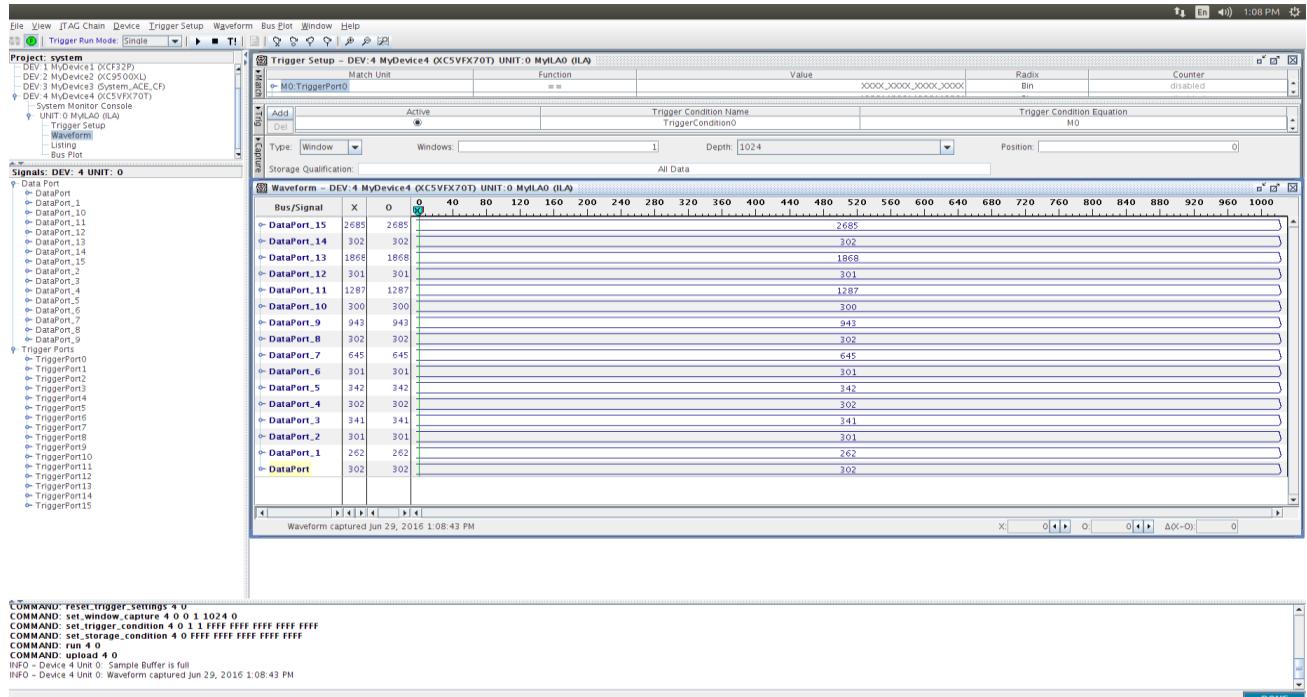
4.6.4 Αποτελέσματα BER

Το παραπάνω σύστημα που υλοποιήθηκε για $N = 16,32,64,128$ είχε ως αποτέλεσμα τα BER που θα παρουσιαστούν παρακάτω, για τις αντίστοιχες στάθμες θορύβου SNR = 0-7db. Οι οποίες πάρθηκαν με την υλοποίηση του IP ChipScope στο ISE στα αντίστοιχα σήματα προς μέτρηση.

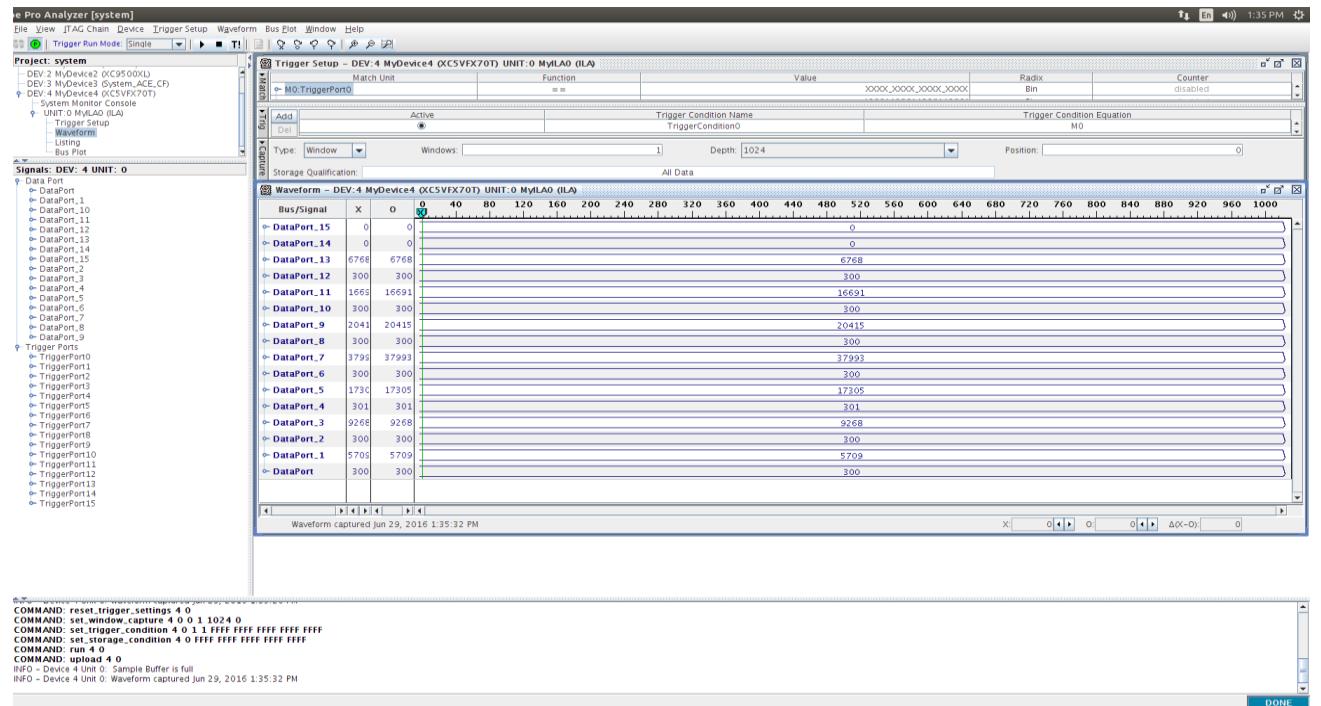
Για κάθε στάθμη θορύβου SNR υπάρχει μέτρηση των λέξεων που πέρασαν προς μέτρηση και μέτρηση των λαθών που έγιναν. Η συνθήκη τερματισμού, που γίνεται μέσω της FSM που αναφέρθηκε, έχει να κάνει με την παρουσίαση ορισμένων λαθών στο εκάστοτε SNR. Όταν συμβεί αυτό αλλάζει μέσω ενός demultiplexer η θέση του καταχωρητή που αποθηκεύονται τα παραπάνω σήματα.

Έτσι έχουμε τις παρακάτω μετρήσεις:

1. Έχουμε για **N=16** και για **300** λάθη ως συνθήκη τερματισμού στο ChipScope του ISE τα σχήμα 4.13,4.14.

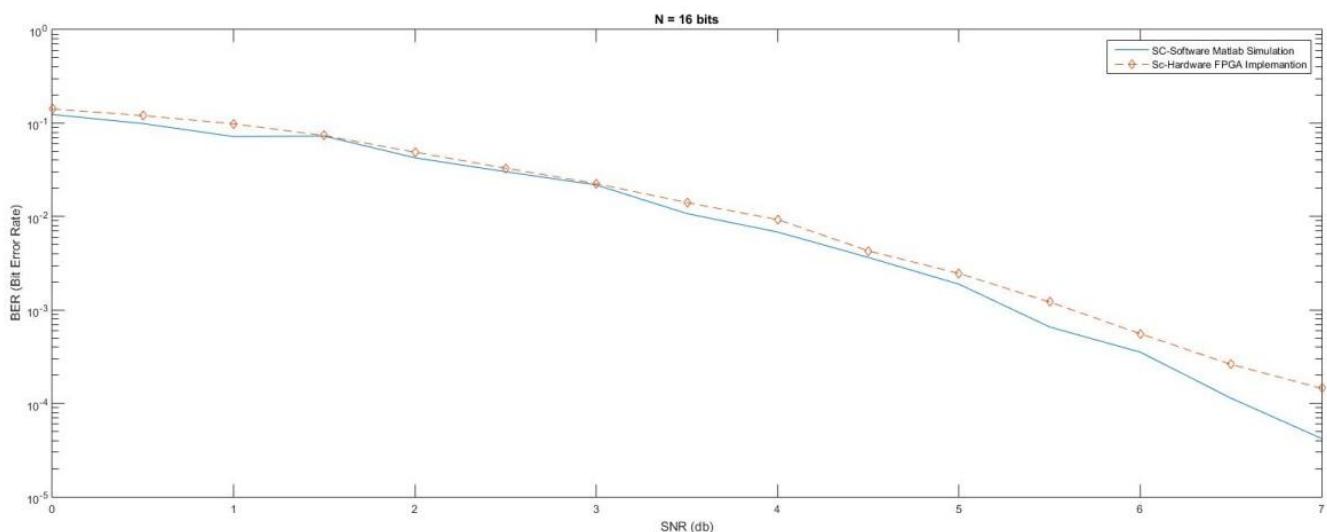


Σχήμα 4.13: 0-3.5 dB, N=16 για τουλάχιστον 300 errors



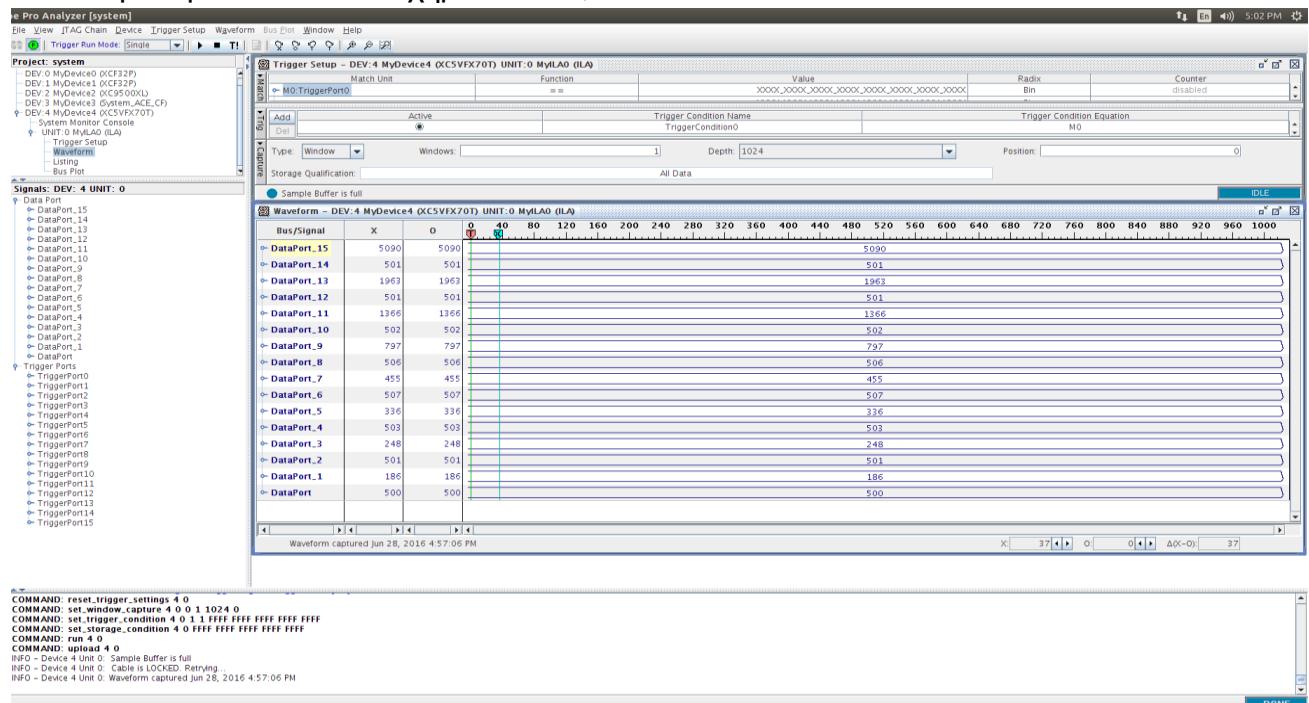
Σχήμα 4.14: 4-7dB, N=16 για τουλάχιστον 300 errors

Οι αντίστοιχες γραφικές σύγκρισης με τα αποτελέσματα της Matlab (για μικρότερα λάθη) όπου έχει μικρότερη ακρίβεια, παρουσιάζεται στο σχήμα 4.15.

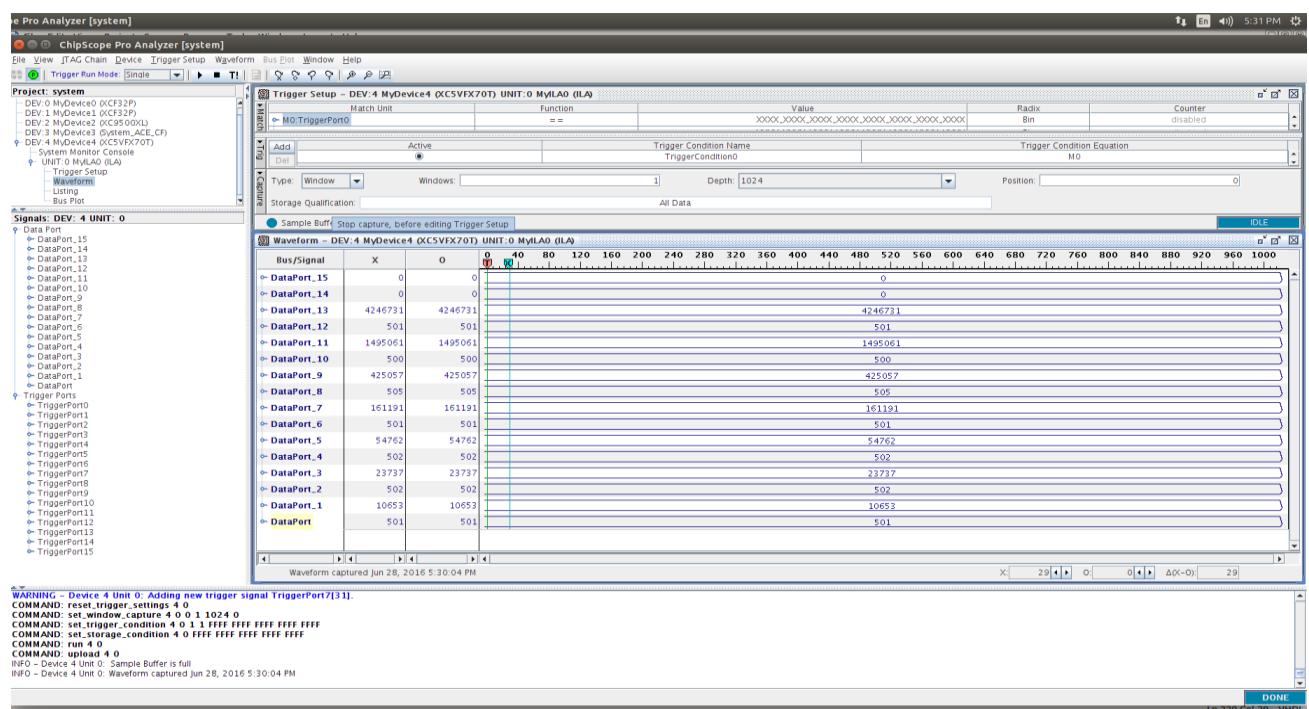


Σχήμα 4.15: Γραφική Παράσταση BER N=16, σε σύγκριση με αποτελέσματα της Matlab

2. Έχουμε για **N=32** και για **500** λάθη ως συνθήκη τερματισμού μέτρησης λαθών στο Chipscope του ISE τα σχήματα 4.16,4.17.

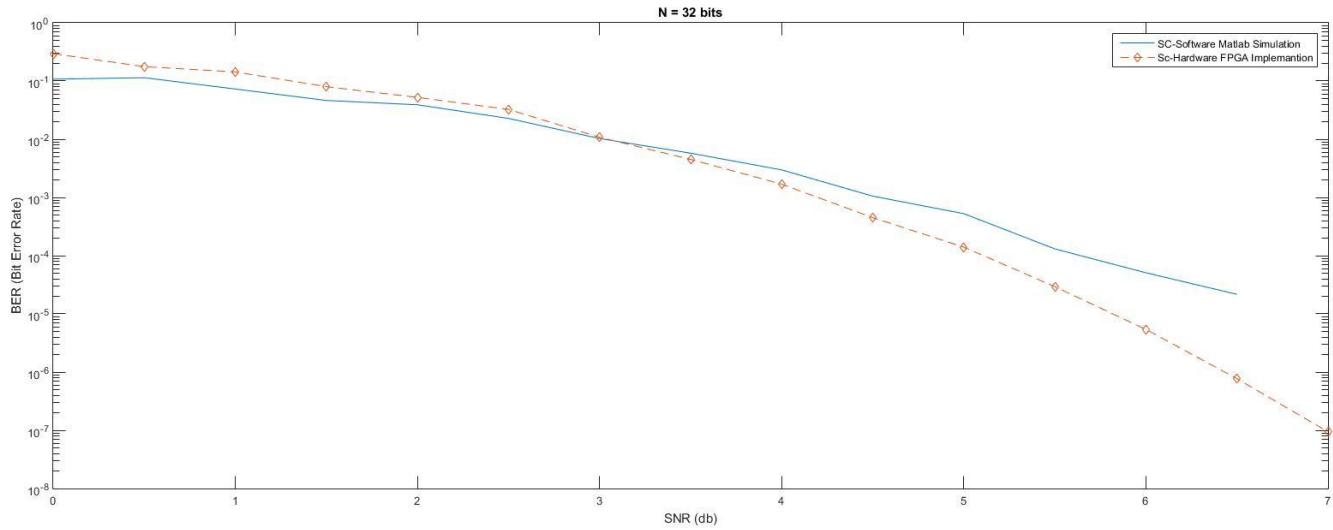


Σχήμα 4.16: 0-3.5 dB, N=32 για τουλάχιστον 500 errors



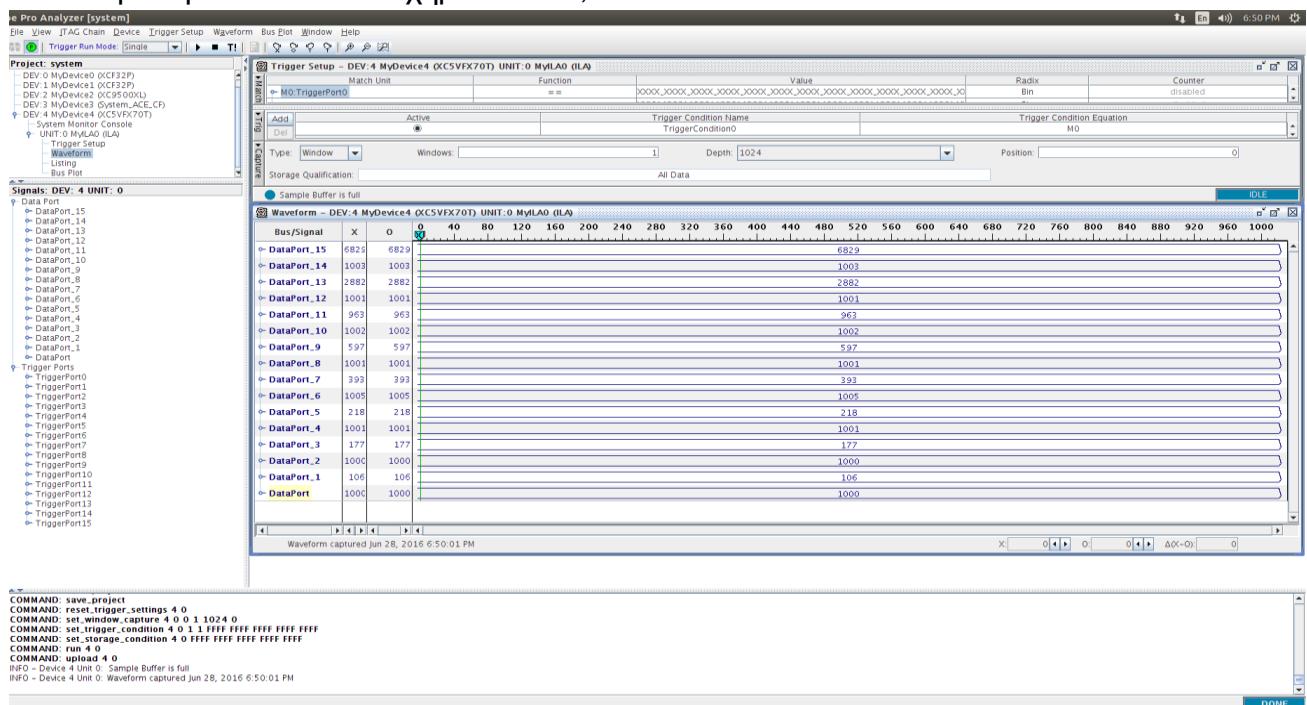
Σχήμα 4.17: 4-7 dB, N=16 για τουλάχιστον 500 errors

Οι ανίστοιχες γραφικές σύγκρισης με τα αποτελέσματα της Matlab (για μικρότερα λάθη) όπου έχει μικρότερη ακρίβεια, παρουσιάζεται στο σχήμα 4.18.

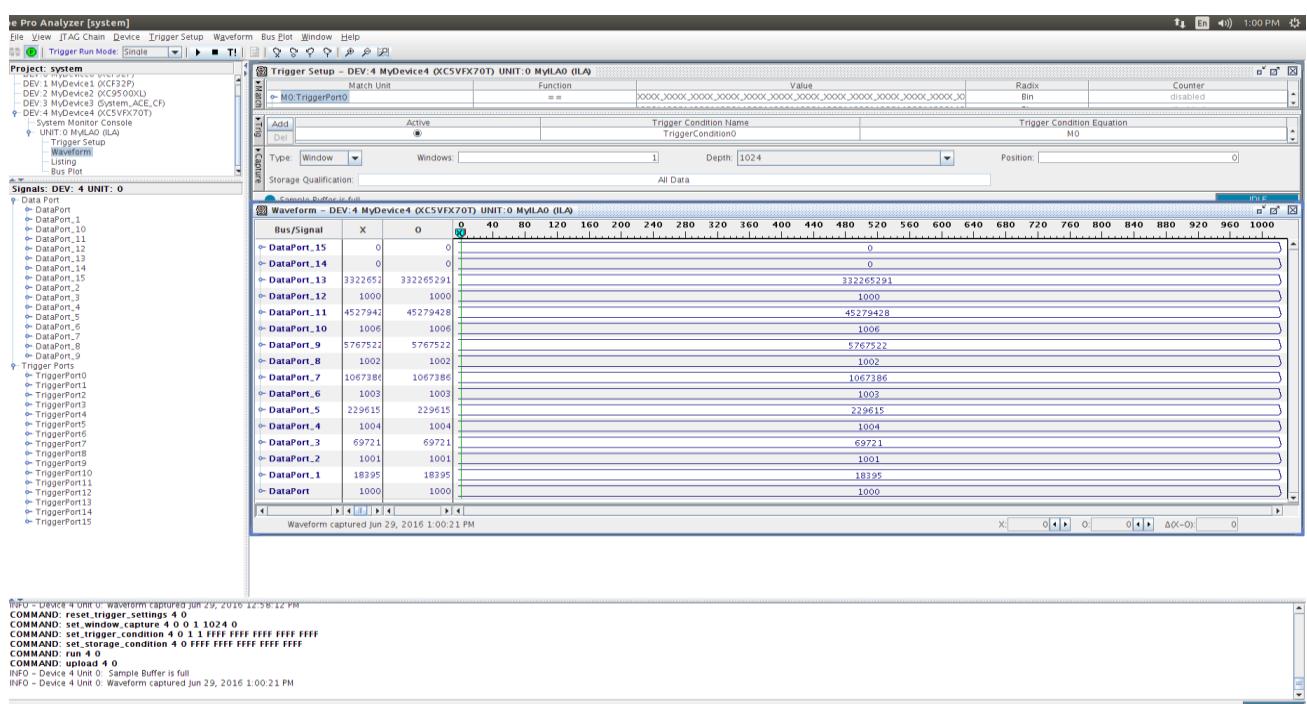


Σχήμα 4.18: Γραφική Παράσταση BER N=32, σε σύγκριση με αποτελέσματα της Matlab

3. Έχουμε για **N=64** και για **1000** λάθη ως συνθήκη τερματισμού μέτρησης λαθών στο Chipscope του ISE τα σχήματα 4.19,4.20.

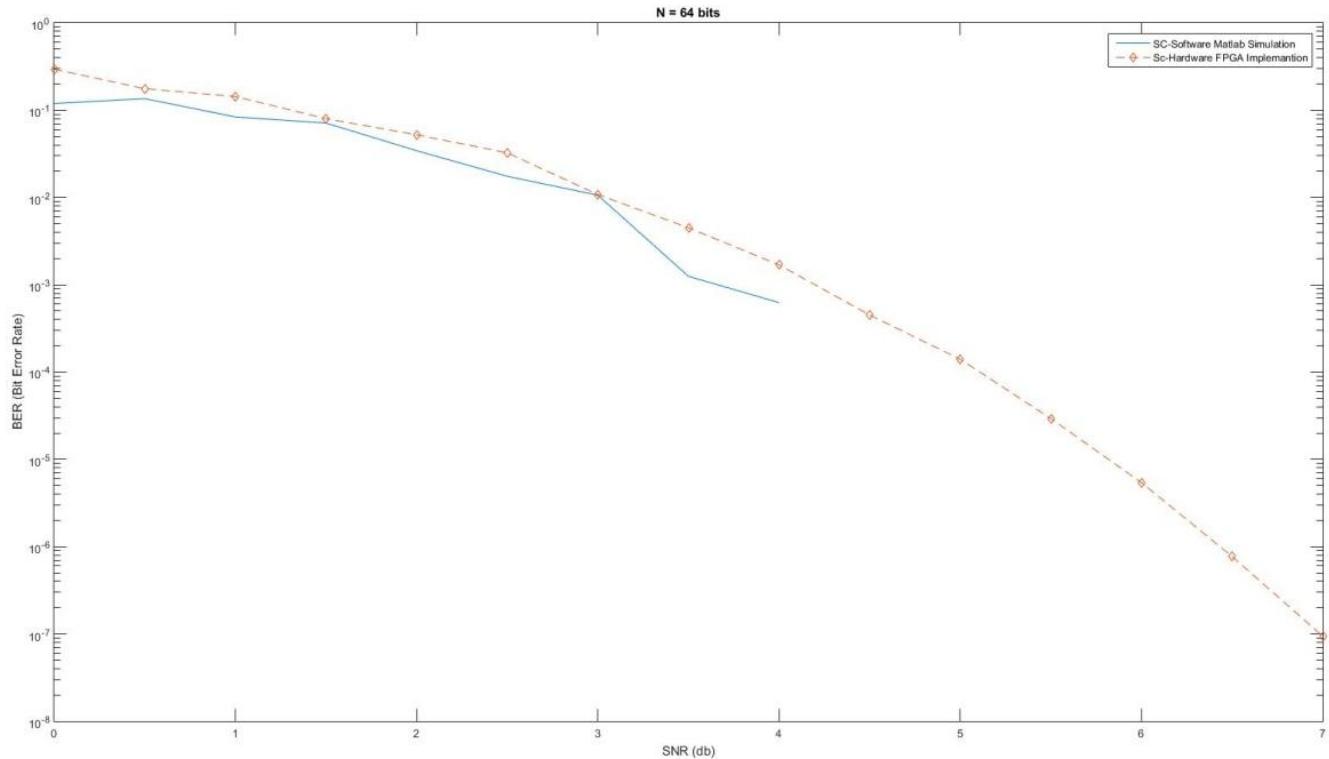


Σχήμα 4.19: 0-3.5 dB, N=64 για τουλάχιστον 1000 errors



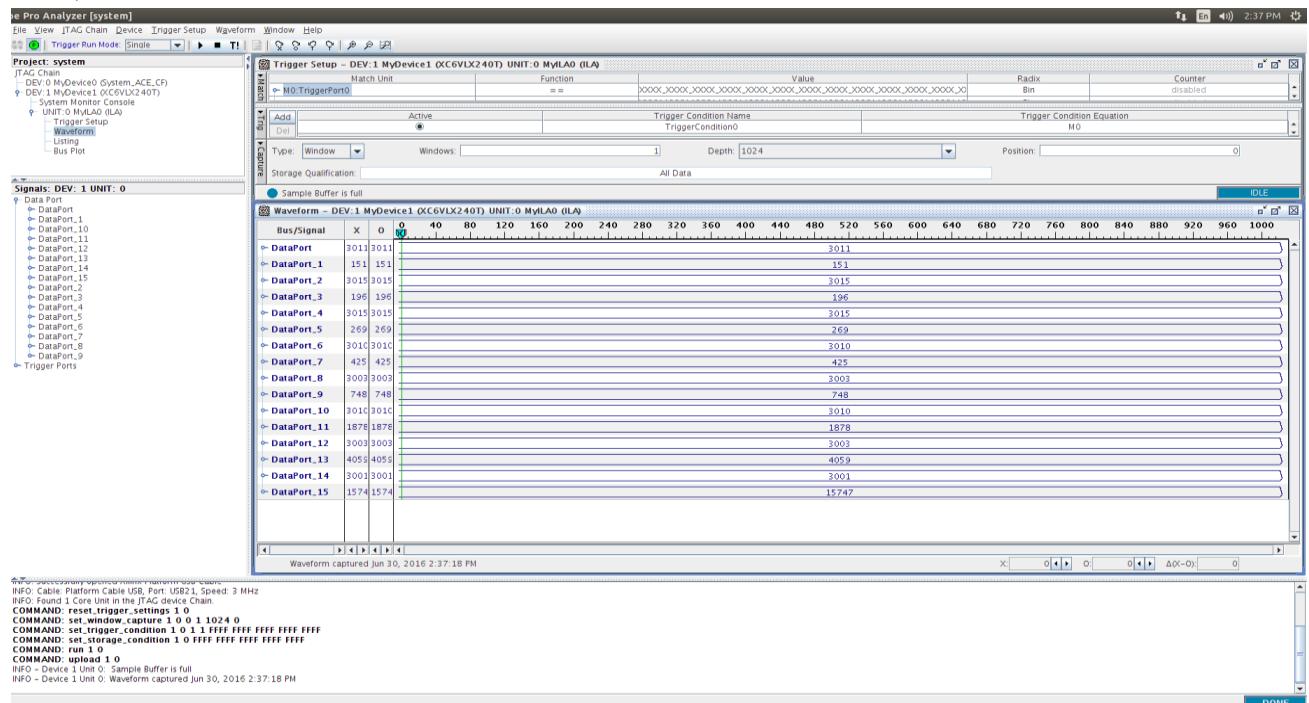
Σχήμα 4.20: 4-7 dB, N=64 για τουλάχιστον 1000 errors

Οι αντίστοιχες γραφικές σύγκρισης με τα αποτελέσματα της Matlab (για μικρότερα λάθη) όπου έχει μικρότερη ακρίβεια, παρουσιάζεται στο σχήμα 4.21.

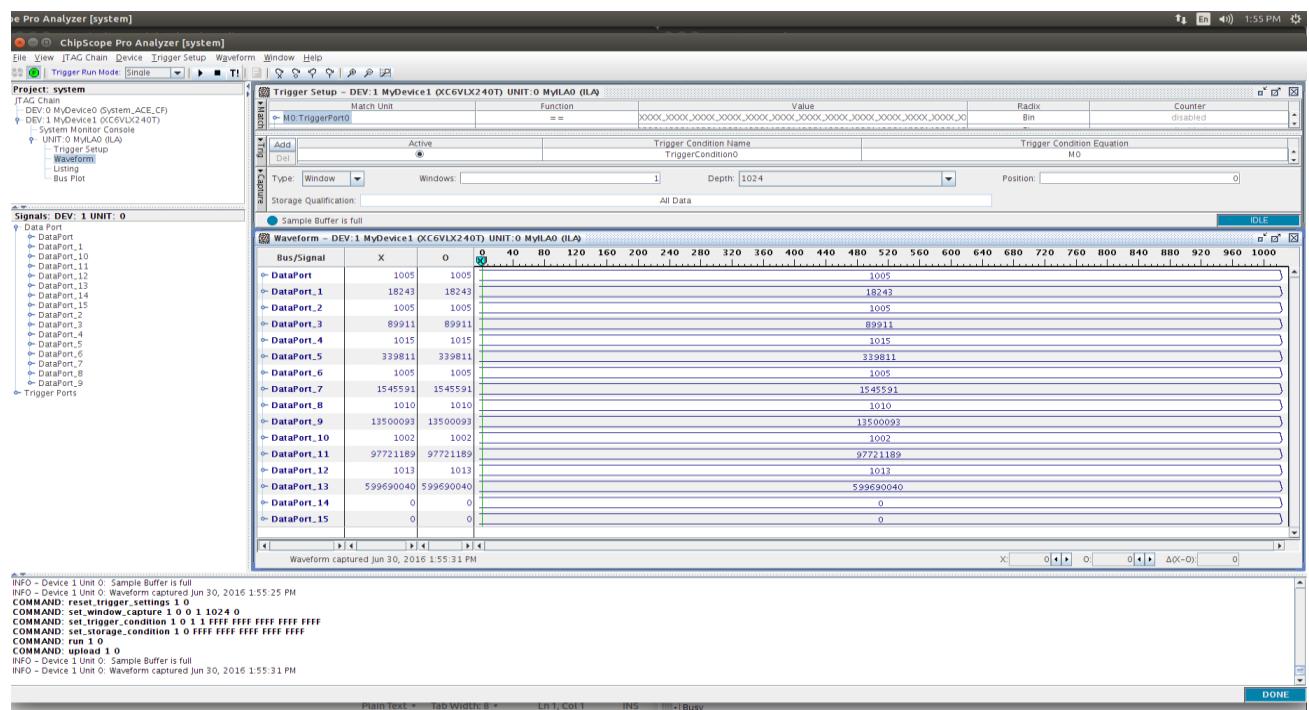


Σχήμα 4.21: Γραφική Παράσταση BER N=64, σε σύγκριση με αποτελέσματα της Matlab

4. Έχουμε για **N=128** και για **3000** λάθη από 0-3.5db και **1000** λάθη από 4-7db, ως συνθήκη τερματισμού μέτρησης λαθών, στο Chipscope του ISE τα σχήματα 4.22,4.23.

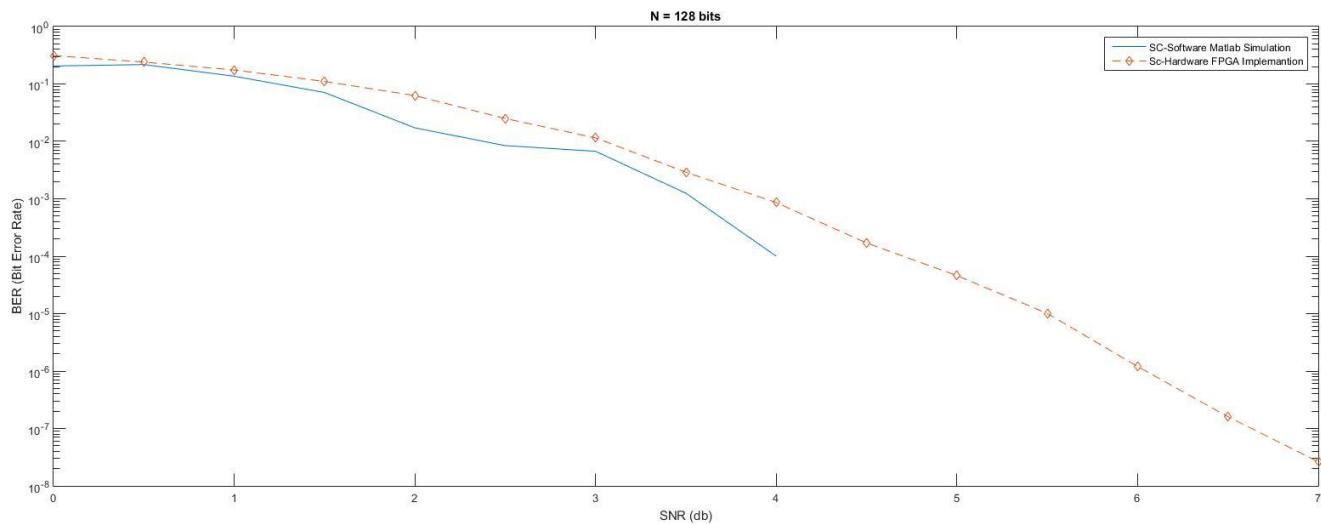


Σχήμα 4.22: 0-3.5 dB, N=128 για τουλάχιστον 3000 errors



Σχήμα 4.23: 4-7 dB, N=128 για τουλάχιστον 1000 errors

Οι ανίστοιχες γραφικές σύγκρισης με τα αποτελέσματα της Matlab (για μικρότερα λάθη) όπου έχει μικρότερη ακρίβεια, παρουσιάζεται στο σχήμα 4.24.



Σχήμα 4.24: Γραφική Παράσταση BER N=128, σε σύγκριση με αποτελέσματα της Matlab

Βλέπουμε λοιπόν ότι επιτυχώς η υλοποίηση μας σε hardware δίνουν περίπου τα ίδια αποτελέσματα με το Simulation στην Matlab και με μεγαλύτερη ακρίβεια και λιγότερες διακυμάνσεις στη γραφική, πράγμα που σημαίνει ότι το κύκλωμα μας δουλεύει όπως προβλέψαμε. Οι μικρές διαφορές στις γραφικές οφείλονται στον λευκό Γκαουσιανό Θόρυβο καθώς και την υλοποίηση του στο hardware.

Κεφάλαιο 5. Συμπεράσματα

Σε αυτήν την διπλωματική είδαμε αρχικά τι είναι οι polar codes καθώς και τα κομμάτια που τους απαρτίζουν, την πόλωση καναλιού, το σχήμα κωδικοποίησης, την υλοποίηση και τους διαφόρους αλγορίθμους αποκωδικοποίησης όπως SC, SCLD, CA-SCLD.

Στην συνέχεια υλοποιήσαμε το σύστημα του SC για $N=16,256$ σε FPGA και πήραμε τα αποτελέσματα χρόνων και αξιοποίησης των εκάστοτε board που χρησιμοποιήθηκαν.

Τέλος καταλήξαμε ότι για πιο ρεαλιστικά συστήματα, θα ήτανε πολύ χρήσιμη η επαναχρησιμοποίηση υλικού (επαναχρησιμοποίηση $W_{N/2}$ δύο φορές) ή ένα κύκλωμα με 1-entity ή P-entity butterfly (f & g), βλέπε [11] καθώς θα μειώνονταν αισθητά ο χώρος του κυκλώματος.

Οι Polar codes, αν και έχουν μερικά ανοιχτά θέματα υλοποίησης ακόμα (π.χ. SCLD $L=32$), θεωρούνται ότι θα αποτελέσουν ένα από τα βασικά σχήματα κωδικοποίησης στα μελλοντικά επικοινωνιακά συστήματα καθώς για μεγάλα μήκη κώδικα N , φτάνουν επιπυχώς την χωρητικότητα του καναλιού του Shannon.

Βιβλιογραφία

- [1] E. Arikan, "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels," in *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051-3073, July 2009.
- [2] K. Niu, K. Chen, J. Lin and Q. T. Zhang, "Polar codes: Primary concepts and practical decoding algorithms," in *IEEE Communications Magazine*, vol. 52, no. 7, pp. 192-203, July 2014.
- [3] A. Balatsoukas-Stimming, M. B. Parizi and A. Burg, "LLR-Based Successive Cancellation List Decoding of Polar Codes," in *IEEE Transactions on Signal Processing*, vol. 63, no. 19, pp. 5165-5179, Oct.1, 2015.
- [4] I. Tal and A. Vardy, "List decoding of polar codes," *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, St. Petersburg, 2011, pp. 1-5.
- [5] Alexandre J.Raymond, "Hardware Implementation of Successive-Cancellation Decoders for Polar Codes", Master Thesis, McGill University, August 2013
- [6] J. Guo, Z. Shi, Z. Liu, Z. Zhang and Q. Liu, "Multi-CRC Polar Codes and Their Applications," in *IEEE Communications Letters*, vol. 20, no. 2, pp. 212-215, Feb. 2016.
- [7] I. Paraskevakos and V. Paliouras, "A flexible high-throughput hardware architecture for a gaussian noise generator," *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Prague, 2011, pp. 1673-1676.
- [8] A. Eslami and H. Pishro-Nik, "A practical approach to polar codes," *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, St. Petersburg, 2011, pp. 16-20.
- [9] J. Guo, Z. Shi, Z. Liu, Z. Zhang and Q. Liu, "Multi-CRC Polar Codes and Their Applications," in *IEEE Communications Letters*, vol. 20, no. 2, pp. 212-215, Feb. 2016.

- [10] C. Leroux, A. J. Raymond, G. Sarkis and W. J. Gross, "A Semi-Parallel Successive-Cancellation Decoder for Polar Codes," in *IEEE Transactions on Signal Processing*, vol. 61, no. 2, pp. 289-299, Jan.15, 2013.
- [11] A. Pamuk, "An FPGA implementation architecture for decoding of polar codes," *Wireless Communication Systems (ISWCS), 2011 8th International Symposium on*, Aachen, 2011, pp. 437-441.
- [12] A. Balatsoukas-Stimming, A. J. Raymond, W. J. Gross and A. Burg, "Hardware Architecture for List Successive Cancellation Decoding of Polar Codes," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 8, pp. 609-613, Aug. 2014.