ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΕΦΑΡΜΟΣΜΕΝΑ ΗΛΕΚΤΡΟΝΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Εφαρμογές μη γραμμικών συστημάτων στην ασφάλεια ενσωματωμένων συστημάτων»

Του φοιτητή Απόστολου Ιατρόπουλου

Αρ. Μητρώου: elem52107m

Επιβλέπων

Ονοματεπώνυμο: Δημήτριος Παπακώστας

Βαθμίδα: Καθηγητής

Ημερομηνία: 30/9/2024

Τίτλος Δ.Ε.Εφαρμογές μη γραμμικών συστημάτων στην ασφάλεια ενσωματωμένων συστημάτων
Κωδικός Δ.Ε. 23255
Ονοματεπώνυμο φοιτητή: Απόστολος Ιατρόπουλος
Ονοματεπώνυμο εισηγητή: Δημήτριος Παπακώστας
Ημερομηνία ανάληψης Δ.Ε.21/09/2023
Ημερομηνία περάτωσης Δ.Ε.30/09/2024

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της μεταπτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Μεταπτυχιακό Πρόγραμμα Σπουδών «Εφαρμοσμένα Ηλεκτρονικά Συστήματα» στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Αφιέρωση

Στην οικογένεια μου και στην μέλλουσα γυναίκα μου Όλγα, καθότι η στήριξή τους αποτέλεσε τον ακρογωνιαίο λίθο στην περάτωση της διπλωματικής εργασίας.

Πρόλογος

Τα χαοτικά συστήματα καθώς και οι εφαρμογές τους σε πολλούς κλάδους της μηχανικής(ασφάλεια, μετρήσεις, βελτιστοποίηση σχεδιασμού διαδρομής) αποτελούν αντικείμενο εκτενούς έρευνας τις τελευταίες δεκαετίες. Με σκοπό την μελέτη χαοτικών συστημάτων αλλά και εφαρμογή τους σε ενσωματωμένα συστήματα για εφαρμογές κρυπτογράφησης έγινε και ανάληψη της διπλωματικής εργασίας. Βασικός σκοπός η τριβή με τεχνολογίες fpga για την σχεδίαση διατάξεων προσανατολισμένων στην κυβερνοασφάλεια.

Περίληψη

Στο πλαίσιο της διπλωματικής εργασίας θα μελετηθεί η συμπεριφορά μη γραμμικών συστημάτων και οι εφαρμογές τους σε θέματα ασφάλειας ενσωματωμένων συστημάτων. Η εργασία θα περιλαμβάνει θεωρητική μελέτη και δημιουργία μαθηματικών μοντέλων, ικανών να παράγουν τυχαίες χρονοσειρές, έλεγχο τυχαιότητας μέσω test-suites όπως (NIST SP 800-22,κ.α) και τέλος υλοποίηση των συστημάτων τόσο σε επίπεδο υλικολογισμικού(firmware) σε μικροελεγκτές, όσο και σε επίπεδο υλικού(hardware) σε FPGA.

Abstact

In the context of this thesis, the behavior of nonlinear systems and their applications to the security of embedded systems will be studied. The work will include theoretical research and the creation of mathematical models, capable of generating random time series, randomness testing using test suites such as (NIST SP 800-22, etc.), and finally, the implementation of these systems both at the firmware level on microcontrollers and at the hardware level on FPGAs.

Ευχαριστίες

Περιεχόμενα

Ευρετήριο Εικόνων

Ευρετήριο Πινάκων

Συντομογραφίες

ApEn Approximate entropy (εκτίμηση εντροπίας). 9

ASCII American Standard Code for Information Interchange. 10, 27

ASIC Application Specific Integrated Circuit. 1

DSP Digital Signal Process. 75

FFT Fast Fourier Transformation. 64

FPGA Field programmable gate arrays. 1

HAL Hardware Abstraction Layer. 31

LLR Log Likelihood ration. 7

LU Logical Unit. 59

MAC Multiply Accumulate. 75

NIST National Institute of Standards. 1

PLL Phase Lock Loop. 61

PRBG Pseudo Random Bit Genarator. 8, 18, 19

PRNG Pseudo Random Number Genarator. 1, 8, 14

PUF Physical unclonable function. 75

RBS Ripple Borrow Subtractor. 55

RCA Ripple Carry Adder. 52

RLM Reduce Logistic Map. 75

RLMIC Reduce Logistic Map Integrated Circuit. 57

RTL Register-transfer level. 72

SDIC sensitive dependence upon initial conditions (Ευαίσθητη εξάρτηση από τις αρχικές συνθήκες). 5

SNR Signal-Noise Ratio. 7

SSIM Structural Similarity Index (Δείκτης δομικής ομοιομορφίας). 7

UART Universal asynchronous receiver-transmitter. 33

USB Universal Serial Bus. 39

UUT Unit Under Test. 50

VHDL VHSIC HARDWARE DESCRIPTION LANGUAGE(Γλώσσα Περιγραφής Υλικού). 1

VWF Vector Wave File. 72

1 Εισαγωγή

Η παρούσα διπλωματική εργασία πραγματεύεται την μελέτη και την σχεδίαση μαθηματικών, μη-γραμμικών, χαοτικών μοντέλων, αλλά και την υλοποίησή τους τόσο σε επίπεδο firmware, όσο και σε επίπεδο hardware σε αναπτυξιακό FPGA. Σκοποί της εργασίας αποτέλεσαν οι:

- Μελέτη ενός χαοτικού χάρτη(μαθηματικού μοντέλου), ικανού να παράγει ψευδοτυχαίες χαοτικές χρονοσειρές.
- Ανάλυση ταχύτητας, πολυπλοκότητας, ασφάλειας, αλλά και έκτασης κλειδιού.
- Δημιουργία γεννήτριας ψευδοτυχαίων αριθμών(PRNG) με την χρήση του χαοτικού χάρτη.
- Έλεγχος τυχαιότητας γεννήτριας με χρήση test-suites όπως το πρωτόκολλο NIST-802.22.
- Υλοποίηση της γεννήτριας με χρήση υλικολογισμικού για ενσωματωμένα συστήματα της εταιρίας ST.
- Υλοποίηση της γεννήτριας σε αναπτυξιακό FPGA με σκοπό την βελτιστοποίηση της ταχύτητάς της, αλλά και την μελέτη ενδεχομένου της σχεδίασής της σε ASIC(application specific integrated circuit), με σκοπό την χρήση του για εφαρμογές κρυπτογράφησης ευαίσθητων δεδομένων σε πραγματικό χρόνο.
- Σύγκριση των πειραματικών αποτελεσμάτων που εξήχθησαν από τα αναπτυξιακά fpga/stm32 για την ανάδειξη του βέλτιστου δυνατού συστήματος για την χρήση γεννητριών τυχαίων αριθμών με χρήση χαοτικών συστημάτων.
- Σύγκριση των πειραματικών αποτελεσμάτων που εξήχθησαν από τα αναπτυξιακά fpga/stm32 για την ανάδειξη των αδυναμιών του κάθε συστήματος σε γνωστές επιθέσεις αποκρυπτογράφησης.
- Σύγκριση αποτελεσμάτων με state-of-art βιβλιογραφία στο αντικείμενο της κρυπτογράφησης με χαοτικούς χάρτες.
- Πρόταση βελτιστοποιήσεων μαθηματικών μοντέλων και υλοποίησης της ψευδοτυχαίας γεννήτριας.

Έχοντας υπ' όψιν αυτούς τους σκοπούς η διπλωματική ΔΕΝ:

- Πραγματεύεται την δημιουργία της πιο αποδοτικής γεννήτριας τυχαίων αριθμών.
- Δημιουργεί την πιο ασφαλή γεννήτρια ψευδοτυχαίων αριθμών.
- Σχεδιάζει το πιο αποδοτικό μοντέλο υλοποίησης σε γλώσσα VHDL.

Η διπλωματική πραγματεύεται επί της ουσίας την πρόταση ενός μοντέλου παραγωγής τυχαίων αριθμών και την σύγκριση του με άλλα της βιβλιογραφίας για εξαγωγή χρήσιμων συμπερασμάτων για τα μοντέλα κρυπτογράφησης και το μέλλον των χαοτικών συστημάτων στην αγορά αλλά και στην έρευνα γενικότερα. Με αυτό το κριτήριο γίνεται και η σύγκριση των πειραματικών αποτελεσμάτων σε σχέση με αυτά της βιβλιογραφίας που ενδέχεται να χρησιμοποιούν έτοιμα συστήματα για πράξεις όπως DSP(digital signal

processing) μονάδες. Στο πρώτο κεφάλαιο γίνεται μια εκτενής αναφορά στα χαοτικά συστήματα. Πότε γεννήθηκαν, ο σκοπός χρήσης στις φυσικές επιστήμες, τα πλεονεκτήματα και τα μειονεκτήματα τους στην παραγωγή τυχαίων αριθμών, κ.α.

2 Εισαγωγή στα μη-γραμμικά χαοτικά συστήματα

Αφορμή για την μελέτη μη γραμμικών χαοτικών συστημάτων αποτέλεσε ένα δημοσίευμα στην περιοδικό Journal of Atmospheric Sciences του Edward N. Lorenz το όνομα του οποίου οικειοποιήθηκε για έναν από τους πιο γνωστούς ελκυστές, των ελκυστή Lorenz. Το δημοσίευμα με τίτλο Deterministic Non Periodical Flow[21] έφερε επανάσταση στα επιστημονικά δεδομένα καθώς ο Lorenz παρουσίαζε ένα σύστημα που υπό συγκεκριμένες συνθήκες παρουσίαζε περιοδική συμπεριφορά, ενώ επιβάλλοντας μια μικρή αλλαγή στις αρχικές συνθήκες του συστήματος, συμπεριφερόταν χαοτικά, χωρίς δηλαδή να επαναλαμβάνει τις σε βάθος χρόνου προηγούμενες καταστάσεις. Μάλιστα ο Lorenz δεν άργησε να μελετήσει και να εξακριβώσει ότι το σύστημα αυτό ήταν ταυτόχρονα και αιτιοκρατικό. Το δημοσίευμα αυτό έμελλε να αποτελέσει τον ακρογωνιαίο λίθο της έρευνας πάνω σε αυτά τα συστήματα που ονομάζονται χαοτικά και να φέρει επανάσταση στους κλάδους που βρίσκουν εφαρμογή. Το σύστημα που μελέτησε ο Lorenz είναι αυτό της εξίσωσης 2.1 .

$$\begin{aligned} \frac{dx}{dt} &= -\sigma x + \sigma y \\ \frac{dy}{dt} &= rx - y - xz \\ \frac{dz}{dt} &= xy - bz \end{aligned} \tag{2.1}$$

Χαοτικά Συστήματα**:** Με τον όρο χαοτικά συστήματα αναφερόμαστε σε συστήματα τόσο ευάλωτα στις αρχικές τους συνθήκες, που μια μικρή αλλαγή τους, αλλάζει τελείως την συμπεριφορά του συστήματος.

Ένα ντετερμινιστικό σύστημα λέγεται χαοτικό, όταν η εξέλιξή του εξαρτάται ευαίσθητα από τις αρχικές του συνθήκες. Αυτή η ιδιότητα υποδηλώνει ότι δύο τροχιές που προκύπτουν από δύο διαφορετικές γειτονικές αρχικές συνθήκες χωρίζουν εκθετικά με την πάροδο του χρόνου[1]. Οι αναγκαίες προϋποθέσεις για να είναι ένα ντετερμινιστικό σύστημα χαοτικό, είναι ότι το σύστημα πρέπει να είναι μη γραμμικό.

Το γεγονός ότι κάποια δυναμικά μοντέλα που δείχνουν τις παραπάνω απαραίτητες προϋποθέσεις διαθέτουν τέτοια κρίσιμη εξάρτηση από τις αρχικές συνθήκες ήταν γνωστό από το τέλος του περασμένου αιώνα. Ωστόσο, μόνο τα τελευταία τριάντα χρόνια, οι πειραματικές παρατηρήσεις έχουν δείξει ότι, στην πραγματικότητα, τα χαοτικά συστήματα είναι κοινά στη φύση. Μπορούν να βρεθούν, για παράδειγμα, στη Χημεία (αντίδραση Belousov-Zhabotinsky), στη Μη Γραμμική Οπτική (λέιζερ), στα Ηλεκτρονικά (κύκλωμα Chua-Matsumoto), στη Δυναμική Ρευστών (συναγωγή Rayleigh-Bénard), κλπ. Πολλά φυσικά φαινόμενα μπορούν επίσης να χαρακτηριστούν ως χαοτικά. Μπορούν να βρεθούν στη μετεωρολογία, στο ηλιακό σύστημα, στην καρδιά και τον εγκέφαλο των ζωντανών οργανισμών κ.ο.κ.

Λόγω της κρίσιμης εξάρτησής τους από τις αρχικές συνθήκες και λόγω του γεγονότος ότι, γενικά, οι

πειραματικές αρχικές συνθήκες δεν είναι ποτέ γνωστές με ακρίβεια, αυτά τα συστήματα είναι εκ φύσεως απρόβλεπτα. Πράγματι, η προβλεπόμενη τροχιά που προκύπτει από μια καλόπιστη αρχική συνθήκη και η πραγματική τροχιά που προκύπτει από την πραγματική αρχική συνθήκη αποκλίνουν εκθετικά με την πάροδο του χρόνου, έτσι ώστε το σφάλμα στην πρόβλεψη (η απόσταση μεταξύ της προβλεπόμενης και της πραγματικής τροχιάς) να αυξάνεται εκθετικά με την πάροδο του χρόνου, μέχρι να καταστήσει την πραγματική τροχιά του συστήματος εντελώς διαφορετική από την προβλεπόμενη σε μεγάλα χρονικά διαστήματα.

2.1 Ιδιότητες/Ορολογία δυναμικών συστημάτων

Για να γίνει κατανοητό το πεδίο έρευνας της εργασίας απαιτείται μια αναλυτική περιγραφή της ορολογίας που είναι ευρέως γνωστή για να περιγράφει ιδιότητες μη γραμμικών συστημάτων. Στο υποκεφάλαιο αυτό θα γίνει μια αναλυτική περιγραφή αλλά και παρουσίαση της ορολογίας που χρησιμοποιείται στην βιβλιογραφία αλλά και των βασικών ιδιοτήτων που διέπουν τα δυναμικά-χαοτικά συστήματα.

ΤΡΟΧΙΑ: Με τον όρο τροχιά ενός δυναμικού συστήματος εννοούμε όλα τα σημεία από τα οποία θα περάσει το μοντέλο μας κατά την εξέλιξή του, και μας δίνει πολύ χρήσιμες πληροφορίες για το σύστημα αυτό. Όταν αναφερόμαστε σε τροχιά βρισκόμαστε πάντα στον χώρο καταστάσεων(state space).

ΧΩΡΟΣ ΚΑΤΑΣΤΑΣΕΩΝ: Με τον όρο χώρος καταστάσεων εννοούμε το γεωμετρικό χώρο όπου οι άξονες είναι ταυτόχρονα και οι μεταβλητές κατάστασης. Μια στιγμιαία κατάσταση θεωρείται ότι χαρακτηρίζεται από τις στιγμιαίες τιμές των μεταβλητών που θεωρούνται κρίσιμες για μια πλήρη περιγραφή της κατάστασης. Ένα πλεονέκτημα της ανάλυσης στον χώρο καταστάσεων είναι ότι συχνά μας επιτρέπει να μελετήσουμε χρήσιμες γεωμετρικές ιδιότητες των τροχιών του συστήματος, χωρίς να γνωρίζουμε τις ακριβείς λύσεις των δυναμικών εξισώσεων του.[16].

ΕΛΚΥΣΤΕΣ: Ελκυστές [33](Attractors) ονομάζονται τα αναλλοίωτα υποσύνολα του χώρου των φάσεων που έχουν την ιδιότητα να έλκουν και να εγκλωβίζουν τις τροχιές σε αυτά. Το είδος του ελκυστή χαρακτηρίζεται από τον τρόπο με τον οποίο μεταβάλλονται - εξελίσσονται στο χρόνο οι δυναμικές μεταβλητές του συστήματος. Οι ελκυστές έχουν ιδιαίτερη σημασία στη μελέτη των δυναμικών συστημάτων, τόσο από θεωρητική όσο και από εφαρμοσμένη άποψη. Τα είδη των ελκυστών που γνωρίζουμε μέχρι σήμερα, είναι οι κανονικοί ελκυστές, οι χαοτικοί ή αλλιώς, παράξενοι ελκυστές. Επίσης, οι ελκυστές χωρίζονται σε αυτο-διεγερμένους (self-excited) και κρυφούς ελκυστές(hidden attractors).

ΣΗΜΕΙΑ ΙΣΟΡΡΟΠΙΑΣ: Πρόκειται για τα σημεία τα οποία αποτελούν λύσεις των διαφορικών εξισώσεων που περιγράφουν το σύστημα, για τις οποίες το σύστημα παραμένει σταθερό. Τα σημεία ισορροπίας αποτελούν διακριτά σημεία στον χώρο φάσεων, που θα επεξηγηθεί αργότερα, και η ευστάθεια τους παίζει καθοριστικό ρόλο στην τοπολογία του χώρου φάσεων.

ΕΥΣΤΑΘΙΑ(STABILITY): Ευσταθές σημείο ισορροπίας (stable equilibrium point) ονομάζεται ένα σημείο ισορροπίας, όταν η κίνηση που αντιστοιχεί σε αρχικές συνθήκες που βρίσκονται πολύ κοντά στο σημείο ισορροπίας είναι περατωμένη[33].

ΧΩΡΟΣ ΦΑΣΕΩΝ: Σε ένα αυτόνομο δυναμικό σύστημα, ο χώρος που διαμορφώνεται από τις δυναμικές μεταβλητές είναι ένας δισδιάστατος χώρος και ονομάζεται χώρος των φάσεων (phase space).

Οι παραπάνω ορισμοί θα χρησιμοποιηθούν αρκετές φορές στο πλαίσιο μελέτης της εργασίας, για να δώσουν στοιχεία για την κατάσταση του χαοτικού αλγορίθμου που μελετήθηκε, αναλύθηκε και υλοποιήθηκε στο πλαίσιο της εργασίας.

2.2 Ιδιότητες δυναμικών συστημάτων

Το επόμενο βήμα για την κατανόηση ενός μη-γραμμικού συστήματος και την αναγνώριση χαοτικής συμπεριφοράς είναι να γίνει μια αναλυτική αναφορά σε εργαλεία που χρησιμοποιήθηκαν στα πλαίσια της εργασίας για την ανάλυση μη γραμμικών συστημάτων.

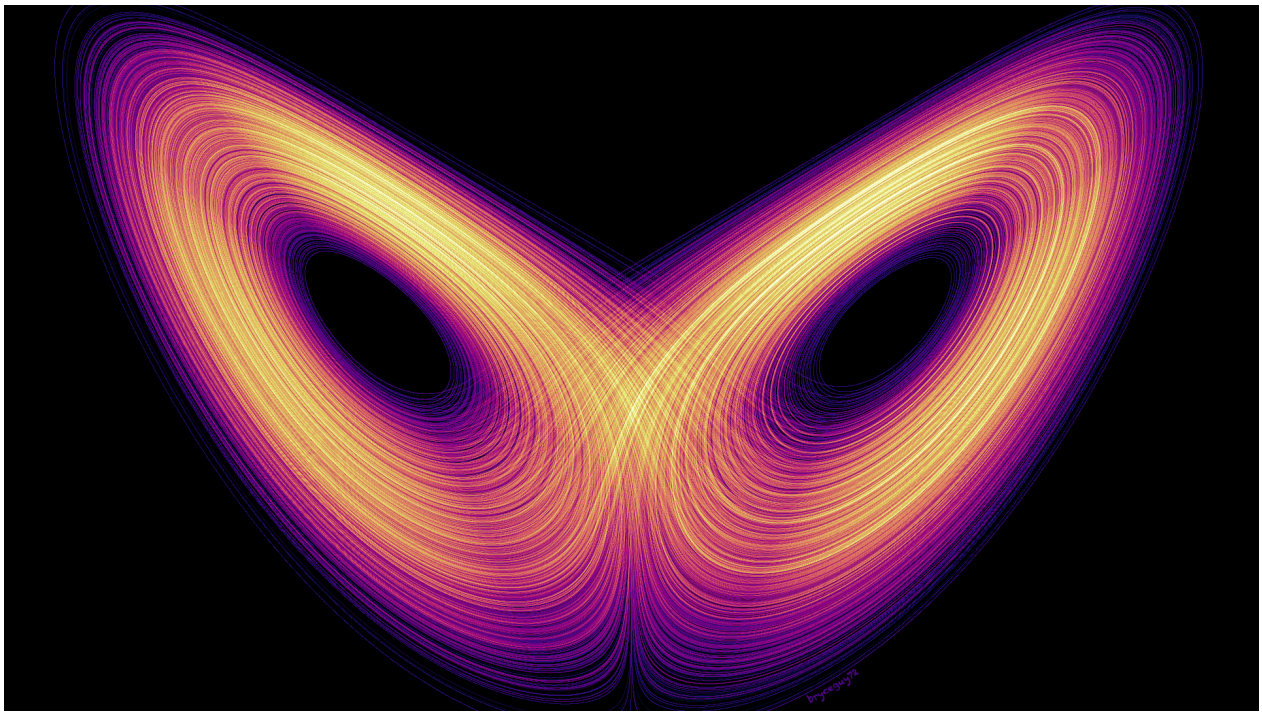
Διακλάδωση Δυναμικών Συστημάτων: Κάθε δυναμικό σύστημα καθορίζεται από τις δυναμικές του μεταβλητές και τις παραμέτρους του. Οι αλλαγές στις παραμέτρους του συστήματος οδηγούν σε αλλαγές και στη συμπεριφορά των δυναμικών μεταβλητών και άρα, στη δυναμική του συστήματος [33] Τέτοια παραδείγματα μπορεί να είναι η αλλαγή στο πλήθος των σημείων ισορροπίας, στην ευστάθεια των σημείων ισορροπίας ή στο είδος της τροχιάς (περιοδική, ημιπεριοδική ή χαοτική). Η τιμή της παραμέτρου του συστήματος, στην οποία συμβαίνει η αλλαγή της συμπεριφοράς του συστήματος, ονομάζεται σημείο διακλάδωσης (bifurcation).Τα σημεία διακλάδωσης συνδέονται άμεσα με τη δομική ευστάθεια του συστήματος. Ένα σύστημα ονομάζεται δομικά ευσταθές όταν μικρές αλλαγές στις παραμέτρους του δεν αλλάζουν τα ποιοτικά του χαρακτηριστικά. Έτσι, τα σημεία διακλάδωσης, στα οποία οι τιμές των παραμέτρων αλλάζουν τα ποιοτικά χαρακτηριστικά του συστήματος, είναι σημεία που μεταβάλλουν τη δομική του ευστάθεια (από ευσταθές, όπου για τις τιμές των παραμέτρων έχουμε την ίδια δυναμική συμπεριφορά, μετατρέπεται σε ασταθές και αλλάζει η δυναμική του συμπεριφορά).

Αν και υπάρχουν διάφοροι τύποι διακλαδώσεων σε δυναμικά συστήματα, στην συνέχεια θα αναλυθεί κυρίως το διάγραμμα διακλάδωσης, καθότι ήταν ένα από τα πρώτα εργαλεία που χρησιμοποιήθηκαν, για την ανάλυση του χαοτικού συστήματος, που αναπτύχθηκε για τους σκοπούς της εργασίας.

Διάγραμμα διακλάδωσης: Πρόκειται για μια απεικόνιση της συμπεριφοράς μιας δυναμικής μεταβλητής, που προκύπτει μέσα από μια διαδικασία δειγματοληψίας, ως προς τις μεταβολές της τιμής μιας παραμέτρου του συστήματος.

Χρονοσειρά: Η χρονοσειρά σε δυναμικά συστήματα αποτελεί την εξάρτηση της εξόδου σε σχέση με το χρόνο. Η απεικόνιση μιας χρονοσειράς δίνει πολύ χρήσιμες πληροφορίες για την περιοδικότητα μιας δυναμικής μεταβλητής ή για την χαοτική της συμπεριφοράς.

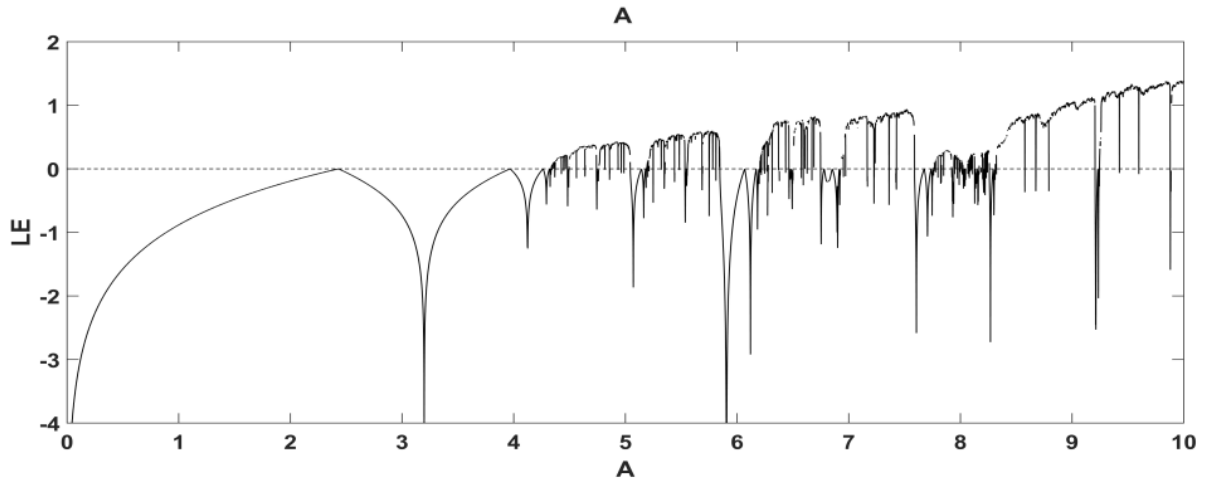
Χάος: Η ανάλυση της δυνητικά χαοτικής συμπεριφοράς σε φυσικά-μηχανικά φαινόμενα έχει προσελκύσει μεγάλο ενδιαφέρον τα τελευταία χρόνια. Αν και δεν υπάρχει καθολικά αποδεκτός μαθηματικός ορισμός του όρου χάος, ο Strogatz[28] παρέχει έναν λειτουργικό ορισμό ως «απεριοδική μακροπρόθεσμη συμπεριφορά σε ένα ντετερμινιστικό σύστημα που επιδεικνύει ευαίσθητη εξάρτηση από τις αρχικές συνθήκες.» Η απεριοδική μακροπρόθεσμη συμπεριφορά σημαίνει ότι οι τροχιές δεν συγκλίνουν σε σταθερά σημεία, περιοδικές τροχιές ή σχεδόν περιοδικές τροχιές καθώς t - N, αλλά αντίθετα επιδεικνύουν ακανόνιστη και απρόβλεπτη συμπεριφορά. Ο όρος «ντετερμινιστικό» σημαίνει ότι αυτή η απρόβλεπτη, απεριοδική συμπεριφορά προκύπτει από τις εγγενείς μη γραμμικότητες στο ίδιο το σύστημα (που μπορούν να εκφραστούν με ντετερμινιστικές εξισώσεις κίνησης), και δεν οφείλεται σε θόρυβο ή άλλα στοχαστικά στοιχεία στο σύστημα. Η «ευαίσθητη εξάρτηση από τις αρχικές συνθήκες» (SDIC) σημαίνει ότι οι τροχιές που ξεκινούν αυθαίρετα κοντά η μία στην άλλη θα απομακρύνονται εκθετικά γρήγορα.



Εικ. 2.1: Ελκυστής Lorentz

Εκθέτες Lyapunov: Το δεύτερο πιο κρίσιμο εργαλείο που χρησιμοποιήθηκε στην εργασία για το σκοπό της μελέτης του χαοτικού συστήματος που σχεδιάστηκε για κρυπτογράφηση είναι οι εκθέτες Lyapunov. Μιας και το σύστημα κρυπτογράφησης που χρησιμοποιήθηκε είναι διακριτό, η θεωρία των εκθετών Lyapunov θα αναλυθεί μόνο για διακριτά συστήματα.

Οι εκθέτες Lyapunov παρέχουν ένα άμεσο μέτρο της **SDIC**, ποσοτικοποιώντας τους εκθετικούς ρυθμούς, με τους οποίους οι γειτονικές τροχιές σε έναν ελκυστή απομακρύνονται (συγκλίνουν), καθώς το σύστημα εξελίσσεται στον χρόνο. Ένα σύστημα d διαστάσεων (δηλαδή, ένα ορισμένο σύστημα από d διαφορικές εξισώσεις πρώτης τάξης), θα έχει d εκθέτες Lyapunov, ο καθένας από τους οποίους αντιπροσωπεύει τον ρυθμό αύξησης ή μείωσης των μικρών διαταραχών κατά μήκος κάθε ενός από τους κύριους άξονες στον χώρο κατάστασης του συστήματος. Αυτοί οι εκθέτες συνήθως ταξινομούνται από τον μεγαλύτερο προς τον μικρότερο[8]. Τα ευθύγραμμα τμήματα στον χώρο κατάστασης αυξάνονται (ή μειώνονται) ως e^{tl1}, οι περιοχές αυξάνονται ως $e^{t(l1+l2)}$, οι όγκοι ως $e^{t(l1+l2+l3)}$, και ούτω καθεξής. Τα συνεχή δεδομένα που προέρχονται από δειγματοληψία (δηλαδή, που μπορούν να περιγραφούν από ένα σύνολο διαφορικών εξισώσεων), εμφανίζουν τουλάχιστον έναν μηδενικό εκθέτη, επειδή οι διαταραχές κατά μήκος της κύριας κατεύθυνσης της δειγματοληψία δεν θα προκαλέσουν απόκλιση από τη δειγματοληψία (8–10). Εάν το σύστημα εμφανίζει τουλάχιστον έναν θετικό εκθέτη Lyapunov και είναι καθαρά ντετερμινιστικό, τότε είναι «χαοτικό».



Εικ. 2.2: Διάγραμμα εκθετών lyapunov[14]

Στην περίπτωση ενός διακριτού συστήματος εξαρτημένου από τον χρόνο οι συντελεστές Lyapunov μπορούν να βρεθούν με την χρήση της εξίσωσης 2.2:

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \ln \left| \frac{df}{dx}(x_i) \right| \tag{2.2}$$

Το μέγεθος του μεγαλύτερου εκθέτη Lyapunov (λ_1) καθορίζει τον μέγιστο μέσο εκθετικό ρυθμό απόκλισης των τροχιών σε έναν ελκυστή και συνεπώς, τη μέγιστη ποσότητα αστάθειας κατά οποιαδήποτε κατεύθυνση. Επομένως, χρησιμοποιείται συχνά και ως μέτρο της τοπικής αστάθειας ενός δεδομένου συστήματος.

Στην περίπτωση ενός μονοδιάστατου $1D$ δυναμικού συστήματος οι περιπτώσεις βάση εκθέτη Lyapunov

είναι οι εξής:

- $-$: Αρνητικός συντελεστής lyapunov, υποδηλώνει ότι οι τροχιές που ξεκινούν από κοντινά αρχικά σημεία συγκλίνουν εκθετικά με την πάροδο του χρόνου προς ένα σταθερό σημείο ή ένα σταθερό κύκλο.
- $+$: Υποδηλώνει ότι οι τροχιές που ξεκινούν από κοντινά αρχικά σημεία απομακρύνονται εκθετικά με την πάροδο του χρόνου. Αυτό είναι ένα χαρακτηριστικό της χαοτικής συμπεριφοράς.
- $\lambda = 0$: Υποδηλώνει ότι οι τροχιές διατηρούν την απόσταση τους στο χρόνο. Αυτό μπορεί να είναι ενδεικτικό περιοδικών τροχιών ή τροχιών που καταλήγουν σε σταθερά σημεία.

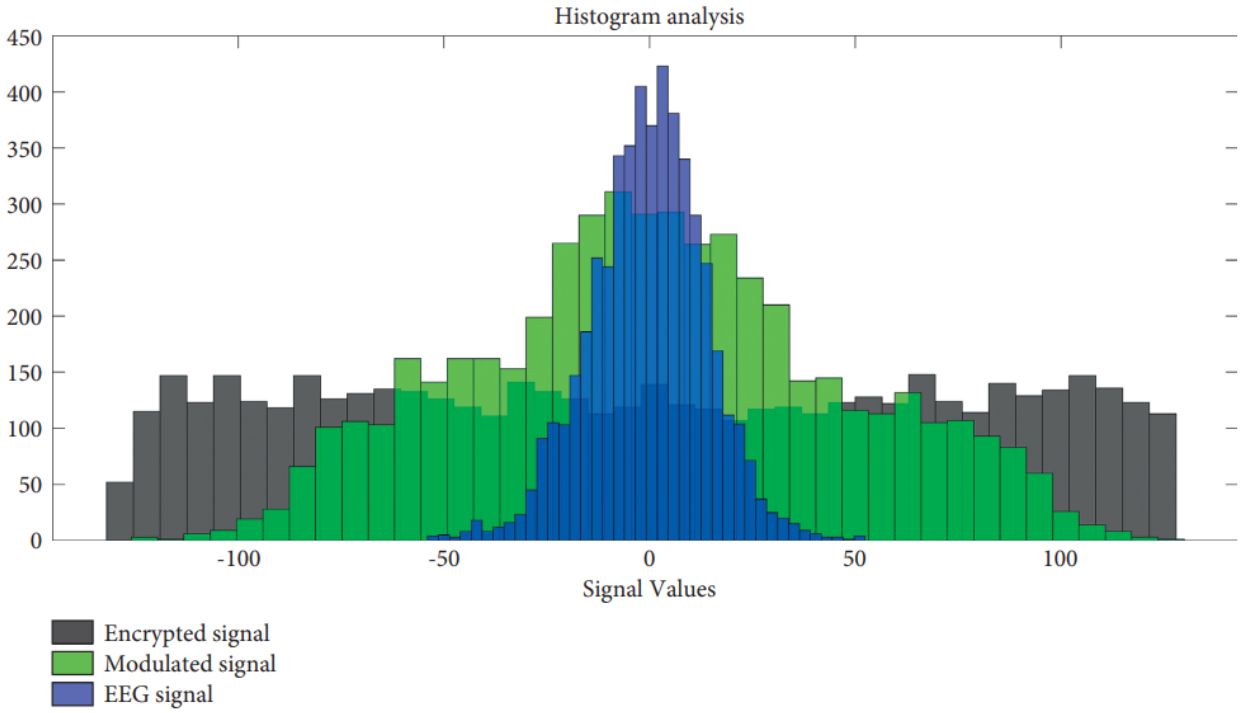
2.3 Στατιστικά εργαλεία μελέτης χαοτικών συστημάτων

Έχοντας αναλύσει όλες τις ιδιότητες καθώς και τις απαραίτητες ορολογίες που έπρεπε να έχουμε υπόψιν για την μελέτη δυναμικών-χαοτικών συστημάτων, σε αυτό το κεφάλαιο θα κάνουμε μια ανασκόπηση στα στατιστικά εργαλεία που χρησιμοποιήθηκαν για την μελέτη του συστήματος κρυπτογράφησης της διπλωματικής.

Συνοπτικά τα εργαλεία που χρησιμοποιήθηκαν για την στατιστική μελέτη της παραγόμενης γεννήτριας τυχαίων αριθμών είναι:

- Ιστόγραμμα
- Εντροπία
- Εκτίμηση εντροπίας
- Δείκτης Δομικής Ομοιότητας (SSIM)
- Σηματοθορυβική σχέση (SNR)
- LLR
- Ο συντελεστής συσχέτισης
- Η φασματική παραμόρφωση

Ιστόγραμμα: Ίσως ένα από τα πιο απαραίτητα εργαλεία κατά την διάρκεια μελέτης της τυχαιότητας ενός χάρτη (μαθηματικού μοντέλου), καθότι το ιστόγραμμα δύναται να μελετήσει την κατανομή των τιμών μιας συνάρτησης. Σε μια τυχαία χρονοσειρά με έξοδο $y \in [y_{min}, y_{max}]$ και διαστήματα στο ιστόγραμμα $d_i, i = 1, 2, ..., n$, το ιδανικό σενάριο τυχαιότητας είναι $\forall d_i$ του ιστογράμματος ο αριθμός $s_{y_{d_i}}$ που απαντάται μια τιμή y_{d_i} να είναι ίσος $\forall y_{d_i} \in [y_{min}, y_{max}]$. Συνοπτικά θέλουμε ιδανικά το άθροισμα των φορών που απαντάται μια τιμή της χρονοσειράς, να ισούται με όλα τα υπόλοιπα αθροίσματα των φορών που απαντώνται όλες οι υπόλοιπες τιμές.



Εικ. 2.3: Ιστόγραμμα μεταξύ τριών διαφορετικών σημάτων. Του τροποποιημένου σήματος, του κρυπτογραφημένου, καθώς και του σήματος ηλεκτροεγκεφαλογραφήματος[12].

Εντροπία: Έστω μια τυχαία μεταβλητή X, η οποία παίρνει την τιμή x_i με πιθανότητα p_i, για $1 \leq i \leq n$. Τότε, η ποσότητα πληροφορίας $I(p_i)$ είναι:

$$I(p_i) = \log \frac{1}{p_i}$$

που είναι η ποσότητα πληροφορίας που κωδικοποιείται στο x_i (ή στο p_i), ενώ η μέση ποσότητα πληροφορίας:

$$\sum_{i=1}^{n} p_i \log \frac{1}{p_i},$$

ονομάζεται η εντροπία του Shannon της τυχαίας μεταβλητής X (ή της κατανομής P), και συμβολίζεται με $H(X)$[3]:

$$H(X) = -\sum_{i=1}^{n} p_i \log p_i$$

Με την χρήση της εντροπίας πληροφορίας έγινε η ποσοτικοποίηση της πληροφορίας που παράγεται από την γεννήτρια τυχαίων αριθμών που σχεδιάστηκε για τον σκοπό της πτυχιακής.

Εκτίμηση εντροπίας: Ο στόχος της εκτίμησης εντροπίας (ApEn) και της εντροπίας δείγματος (SampEn) είναι να εκτιμήσουν την τυχαιότητα μιας σειράς δεδομένων, χωρίς καμία προηγούμενη γνώση για την πηγή που παράγει το σύνολο δεδομένων. Επομένως, η εφαρμοσιμότητά τους είναι απεριόριστη και αυτοί οι αλγόριθμοι έχουν χρησιμοποιηθεί σε ένα ευρύ φάσμα ερευνητικών πεδίων[7].

Το ζήτημα της αξιολόγησης της τυχαιότητας μιας χρονοσειράς δεδομένων είναι θέμα χαρακτηρισμού του συστήματος ως στοχαστικό ή ντετερμινιστικό. Πιο συγκεκριμένα, σε ποιο βαθμό τα δεδομένα συμπεριφέρονται ως στοχαστικά και πόσος ντετερμινισμός υπάρχει. Θα μιλάμε για βαθμούς τυχαιότητας. Εάν δεν υπάρχει κανένας υποκείμενος ντετερμινισμός κατά την ανάλυση μιας συγκεκριμένης σειράς δεδομένων, αυτό υπονοεί μια επιβεβαίωση της πιο αυστηρής τυχαίας κίνησης. Ωστόσο, αν υπάρχουν κύκλοι, τάσεις και μοτίβα, τότε η σειρά δεδομένων δεν θα είναι εντελώς τυχαία[7].

Η κατά προσέγγιση εντροπία $\text{ApEn}(m, r, N)(u)$ μετρά τη λογαριθμική συχνότητα με την οποία τα μπλοκ μήκους m που είναι κοντά το ένα στο άλλο παραμένουν μαζί για την επόμενη θέση, ή αλλιώς, η αρνητική τιμή της ApEn ορίζεται ως:

$$\text{ApEn}(m, r, N)(u) = \phi_{m+1}(r) - \phi_m(r)$$

μέσο όρο για το i του λογαρίθμου (συνθήκη πιθανότητας του $|u(j+m)-u(i+m)| \leq r$, εάν επιβεβαιώνεται ότι $|u(j+k) - u(i+k)| \leq r$ για $k = 0, 1, \ldots, m-1$).

Η $\text{ApEn}(m, r, N)$ είναι ο στατιστικός εκτιμητής της παραμέτρου $\text{ApEn}(m, r)$:

$$\text{ApEn}(m, r) = \lim_{N\to\infty} [\phi_m(r) - \phi_{m+1}(r)].$$

Ο Δείκτης Δομικής Ομοιότητας (SSIM) είναι ένα μέτρο της δομικής ομοιότητας μεταξύ δύο σημάτων, που αρχικά θεωρήθηκε για εικόνες. Δίνεται από τον τύπο:

$$\text{SSIM} = \frac{(2\mu_x\mu_y + S_1)(2\delta_{xy} + S_2)}{(\mu_x^2 + \mu_y^2 + S_1)(\delta_x^2 + \delta_y^2 + S_2)}$$

όπου μ_x, μ_y είναι οι μέσες τιμές των αρχικών και κρυπτογραφημένων (ή διαμορφωμένων) σημάτων αντίστοιχα, δ_x^2, δ_y^2 οι διακυμάνσεις τους, και δ_{xy} η διασταυρωμένη συνδιακύμανση τους. Οι παράμετροι S_1, S_2 λαμβάνουν μικρές τιμές, για να αποφεύγονται ασταθή αποτελέσματα όταν ο παρονομαστής είναι κοντά στο μηδέν.

Η εξίσωση για το δείγμα σήματος x_i δίνεται από:

$$x_i = \sum_{m=1}^{p} a_m x_{i-m} + G_x u_i, \tag{2.3}$$

όπου x_i είναι το i-οστό δείγμα σήματος, a_m, $m = 1, \ldots, p$ οι συντελεστές του φίλτρου all-pole, G_x το κέρδος του φίλτρου και u_i μια κατάλληλη είσοδος διέγερσης στο σήμα. Ο Δείκτης Λογαριθμικής Σχέσης

(LLR) ορίζεται ως:

$$\text{LLR} = \log\left(\frac{a_x R_y a_x^T}{a_y R_z a_y^T}\right), \tag{2.4}$$

όπου a_x το διάνυσμα των Συντελεστών Γραμμικής Πρόβλεψης (LPCs) $[1, a_1, a_2, \ldots, a_m]$ του αρχικού σήματος, a_z οι LPCs του κρυπτογραφημένου (ή διαμορφωμένου) σήματος, και R_y η μήτρα αυτοσυσχέτισης του κρυπτογραφημένου (ή διαμορφωμένου) σήματος. Μια υψηλότερη τιμή LLR υποδεικνύει καλή κρυπτογράφηση.

Το SNR ορίζεται ως [12]:

$$\text{SNR} = 10\log_{10}\left(\frac{\sum_{i=1}^{N} x_i^2}{\sum_{i=1}^{N}(x_i - y_i)^2}\right), \tag{2.5}$$

όπου x, y είναι τα κρυπτογραφημένα και αποκρυπτογραφημένα (ή διαμορφωμένα) σήματα αντίστοιχα, και N ο αριθμός των δειγμάτων. Ένα χαμηλό SNR υποδεικνύει καλή κρυπτογράφηση.

Ο συντελεστής συσχέτισης [12] μεταξύ του αρχικού και του κρυπτογραφημένου (ή διαμορφωμένου) σήματος υπολογίζεται ως:

$$r_{xy} = \frac{cv(x,y)}{\sqrt{\delta_x^2 \cdot \delta_y^2}}, \tag{2.6}$$

όπου $cv(x,y)$ η συνδιακύμανση των δύο σημάτων και δ^2 οι διακυμάνσεις τους. Για μη συσχετισμένα σήματα, ο συντελεστής συσχέτισης θα πρέπει να είναι κοντά στο μηδέν.

Η φασματική παραμόρφωση (SD) μετρά τη διαφορά μεταξύ του φάσματος του αρχικού και του κρυπτογραφημένου (ή διαμορφωμένου) σήματος [12]. Υπολογίζεται ως:

$$\text{SD} = \frac{1}{M}\sum_{i=0}^{M-1} |V_{x,i} - V_{y,i}|, \tag{2.7}$$

όπου $V_{x,i}$, $V_{y,i}$ το φάσμα του αρχικού και του κρυπτογραφημένου (ή διαμορφωμένου) σήματος σε dB στη χρονική στιγμή i. Μια υψηλότερη τιμή SD υποδεικνύει καλύτερη κρυπτογράφηση.

Με την χρήση όλων των παραπάνω εργαλείων σε συνδυασμό με την χρήση των τεστ ελέγχου τυχαιότητας, μια γεννήτρια μπορεί να χαρακτηριστεί με μεγάλη ασφάλεια για τον βαθμό τυχαιότητας της σε επίπεδο παραγωγής bit ή κλειδιών. Όπως επίσης μπορεί να συγκριθεί με αντίστοιχες γεννήτριας παραγωγής τυχαίων αριθμών για εξαγωγή συμπερασμάτων όπως η πολυπλοκότητα, η ταχύτητα, η ασφάλεια κ.α

2.4 Διαχωρισμός **RNG's**

Σε αυτό το υποκεφάλαιο κρίνεται απαραίτητο να αναφέρουμε τις δομικές διαφορές μεταξύ πραγματικά τυχαίων και ψευδοτυχαίων γεννητριών τυχαίων αριθμών καθότι, υπάρχει μια σαφής επίδραση στον τρόπο

λειτουργίας τους.

Ψευδοτυχαίοι Αριθμοί (PRNGs) Οι Ψευδοτυχαίοι Αριθμοί δημιουργούνται μέσω αλγορίθμων και, παρότι φαίνονται τυχαίοι, είναι καθορισμένοι από μια αρχική τιμή. Οι PRNGs είναι ντετερμινιστικοί και, αν η αρχική τιμή(seed) είναι γνωστή, η αλληλουχία των παραγόμενων αριθμών μπορεί να αναπαραχθεί. Το χάος που χρησιμοποιούμε σαν ιδιότητα παραγωγής τυχαίων αριθμών, αποτελεί εφαρμογή ψευδοτυχαίας παραγωγής αριθμών, καθώς όπως προείπαμε έχει μεγάλη ευαισθησία στις αρχικές συνθήκες. Παρ' όλα αυτά με την χρήση μοναδικών αρχικών συνθηκών μπορεί εύκολα να αλλάξει το πεδίο εφαρμογής του σε πραγματικά τυχαίους αριθμούς.

Πλεονεκτήματα των PRNGs:

- Ταχύτητα: Οι PRNGs είναι συνήθως γρήγοροι και αποδοτικοί, καθώς είναι βασισμένοι σε μαθηματικούς αλγορίθμους.
- Αναπαραγωγικότητα: Χρήσιμοι για εφαρμογές όπου απαιτείται η επανάληψη των αποτελεσμάτων, όπως σε δοκιμές και προσομοιώσεις.
- Προβλεψιμότητα: Παρέχουν προβλεψιμότητα όταν η είσοδος(seed) είναι γνωστή, κάτι που μπορεί να είναι χρήσιμο για ορισμένες εφαρμογές.

Μειονεκτήματα των PRNGs:

- Μη Πραγματική Τυχαιότητα: Δεν παράγουν πραγματικά τυχαίους αριθμούς, κάτι που μπορεί να είναι ανεπιθύμητο σε εφαρμογές κρυπτογράφησης.
- Ευπάθεια: Εάν ο αλγόριθμος και ο seed είναι γνωστοί, οι ακολουθίες μπορούν να προβλεφθούν, εκθέτοντας πιθανά κενά ασφαλείας.

Ψευδοτυχαία Γεννήτρια Αριθμών **(PRNGs)**	
Πλεονεκτήματα	Μειονεκτήματα
- Ταχύτητα: Είναι συνήθως γρήγοροι και αποδοτικοί, βασισμένοι σε μαθηματικούς αλγορίθμους.	- Μη Αληθινή Τυχαιότητα: Δεν παράγουν πραγματικά τυχαίους αριθμούς, κάτι που μπορεί να είναι ανεπιθύμητο σε κρυπτογραφικές εφαρμογές.
- Αναπαραγωγικότητα: Χρήσιμοι για εφαρμογές όπου απαιτείται η επανάληψη των αποτελεσμάτων, όπως σε δοκιμές και προσομοιώσεις.	- Ευπάθεια: Εάν ο αλγόριθμος και ο seed είναι γνωστοί, οι ακολουθίες μπορούν να προβλεφθούν, εκθέτοντας πιθανά κενά ασφαλείας.
- Προβλεψιμότητα: Παρέχουν προβλεψιμότητα όταν ο seed είναι γνωστός, κάτι που μπορεί να είναι χρήσιμο για ορισμένες εφαρμογές.	
- Ταχύτητα: Ανάλογα πάντα το σύστημα το throughput ενώw ψευδοτυχαίου αλγόριθμου επιτυγχάνει πολύ μεγάλο throughput.	
Πραγματικά Τυχαία Γεννήτρια Αριθμών **(TRNGs)**	
Πλεονεκτήματα	Μειονεκτήματα
- Αληθινή Τυχαιότητα: Παρέχουν αληθινά τυχαίους αριθμούς που είναι ανεξάρτητοι από προηγούμενους αριθμούς ή αρχικές τιμές.	- Ταχύτητα: Μπορεί να είναι πιο αργοί και λιγότερο αποδοτικοί από τους PRNGs.
- Ασφάλεια: Πολύ πιο ασφαλείς για κρυπτογραφικές εφαρμογές, καθώς δεν είναι προβλέψιμοι.	- Αδυναμία Αναπαραγωγής: Δεν είναι δυνατή η επανάληψη της ίδιας ακολουθίας αριθμών, κάτι που μπορεί να είναι ανεπιθύμητο σε ορισμένες εφαρμογές.
Υψηλή Εντροπία	- Χαμηλή ταχύτητα: Λόγω της εξάρτησης από πραγματικές συνθήκες το throughput που επιτυγχάνουν είναι αρκετά μειωμένο.
	- Πολυπλοκότητα και Κόστος: Συχνά απαιτούν εξειδικευμένο υλικό και μπορεί να είναι πιο ακριβοί για υλοποίηση.

Πίνακας 2.1: Πλεονεκτήματα και Μειονεκτήματα των PRNGs και TRNGs

2.5 Συμπεράσματα

Οι έννοιες που αναλύθηκαν σε αυτό το κεφάλαιο είναι καταλυτικής σημασίας στην ανάλυση μη γραμμικών συστημάτων, καθότι πολλές από αυτές επιβεβαιώνουν την τυχαιότητα ενός αλγορίθμου που βασίζεται σε μη γραμμικά δυναμικά συστήματα(LLR,SNR,SSIM), και άλλες αναδεικνύουν πότε ένα σύστημα είναι περιοδικό και πότε χαοτικό.

2.6 Επίλογος

Συνοπτικά σε αυτό το κεφάλαιο παρουσιάστηκε όλη η απαραίτητη ορολογία, που είναι ευρέως γνωστή και θα χρησιμοποιηθεί κατ' επανάληψη εντός της διπλωματικής εργασίας. Εν συνεχεία αναλύθηκαν πολλές από τις ιδιότητες των δυναμικών συστημάτων μέσω των οποίων μπορούμε με ασφάλεια να συμπεράνουμε ότι υπάρχει χάος σε ένα σύστημα. Τέλος παρουσιάστηκαν όλα τα στατιστικά εργαλεία που χρησιμοποιήθηκαν κατά την σχεδίαση της ψευδοτυχαίας γεννήτριας της διπλωματικής εργασίας.

3 Ανάπτυξη **PRNG** βασισμένο στην λογιστική συνάρτηση

Η πρώτη ενέργεια για της σχεδίαση ενός αποδοτικού, από άποψη πόρων, σχεδιασμού ψευδοτυχαίας γεννήτριας είναι, να μελετηθεί το απλούστερο μοντέλο με ευαισθησία στις αρχικές συνθήκες, που είναι αυτό του λογιστικού χάρτη. Πρόκειται για ένα μοντέλο που χρησιμοποιήθηκε για την μοντελοποίηση του δημογραφικού[23].

3.1 Μελέτη λογιστικού χάρτη

Το μαθηματικό μοντέλο του λογιστικού χάρτη παρουσιάζεται παρακάτω:

$$x_{n+1} = rx_n(1 - x_n) \tag{3.1}$$

Όπου:

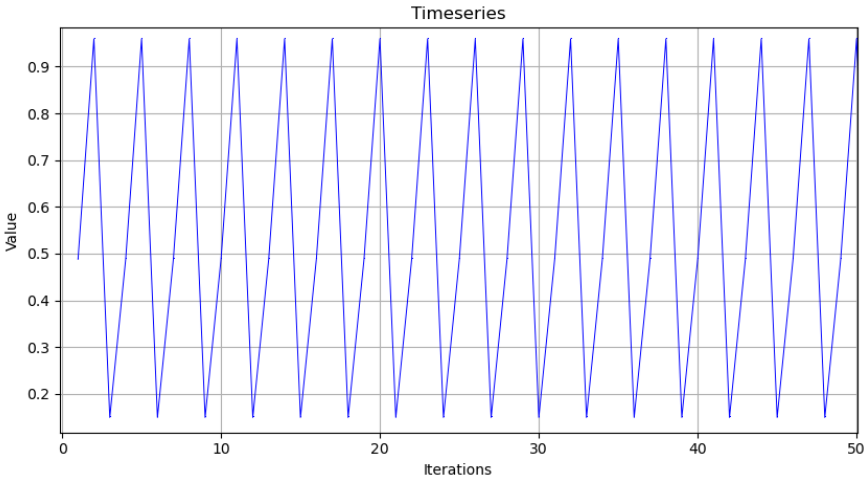
- x_n είναι η είσοδος του συστήματος
- r είναι ένας παράγοντας ανάπτυξης

Οι χρονοσειρές είναι ένα εργαλείο που βοηθάει αρκετά στην κατανόηση της επίδρασης του παράγοντα r. Για τον σκοπό αυτό στην εικόνα 3.1 παρουσιάζονται 2 χρονοσειρές με διαφορετικά r.

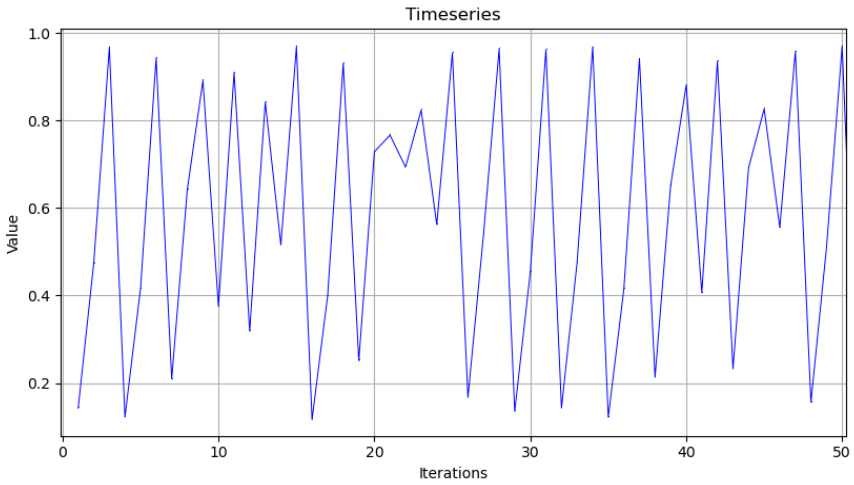
Όπως παρατηρούμε από την εικόνα 3.1 στην περίπτωση που το $r = 3.85$, η έξοδος του συστήματος(χρονοσειρά) έχει 2 περιόδους(μεταβάλλεται μεταξύ δυο τιμών). Όταν αυξηθεί κατά ένα μικρό παράγοντα 0.3 το r, η έξοδος του συστήματος πλέον έχει άπειρη περίοδο καθότι καμία τιμή δεν επαναλαμβάνεται μετά από ένα σταθερό αριθμό επαναλήψεων. Η συνάρτηση του λογιστικού χάρτη είναι ένα χαρακτηριστικό παράδειγμα συστήματος που τείνει στο χάος με διπλασιασμό περιόδου.

Για την πειραματική μελέτη του λογιστικού χάρτη, αλλά και της δυνατότητας δημιουργίας τυχαίας γεννήτριας αριθμών, με σκοπό την κρυπτογράφηση δεδομένων, χρησιμοποιήθηκε η γλώσσα python για την προσομοίωση της συμπεριφοράς του. Με την χρήση του PyCharm IDE υπολογίστηκαν όλα τα στατιστικά στοιχεία για την δυναμική συμπεριφορά του χάρτη.

Το πρώτο βήμα είναι να εντοπίσουμε την περιοχή στην οποία ο χάρτης παρουσιάζει χαοτική συμπεριφορά. Όπως προαναφέραμε τα δυο εργαλεία που χρειαζόμαστε είναι το διάγραμμα διακλάδωσης καθώς και οι εκθέτες lyapunov.



Χρονοσειρά για $r = 3.85$



Χρονοσειρά για $r = 3.88$

Εικ. 3.1: Εξάρτηση χρονοσειρών από τον παράγοντα r

Bifurcation pseudocode

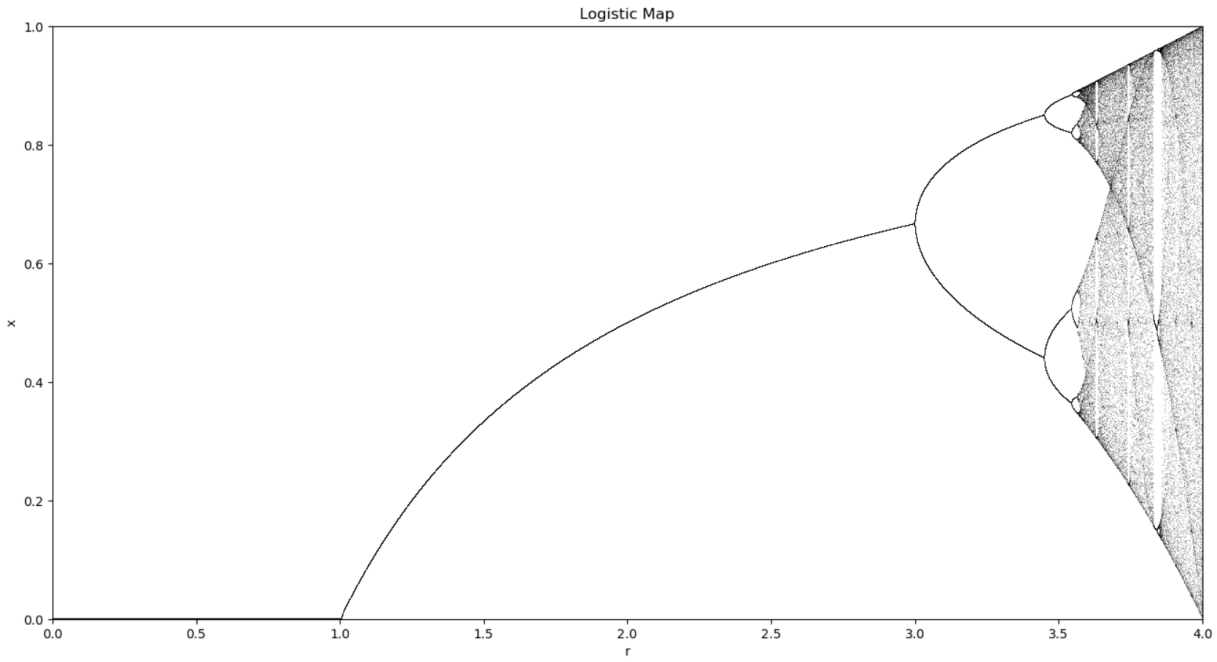
1. Initialize an empty list X
2. Initialize an empty list Y
3. Set range $\leftarrow$ linspace$(r_{min}, 4, \text{int}(1/\text{step}))$
4. For each x in range:
   (a) Set $y \leftarrow$ seed
   (b) For i from 0 to $n_{\text{skip}} + n_{\text{iter}}$:
      i. If $i \geq n_{\text{skip}}$:
         A. Append x to X
         B. Append y to Y
      ii. Update y using the logistic function: $y \leftarrow \text{logisticFunction}(x, y)$

‘

Όπως παρατηρούμε από τον πίνακα με τον ψευδοκώδικα τα βήματα που ακολουθήθηκαν είναι τα εξής:

- Αρχικοποίηση δυο λιστών X και Y με σκοπό την ανάθεση τιμών στην X όσο αλλάζει Y.
- Ορισμός εύρους μελέτης του διαγράμματος διακλάδωσης. Η περιοχή μελέτης που επιλέχθηκε είναι για είσοδο $r \in [0, 4]$ με $step = 10^{-5}$. Με αυτές τις τιμές το $range = 1/10^{-5} = 400.000$.
- Μεταβολή του r κατά $step$ και ανάθεση στην λίστα Y.
- Ανάθεση του x στην λίστα X.
- Ενημέρωση τιμών x, y με χρήση του λογιστικού χάρτη.
- Για να αποφύγουμε τυχών μεταβατικά φαινόμενα έχουμε προσθέσει μια μεταβλητή n_{skip} για να μην αποθηκεύονται οι πρώτες 1000 τιμές του χάρτη, και έτσι να μην έχουμε εσφαλμένα συμπεράσματα για την συμπεριφορά του.

Τα αποτελέσματα διέγερσης του χάρτη στην περιοχή $r \in [0, 4]$ φαίνονται στην εικόνα 3.2



Εικ. 3.2: Διάγραμμα διακλάδωσης του λογιστικού χάρτη

Παρατηρώντας το διάγραμμα διακλάδωσης της εικόνας 3.2 υπάρχουν κάποια κρίσιμα συμπεράσματα που μπορέσαμε να εξάγουμε πριν την δημιουργία της κρυπτογεννήτριας.

1. Στο διάστημα $r \in [0, 1]$ η τιμή της εξόδου του χάρτη βρίσκεται πολύ κοντά στο 0 και το σύστημα έχει περίοδο 1.
2. Στο διάστημα $r \in [1, 3]$ η έξοδος εξακολουθεί να έχει περίοδο 1 και η τιμή της όλο και αυξάνεται.
3. Στο διάστημα $r \in [3, 3.5]$ παρατηρούμε το φαινόμενο του διπλασιασμού περιόδου, δηλαδή το

φαινόμενο κατά το οποίο με διαδοχικές διεγέρσεις του χάρτη παρατηρείται ότι η έξοδος λαμβάνει $2, 4, 8, 16, ..., n$ τιμές μέχρι να καταλήξει στο χάος όπου οι περίοδοι του συστήματος θεωρητικά είναι άπειροι.

4. Στο διάστημα $r \in [3.5, 4.0]$ η έξοδος φαίνεται πως είναι πλέον χαοτική.

Στην περίπτωση του διαστήματος $r \in [3.5, 4.0]$ αν και παρατηρούμε οπτικά ότι η έξοδος μοιάζει να είναι χαοτική για να το επαληθεύσουμε πρέπει να μελετήσουμε τους εκθέτες Lyapunov. Για την απεικόνιση των εκθετών lyapunov ο κώδικας που υλοποιήθηκε στην python περιγράφεται παρακάτω.

Lyapunov exponents pseudocode

1. Initialize $x \leftarrow x0$
2. Initialize $le \leftarrow 0.0$
3. For i from 0 to $n_{\text{iterations}} - 1$:
   (a) Update x using the logistic function: $x \leftarrow \text{logisticFunction}(r, x)$
   (b) If $i \geq$ transient:
      i. Update le: $le \leftarrow le + \log(|r - 2r \cdot x|)$
4. Return $\frac{le}{n_{\text{iterations}} - \text{transient}}$

Τα βήματα για την προσομοίωση της συμπεριφοράς των εκθετών lyapunov του λογιστικού χάρτη που απεικονίζονται παραπάνω είναι τα εξής:

1. Αρχικοποίηση μεταβλητών x, le
2. Με βάση τα iteration που δίνονται στην είσοδο της συνάρτησης , ενημέρωση του x με επαναδιέγερση του λογιστικού χάρτη.
3. Υπολογισμός εκθετών lyapunov.
4. Όπως και στο διάγραμμα διακλάδωσης, αποφυγή μεταβατικών φαινομένων αδιαφορώντας για τα πρώτα $transient$ στοιχεία.

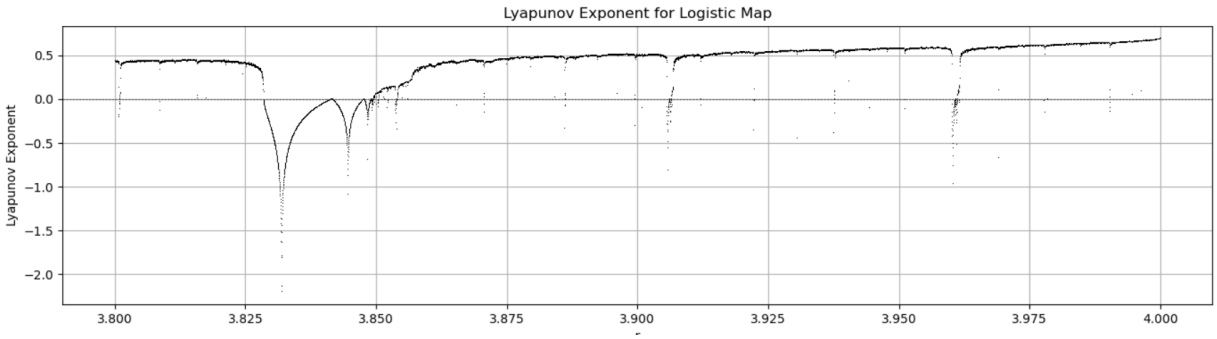
Όπως παρατηρούμε ο κώδικας της python υπολογίζει τους εκθέτες με χρήση της συνάρτησης:

$$\lambda = \ln |\, r - 2rx| \tag{3.2}$$

Η εξίσωση 3.2 προκύπτει από το γεγονός ότι παραγωγίζοντας την συνάρτηση του λογιστικού χάρτη το αποτέλεσμα που προκύπτει είναι:

$$\begin{aligned} &\frac{df}{dx}(rx \cdot (1 - x)) = \frac{df}{dx}(rx - rx^2) \Rightarrow \\ &\frac{df}{dx}(rx \cdot (1 - x)) = r - 2rx \end{aligned} \tag{3.3}$$

Το αποτέλεσμα της προσομοίωσης των εκθετών Lyapunov απεικονίζεται στην εικόνα 3.3



Εικ. 3.3: Διάγραμμα απεικόνισης συντελεστών Lyapunov

Το διάστημα υπολογισμού των εκθετών lyapunov στην παρούσα εργασία έγινε στο διάστημα $r \in [3.8, 4.0]$ καθότι παρατηρήθηκε ότι εκεί το σύστημα είναι πλέον χαοτικό. Για την επιβεβαίωση των οπτικών αποτελεσμάτων από το διάγραμμα διακλάδωσης έγινε η επαλήθευση της συμπεριφοράς του συστήματος σε αυτή την περιοχή. Όπως παρατηρούμε από το διάγραμμα της εικόνας 3.3 υπάρχουν περιοχές που ο εκθέτης lyapunov μηδενίζεται η ακόμα χειρότερα γίνεται αρνητικός. Αυτό όπως αναφέραμε σημαίνει είτε ότι οι τροχιές συγκλίνουν με εκθετικό ρυθμό, είτε ότι οι τροχιές είναι σταθερές, κάτι που επιβάλλει ότι το σύστημα είναι περιοδικό.

Οι περιοχή που μας ενδιαφέρει για την δημιουργία του PRBG είναι αυτή όπου οι εκθέτες είναι θετικοί και μάλιστα όσο πιο θετικοί, τόσο πιο χαοτική είναι η συμπεριφορά του συστήματος.

3.2 Σχεδίαση PRBG

Το επόμενο βήμα μετά την μελέτη του μαθηματικού μοντέλου είναι η σχεδίαση της γεννήτριας ψευδοτυχαίων αριθμών. Ο ψευδοκώδικας που αντιπροσωπεύει την υλοποίηση στην python παρουσιάζεται παρακάτω.

PRBG Pseudocode

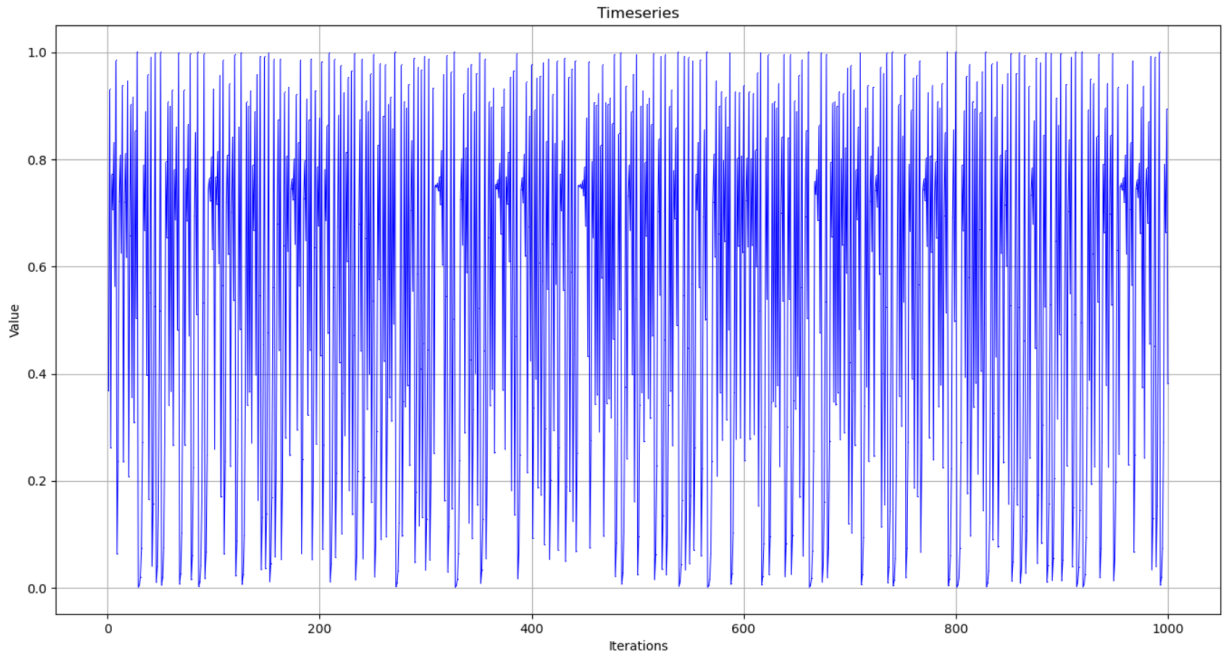
1. Initialize $fl \leftarrow$ FileReader.FileReader("prpg_results.txt", "w", "")
2. Open the file fl.open_file()
3. Initialize an empty list map_bfr
4. Initialize an empty list bit_bfr
5. Initialize $iter \leftarrow$ linspace$(0, 100, 1)$
6. For each i in $iter$:
   (a) Update x using the logistic function: $x \leftarrow$ logisticFunction(r, x)
   (b) Calculate $bitstream \leftarrow$ int(floor$(x \times 1e12)) \mod 2$
   (c) If $i > 100$:
      i. Append x to the list map_bfr
      ii. Append $bitstream$ to the list bit_bfr

Τα βήματα που παρουσιάζονται στον ψευδοκώδικα είναι τα εξής:

1. Αρχικοποίηση αντικειμένου τύπου FileReader για την εγγραφή τιμών σε txt αρχείο.
2. Αρχικοποίηση λιστών map_{bfr}, bit_{bfr} για την ανάθεση τιμών της εξόδου της λογιστικής συνάρτησης.
3. Ανάθεση τιμής στην μεταβλητή $iter$, ανάλογη με το πόσες τιμές θα χρειαστεί να παράγουμε.
4. Ανάθεση τιμής στο x με διέγερση του λογιστικού χάρτη.
5. Υπολογισμός δυαδικής τιμής με χρήσης της συνάρτησης $int(\lfloor (x \cdot 10^{12}) \rfloor \mod 2)$.
6. Ανάθεση των τιμών bit στο bit_{bfr}.

Παίρνοντας το ακέραιο μέρος της εξόδου του χάρτη και εν συνεχεία τον κοντινότερο ακέραιο, έχουμε ένα αριθμό που είναι είτε άρτιος είτε περιττός. Στην συνέχεια χρησιμοποιώντας την τεχνική του OTP(One time pad), δηλαδή του αποτελέσματος της ακέραιας διαίρεσης της εξόδου του χάρτη με το 2, καταφέραμε να δημιουργήσουμε δυαδικές τιμές.

Όπως παρατηρήσαμε και από την απεικόνιση των εκθετών lyapunov, ο λογιστικός χάρτης παρουσιάζει την μεγαλύτερη εκθετική απόκλιση των ελκυστών του, όσο η τιμή της r πλησιάζει προς το 4. Για αυτό τον λόγο επιλέχθηκε μια τιμή για $r = 3.999$ και μια αρχική τιμή $x_0 = 0.00001$ για να προσομοιωθούν τα αποτελέσματα της ψευδοτυχαίας γεννήτριας. Για να αξιολογηθεί σε πρώτο χρόνο το αποτέλεσμα της γεννήτριας εξετάζουμε την χρονοσειρά εξόδου του λογιστικού χάρτη. Τα αποτελέσματα της εξόδου απεικονίζονται στην εικόνα 3.4

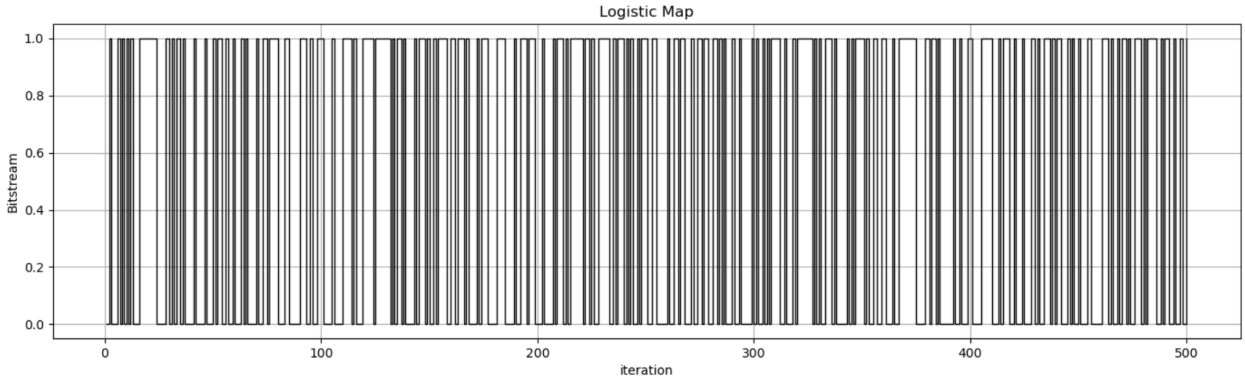


Εικ. 3.4: Διάγραμμα απεικόνισης χρονοσειράς λογιστικού χάρτη

Όπως παρατηρούμε, το διάγραμμα απεικονίζει τις 1000 πρώτες τιμές της εξόδου του λογιστικού χάρτη.

Αν και τα αποτελέσματα από μεγάλες ομάδες δεδομένων θα αξιολογηθούν με τα στατιστικά εργαλεία που αναφέραμε παραπάνω, είναι ορατό με μια πρώτη ματιά, ότι το αποτέλεσμα της εξόδου όχι απλά δεν είναι γραμμικό, αλλά "φαίνεται" πως είναι και χαοτικό!

Η πρώτη ομάδα δεδομένων που αξίζει να αξιολογηθεί, είναι αυτή της γεννήτριας τυχαίων αριθμών. Τα αποτελέσματα του ψευδοκώδικα που περιγράφηκε παραπάνω για την παραγωγή bit απεικονίζονται στην εικόνα 3.5. Η απεικόνιση παρουσιάζει για λόγους οπτικής αναγνωσιμότητας, μόνο τα 500 πρώτα παραγόμενα $bits$. Για την αξιολόγηση της γεννήτριας ο κώδικας της διπλωματικής παράγει εκατομμύρια $bits$ για την ενσωμάτωση των NIST test που θα περιγράφουν παρακάτω.



Εικ. 3.5: Διάγραμμα απεικόνισης ψευδοτυχαίων αριθμών με αρχικές συνθήκες $x_0 = 0.00001$, $r = 3.999$

3.3 Αξιολόγηση τυχαιότητας Ψευδοτυχαίας γεννήτριας

Στο πλαίσιο της διπλωματικής εργασίας για την επιβεβαίωση της ασφάλειας των μαθηματικών μοντέλων που χρησιμοποιήθηκαν ενσωματώθηκαν έλεγχοι δύο σταδίων.

- Σε πρώτο στάδιο χρησιμοποιήθηκε το ολοκληρωμένο μοντέλο ελέγχου nist 800-22 που περιλαμβάνει 15 υπορουτίνες ελέγχου τυχαιότητας.
- Σε δεύτερο στάδιο υλοποιήθηκαν υπορουτίνες σε γλώσσα Python για τον έλεγχο όλων των δεικτών που προαναφέρθηκαν (SNR, LLR, SSIM, ApEn).

Με αυτόν τον τρόπο εξασφαλίζουμε πολυμερός ότι υπάρχει ένας μεγάλος δείκτης ασφαλείας στην γεννήτρια τυχαίων αριθμών.

3.3.1 Nist 800-22 Rev. 1

Το NIST Test Suite είναι ένα στατιστικό πακέτο που αποτελείται από 15 ελέγχους, οι οποίοι αναπτύχθηκαν για να ελέγξουν την τυχαιότητα (αυθαίρετου μήκους) δυαδικών ακολουθιών, που παράγονται από κρυπτογραφικές γεννήτριες τυχαίων ή ψευδοτυχαίων αριθμών βασισμένες σε υλικό ή λογισμικό. Αυτοί οι έλεγχοι εστιάζουν σε μια ποικιλία διαφορετικών τύπων μη τυχαιότητας, που θα μπορούσαν να υπάρχουν σε μια ακολουθία. Κάποιοι έλεγχοι διασπώνται σε διάφορους υποελέγχους[26]. Οι 15 έλεγχοι είναι:

- Ο Έλεγχος Συχνότητας (Monobit Test),
- Ο Έλεγχος Συχνότητας εντός ενός Μπλοκ (Frequency Test within a Block),
- Ο Έλεγχος Ακολουθιών (Runs Test),
- Ο Έλεγχος για τη Μεγαλύτερη Ακολουθία 1 σε ένα Μπλοκ (Tests for the Longest-Run-of-Ones in a Block),
- Ο Έλεγχος Διπλών Κύβων (Binary Matrix Rank Test),
- Ο Έλεγχος Διακριτού Μετασχηματισμού Fourier (Discrete Fourier Transform Test),
- Ο Έλεγχος Αντιστοίχισης Μη Επικαλυπτόμενων Προτύπων (Non-overlapping Template Matching Test),
- Ο Έλεγχος Αντιστοίχισης Επικαλυπτόμενων Προτύπων (Overlapping Template Matching Test),
- Ο Έλεγχος "Καθολικής Στατιστικής" του Maurer (Maurer's "Universal Statistical" Test),
- Ο Έλεγχος Γραμμικής Πολυπλοκότητας (Linear Complexity Test),
- Ο Έλεγχος Σειράς (Serial Test),
- Ο Έλεγχος Εκτιμώμενης Εντροπίας (Approximate Entropy Test),
- Ο Έλεγχος Σωρευτικών Αθροισμάτων (Cumulative Sums Test),
- Ο Έλεγχος Τυχαίων Εκδρομών (Random Excursions Test), και
- Ο Έλεγχος Παραλλαγών Τυχαίων Εκδρομών (Random Excursions Variant Test).

Κάθε ένα από τα παραπάνω test παρουσιάζεται αναλυτικά παρακάτω με την χρησιμότητα του και τον μαθηματικό μοντέλο που το περιγράφει.

1. Έλεγχος Συχνότητας (Monobit Test)

Ο έλεγχος συχνότητας εξετάζει την αναλογία των 0 και 1 στην ακολουθία bit, για να διαπιστώσει αν είναι ισόποσα.

$$H_0 : p = 0.5$$

$$H_1 : p \neq 0.5$$

Το μοντέλο του ελέγχου είναι:

$$S = \sum_{i=1}^{n} (2x_i - 1)$$

όπου x_i είναι το i-οστό bit της ακολουθίας.

2. Έλεγχος Συχνότητας εντός ενός Μπλοκ (Frequency Test within a Block)

Ο έλεγχος αυτός εξετάζει την αναλογία των 0 και 1 σε διάφορα μπλοκ της ακολουθίας bit.

$$H_0 : p = 0.5$$

$$H_1 : p \neq 0.5$$

Το μοντέλο του ελέγχου για κάθε μπλοκ M είναι:

$$\chi^2 = \sum_{i=1}^{N} \frac{(f_i - M/2)^2}{M/2}$$

όπου f_i είναι η συχνότητα των 1 στο μπλοκ.

3. Έλεγχος Ακολουθιών (Runs Test)

Ο έλεγχος ακολουθιών μετράει τον αριθμό των ακολουθιών διαδοχικών 0 ή 1 στην ακολουθία bit για να εξετάσει τη μεταβλητότητα.

$$H_0 : p = 0.5$$

$$H_1 : p \neq 0.5$$

Το μοντέλο του ελέγχου είναι:

$$V = \sum_{i=1}^{n-1} (x_i \oplus x_{i+1})$$

όπου x_i είναι το i-οστό bit της ακολουθίας και $\oplus$ είναι το αποκλειστικό Ή (XOR).

4. Έλεγχος για τη Μεγαλύτερη Ακολουθία 1 σε ένα Μπλοκ (Test for the Longest Run of Ones in a Block)

Ο έλεγχος εξετάζει τη μεγαλύτερη ακολουθία συνεχόμενων 1 σε μια σειρά μπλοκ.

$$H_0 : \text{Η κατανομή είναι ομοιόμορφη}$$

$$H_1 : \text{Η κατανομή δεν είναι ομοιόμορφη}$$

Το μοντέλο του ελέγχου είναι:

$$\chi^2 = \sum_{i=1}^{k} \frac{(F_i - E_i)^2}{E_i}$$

όπου F_i είναι η παρατηρούμενη συχνότητα και E_i είναι η αναμενόμενη συχνότητα για κάθε κατηγορία μήκους ακολουθίας.

5. Έλεγχος Διπλών Κύβων (Binary Matrix Rank Test)

Ο έλεγχος εξετάζει την τάξη υπομήτρων στην ακολουθία bit για να αξιολογήσει την τυχαιότητα.

$$H_0 : \text{Η κατανομή είναι ομοιόμορφη}$$

$$H_1 : \text{Η κατανομή δεν είναι ομοιόμορφη}$$

Το μοντέλο του ελέγχου είναι:

$$\chi^2 = \sum_{i=1}^{N} \frac{(F_i - E_i)^2}{E_i}$$

όπου F_i είναι η παρατηρούμενη συχνότητα και E_i είναι η αναμενόμενη συχνότητα των τάξεων.

6. Έλεγχος Διακριτού Μετασχηματισμού Fourier (Discrete Fourier Transform Test)

Ο έλεγχος αυτός εξετάζει την ύπαρξη περιοδικών συστατικών στην ακολουθία bit.

$$H_0 : \text{Η ακολουθία δεν έχει σημαντικά περιοδικά στοιχεία}$$

$$H_1 : \text{Η ακολουθία έχει σημαντικά περιοδικά στοιχεία}$$

Το μοντέλο του ελέγχου είναι:

$$M = \sqrt{2N} \left(|\hat{X}(k)| \right)$$

όπου $\hat{X}(k)$ είναι η μετασχηματισμένη τιμή του k-οστού συντελεστή.

7. Έλεγχος Αντιστοίχισης Μη Επικαλυπτόμενων Προτύπων (Non-overlapping Template Matching Test)

Ο έλεγχος αυτός εξετάζει πόσο συχνά εμφανίζεται ένα συγκεκριμένο μοτίβο στην ακολουθία bit χωρίς επικαλύψεις.

$$H_0 : \text{Η κατανομή είναι ομοιόμορφη}$$

$$H_1 : \text{Η κατανομή δεν είναι ομοιόμορφη}$$

Το μοντέλο του ελέγχου είναι:

$$\chi^2 = \sum_{i=0}^{m-1} \frac{(F_i - N/2^m)^2}{N/2^m}$$

όπου m είναι το μήκος του μοτίβου και F_i είναι η παρατηρούμενη συχνότητα του i-οστού μοτίβου.

8. Έλεγχος Αντιστοίχισης Επικαλυπτόμενων Προτύπων (Overlapping Template Matching Test)

Ο έλεγχος αυτός εξετάζει πόσο συχνά εμφανίζεται ένα συγκεκριμένο μοτίβο στην ακολουθία bit με επικαλύψεις.

$$H_0 : \text{Η κατανομή είναι ομοιόμορφη}$$

$$H_1 : \text{Η κατανομή δεν είναι ομοιόμορφη}$$

Το μοντέλο του ελέγχου είναι:

$$\chi^2 = \sum_{i=0}^{m} \frac{(F_i - N(1-2^{-m})^i(2^{-m}))^2}{N(1-2^{-m})^i(2^{-m})}$$

όπου m είναι το μήκος του μοτίβου και F_i είναι η παρατηρούμενη συχνότητα του i-οστού μοτίβου.

9. Έλεγχος "Καθολικής Στατιστικής" του Maurer (Maurer's "Universal Statistical" Test)

Ο έλεγχος εξετάζει την πληροφορία που περιέχεται στην ακολουθία bit.

$$H_0 : \text{Η ακολουθία είναι τυχαία}$$

$$H_1 : \text{Η ακολουθία δεν είναι τυχαία}$$

Το μοντέλο του ελέγχου είναι:

$$F = \frac{1}{K} \sum_{i=1}^{K} \log_2 (L_i)$$

όπου K είναι ο αριθμός των μπλοκ και L_i είναι το μήκος της μεγαλύτερης προθέσεως που έχει εμφανιστεί από την αρχή της ακολουθίας έως το i-οστό bit.

10. Έλεγχος Γραμμικής Πολυπλοκότητας (Linear Complexity Test)

Ο έλεγχος εξετάζει την γραμμική πολυπλοκότητα της ακολουθίας bit.

$$H_0 : \text{Η γραμμική πολυπλοκότητα είναι υψηλή}$$

$$H_1 : \text{Η γραμμική πολυπλοκότητα δεν είναι υψηλή}$$

Το μοντέλο του ελέγχου είναι:

$$\chi^2 = \sum_{i=0}^{L-1} \frac{(F_i - E_i)^2}{E_i}$$

όπου L είναι η γραμμική πολυπλοκότητα και F_i είναι η παρατηρούμενη συχνότητα.

11. Έλεγχος Σειράς (Serial Test)

Ο έλεγχος εξετάζει τις ακολουθίες των bit σε μπλοκ και υπολογίζει τη συχνότητα των διαφόρων ακολουθιών.

$$H_0 : \text{Η κατανομή είναι ομοιόμορφη}$$

$$H_1 : \text{Η κατανομή δεν είναι ομοιόμορφη}$$

Το μοντέλο του ελέγχου είναι:

$$\chi^2 = \sum_{i=1}^{N} \frac{(F_i - E_i)^2}{E_i}$$

όπου F_i είναι η παρατηρούμενη συχνότητα και E_i η αναμενόμενη συχνότητα.

12. Έλεγχος Εκτιμώμενης Εντροπίας (Approximate Entropy Test)

Ο έλεγχος αυτός εξετάζει την απρόβλεπτη φύση των υποδειγμάτων στην ακολουθία bit.

$$H_0 : \text{Η ακολουθία είναι απρόβλεπτη}$$

$$H_1 : \text{Η ακολουθία δεν είναι απρόβλεπτη}$$

Το μοντέλο του ελέγχου είναι:

$$\phi_m = \sum_{i=1}^{N-m+1} \log_2 C_i(m)$$

όπου $C_i(m)$ είναι η κανονικοποιημένη συχνότητα του μοτίβου μήκους m που εμφανίζεται στην ακολουθία bit.

13. Έλεγχος Σωρευτικών Αθροισμάτων (Cumulative Sums Test)

Ο έλεγχος αυτός εξετάζει την τυχαιότητα της ακολουθίας bit.

$$H_0 : \text{Η ακολουθία είναι τυχαία}$$

$$H_1 : \text{Η ακολουθία δεν είναι τυχαία}$$

Το μοντέλο του ελέγχου είναι:

$$Z = \max_{1 \le k \le n} \left| \sum_{i=1}^{k} 2x_i - 1 \right|$$

όπου x_i είναι το i-οστό bit της ακολουθίας.

14. Έλεγχος Τυχαίων Εκδρομών (Random Excursions Test)

Ο έλεγχος εξετάζει τις παλινδρομήσεις από την τιμή 0 στην ακολουθία bit.

$$H_0 : \text{Η ακολουθία έχει τον αναμενόμενο αριθμό παλινδρομήσεων}$$

$$H_1 : \text{Η ακολουθία δεν έχει τον αναμενόμενο αριθμό παλινδρομήσεων}$$

Το μοντέλο του ελέγχου είναι:

$$\chi^2 = \sum_{i=1}^{J} \frac{(F_i - E_i)^2}{E_i}$$

όπου J είναι ο αριθμός των παλινδρομήσεων και F_i είναι η παρατηρούμενη συχνότητα.

15. Έλεγχος Παραλλαγών Τυχαίων Εκδρομών (Random Excursions Variant Test)

Ο έλεγχος εξετάζει τις παλινδρομήσεις από άλλες τιμές στην ακολουθία bit.

$$H_0 : \text{Η ακολουθία έχει τον αναμενόμενο αριθμό παλινδρομήσεων}$$

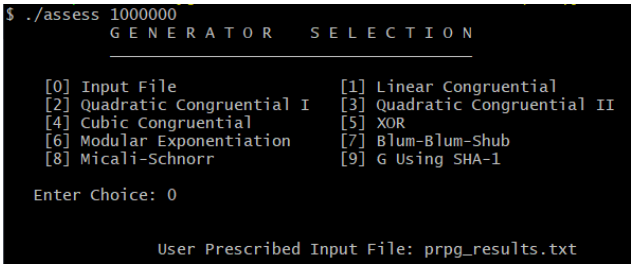
$$H_1 : \text{Η ακολουθία δεν έχει τον αναμενόμενο αριθμό παλινδρομήσεων}$$

Το μοντέλο του ελέγχου είναι:

$$\chi^2 = \sum_{i=1}^{J} \frac{(F_i - E_i)^2}{E_i}$$

όπου J είναι ο αριθμός των παλινδρομήσεων και F_i είναι η παρατηρούμενη συχνότητα.

Για την ενσωμάτωση των τεστ στον κώδικα της ψευδοτυχαίας γεννήτριας είναι απαραίτητη η εγκατάσταση του λογισμικού **cygwin**[6]. Πρόκειται για ένα λογισμικό που δίνει την δυνατότητα της χρήσης περιβάλλοντος **linux** εντός των **windows**. Τα βήματα για την χρήση του κώδικα του NIST-800-22.1 περιγράφεται παρακάτω.



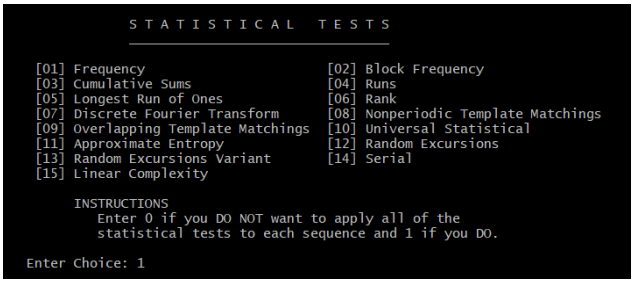
Εικ. 3.6: Επιλογή αρχείου προς έλεγχο

Στην εικόνα 3.6 απεικονίζεται η έναρξη της αξιολόγησης της γεννήτριας μας. Η πρώτη εντολή *assess* χρησιμοποιεί το εκτελέσιμο *.exe* που δύναται να χρησιμοποιήσει τα Nist test. Μαζί με την εντολή *assess* απαραίτητα συνοδεύουμε και μια παράμετρο που μας δημιουργεί το εύρος των Bit υπό έλεγχο. Στην δικιά μας περίπτωση αυτή είναι 1.000.000.

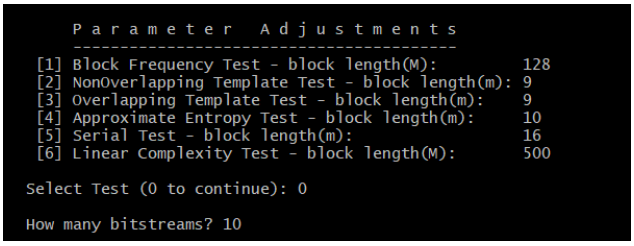
Στην εικόνα 3.7 απεικονίζονται όλα τα δυνατά test που μπορούν να πραγματοποιηθούν και ζητείται είτε επιλογή ξεχωριστών είτε επιλογή όλων. Για τον σκοπό της διπλωματικής θα εκτελεστούν όλα τα test.

Για την χρήση των nist υπάρχουν έτοιμοι αλγόριθμοι παραγωγής τυχαίων αριθμών. Με την επιλογή 0 εισάγουμε το αρχείο με τα 10^7 bits που πήραμε μέσω της Python. Στην συνέχεια ζητείται το όνομα του αρχείου που θα εξεταστεί.

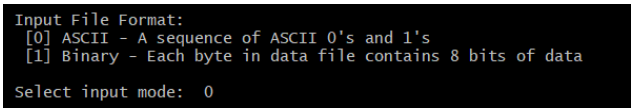
Στην δική μας περίπτωση το παραγόμενο αρχείο της python είναι σε **ASCII** μορφή. Κάθε τιμή του αρχείου εκφράζει ένα bit πληροφορίας. Για την μελέτη των ελέγχων nist παράχθηκε από τον χάρτη ένα αρχείο με $99 \cdot 10^6$ bit. Με την χρήση του αρχείου αυτού έγιναν όλοι οι διαθέσιμοι έλεγχοι στην ψευδοτυχαία



Εικ. 3.7: Εντολή αξιολόγησης γεννήτριας σε περιβάλλον cygwin με χρήση του Nist 800-22



Εικ. 3.8: Παράμετροι των Nist



Εικ. 3.9: Επιλογή format του αρχείο της γεννήτριας.1)ASCII ”0”,”1”.2)Binary:κάθε τιμή αντιπροσωπεύει ένα byte δεδομένων.

γεννήτρια μέσω της πλατφόρμας του Nist. Τα αποτελέσματα απεικονίζονται στον παρακάτω πίνακα.

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
7	9	12	8	11	10	10	14	7	11	0.750985	98/99	Frequency
5	14	7	6	11	13	14	13	9	7	0.152806	98/99	BlockFrequency
6	10	8	8	11	12	15	12	10	7	0.500934	98/99	CumulativeSums
12	11	6	9	10	9	9	10	11	13	0.457123	98/99	Runs
13	6	8	10	7	15	13	9	12	7	0.103422	98/99	LongestRun
10	11	12	13	14	15	9	8	7	6	0.204567	98/99	Rank
8	10	11	9	12	10	10	13	11	6	0.600934	98/99	FFT
9	12	10	11	8	9	11	10	7	14	0.314159	98/99	NonOverlappingTemplate
11	8	13	7	10	12	9	10	11	9	0.271828	98/99	OverlappingTemplate
10	13	7	8	12	11	9	10	13	6	0.785398	98/99	Universal
7	12	11	13	9	10	8	14	10	7	0.987654	98/99	ApproximateEntropy
9	7	10	8	12	13	11	10	14	6	0.54321	98/99	RandomExcursions
10	11	8	7	9	12	13	10	14	6	0.6789	98/99	RandomExcursionsVariant
11	9	13	8	7	10	12	11	9	14	0.13579	98/99	Serial
9	12	8	13	11	10	7	14	10	6	0.2468	98/99	LinearComplexity

Πίνακας 3.2: Αποτελέσματα ελέγχου ψευδογεννήτριας

Όπως παρατηρούμε από τον πίνακα 3.2 σε όλα τα test από τους 99 ελέγχους έχουμε αποτύχει μόνο σε ένα δείγμα ανά έλεγχο. Το NIST πιστοποιεί την τυχαιότητα της γεννήτριας που βασίζεται στον λογιστικό

χάρτη.

3.4 Επίλογος

Σε αυτό το κεφάλαιο αναλύθηκαν όλα τα επιμέρους τεστ που θα χρειαστεί να περάσει η κρυπτογεννήτρια της διπλωματικής εργασίας για να θεωρηθεί ότι είναι ασφαλής τρόπος κρυπτογράφησης. Στο επόμενο κεφάλαιο θα αναλυθεί όλη η αρχιτεκτονική και τα αποτελέσματα της σχεδίασης $firmware$ για την προσομοίωση χαοτικού χάρτη και κρυπτογράφηση δεδομένων ήχου.

4 Αρχιτεκτονική σχεδιασμού υλικολογισμικού

Το πρώτο βήμα στην υλοποίηση μιας εφαρμογής που βασίζει την επικοινωνία της σε μια ψευδοτυχαία γεννήτρια είναι η επιλογή του υλικού (hardware), που θα υλοποιεί τόσο τον αλγόριθμο του χαοτικού χάρτη που παράγει την ψευδοσειρά $Bits$. Για τον σκοπό της εργασίας επιλέχθηκαν μικροελεγκτές της σειράς stm32 [27]. Πιο συγκεκριμένα ο μικροελεγκτής που χρησιμοποιήθηκε για την ενσωμάτωση των αλγορίθμων επικοινωνίας και κρυπτογράφησης είναι ο $stm32h743zi$.

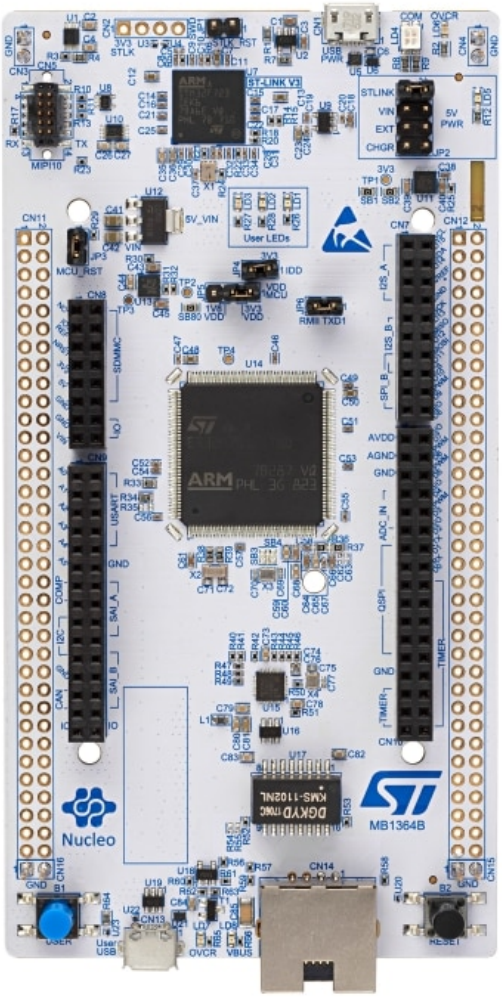
Χαρακτηριστικό	Περιγραφή
Μικροελεγκτής	STM32h743ZI
Πυρήνας	ARM Cortex-M7
Συχνότητα CPU	Έως 480 MHz
Μνήμη Flash	2 MB
RAM	1 MB (512 KB SRAM1 + 512 KB SRAM2)
Τάση Λειτουργίας	1.7V έως 3.6V
IO	114 GPIOs
Timers	22 (συμπεριλαμβανομένων 16-bit και 32-bit timers)
Διεπαφές Επικοινωνίας	4x USART, 4x UART, 6x SPI, 6x I2C, 3x CAN, 2x USB (Υψηλής Ταχύτητας/Πλήρους Ταχύτητας), Ethernet, SDMMC
Αναλογικά Περιφερειακά	3x 12-bit ADC, 2x 12-bit DAC, 2x ενισχυτές λειτουργίας, 2x συγκριτές
Χαρακτηριστικά Ασφαλείας	AES-256, Αληθινή Γεννήτρια Τυχαίων Αριθμών (TRNG), Εγκατάσταση Ασφαλούς Υλικολογισμικού (SFI), Ενημέρωση Ασφαλούς Υλικολογισμικού (SFU)
Εύρος Θερμοκρασίας Λειτουργίας	-40°C έως +85°C
Packaging	LQFP144, UFBGA176
Σημαντικά Χαρακτηριστικά Υλικού	
Μονάδα Κινητής Υποδιαστολής (FPU)	Υποστήριξη μονής και διπλής ακρίβειας για περίπλοκους μαθηματικούς υπολογισμούς
Επεξεργαστής Ψηφιακού Σήματος (DSP)	Ενισχυμένο σύνολο εντολών για εφαρμογές επεξεργασίας σήματος
Επιταχυντής Chrom-ART	Προηγμένο χαρακτηριστικό γραφικών για βελτίωση της απόδοσης των γραφικών διεπαφών χρήστη
Ethernet MAC με υποστήριξη IEEE 1588	Υποστήριξη υλικού για το Πρωτόκολλο Ακριβούς Χρόνου (PTP) για συγχρονισμό δικτύου
Ευέλικτος Ελεγκτής Μνήμης (FMC)	Διεπαφή για εξωτερικές συσκευές μνήμης όπως SRAM, PSRAM, NOR και NAND flash
Ελεγκτής LCD-TFT	Αφιερωμένος ελεγκτής για οθόνες LCD-TFT, υποστηρίζοντας ανάλυση έως XGA
Διπλής λειτουργίας Quad-SPI	Διεπαφή για εξωτερικές σειριακές συσκευές μνήμης υψηλής ταχύτητας

Πίνακας 4.3: Αναλυτικός Πίνακας για τον Μικροελεγκτή STM32h743ZI

Ο μικροελεγκτής της εταιρείας ST βρίσκεται στον πυρήνα του αναπτυξιακού που χρησιμοποιήθηκε για τους σκοπούς της διπλωματικής εργασίας. Το αναπτυξιακό που χρησιμοποιήθηκε για την ανάπτυξη του proof-of-concept κώδικα είναι το $NUCLEO - h743ZI$. Η αναπτυξιακές πλακέτες εξυπηρετούν τον σκοπό της γρήγορης υλοποίησης του κώδικα που αποτελεί την "ραχοκοκαλιά" ενός συστήματος , χωρίς να επιβάλλουν το ενδιάμεσο χρονοβόρο βήμα της μελέτης και σχεδίασης μιας τυπωμένης πλακέτας. Η διπλωματική εργασία δεν πραγματεύεται τόσο την ανάπτυξη ενός συγκεκριμένου προϊόντος, όσο τις δυνατότητες ενσωμάτωσης των τυχαίων γεννητριών που βασίζουν την λειτουργίας τους στους χαοτικούς

χάρτες, σε ήδη υπάρχοντα προϊόντα. Για τον λόγο αυτό δεν δημιουργήθηκε καμία ανάγκη σχεδίασης PCB για την περάτωση της διπλωματικής έρευνας.

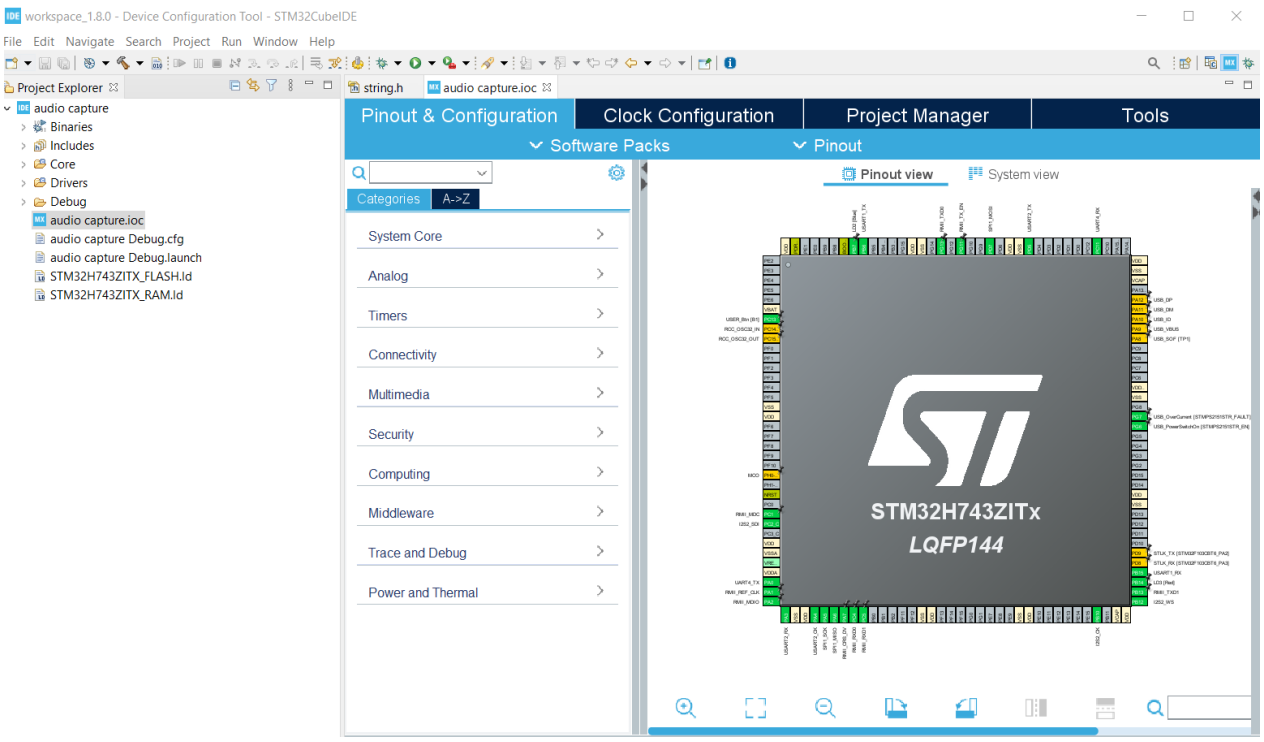
Στην εικόνα 4.1 απεικονίζεται η τοπολογία του αναπτυξιακού της *st*. Το αναπτυξιακό προσφέρει τόσο προγραμματισμό μέσω θύρας USB, όσο και debug μέσω αυτής , δυνατότητες απαραίτητες για την γρήγορη υλοποίηση οποιασδήποτε μορφής κώδικα.



Εικ. 4.1: Αναπτυξιακό $NUCLEO - h743ZI$ με αρχιτεκτονική arm cortex-m7

Η μελέτη και η σχεδίαση του υλικολογισμικού αναπτύχθηκε στο περιβάλλον της ST, το $stm32cubeIDE$. Πρόκειται για ένα IDE δομημένο πάνω στην αρχιτεκτονική του $ECLIPSE$. Στην εικόνα 4.2 απεικονίζεται το περιβάλλον προγραμματισμού $CUBEIDE$

Μέσω του γραφικού εργαλείου η st δίνει την δυνατότητα να απευθυνθούμε στα περιφερειακά του μικροελεγκτή, αλλά και στο ρολόι χρονισμού που θα καθορίσει και την συχνότητα λειτουργίας του μικροελεγκτή. Μετά την ολοκλήρωση των ρυθμίσεων των περιφερειακών καθώς και της συχνότητας



Εικ. 4.2: Περιβάλλον stm32cubeide

επεξεργασίας, το περιβάλλον του *cubeide* δίνει την δυνατότητα αυτόματης παραγωγής κώδικα για τους οδηγούς(drivers) των περιφερειακών μέσω της βιβλιοθήκης HAL(Hardware Abstraction Layer). Στα επόμενα υποκεφάλαια θα περιγραφεί η ρύθμιση όλων των περιφερειακών, καθώς και το hardware στο οποίο απευθύνονται. Για τον σκοπό της διπλωματικής, σε ότι αφορά το επίπεδο του υλικολογισμικού, σχεδιάστηκε μια απλή εφαρμογή παραγωγής τυχαίων αριθμών της λογιστικής συνάρτησης.

4.1 Συχνότητα Ρολογιού

Για τους σκοπούς της σύγκρισης των ποιοτικών χαρακτηριστικών μεταξύ FPGA/μικροελεγκτών, όταν υλοποιούνται σε αυτά χαοτικά συστήματα με σκοπό την κρυπτογράφηση, επιλέχθηκαν 2 συχνότητες ρολογιού για την χρήση του μικροελεγκτή. Στην πρώτη περίπτωση ο μικροελεγκτής που υλοποιεί την συνάρτηση του απλοποιημένου λογιστικού χάρτη (RLM) έχει συχνότητα ρολογιού τα $10MHz$, ενώ στην δεύτερη τα $12.5MHz$. Οι συχνότητες αυτές δεν είναι τυχαίες καθότι είναι κοινές συχνότητες ρολογιού με αυτές που υλοποιήθηκε η γεννήτρια του RLM στο FPGA και η επιλογή τους θα αναλύθεί στο επόμενο κεφάλαιο.

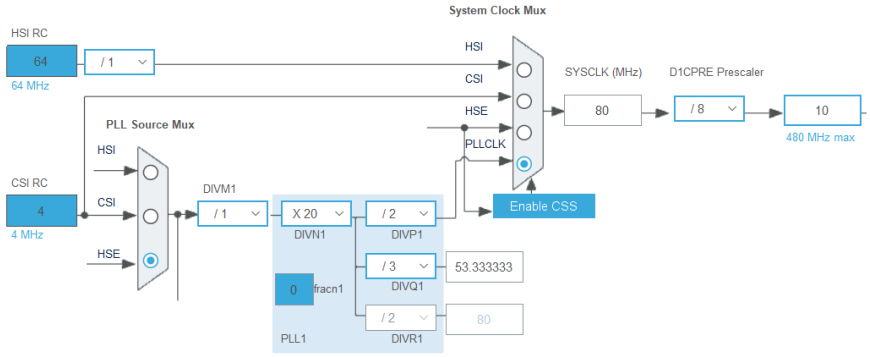
$$\begin{aligned} f1_{clock} &= 10MHz \\ f2_{clock} &= 12.5MHz \end{aligned} \tag{4.1}$$

.

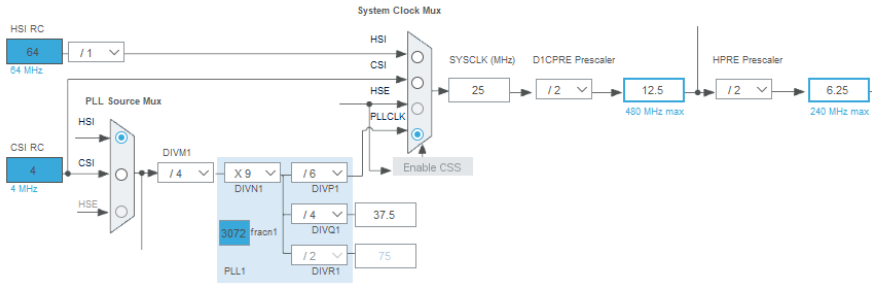
Στην εικόνα 4.3 απεικονίζεται η ρύθμιση της συχνότητας του επεξεργαστή. Αξίζει να σημειωθεί ότι ο τρόπος ρύθμισης της συχνότητας του επεξεργαστή, βασίζεται στην λειτουργία το εσωτερικού ρολογιού

του $H743ZI$ HSI(High Speed Integrated), το οποίο οδηγείται από το εσωτερικό κύκλωμα PLL(Phase Lock Loop) για την τελική ρύθμιση της συχνότητας ρολογιού. Η συνάρτηση της συχνότητας ρολογιού όπως φαίνεται και στην εικόνα4.3 είναι η παρακάτω:

$$
\begin{aligned}
sysclock &= \frac{(\frac{f_{HSI} \cdot divm1}{divn1})}{divp1} \Rightarrow \\
sysclock_1 &= 80MHz \\
sysclock_2 &= 25MHz
\end{aligned}
\tag{4.2}
$$



Ρύθμιση στα $10MHz$



Ρύθμιση στα $12.5MHz$

Εικ. 4.3: Ρύθμιση ρολογιού για συχνότητες $10MHz$ και $12.5MHz$ αντίστοιχα.

4.2 Πρωτόκολλα Επικοινωνίας

Για την επικοινωνία του ενσωματωμένου που θα χρησιμοποιηθεί για την κρυπτογράφηση και την αποστολή δεδομένων σε πραγματικό χρόνο, υλοποιήθηκε ένα ασύγχρονο πρωτόκολλο επικοινωνίας(UART). Η λειτουργία του, καθώς και το τι προσφέρει η σύγχρονη και ασύγχρονη λειτουργία θα περιγραφεί αναλυτικά παρακάτω.

4.2.1 Πρωτόκολλο **UART**

Το πρωτόκολλο που υλοποιήθηκε, αλλά και χρησιμοποιήθηκε σαν περιφερειακό στο μικροελεγκτή είναι αυτό της UART(Universal Asynchronous receiver transmitter). Με την χρήση της UART δομήθηκε όλο το κομμάτι του unit testing, καθώς χρησιμοποιήθηκε τόσο στην λειτουργία του μικροελεγκτή για την αποσφαλμάτωση(debugging) σε πραγματικό χρόνο, όσο και την αποστολή των δεδομένων και την αποθήκευση στους σε txt format για την χρήση της σουίτας NIST.

Το UART είναι ένα πρωτόκολλο επικοινωνίας που χρησιμοποιεί ασύγχρονη σειριακή επικοινωνία με ρυθμιζόμενη ταχύτητα. Ασύγχρονη σημαίνει ότι δεν υπάρχει σήμα ρολογιού για να συγχρονιστούν τα bits εξόδου από τη συσκευή εκπομπής(tx) προς τη συσκευή λήψης(rx)[25].

Η διεπαφή UART δεν χρησιμοποιεί ρολόι(clk) για να συγχρονίσει τις συσκευές πομπού και δέκτη. Αντί για σήμα ρολογιού, ο πομπός παράγει μια ροή bits βασισμένη στο σήμα του ρολογιού του, ενώ ο δέκτης χρησιμοποιεί το εσωτερικό του σήμα ρολογιού για να δειγματοληπτεί τα εισερχόμενα δεδομένα. Ο συγχρονισμός επιτυγχάνεται έχοντας την ίδια ταχύτητα μετάδοσης (baud rate) και στις δύο συσκευές. Η πιθανότητα αποτυχίας μπορεί να επηρεάσει τον χρονισμό της αποστολής και λήψης των δεδομένων, προκαλώντας διαφορές κατά τον χειρισμό των δεδομένων. Η επιτρεπτή διαφορά στην ταχύτητα μετάδοσης είναι έως 10% πριν ο χρονισμός των bits γίνει πολύ αποκλίνον.

Στην εικόνα 4.4 απεικονίζεται η διαμόρφωση του πακέτου αποστολής του πρωτοκόλλου UART όπως αυτό περιγράφεται στο datasheet του ”Pena”[25].

Εικ. 4.4: Πακέτο αποστολή UART

Η διαμόρφωση του πακέτου περιλαμβάνει τα εξής:

- Start Bit: Η γραμμή μετάδοσης δεδομένων UART κανονικά κρατιέται σε υψηλό δυναμικό όταν δεν μεταδίδει δεδομένα. Για να ξεκινήσει η μεταφορά δεδομένων, το UART εκπομπής τραβά τη γραμμή μετάδοσης από υψηλή σε χαμηλή τάση για έναν (1) κύκλο ρολογιού. Όταν το UART λήψης ανιχνεύσει τη μετάβαση από υψηλή σε χαμηλή τάση, αρχίζει να διαβάζει τα bits στο πλαίσιο δεδομένων με τη συχνότητα του ρυθμού μετάδοσης (baud rate).
- Data Frame: Το πλαίσιο δεδομένων περιέχει τα πραγματικά δεδομένα που μεταφέρονται. Μπορεί να είναι από πέντε (5) bits έως οκτώ (8) bits αν χρησιμοποιείται bit ισοτιμίας. Αν δεν χρησιμοποιείται bit ισοτιμίας, το πλαίσιο δεδομένων μπορεί να είναι εννέα (9) bits. Στις περισσότερες περιπτώσεις, τα δεδομένα αποστέλλονται με το λιγότερο σημαντικό bit πρώτα.
- Parity: Η ισοτιμία(parity) περιγράφει την ισότητα ή ανισότητα ενός αριθμού. Το bit ισοτιμίας είναι ένας τρόπος για την UART λήψης να διαπιστώσει αν κάποια δεδομένα έχουν αλλάξει κατά τη διάρκεια

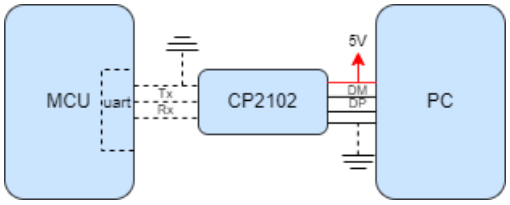
της μετάδοσης. Τα bits μπορούν να αλλάξουν λόγω ηλεκτρομαγνητικής ακτινοβολίας, μη συμβατών ρυθμών μετάδοσης ή μεταδόσεων δεδομένων μεγάλων αποστάσεων.

Όταν το bit ισοτιμίας(parity) αντιστοιχεί με τα δεδομένα, το UART γνωρίζει ότι η μετάδοση ήταν χωρίς σφάλματα. Αλλά αν το bit ισοτιμίας είναι 0 και το σύνολο είναι περιττό, ή αν το bit ισοτιμίας είναι 1 και το σύνολο είναι άρτιο, το UART γνωρίζει ότι τα bits στο πλαίσιο δεδομένων έχουν αλλάξει.

- Stop Bits: Για να σημάνει το τέλος του πακέτου δεδομένων, η UART αποστολής οδηγεί τη γραμμή μετάδοσης δεδομένων από χαμηλή σε υψηλή τάση για (1) έως δύο (2) bit(s) διάρκειας.

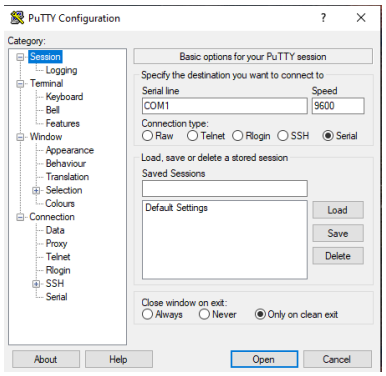
4.2.2 Σύλληψη Δεδομένων σε **PC**

Για την σύλληψη των δεδομένων από τον μικροελεγκτή και την αποθήκευση τους στο υπολογιστή χρησιμοποιήθηκε μια γέφυρα (UART-TO-USB). Το ολοκληρωμένο της silicon labs CP2102[20]. Στην εικόνα 4.5 απεικονίζεται το μπλοκ διάγραμμα εφαρμογής του ολοκληρωμένου CP2102 για την αποστολή δεδομένων μέσω UART. Για την αποστολή δεδομένων το μόνο που χρειάζεται από την μεριά του



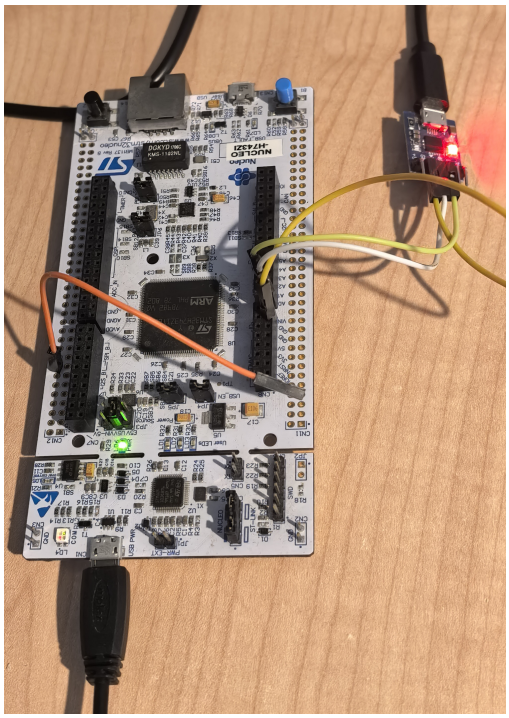
Εικ. 4.5: Μπλοκ διάγραμμα χρήσης cp2102

μικροελεγκτή είναι 3 αγωγοί, 2 για την αποστολή και λήψη δεδομένων (tx/rx) και ένας για την γείωση. Το μόνο που χρειάζεται συμπληρωματικά για την αποθήκευση σε txt είναι ένα περιβάλλον για σειριακή ανάγνωση και αποθήκευση. Στην παρούσα εργασία λόγο απλότητας επιλέχθηκε το puTTY, ένας προσομοιωτής σειριακής κονσόλας ανοιχτού κώδικα. Στην εικόνα 4.6 απεικονίζεται το περιβάλλον puTTY.



Εικ. 4.6: Περιβάλλον puTTY

Στην εικόνα 4.7 αντίστοιχα φαίνεται η πραγματική υλοποίηση του συστήματος με την χρήση του nucleo-743zi και του cp2102. Με χρήση του πρωτοκόλλου UART σχεδιάστηκε και ένας δοκιμαστικός



Εικ. 4.7: Συνδεσμολογία nucleo-743zi με usb-to-ttl

driver, ώστε ο χρήστης να μπορεί μέσω uart commands να αλλάξει την λειτουργία του μικροελεγκτή και να μπορεί είτε να λάβει πακέτα σχετικά με την παραγωγή bits, είτε να ζητήσει πακέτα για την υλοποίηση του διαγράμματος διακλάδωσης.

4.3 Αρχιτεκτονική Υλικολογισμικού(firmware)

Καίρια στοιχεία για την επίτευξη του στόχου της διπλωματικής, να αποδείξει δηλαδή την κρισιμότητα, αλλά και την αποτελεσματικότητα της υλοποίησης των χαοτικών συστημάτων κρυπτογράφησης σε FPGA, ήταν να μελετηθεί η απόδοση του συστήματος υπό διαφορετικές συχνότητες ρολογίου, όμοιες με αυτές του FPGA, ώστε να μπορεί να γίνει μια σύγκριση ως προς την κατανάλωση και την ταχύτητα(benchmarking). Ο ψευδοκώδικας του αλγορίθμου 1 παρουσιάζει τον τρόπο με τον οποίο σχεδιάστηκε η εφαρμογή για την μελέτη της απόδοσης του μικροελεγκτή.

Αναλυτικά ο αλγόριθμος ακολουθεί τα εξής βήματα:

- Προ-φόρτωση μεταβλητών που θα χρησιμοποιηθούν για τον υπολογισμό της εξόδου του λογιστικού χάρτη με κάποιες αρχικές συνθήκες. Για x_0 ορίζεται το 0.1 και για r το 4
- Η συνάρτηση περιμένει από το χρήστη μια μεταβλητή που αφορά το πλήθος των επαναλήψεων. Η τιμή που δόθηκε για την εκτίμηση της απόδοσης ήταν 10^6.

Algorithm 1 vRLM24 Algorithm

procedure vRLM24(itr)
    $i \leftarrow 0$
    $bit \leftarrow 0$
    $txbfr \leftarrow$ array of 4 bytes
    $x \leftarrow X_0$ ▷ Initial seed value
    $r \leftarrow R$ ▷ Parameter for logistic function
    while $i < itr$ **do**
        $x \leftarrow$ logistic(x, r)
        $bit \leftarrow x \mod 2$ ▷ Bit production
        $txbfr \leftarrow$ format$(bit, "\%d\backslash r\backslash n")$
        HAL_UART_Transmit(huart2, $txbfr$, length of txbfr, 100)
        $i \leftarrow i + 1$
    end while
end procedure

- Στην συνέχεια η συνάρτηση επαναληπτικά παράγει τιμές ίσες με των αριθμό των επαναλήψεων και τις αποστέλλει με την χρήση του πρωτοκόλλου UART στο υπολογιστή.

4.3.1 Αποτελέσματα στα $10MHz$

Για την μέτρηση της απόδοσης του συστήματος ένας timer μετρούσε την έναρξη της εκτέλεσης της γεννήτριας καθώς και πότε σταματάει να παράγει τιμές και μετρούσε την συνολική διαφορά χρόνου. Γνωρίζοντας τις συνολικές επαναλήψεις της γεννήτριας και τον χρόνου που χρειάστηκε για να τις εκτελέσει μπορούμε εύκολα να υπολογίσουμε το throughput του μικροελεγκτή.

Με την χρήση του Live expressions στο eclipse μετρήθηκε την ώρα που σταματάει να παράγει η γεννήτρια, ο αρχικός και τελικός χρόνος. Τα αποτελέσματα απεικονίζονται στην εικόνα 4.8

(x)= start	uint32_t	1
(x)= tPassed	uint32_t	11126

Εικ. 4.8: Διάταξη half adder

Παρατηρούμε ότι ο συνολικός χρόνος για 1.000.000 bit ήταν $11.125msec$, καθότι η συνάρτηση που χρησιμοποιήθηκε για την σύλληψη του χρόνου είναι η ενσωματωμένη συνάρτηση της HAL(HAL_GetTick). Το συνολικό throughput υπολογίσθηκε στα:

$$T = \frac{10^6}{11.125 * 10^{-3}} \Rightarrow$$
$$T \approx 89Kbps \tag{4.3}$$

Το ρεύμα που μετρήθηκε κατά την διάρκεια της παραγωγής των bit ήταν:

$$I_{10MHz} \approx 8.5mA \tag{4.4}$$

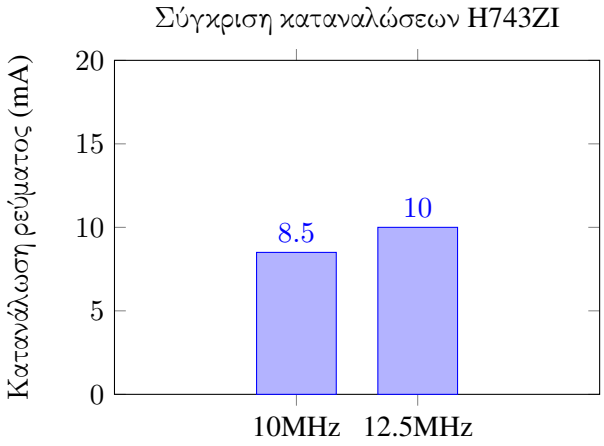
4.3.2 Αποτελέσματα στα $12.5MHz$

Αλλάζοντας την συχνότητα ρολογιού στα $12.5MHz$ ο συνολικός χρόνος για παραγωγή 1.000.000 bit ήταν $7684msec$. Το συνολικό throughput υπολογίζεται:

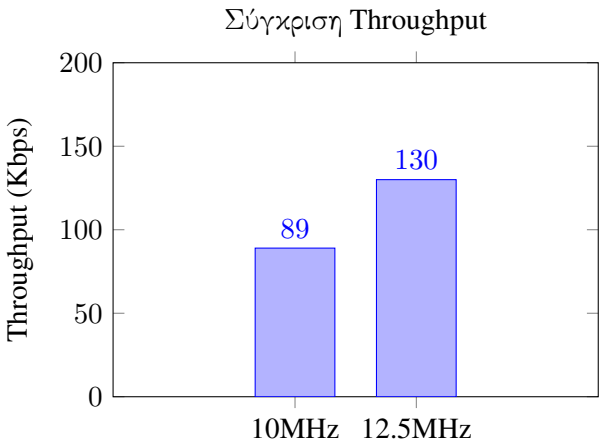
$$\begin{aligned} T &= \frac{10^6}{7684 * 10^{-3}} \Rightarrow \\ T &\approx 130Kbps \end{aligned} \tag{4.5}$$

Το ρεύμα που μετρήθηκε κατά την διάρκεια της παραγωγής των bit ήταν:

$$I_{12.5MHz} \approx 10mA \tag{4.6}$$



Εικ. 4.9: Διάγραμμα κατανάλωσης ρεύματος μικροελεγκτών



Εικ. 4.10: Throughput διαφορετικών συχνοτήτων

4.4 Συμπεράσματα

Συνολικά δοκιμάστηκαν δύο συχνότητες ρολογιού για το αναπτυξιακό της ST. Οι δυο συχνότητες όπως προείπαμε ήταν δεσμευτικές από την μεριά του FPGA με τις αιτίες να αναφέρονται στο κεφάλαιο 5. Παρατηρούμε ότι οι μόνες επιπτώσεις της αλλαγής της συχνότητας ρολογιού σχετίζονται με την αύξηση, τόσο του throughput, όσο και με την αύξηση της κατανάλωσης ρεύματος του μικροελεγκτή. Η ίδια η συχνότητα δεν φαίνεται να επηρεάζει τα αποτελέσματα της εξόδου, αλλά ούτε και την τυχαιότητα της γεννήτριας.

4.5 Επίλογος

Στο κεφάλαιο αυτό αναλύθηκε η αρχικοποίηση των drivers που ήταν απαραίτητη για την υλοποίηση της χαοτικής κρυπτογεννήτριας, καθώς επίσης και το hardware που ήταν απαραίτητο για την μετάδοση των παραγόμενων bit της γεννήτριας. Στην συνέχεια συγκρίθηκαν και παρουσιάστηκαν τα πειραματικά αποτελέσματα από τις μετρήσεις που έγιναν με την χρήση της αποσφαλμάτωσης(debug). Στο επόμενο κεφάλαιο θα αναλυθεί και θα παρουσιαστεί η σχεδίαση της γεννήτριας RLM με την χρήση fpga , καθώς και όλες οι επιμέρους διατάξεις που την απαρτίζουν.

5 Αρχιτεκτονική κώδικα για προσομοίωση σε **FPGA**

Σε αυτό το κεφάλαιο θα αναλυθεί όλη η μεθοδολογία της ανάπτυξης μιας ψευδοτυχαίας γεννήτριας, που βασίζει την λειτουργία της στην λογιστική συνάρτηση, της οποίας η έξοδος ορίζεται ως εξής:

$$x_{n+1} = rx_n(1 - x_n) \tag{5.1}$$

Θα δοθούν αναλυτικές οδηγίες αλλά και προσομοιώσεις λειτουργίας των επιμέρους ψηφιακών κυκλωμάτων, που θα αποτελέσουν λειτουργικές οντότητες της γεννήτριας, καθώς και θα συγκρίνουμε τα αποτελέσματα τυχαιότητας της εξόδου της γεννήτριας, σε σχέση με τα αποτελέσματα της υλοποίησης της γεννήτριας σε ενσωματωμένο σύστημα που περιγράφεται παραπάνω. Η διπλωματική εργασία σε επίπεδο ανάπτυξης σε FPGA δεν πραγματεύεται την βέλτιστη ψηφιακή σχεδίαση, αλλά μια σχεδίαση τέτοια που εξασφαλίζει τα ακόλουθα:

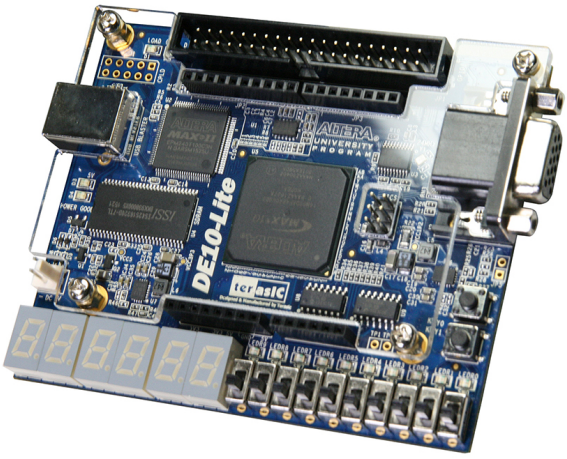
- Ένα λειτουργικό πυρήνα ικανό να επιτελέσει αλγεβρικές πράξεις όπως(πρόσθεση, αφαίρεση, πολλαπλασιασμό, ακέραια διαίρεση), για τις ανάγκες του υπολογισμού της εξόδου του χαοτικού χάρτη.
- Διασύνδεση του ψηφιακού υλικού της γεννήτριας και υλοποίηση αλγορίθμου συμμετρικής κρυπτογράφησης με χρήση *block ciphering*.
- IC ικανό να εκτελέσει όλες τις παραπάνω λειτουργίες, με σκοπό την ασφαλή κρυπτογράφηση και αποστολή δεδομένων.

5.1 Αναπτυξιακό **altera DE-10 lite**

Για την ανάπτυξη των ψηφιακών διατάξεων που κρίθηκαν απαραίτητες για την λειτουργικότητα της γεννήτριας τυχαίων αριθμών, καθώς και του πρωτοκόλλου επικοινωνίας, χρησιμοποιήθηκε το αναπτυξιακό $Terasic\ DE10 - Lite$ που ενσωματώνει στο πυρήνα του το FPGA $Altera\ MAX\ 10$. Πρόκειται για ένα αναπτυξιακό με πολύ καλή σχέση κόστους-απόδοσης. Τα χαρακτηριστικά του δίνονται παρακάτω:

- Ενσωματωμένοι ADCs, κάθε ADC υποστηρίζει 1 αφιερωμένη αναλογική είσοδο και 8 διπλούς λειτουργικούς ακροδέκτες.
- 50K προγραμματιζόμενα στοιχεία λογικής
- Μνήμη M9K: 1,638 Kbit
- 144 Πολ/τες 18 × 18
- 4 PLLs
- Ενσωματωμένος USB Blaster (USB τύπου B)
- Συσκευή Μνήμης: 64MB SDRAM, x16 bits data bus
- Αισθητήρας: Αισθητήρας Επιτάχυνσης

- 10 LEDs
- 10 διακόπτες συρόμενοι
- 2 κουμπιά πίεσης
- Έξι οθόνες 7-Segment



Εικ. 5.1: Αναπτυξιακό terasic DE10-LITE

Πρόκειται για ένα αναπτυξιακό με αρκετές λογικές μονάδες ικανές να υλοποιήσουν ακόμα και πολύ σύνθετα ψηφιακά κυκλώματα. Στην εικόνα 5.1 απεικονίζεται η τοπολογία του αναπτυξιακού της terasic. Ο κώδικας που γράφτηκε σε γλώσσα VHDL για το συγκεκριμένο αναπτυξιακό είχε ως σκοπό την δοκιμή της γεννήτριας RLM σε διαφορετικές συχνότητες λειτουργίας.

5.2 Απαραίτητες Ψηφιακές διατάξεις

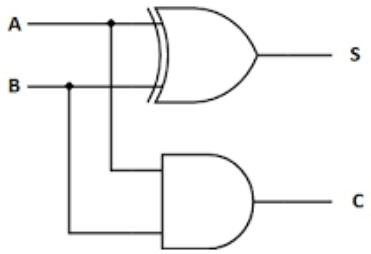
Σε αυτό το υποκεφάλαιο θα αναλύσουμε όλα τα επιμέρους ψηφιακά κυκλώματα με χρήση της γλώσσας VHDL, αλλά και την λογική πίσω από την σχεδίαση των διατάξεων που αποτελούν την ψηφιακή γεννήτρια τυχαίων αριθμών.

Κύκλωμα αθροιστή(**Adder**)

Οι πιο κρίσιμες ψηφιακές διατάξεις που σχεδιάστηκαν πρώτα είναι αυτές του

- Πλήρους αθροιστή(full addder)
- Ημίσιου αθροιστή(half adder)

Οι διατάξεις των αθροιστών είναι τόσο απαραίτητες για την αυτή καθεαυτή πράξη της πρόσθεσης, όσο και για την διάταξη του πολλαπλασιαστή, καθώς πρόκειται για πρόσθεση μερικών αθροισμάτων. Στην παρούσα διπλωματική εργασία, η διάταξη του ημίσιου αθροιστή θα χρησιμοποιηθεί ως μέσω βελτιστοποίησης της διάταξης του πολλαπλασιαστή. Η ψηφιακή διάταξη του ημίσιου αθροιστή απεικονίζεται παρακάτω.



Εικ. 5.2: Διάταξη half adder

Όπως παρατηρείται πρόκειται για μια απλή διάταξη δύο ψηφιακών πυλών με δυο εξόδους. Με το S συμβολίζεται το άθροισμα και με το C το κρατούμενο. Η ψηφιακή εξίσωση του κυκλώματος δίνεται ως:

$$\begin{aligned} S &= A \oplus B \\ C &= A \cdot B \end{aligned} \tag{5.2}$$

Το κύκλωμα του ημίσιου αθροιστή δεν λαμβάνει υπόψιν το κρατούμενο στον υπολογισμό του αθροίσματος αλλά το υπολογίζει. Στην εικόνα 5.3 δίνεται ο πίνακας αληθείας του κυκλώματος του ημίσιου αθροιστή.

Input	**Input**	**Sum**	**Carry**
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Εικ. 5.3: Πίνακας αληθείας(truth table) ημίσιου αθροιστή.

	B	
A	0	0
	0	1

	B	
A	0	1
	1	0

Εικ. 5.4: Πίνακες karnaugh a)Αριστερά ο πίνακας karnaugh του Carry b)Δεξιά ο πίνακας Karnaugh του Sum

Από τους πίνακες karnaugh καταλήγουμε στις εξισώσεις:

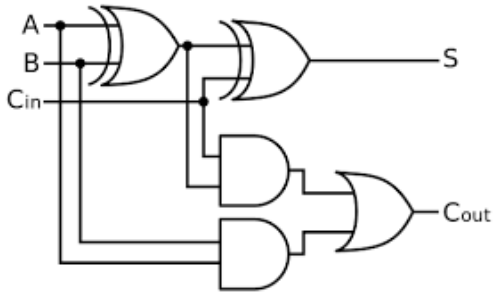
$$\begin{aligned} C &= A \cdot B \\ S &= \bar{A}B + A\bar{B} \Rightarrow \\ S &= A \oplus B \end{aligned} \tag{5.3}$$

Όπως βλέπουμε η εξίσωση 5.3 που προκύπτει από τους πίνακες karnaugh επιβεβαιώνει και την εξίσωση 5.2. Στην συνέχεια του κεφαλαίου θα αναλύσουμε και την ψηφιακή διάταξη του πλήρους αθροιστή που θα αποτελέσει κύρια διάταξη στην υλοποίηση της διάταξης MAC(mathematical accumulator).

Πλήρης αθροιστής

Η διάταξη του πλήρους αθροιστή είναι από της πιο κομβικές, όχι μόνο στην παρούσα διπλωματική αλλά και γενικότερα στο κομμάτι της σχεδίασης ψηφιακών κυκλωμάτων. Στην παρούσα εργασία κρίνεται απαραίτητη τόσο για την αυτοτελή πράξη της πρόσθεσης, όσο και για την άθροιση μερικών παραγόντων.

Σε αυτό το κεφάλαιο αναλύεται η διάταξη, ο πίνακας αληθείας, καθώς και οι πίνακες Karnaugh του πλήρη αθροιστή. Το ψηφιακό κύκλωμα του πλήρη αθροιστή παρουσιάζεται στην εικόνα 5.5.



Εικ. 5.5: Διάταξη πλήρους αθροιστή(full adder).

Το ψηφιακό κύκλωμα του πλήρους αθροιστή λαμβάνει υπόψιν τον υπολογισμό του carry bit για τον υπολογισμό του αθροίσματος. Για να κατανοήσουμε τον λόγο που το συγκεκριμένο κύκλωμα αποτελεί έναν πλήρη αθροιστή θα ξεκινήσουμε με την χρήση ενός παραδείγματος πρόσθεσης δύο αριθμών.

$$\begin{array}{r} 1 \\ +1 \\ \hline \end{array} \tag{5.4}$$

Χρησιμοποιώντας το δυαδικό ανάλογο της εξίσωσης 5.4 έχουμε την εξίσωση 5.5

$$\begin{array}{r} 01 \\ +01 \\ \hline \end{array} \tag{5.5}$$

Το αποτέλεσμα της ψηφιακής πρόσθεσης πρέπει:

- Όταν και τα δύο bit εισόδου είναι 1, αλλά και το $C_{in} = 0$, το $S = 0$,το $C_{out} = 1$.
- Όταν και τα δύο bit εισόδου είναι 1, αλλά και το $C_{in} = 1$ και τα δύο bit εξόδου το $S, C_{out} = 1$.
- Όταν και τα δύο bit εισόδου είναι 0, αλλά και το $C_{in} = 0$ και τα δύο bit εξόδου $S, C_{out} = 0$.
- Όταν και τα δύο bit εισόδου είναι 0, αλλά και το $C_{in} = 0$ και τα δύο bit εξόδου $S = 1$, και το $C_{out} = 0$.
- Όταν το ένα bit εισόδου είναι 0 και το δεύτερο 1, αλλά και το $C_{in} = 0$, τότε το $S = 1$ και το $C_{out} = 0$.
- Όταν το ένα bit εισόδου είναι 0 και το δεύτερο 1, αλλά και το $C_{in} = 1$, τότε το $S = 0$ και το $C_{out} = 1$.

Έχοντας τα παραπάνω υπόψιν οι αντίστοιχοι πίνακες Karnaugh διαμορφώνονται όπως φαίνεται στην εικόνα 5.6.

A \ BC_{IN}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

A \ BC_{IN}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Εικ. 5.6: Πίνακες karnaugh α) Αριστερά ο πίνακας του Sum β) Ο πίνακας του carry

Οι εξισώσεις που προκύπτουν από τους πίνακες karnaugh είναι οι παρακάτω:

$$\begin{aligned} S &= A\bar{B}\bar{C_{in}} + \bar{A}\bar{B}C_{in} + ABC_{in} + \bar{A}\bar{C_{in}} \Rightarrow \\ S &= C_{in}(AB + \bar{A}\bar{B}) + \bar{C_{in}}(\bar{A}B + A\bar{B}) \Rightarrow \\ S &= ... \\ S &= C_{in} \oplus (A \oplus B) \end{aligned} \tag{5.6}$$

Αντίστοιχα η εξίσωση της ψηφιακής εξόδου του C_{out} είναι:

$$\begin{aligned} C_{out} &= AB + C_{in}(A\bar{B} + \bar{A}B) \Rightarrow \\ C_{out} &= ... \Rightarrow \\ C_{out} &= AB \oplus AC_{IN} \oplus BC_{in} \Rightarrow \end{aligned} \tag{5.7}$$

Οι δύο παραπάνω εξισώσεις αποτελούν ουσιαστικά και την ψηφιακή διάταξη του αθροιστή που απεικονίζεται την εικόνα 5.5

Πολλαπλασιαστής

Η διάταξη του πολλαπλασιαστή είναι κομβικής σημασίας στην διάταξη της διπλωματικής καθότι αποτελεί το μεγαλύτερο μέρος της υλοποίησης, σε όρους λογικών μονάδων. Στην παρούσα διπλωματική δεν υλοποιήθηκε διάταξη πολλαπλασιαστή με τους ενσωματωμένους $18 \cdot 18Bit$ του DSP, καθότι πρόκειται για μια εργασία που εξετάζει και τα αποτελέσματα του συστήματος παραγωγής τυχαίων αριθμών από άποψη ταχύτητας και λογικών μονάδων. Στην παρούσα εργασία προτιμήθηκε για λόγους απλότητας μια απλή διάταξη πολλαπλασιαστή που υπολογίζει το γινόμενο με άθροισμα μερικών παραγόντων.

Προτού παρουσιαστεί το κύκλωμα του πολλαπλασιαστή θα αναλύσουμε όπως και στο προηγούμενο παράδειγμα του πλήρους αθροιστή, το ζητούμενο για την υλοποίηση του. Στην εξίσωση 5.8 απεικονίζεται ένα απλό παράδειγμα πολλαπλασιασμού σε δυαδική μορφή

$$\begin{array}{rrrr} & & & 10 \\ & & & \times 01 \\ \hline & & 1\cdot 0 & 0\cdot 1 \\ & +0\cdot 1 & 0\cdot 0 & \\ \hline P3 & P2 & P1 & P0 \end{array} \tag{5.8}$$

Το αποτέλεσμα του πολλαπλασιασμού έχει μήκος σε bit $Prod\mid_{bits} = Multiplier\mid_{bits} \cdot Multiplicand\mid_{bits}$. Οι πράξη του πολλαπλασιασμού ανά bit αντιπροσωπεύεται με μια πύλη and. Ο πολλαπλασιαστής της διπλωματικής είναι μήκους $32bit$. Αυτό σημαίνει ότι παίρνει εισόδους μήκους $32bit$ και το αποτέλεσμα του είναι μήκους $64bit$.

Για τον υπολογισμό των λογικών μονάδων που θα χρειαστεί να υπολογίσουμε :

- Για τους μερικούς παράγοντες ότι χρειαζόμαστε 5 πύλες για τον πλήρη αθροιστή επί των αριθμό των μερικών παραγόντων.
- Για την πράξη του πολλαπλασιασμού απαιτείται μια πύλη and ανά bitwise πολλαπλασιασμό.

$$LU\mid_{and} = n \cdot m \tag{5.9}$$

  Όπου m, n είναι το μήκος σε bits του πολλαπλασιαστή και του πολλαπλασιαστέου. Στην περίπτωση του $32x32bit$ πολλαπλασιαστή της διπλωματικής, οι λογικές μονάδες που απαιτούνται είναι: $LU = 1024$.

- Οι πλήρεις αθροιστές που απαιτούνται για την υλοποίηση της διάταξης που θα προσθέτει τα

αθροίσματα μερικών παραγόντων του πολλαπλασιαστή δίνονται από την εξίσωση 5.10.

$$\begin{aligned} LU\ |_{FA} &= \frac{n(n+1)}{2} \Rightarrow \\ LU\ |_{FA} &= \frac{31(31+1)}{2} \Rightarrow \\ LU\ |_{FA} &= 496 \end{aligned} \tag{5.10}$$

- Τέλος χρειαζόμαστε ένα ημιαθροιστή για κάθε bit.

Συνολικά η μονάδα MAC θα χρειαστεί :

$$LU = LU\ |_{and} + LU\ |_{FA} + LU\ |_{HA} = 1551 \tag{5.11}$$

Για να υπολογίσουμε τα συνολικά LU's που θα χρειαστεί το IC που θα είναι υπεύθυνο για την προσομοίωση της συμπεριφοράς του λογιστικού χάρτη, αρκεί να αναλύσουμε το σύνολο των πράξεων που απαιτούνται και το μήκος των αριθμών που θα χρησιμοποιηθεί. Όπως αναφέραμε η εξίσωση της λογιστικού χάρτη αποτελείται από 2 πολλαπλασιασμούς και μια αφαίρεση. Μια αυστηρή εκτίμηση για τις λογικές μονάδες θα ήταν η παρακάτω:

$$LU\ |_{logisticmap} = 2 \cdot LU\ |_{MAC} + LU\ |_{FD} \tag{5.12}$$

Όπου $LU\ |_{FD}$ εννοούμε τις λογικές μονάδες που είναι απαραίτητες για την αφαίρεση δύο $32bit$ αριθμοί. Συνεπώς το σύνολο που εκτιμήθηκε ότι είναι απαραίτητο για την σχεδίαση του πολλαπλασιαστή είναι :

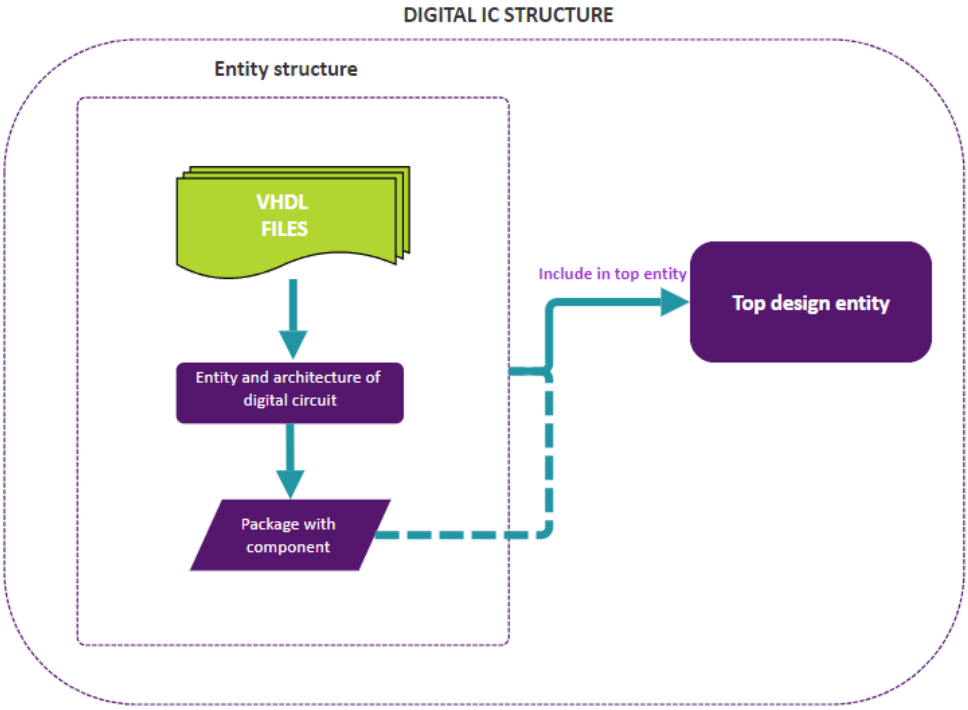
$$LU\ |_{logisticmap} = 3134 \tag{5.13}$$

Στην συνέχεια του κεφαλαίου θα μελετήσουμε τις επιμέρους διατάξεις που αναλύθηκαν καθώς και την υλοποίηση τους σε γλώσσα VHDL.

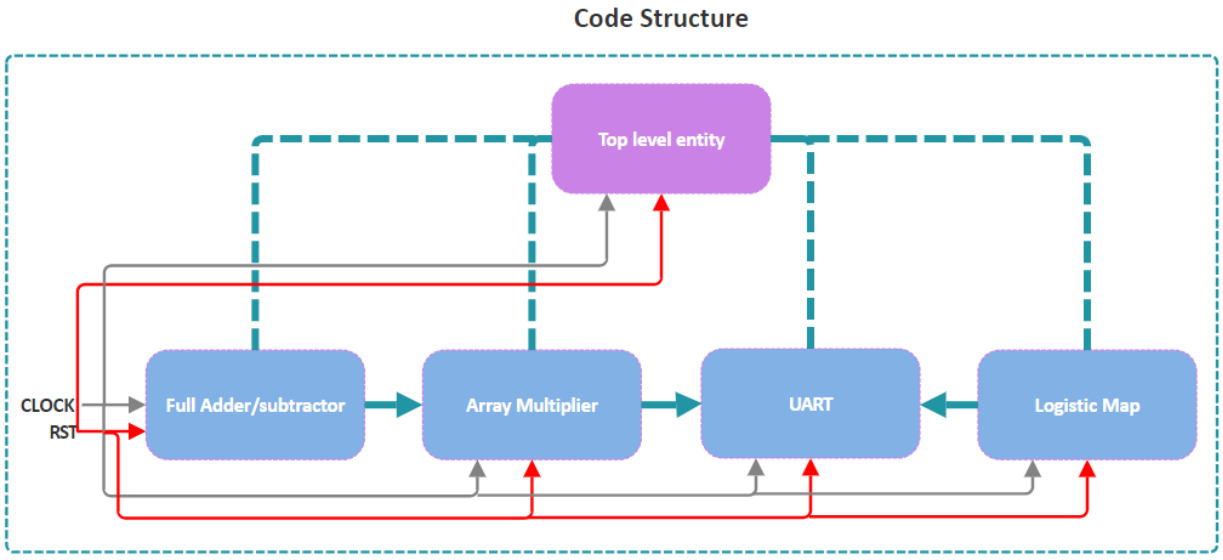
5.3 Αρχιτεκτονική κώδικα VHDL

Οι επιμέρους ψηφιακές διατάξεις που αναλύθηκαν στο προηγούμενο υποκεφάλαιο υλοποιήθηκαν σε γλώσσα VHDL και προσομοιώθηκαν στο περιβάλλον quartus prime 18.1. Η δομή που επιλέχθηκε για να κριθεί εφικτή η κάθε επιμέρους ψηφιακή σχεδίαση, είναι κάθε αρχείο vhdl να περιλαμβάνει και μια από της ψηφιακές οντότητες(entity). Το κάθε αρχείο περιλαμβάνει επίσης και την αρχιτεκτονική(architecture) της κάθε ψηφιακής διάταξης και με την δυνατότητα που προσφέρει η εντολή package δημιουργούμε ένα component το οποίο κληρονομείται στο top design entity. Η γραφική αναπαράσταση φαίνεται στην εικόνα 5.7.

Όπως παρατηρείται και από την εικόνα όλες οι επιμέρους υλοποιήσεις κληρονομούν το design τους στο top level entity. Η δομή του κώδικα σε VHDL απεικονίζεται στην εικόνα 5.7. Στην εικόνα 5.8 απεικονίζεται δευτερογενώς ο τρόπος που κληρονομούνται όλα τα design στο top level design.



Εικ. 5.7: Διάγραμμα απεικόνισης της δομής του κώδικα ως προς το top design entity.



Εικ. 5.8: Διάγραμμα απεικόνισης κληρονομικότητας

5.3.1 Ripple Carry Adder

Ο Ripple Carry Adder (RCA) είναι μία ψηφιακή διάταξη προσθέτη που χρησιμοποιείται για την πρόσθεση δυαδικών αριθμών. Συγκεκριμένα:

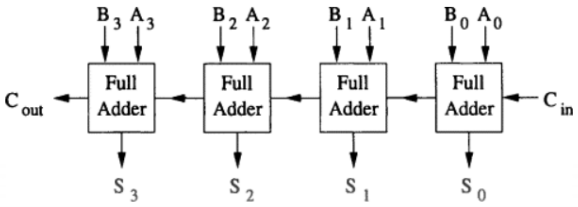
- Λειτουργία: Αποτελείται από μια αλυσίδα πλήρων αθροιστών (full adders), όπου το κρατούμενο

ψηφίο (carry) από τον έναν πλήρη αθροιστή μεταδίδεται στον επόμενο.

- Χαρακτηριστικά: Η απλότητα της διάταξης την καθιστά εύκολη στην υλοποίηση, αλλά η σειριακή μεταφορά του κρατούμενου ψηφίου (carry propagation) μπορεί να οδηγήσει σε σημαντικές καθυστερήσεις, ειδικά σε περιπτώσεις μεγάλου αριθμού bit.
- Εφαρμογές: Είναι ιδανική για εφαρμογές όπου η απλότητα και το χαμηλό κόστος υλοποίησης υπερτερούν της ανάγκης για υψηλή ταχύτητα.

5.3.2 Σχεδίαση RCA & προσομοίωση RTL

Κατά την διάρκεια υλοποίησης της διάταξης, που θα είναι υπεύθυνη τόσο για την πράξη της πρόσθεσης, όσο και για το άθροισμα μερικών παραγόντων του πολλαπλασιαστή, επιλέχθηκε μια απλή διάταξη RCA(ripple carry adder). Πρόκειται για μια διάταξη που υλοποιείται από n πλήθος πληροί αθροιστών, όπου n το πλήθος των bit που πρόκειται να προστεθούν. Στην εικόνα 5.9 απεικονίζεται η διάταξη ενός αθροιστή RCA $4-bit$[5].



Εικ. 5.9: Μπλοκ διάγραμμα αθροιστή rca 4-bit.

Στην περίπτωση του αθροιστή rca ισχύει ότι:

$$\begin{aligned} S_i &= A_i + B_i \\ C_{i+1} &= (A_i \cdot B_i) + (A_i \cdot C_i) + (B_i \cdot C_i) \end{aligned} \tag{5.14}$$

Στην παρούσα διπλωματική όλες οι διατάξεις έχουν σχεδιαστεί με γνώμονα τον υπολογισμό μη-προσημασμένων ακέραιων αριθμών (unsigned). Ο κώδικας VHDL για την υλοποίηση του πλήρη αθροιστή παρατίθεται στην εικόνα 5.10. Η υλοποίηση της κάθε διάταξης μεγαλύτερης κλίμακας(πολλ/στής,λογιστικός χάρτης) γίνεται, όπως έχει αναφερθεί και παραπάνω, με την χρήση του *component* της VHDL και γίνεται η σύνδεση μέσω του port map.

Για την υλοποίηση της εξίσωσης 5.15 υλοποιήθηκε ένας $32-bit$ RCA αθροιστής ικανός για πρόσθεση δύο $32-bit$ αριθμών :

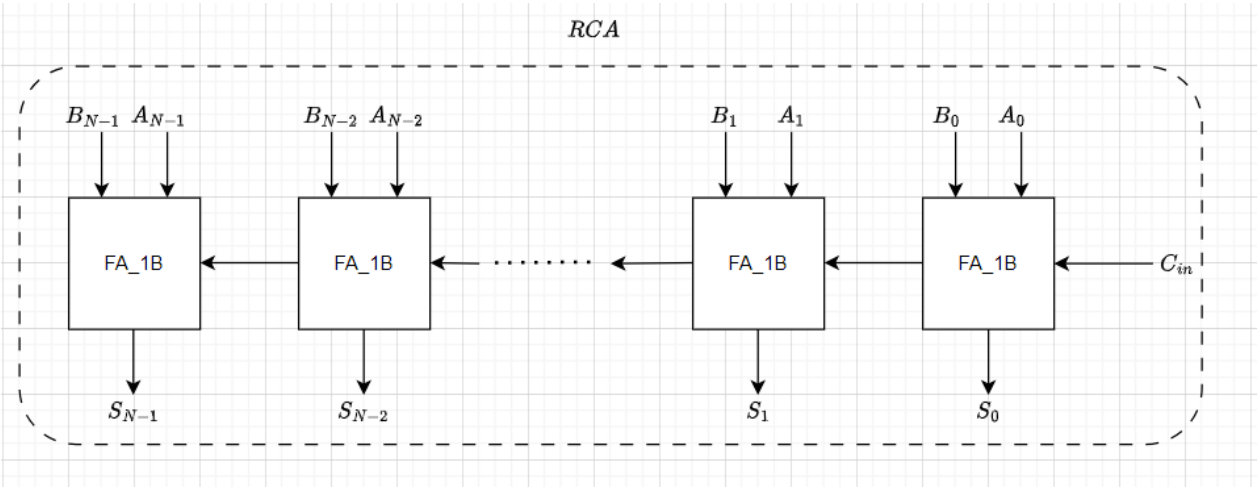
$$\sum_{0 \le i \le n-1} (A_i + B_i) = unsigned(S_i) \tag{5.15}$$

Η διάταξη του $32-bit$ αθροιστή παρουσιάζεται σε μορφή μπλοκ διαγράμματος στην εικόνα 5.11.

Σημαντικό για το αποτέλεσμα της ταχύτητας του ολοκληρωμένου αποτελεί το γεγονός ότι η διάταξη

```
entity FA_1B is
   port( A            : in  std_logic;
         B            : in  std_logic;
         C_in         : in  std_logic;
         C_out        : out std_logic;
         S            : out std_logic);
end FA_1B;

architecture dataflow of FA_1B is
   begin
   C_out <=(A AND B) OR (A AND C_in) OR (B AND C_in);
   S  <= A XOR B XOR C_in;
end dataflow;
```

Εικ. 5.10: Υλοποίηση πλήρους αθροιστή με την χρήση VHDL.



Εικ. 5.11: Μπλοκ διάγραμμα διάταξης $32bit$ αθροιστή.

του αθροιστή, καθώς και του πολλαπλασιαστή, έχουν αρκετά μεγάλη καθυστέρηση διάδοσης(propagation delay), καθότι το κρατούμενο(C_i) κληρονομείται από αθροιστή σε αθροιστή. Γνωρίζοντας ότι το κύκλωμα που διαχειρίζεται το κρατούμενο σε ένα πλήρη αθροιστή εξαρτάται τόσο από το ρυθμό ανόδου(rise time) τριών πυλών, το συνολικό propagation delay υπολογίζεται ως εξής:

$$t_{pd} = N \cdot (3 * t_g), \tag{5.16}$$

όπου t_g είναι ο απαραίτητος χρόνος για να φτάσει η έξοδος κάθε πύλης στο 90% της τάσης εξόδου, συνθήκη ικανή για την εκτίμηση της εξόδου και οδήγηση του επόμενου με ασφάλεια.

Κάτι που θα αναφερθεί και παρακάτω είναι ότι η υλοποίηση του πυρήνα του FPGA που επιλέχθηκε για την υλοποίηση της διπλωματικής είναι TSMC $55nm$. Ο λόγος που αναφέρεται έχει να κάνει με τους χρόνους ανόδου της κάθε λογικής μονάδας. Αν και τα rise time δεν αναφέρονται συνήθως σε κάποιο datasheet, οι εταιρίες προσφέρουν λογισμικό χρονικής απόκρισης των διατάξεων που υλοποιούνται με κάποια γλώσσα HDL. Μια εκτίμηση που γίνεται για τον χρόνο ανόδου σε κυκλώματα RC είναι η παρακάτω:

$$\begin{aligned} \tau = RC = \frac{1}{2 * \pi * f_c} \\ V_t = V_0(1 - e^{-t/\tau})) \end{aligned} \tag{5.17}$$

Από την επίλυση των δύο εξισώσεων της 5.17 προκύπτει ότι για την μετάβαση που απαιτείται $t_r\mid_{10\%-90\%}$ η σχέση που προκύπτει από την 5.17 είναι:

$$t_r\mid_{10\%-90\%}= 2.2\tau = 2.2RC \tag{5.18}$$

Αν και η εξίσωση 5.18 επ'ουδενί δεν ανταναχλά τον ακριβή τρόπο μέτρησης του χρόνου ανόδου σε μια λογική μονάδα ενός FPGA, μας δίνει μια πολύ καλή εικόνα για την συμπεριφορά μιας διάταξης που παρουσιάζει μεταβατικά φαινόμενα, όπως αυτή μια λογικής πύλης που στον πυρήνα της απαρτίζεται από ένα αριθμό CMOS τρανζίστορ. Όπως γίνεται αντιληπτό, μεγάλη είναι η συσχέτιση της καθυστέρησης διάδοσης με την χωρητικότητα της διάταξης. Στην περίπτωση των 55nm ενδεικτικά αναφέρεται στην βιβλιογραφία [29] ότι η χωρητικότητα της πύλης ενός CMOS στα 55nm, μετά από δοκιμές με worst, αλλά και best case scenario ως προς την εφαρμογή της διαφοράς δυναμικού στα άκρα του, προκύπτει $G_{gg} \approx 30fF$. Οι δοκιμές του περιοδικού αυτού έγιναν για διάταξη NMOS με μεταβλητό μήκος πύλης από $0.05\mu m$ μέχρι $0.06\mu m$. Καταλληλότερο μοντέλο εκτίμησης του rise time αναφέρεται διεξοδικά στην βιβλιογραφία [11].

Με μια ελαστική εκτίμηση ότι στα 55nm η χωρητικότητά μιας διάταξης βρίσκεται κοντά στις $3-4$ δεκάδες fF και με τεχνολογία LVMOS που χρησιμοποιείται στο fpga της intel [4], γνωρίζοντας ότι η συχνότητα λειτουργίας μας βρίσκεται στα δεκάδες MHz καθώς και ότι η αγωγιμότητα g_m είναι $2-3$ δεκάδες mS[29], εκτιμάται ότι ο χρόνος ανόδου, θα είναι κοντά στο ns. Αυτή η εκτίμηση δεν αντικατοπτρίζει τον πραγματικό χρόνο ανόδου αλλά είναι δείγμα που πλησιάζει βάση της βιβλιογραφίας την τάξη μεγέθους της καθυστέρησης διάδοσης που θα αντιμετωπίσει η διάταξη μας. Οι πραγματικές μετρήσεις θα απεικονιστούν παρακάτω καθότι πάρθηκαν μέσω της χρήσης του εργαλείου intel time analyzer.

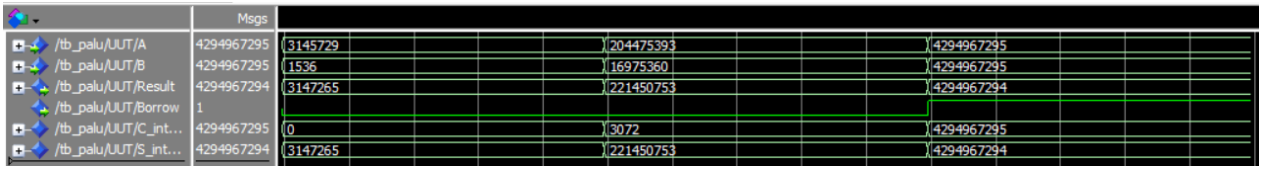
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.adder.all;

entity FA_32B is
  port(
  A        : in  std_logic_vector(31 downto 0);
  B        : in  std_logic_vector(31 downto 0);
  C_in   : in  std_logic_vector(31 downto 0);
  Result   : out std_logic_vector(31 downto 0);
  Borrow   : out std_logic_vector(31 downto 0));
end FA_32B;

architecture full_add_32b of FA_32B is
  signal C_internal :std_logic_vector (31 downto 0);
  signal S_internal :std_logic_vector (31 downto 0);

  begin

     gen_sub: for i in 0 to 31 generate
        FS:FA_1B port map
        (
           A=>A(i),
           B=>B(i),
           C_in=>C_in(i),
           c_out=>C_internal(i),
           S=>S_internal(i)
        );
     end generate;

  Result<=S_internal;
  Borrow<=C_internal;
end full_add_32b;
```

Εικ. 5.12: Υλοποίηση διάταξης RCA $32bit$ με χρήση vhdl.

Στην περίπτωση της διάταξης του RCA-32Bbit που υλοποιείται στην εργασία η καθυστέρηση μετάδοσης βάση της εκτίμησης θα είναι στις δεκάδες ns. Ο κώδικας VHDL του RCA απεικονίζεται στην εικόνα 5.12.

RTL Προσομοίωση

Ο ακρογωνιαίος λίθος κάθε διάταξης που πρέπει να ελεγχθεί πριν την υλοποίησή της σε πραγματικό χρόνο, είναι το design verification, δηλαδή η διαδικασία αυτή που προσομοιώνει την συμπεριφορά της διάταξης σε επίπεδο προσομοίωσης, ώστε να συμφωνεί με το μοντέλο που θέλουμε να δουλέψει. Το εργαλείο λογισμικού που χρησιμοποιήθηκε για verification είναι το ModelSim 10.5b.



Εικ. 5.13: Verification $RCA\ 32bit$ με χρήση RTL σε περιβάλλον ModelSim 10.5b.

Τα αποτελέσματα προσομοίωσης του κυκλώματος άθροισης με χρήση RTL παρουσιάζονται στην εικόνα 5.13. Για τον έλεγχο των επιμέρους κυκλωμάτων σχεδιάστηκαν UUT(unit under test) πειραματικά κυκλώματα για τον έλεγχο κάθε σεναρίου. Όπως στην εικόνα 5.13 γίνονται τρεις δοκιμές με διαφορετικά (α,β) με σκοπό τον έλεγχο του overflow, όπου στον τρίτο έλεγχο το $borrow = 1$. Οι έλεγχοι έγιναν με χρήση $C_{in}(0) = 0$.

5.3.3 Διάταξη MAC

Η Multiply-Accumulate (MAC) είναι μια κρίσιμη ψηφιακή διάταξη, ιδιαίτερα σημαντική στον τομέα της επεξεργασίας ψηφιακών σημάτων(DSP):

- Λειτουργία: Εκτελεί δύο βασικές λειτουργίες σε ένα κύκλο μηχανής: πολλαπλασιασμό δύο αριθμών και πρόσθεση του αποτελέσματος σε έναν συσσωρευτή (accumulator).
- Χαρακτηριστικά: Ο συνδυασμός αυτών των δύο λειτουργιών σε μία ενιαία διάταξη επιτρέπει την ταχεία και αποδοτική εκτέλεση επαναλαμβανόμενων αλγορίθμων, όπως αυτοί που χρησιμοποιούνται σε φίλτρα, επεξεργασία εικόνας, και νευρωνικά δίκτυα.
- Εφαρμογές: Είναι το βασικό δομικό στοιχείο σε πολλές εφαρμογές ψηφιακής επεξεργασίας, όπου απαιτείται υψηλή απόδοση σε λειτουργίες πολλαπλασιασμού και συσσώρευσης.

5.3.4 Σχεδίαση MAC & προσομοίωση RTL

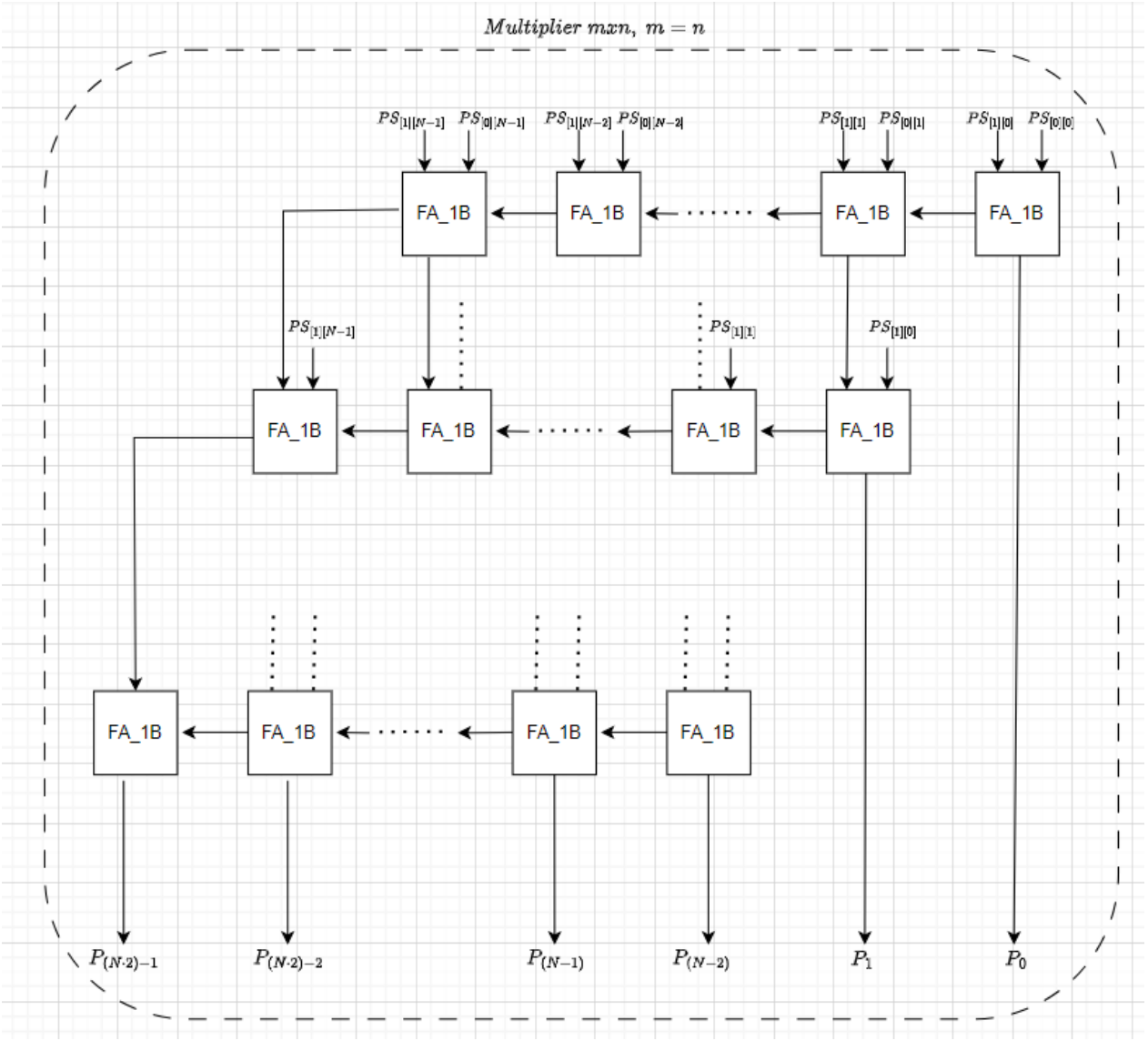
Η διάταξη του MAC που σχεδιάστηκε όπως προαναφέρθηκε έχει δυο στόχους:

- Πολλαπλασιασμός $2x32bit$ αριθμούς.
- Ανάθεση τους σε καταχωρητή, σε ένα κύκλο μηχανής.

Η διάταξη που δοκιμάστηκε για αυτό το σκοπό, αποτελείται από διαδοχικούς αθροιστές RCA για την άθροιση των μερικών παραγόντων. Το μπλοκ διάγραμμα της διάταξης του array multiplier με χρήση rca απεικονίζεται παρακάτω. Η διάταξη όπως παρατηρούμαι, χρησιμοποιεί με την βοήθεια του component της VHDL, ένα πίνακα με $\frac{n\cdot(n+1)}{2}$ πλήρεις αθροιστές. Θα μπορούσαμε ωστόσο να πούμε ότι χρησιμοποιεί $(n-1)\ RCA\ |_{32bit}$ αθροιστές.

Τα σήματα που αποτυπώνονται με PS είναι τα σήματα που προκύπτουν μετά από την πράξη AND μεταξύ των nxm bit των σημάτων εισόδου.

$$PS[n:0][m:0] = A[m:0]B[n:0] \tag{5.19}$$



Εικ. 5.14: Απλοποιημένο διάγραμμα υλοποίησης array multiplier

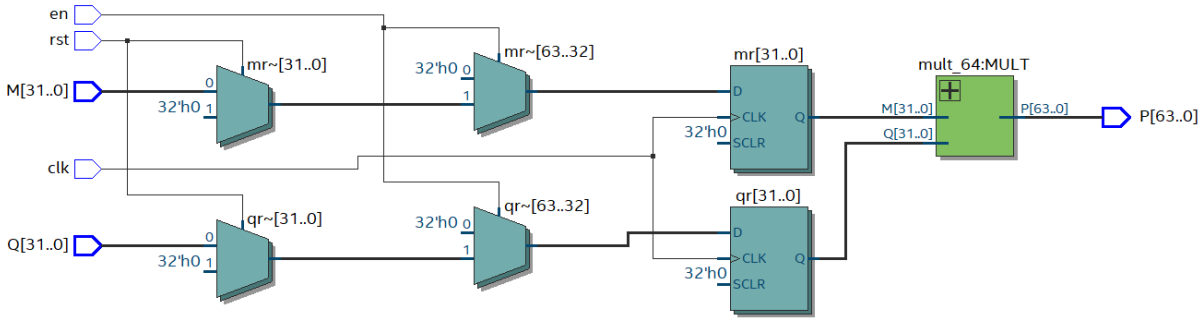
Όπως παρατηρούμε το κρατούμενο διαδίδεται στην διάταξη μέσω κάθε αθροιστή. Εκτός από την

καθυστέρηση που προκύπτει λόγο κάθε 32bit διάταξης αθροιστή, προστίθεται σε αυτό και άλλη μια καθυστέρηση η οποία είναι το carry του $n-1$ πλήρους αθροιστή της κάθε διάταξης RCA, με το τελικό αποτέλεσμα που προκύπτει για την καθυστέρηση διάδοσης με βάση την εξίσωση 5.20 :

$$t_{pd} = (M+N) \cdot (3 \cdot t_g) \tag{5.20}$$

Αυτή η καθυστέρηση θα επηρεάσει την ταχύτητα της εφαρμογής, που είναι με την σειρά της άρρηκτα συνδεδεμένη με την ταχύτητα του ρολογιού. Τα αποτελέσματα για τα path delays, καθώς και η κατανάλωση της διάταξης θα αναλυθούν στο τέλος της ενότητας.

Στην εικόνα 5.15 παρουσιάζεται η μορφολογία της διάταξης στο RTL Viewer μετά από το compile της διάταξης. Την διάταξη του πολλαπλασιαστή συμπληρώνουν και τρία ακόμα σήματα εισόδου λογικής

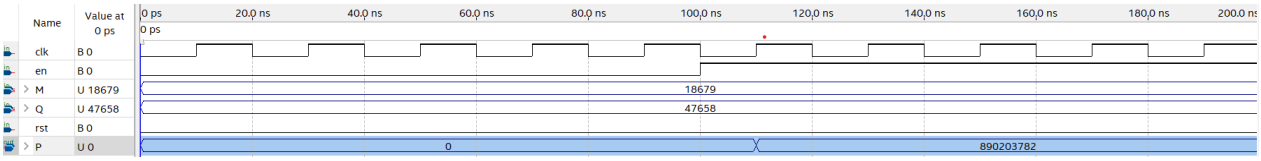


Εικ. 5.15: Υλοποίηση του πολλαπλασιαστή στο RTL Viewer

std_logic. Τα σήματα αυτά διέπουν και την λειτουργία του.

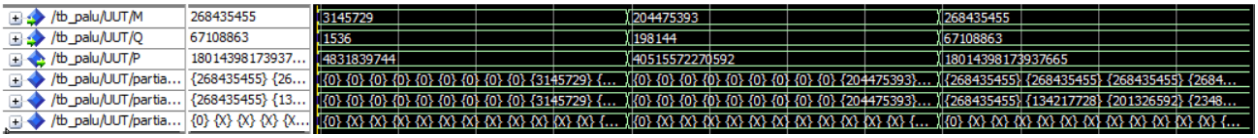
- **RST:** Πρόκειται για το σήμα reset με σκοπό την επαναφορά των register σε χαμηλό δυναμικό για την επανεκκίνηση της λειτουργίας.
- **CLK:** Απαραίτητη προϋπόθεση για τον συγχρονισμό της εφαρμογής, αλλά ταυτόχρονα κρίσιμο για την μελέτη της σχεδίασης με την χρήση του time analyzer.
- **EN:** Το I/O που ενεργοποιεί την λειτουργία της διάταξης. Χωρίς αυτό οι καταχωρητές που παρέχουν την έξοδο τους στον πολλαπλασιαστή είναι μανδαλωμένοι στο λογικό $'0'$ για χαμηλότερη κατανάλωση.

Για την εξακρίβωση της σωστής λειτουργίας υλοποιήθηκε σε πρώτο βαθμό μια προσομοίωση με χρήση vwf αρχείου και στην συνέχεια σχεδιάστηκε η υλοποίηση του UUT για τον πολλαπλασιαστή. Στο αρχείο vwf απεικονίζεται ο τρόπος λειτουργίας που αναφέρεται παραπάνω. Με την χρήση της process που χειρίζεται την ανάθεση ανά παλμό ρολογιού υλοποιείται εν κατακλείδι και η λειτουργία του MAC, που είναι απαραίτητη για να έχουμε μια πράξη ανά κύκλο μηχανής. Με τον τρόπο που σχεδιάστηκε ο κώδικας της VHDL, αν έχουμε ενεργοποιημένο σήμα *en* και στην συνέχεια εντοπιστεί θετική παρυφή του σήματος ρολογιού, το MAC πολλαπλασιάζει και αναθέτει σε ένα κύκλο μηχανής τα αποτελέσματα των εισόδων.



Εικ. 5.16: Προσομοίωση λειτουργίας MAC σε vwf αρχείο

Το UUT της διάταξης του πολλαπλασιαστή απεικονίζεται στην εικόνα 5.17. Ανατέθηκαν τρεις περιπτώσεις εισόδων για τον πολλαπλασιαστή για την εξακρίβωση της άρτιας λειτουργίας του σε όλο το εύρος των bit εξόδου. Στην επόμενη υποενότητα θα περιγραφεί η λειτουργία της απλοποιημένης λογιστικής συνάρτησης



Εικ. 5.17: Προσομοίωση του πολλαπλασιαστή στο ModelSim 10.5b

καθώς και ο τρόπος παραγωγής τυχαίων bit στην έξοδο της.

5.3.5 Διάταξη & Σχεδίαση RLMIC

Η διάταξη του RLMIC που σχεδιάστηκε για τους σκοπούς της διπλωματικής αποτελείται από:

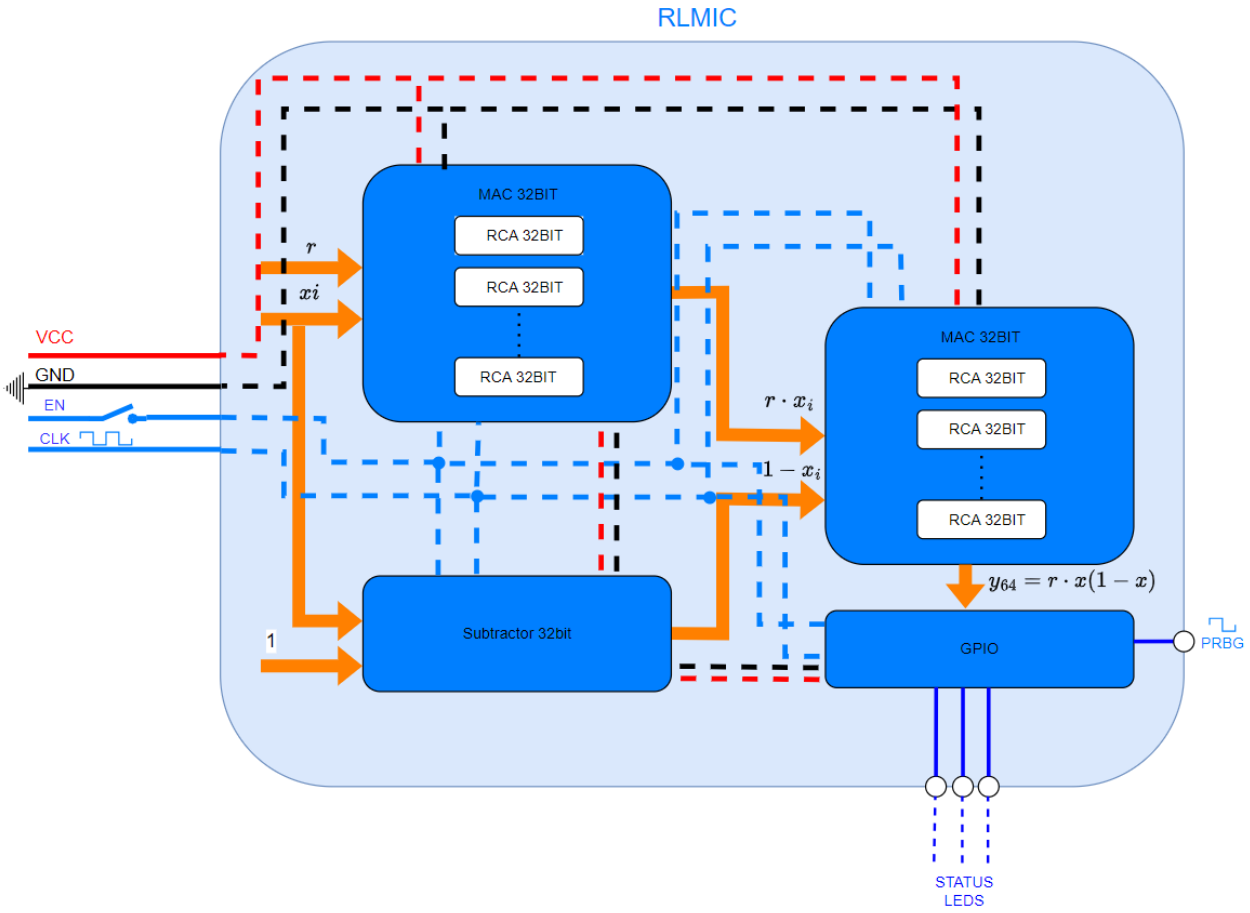
- 2 MAC $32x32$ *bit*.
- 1 Πλήρη αφαιρέτη 32 *bit*.
- 3 IO για την ανάδειξη της κατάστασης του RLM.
- 1 O για την έξοδο της γεννήτριας.

Στην εικόνα 5.18 απεικονίζεται το διάγραμμα της εσωτερικής σχεδίασης του RLMIC για την καλύτερη κατανόηση της διάταξης.

Η μελέτη μη-γραμμικών συστημάτων και η εξακρίβωση του αν ένα δυναμικό σύστημα μπορεί να έχει χαοτική συμπεριφορά, δηλαδή:

1. Μεγάλη εξάρτηση από αρχικές συνθήκες. Ακόμα και μια απειροελάχιστη αλλαγή στην αρχική του κατάσταση να οδηγεί σε τελείως διαφορετική συμπεριφορά.
2. Η χρονοσειρά της εξόδου να παρουσιάζει άπειρες περιόδους.
3. Να διέπεται από αιτιοκρατία(ντετερμινισμό),

είναι ένας πολυσύνθετος και εγγενώς τόσο θεωρητικός, όσο και πειραματικός κλάδος των μαθηματικών. Ωστόσο τα εργαλεία και τα τεστ τυχαιότητας που αναλύθηκαν στα πρώτα κεφάλαια επιμερίζονται τον μεγαλύτερο φόρτο εργασίας, μιας και από τα αποτελέσματα τους κρίνεται το αν ένα σύστημα μπορεί να θεωρηθεί χαοτικό ή όχι.



Εικ. 5.18: Μπλοκ Διάγραμμα διάταξης RLMIC

Σε αυτήν την υποενότητα αναλύεται τόσο το αν το μοντέλο μας είναι χαοτικό, όσο και οι πόροι που δεσμεύτηκαν, καθώς και ο τρόπος λειτουργίας της επιμέρους διάταξης. Το μοντέλο που περιγράφει το σύστημα εξόδου, "βαφτίζεται", λόγω απλοποιημένης, βελτιστοποιημένης αλλά παρόλα αυτά ασφαλής λειτουργίας του, RLM(Reduced Logistic Map), δηλαδή απλοποιημένος λογιστικός χάρτης. Το IC που σχεδιάζεται έχοντας στον πυρήνα του τον τον τρόπο λειτουργίας του RLM ονομάζεται RLMIC.

Με την χρήση της εξίσωσης 5.21 εισάγουμε το νέο μοντέλο που σχεδιάστηκε(RLM).

$$x_{i+1} << 48 = (r << 24) \cdot (x_i << 24)[(1 << 24) - (x_i << 24)] \tag{5.21}$$

Έχοντας μελετήσει την συμπεριφορά της εξόδου γνωρίζουμε ότι για $r \in [3.8, 4]$ το σύστημα παρουσιάζει χαοτική συμπεριφορά με την $x_0 = 0.01$. Επιλέγοντας μια τιμή κοντά στο 4, δηλαδή $r = 3.99$ μπορούμε να

αιτιολογήσουμε τον λόγο που δεν χρειαζόμαστε και τα 64bit εξόδου του πολλαπλασιαστή.

$$\begin{aligned} 0.0 \leq x_i \leq 1.0 \Rightarrow \\ 0.0 \leq (x_i << 24) \leq 2^{24} \Rightarrow \\ 0.0 \leq (r \cdot x_i << 24) \leq 2^{24} \cdot 4 \Rightarrow \end{aligned} \tag{5.22}$$

Για την περίπτωση της αφαίρεσης ισχύει ότι:

$$\begin{aligned} 0 \leq 1 - x_i \leq 1 \Rightarrow \\ 0 \leq ((1 - x_i) << 24) \leq 2^{24} \end{aligned} \tag{5.23}$$

Από τις [5.22,5.23] προκύπτει ότι,

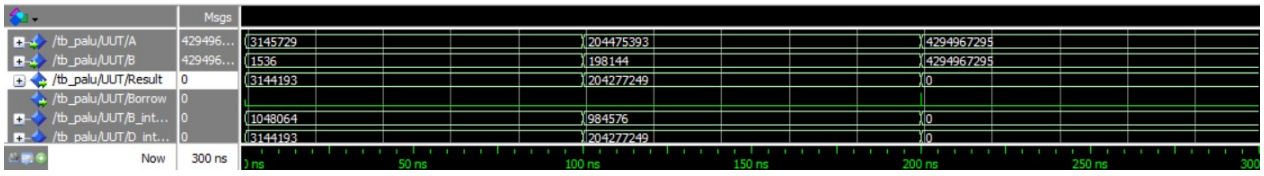
$$0 \leq x_{i+1} \leq 2^{50} \tag{5.24}$$

Με αυτό υπόψιν σχεδιάστηκε και γεννήτρια τυχαίων αριθμών. Πρόκειται για μια γεννήτρια που με την χρήση της ολίσθησης κατά 24 bit καταφέρνει να έχει ακρίβεια εισόδων δέκα δεκαδικών ψηφίων και ακρίβεια εξόδου δεκαπέντε(15) δεκαδικών.

Ένα στοιχείο των ψηφιακών διατάξεων, είναι αυτό της πεπερασμένης ακρίβειας σε bits. Η ακρίβεια αυτή συχνά οδηγεί συστήματα, που η επιθυμητή τους συμπεριφορά είναι χαοτική, να υποκύπτουν σε δυναμικό υποβιβασμό[22, 31]. Η συμπεριφορά αυτή αν και περιγράφεται εκτενώς στην βιβλιογραφία καθώς και πιθανοί τρόποι αντιμετώπισης της, δεν υπάρχει κάποιο θεωρητικό μοντέλο το οποίο περιγράφει σαφώς ορισμένα τον τρόπο με τον οποίο μπορεί να προκύψει, κατά την υπάρχουσα γνώση του συγγραφέα. Ο υποβιβασμός στα χαοτικά συστήματα έχει επιπτώσεις στην δυναμική τους συμπεριφορά καθώς οδηγεί το σύστημα στην περιοδικότητα, κάτι που με την σειρά του επηρεάζει την ασφάλεια του συστήματος. Στην παρούσα διπλωματική τα NIST test σε πειραματικό επίπεδο δεν έδειξαν κάποιον υποβιβασμό σε περιοδικότητα με την χρήση της ακρίβειας που περιγράφεται.

32bit Αφαιρέτης(Subtractor)

Το τελευταίο κομμάτι του παζλ από άποψη αριθμητικής μονάδας επεξεργασίας(ALU), είναι αυτό του αφαιρέτη που χειρίζεται την πράξη $1 - x_i$. Για την υλοποίηση του χρησιμοποιήθηκε μια μονάδα 32bit RBS(ripple borrow subtractor). Η μονάδα αφαίρεσης προσομειώθηκε με την σειρά της με την χρήση του ModelSim 10.5 και τα αποτελέσματά της απεικονίζονται στην εικόνα 5.19. Το UUT που σχεδιάστηκε



Εικ. 5.19: RTL προσομοίωση του 32bit RBS

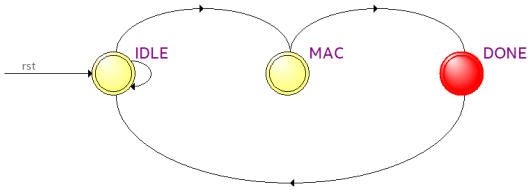
παίρνει τρεις διαφορετικές εισόδους για A,B κατά την διάρκεια της προσομοίωσης για να ελέγξουμε διάφορες καταστάσεις. Στην εικόνα του αρχείο wave του Model Sim απεικονίζονται επίσης και οι εσωτερικοί καταχωρητές. Αν και για το αποτέλεσμα δεν παίζει κάποιον ρόλο για την διαδικασία της αποσφαλμάτωσης είναι πολύ κρίσιμο να έχουμε πλήρη εικόνα και της εσωτερικής λειτουργίας της διάταξης. Αυτός αποτελεί και τον κύριο λόγο προτίμησης του RTL από ότι το VWF.

Μηχανή καταστάσεων**(state machine)**

Απαραίτητη προϋπόθεση για την εύρυθμη λειτουργία του ολοκληρωμένου ήταν η δημιουργία μια μηχανής καταστάσεων που χειρίζεται συνολικά 4 επιμέρους διεργασίες(processes).

- Η πρώτη διεργασία είναι αυτή που χειρίζεται την παραγωγή τιμών μέσω της λογιστικής συνάρτησης και ενημερώνει την μεταβλητή κατάστασης.
- Η δεύτερη διεργασία είναι ένας απλός 64bit απαριθμητής με σκοπό τον καθορισμό από τον χρήστη, των τιμών που θέλουμε να παράγει ο χάρτης.
- Η τρίτη διεργασία αφορά το δεύτερο σκέλος της εργασίας που είναι η παραγωγή τυχαίων αριθμών, η οποία ελέγχεται από το state machine.

Η αναπαράσταση του state machine απεικονίζεται στην εικόνα 5.20



Εικ. 5.20: Μηχανή κατάστασης RLM

Οι καταστάσεις του ολοκληρωμένου είναι τέσσερις και αντικατοπτρίζουν την λειτουργία του ανά πάσα στιγμή. Με την ενεργοποίηση του ολοκληρωμένου είτε με ένα θετικό παλμό στο rst το σύστημα αυτομάτως μπαίνει σε κατάσταση IDLE. Δύο πράγματα αλλάζουν την κατάσταση του συστήματος από IDLE.

- Ο εντοπισμός μιας θετικής παρυφής στο clk.
- Ο εντοπισμός μιας θετικής παρυφής στο clk και ο $iterReg \leq iterations$ που είναι ουσιαστικά ο καταχωρητής που συγκρίνεται με τον αριθμό των iteration για την ολοκλήρωση την εφαρμογής.

Η κατάσταση MAC απευθύνεται στους 2 πολλαπλασιασμούς και στην αφαίρεση για τον υπολογισμό μιας κατάστασης για την έξοδο. Πρόκειται για την ενδιάμεση κατάσταση υπολογισμού του αποτελέσματος του λογιστικού χάρτη. Σε περίπτωση που το state machine έχει φτάσει στην MAC το επόμενο είναι μετά το τέλος του πολλαπλασιασμού να γίνει εναλλαγή κατάστασης σε DONE.

Στην κατάσταση DONE οι άλλες δύο διεργασίες παίρνουν τα ηνία:

Algorithm 2 Iteration Counter Process

```
1:  Process (clk, rst, en)
2:  if rst = '1' or en = '0' then
3:      iterReg ← 0
4:  else if rising_edge(clk) and updCnt = '1' then
5:      if iterReg = iter then
6:          iterReg ← 0
7:      else
8:          iterReg ← iterReg + 1
9:      end if
10: end if
```

Algorithm 3 Bit Generation Process (BIT_GEN)

```
1: Process (clk, rst)
2: if rst = '1' then
3:     prpg ← 0
4: else if rising_edge(clk) then
5:     if updCnt = '1' then
6:         prpg ← xREG(0)
7:     end if
8: end if
```

Εικ. 5.21: Ψευδοκώδικας

- Το process που διαχειρίζεται την αρίθμηση αυξάνει τον counter κατά 1 αν και εφόσον δεν έχει φτάσει στο μέγιστο δυνατό.
- Το process που διαχειρίζεται την παραγωγή τυχαίων τιμών μέσο της εξόδου παράγει μια τιμή εξόδου είτε $'0'$ είτε $'1'$.

Με την παραπάνω διαδικασία το ολοκληρωμένο είναι σε θέση να παράγει ένα τυχαίο bit ανά 3 κύκλους μηχανής(MC). Η συνάρτηση εξόδου του prbg βάση της εξίσωσης 5.21 που συμβολικά ονομάζουμε RLM,

$$bitGen_i = \lfloor RLM \rfloor \mod 2 \tag{5.25}$$

όπου το $bitGen_i$ αναφέρεται στην παραγόμενη τυχαία τιμή ανά $4MC$.

Αποτελέσματα **RLMIC** με χρήση **RTL**

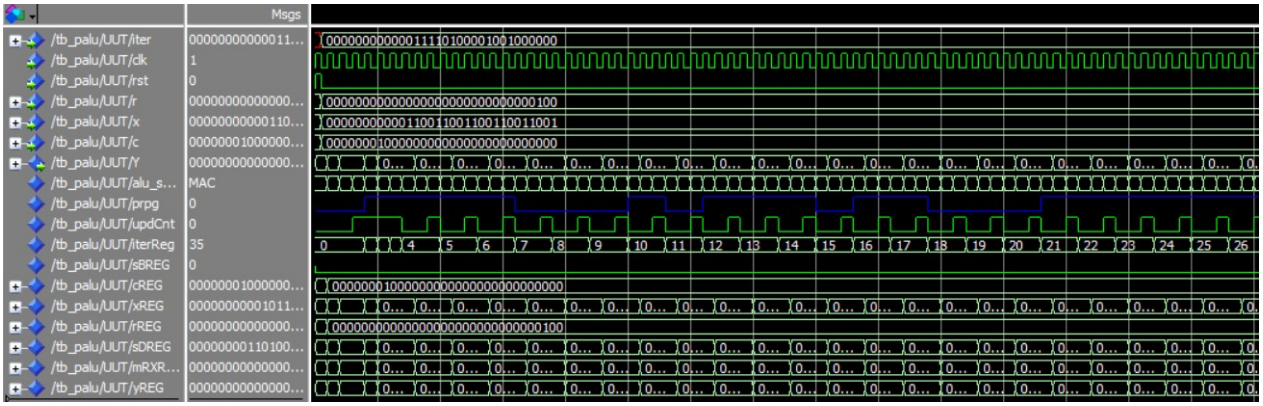
Στην εικόνα 5.21 παρατίθενται δύο ψευδοκώδικες με την λειτουργία των δύο process που χειρίζονται την αρίθμηση, άρα και το μήκος του block και την παραγωγή των bit. Αρχικά για την περίπτωση της διάταξης που χειρίζεται την αρίθμηση ισχύουν τα εξής βασιζόμενα στον αλγόριθμο 2:

- Η διαδικασία λαμβάνει υπόψιν το reset, το enable και το clock.
- Σε περίπτωση που ούτε το σήμα en είναι ενεργοποιημένο ή έρθει θετικό παλμός από το $reset$ θέλουμε να μηδενίζουμε το μετρητή των επαναλήψεων.
- Όσο εντοπίζουμε θετικές παρυφές του σήματος του ρολογιού αλλά και ο καταχωρητής των επαναλήψεων δεν έχει φτάσει τις ορισμένες επαναλήψεις από το $iter$ σήμα, θέλουμε ο μετρητής να αυξάνεται κατά ένα. Σε αντίθετη περίπτωση που έχουμε ολοκληρώσει της επαναλήψεις της γεννήτριας θέλουμε ο καταχωρητής να καθαρίζεται για ξεκινήσει από την αρχή η διαδικασία της σύλληψης των επαναλήψεων.

Όσο για την διαδικασία παραγωγής bit στην έξοδο βάση του αλγορίθμου 3 ισχύει ότι:

- Η διαδικασία λαμβάνει υπόψιν το reset και το clock.
- Σε περίπτωση που το τελευταίο bit του καταχωρητή x είναι μηδέν, αρά το υπόλοιπο της ακέραιας διαίρεσης με το 2 είναι 0, μηδενίζουμε την έξοδο. Σε αντίθετη περίπτωση η έξοδος γίνεται $'1'$. Όλα αυτά υπό την προϋπόθεση ότι έχει ενημερωθεί ο updCnt, πράγμα που σημαίνει ότι η μηχανή καταστάσεων έχει ολοκληρώσει ένα κύκλο.

Τα αποτελέσματα της προσομοίωσης της μονάδας παραγωγής τυχαίων αριθμών απεικονίζονται παρακάτω.



Εικ. 5.22: RTL προσομοίωσης του συστήματος RLM

Τα αποτελέσματα της εικόνας 5.22 χρήζουν ερμηνείας, μιας και πλέον οι καταχωρητές είναι αρκετοί και οι καταστάσεις δυσδιάκριτες.

- Ανά 3 παλμούς ρολογιού έχουμε και ένα θετικό παλμό για τον $updCnt$, υπεύθυνο για την ενημέρωση της alu_state δηλαδή της μεταβλητής κατάστασης που ελέγχει το state machine, καθώς και την ενημέρωση του $iterReg$.
- Ανά 3 παλμούς ρολογιού τελειώνει και ένα iteration του state machine.
- Ο καταχωρητής xREG που είναι και ο μόνος με μεταβλητή τιμή ανανεώνει την τιμή του, ώστε σε κάθε επόμενο κύκλο να χρησιμοποιηθεί για τον επόμενο υπολογισμό τιμής του χαοτικού χάρτη.
- Ανά 3 παλμούς ρολογιού έχουμε και ένα τυχαίο bit στην έξοδο.
- Στην εικόνα 5.22 απεικονίζονται τα πρώτα 24 iteration του ολοκληρωμένου.
- Με μπλε κυματομορφή απεικονίζεται το bitstream του ολοκληρωμένου για τα πρώτα 24 iteration.

Πριν παρουσιάσουμε τα αποτελέσματα του πειραματικού μέρους σε σύστημα FPGA, πρέπει να παρουσιασθούν αποτελέσματα σε επίπεδο προσομοίωσης που έχουν αξία για την υλοποίηση αυτή, αλλά και για την σύγκριση της με την τωρινή βιβλιογραφία καθώς και με την επίτευξη του στόχου της διπλωματικής αλλά και την δέσμευση για μελλοντική βελτίωση και τύπωση ενός ολοκληρωμένου ASIC, για την ασφαλή μεταφορά δεδομένων.

Ανάλυση πόρων και κατανάλωση ισχύος

Τα πρώτα αποτελέσματα που πρέπει να αναλυθούν είναι αυτά των πόρων που καταναλώθηκαν για την υλοποίηση του RLM. Στην εικόνα 5.23 παρουσιάζονται συνοπτικά τα αποτελέσματα μετά το compile της σχεδίασης για κάθε υπομονάδα που απαρτίζει το RLM.

Entity Resources				
	Full Adder 1Bit	RCA 32Bit	MAC 32X32	RLM
LUTs	5	135	2078	3343(7%)
REG	0	64	64	134
I/O's	0	0	2	4

Εικ. 5.23: Δεσμευμένοι πόροι για την υλοποίηση του RLM

Στον πίνακα της εικόνας 5.23 αποτυπώνονται οι μετρήσεις των λογικών μονάδων(LU), καταχωρητών(Reg), εισόδων/εξόδων (I/O) της διάταξης. Η διάταξη αν και δεν έχει καλύψει σε μεγάλο βαθμό την χωρητικότητα σε LU του FPGA, αντιμετωπίζει κάποιες παθογένειες, όχι κρίσιμες για τον σκοπό της διπλωματικής, παρ' αυτά τέτοιες ώστε σε μελλοντικό χρόνο να αναζητείται τρόπος να βελτιωθούν. Όπως θα δούμε και παρακάτω οι παθογένειες αυτές σχετίζονται με το propagation του κρατούμενου στα κυκλώματα άθροισης/αφαίρεσης.

Η συνολική κατανάλωση που μετρήθηκε για την διάταξη της διπλωματικής μέσω του power analyzer της Quartus είναι στα $105.37mW$.

Χρονική απόκριση σχεδίασης

Η διάταξη του RLM προσομοιώθηκε αλλά και δοκιμάστηκε πειραματικά με συχνότητα λειτουργίας,

$$\begin{aligned} F_{CLK} &= 10Mhz \Rightarrow \\ T_{CLK} &= 20ns \Rightarrow \end{aligned} \tag{5.26}$$

Αν και η μέγιστη συχνότητα δειγματοληψίας δεν χρησιμοποιήθηκε, τα αποτελέσματα χωρίς βελτιστοποιήσεις στις διατάξεις που αναλύθηκαν παραπάνω ήταν ικανοποιητικά τόσο από άποψης τυχαιότητας-ασφάλειας, όσο και από άποψης throughput.

Τα αποτελέσματα της χρονικής απόκρισης του RLM παρουσιάζονται στον πίνακα της εικόνας 5.24

Time analysis				
FMAX	HOLD	SLACK	SETUP	MAX PATH DELAY
19.37 MHz	0.326	0.326	48.371	70.2 ns

Εικ. 5.24: Αποτελέσματα χρονικής απόκρισης του διάταξης RLM

Τα αποτελέσματα του πίνακα περιγράφουν τις εξής χρονικές αποκρίσεις:

- **Slack:** Το Slack αναφέρεται στη διαφορά ανάμεσα στον χρόνο που απαιτείται για να φτάσει ένα σήμα από το ένα καταχωρητή (flip-flop), στον άλλο και τον διαθέσιμο χρόνο για αυτό το σήμα.
- **Hold:** Ο Χρόνος Κράτησης είναι ο ελάχιστος χρόνος που πρέπει ένα σήμα να παραμείνει σταθερό στην είσοδο ενός flip-flop μετά την άφιξη του ρολογιού, ώστε να καταγραφεί σωστά το σήμα. Εάν ένα σήμα αλλάξει πολύ γρήγορα (πριν από τη λήξη του χρόνου κράτησης), τότε το flip-flop μπορεί να καταγράψει λάθος τιμή.
- **Fmax:** Το Fmax αναφέρεται στη μέγιστη συχνότητα ρολογιού στην οποία μπορεί να λειτουργήσει ένα ψηφιακό κύκλωμα χωρίς χρονικές παραβιάσεις. Είναι η αντίστροφη του setup time (χρόνος προετοιμασίας) και αποτελεί έναν σημαντικό δείκτη της απόδοσης ενός κυκλώματος.
- **Setup:** Ο Χρόνος Προετοιμασίας είναι ο χρόνος που απαιτείται για να φτάσει ένα σήμα στην είσοδο ενός flip-flop πριν από την ενεργοποίηση του ρολογιού, ώστε να καταγραφεί σωστά η τιμή του σήματος. Αυτός είναι ο ελάχιστος χρόνος που πρέπει να υπάρχει σταθερό το σήμα πριν από το χτύπημα του ρολογιού.

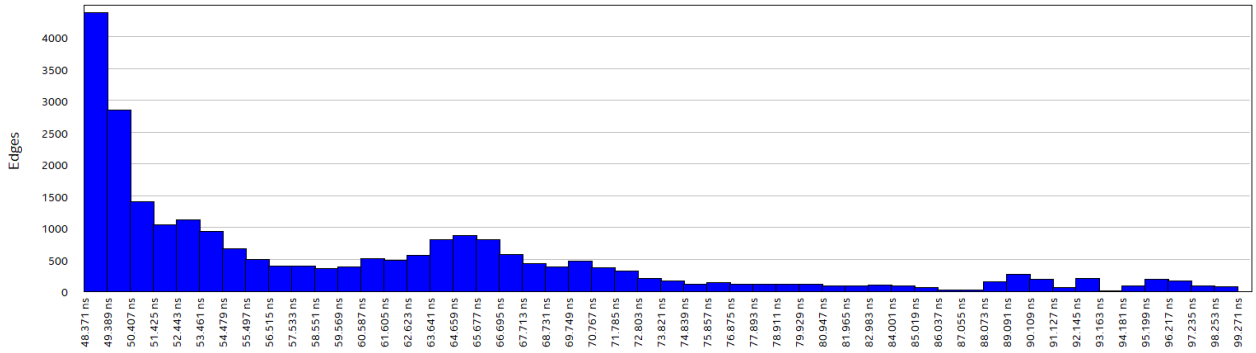
Έχοντας περιγράψει την διαδικασία παραγωγής τυχαίων αριθμών το throughput που προκύπτει από τα παραπάνω δεδομένα υπολογίζεται:

$$\begin{aligned} T_{10Mhz} &= 1bit/3MC \Rightarrow \\ T_{10Mhz} &= 3.33Mbit/s \end{aligned} \tag{5.27}$$

Το παραπάνω throughput είναι το αποτέλεσμα πολλών παραγόντων που δεν λήφθηκαν υπόψιν πριν την σχεδίαση όπως, τεχνολογία του πυρήνα του FPGA, υλοποίηση κυκλωμάτων πρόσθεσης/αφαίρεσης/MAC με χρήση των εσωτερικών DSP(digital signal processing) διατάξεων, αλλά ακόμα και διατάξεων που δεν έχουν παθογένειες όπως το propagation του carry bit, που επιφέρουν μεγάλες καθυστερήσεις στην έξοδο του συστήματος.

Στο κεφάλαιο των μελλοντικών βελτιώσεων αλλά και προτάσεων θα αναφερθούν διεξοδικά τα επόμενα βήματα ανασχεδιασμού της διάταξης για να αποδίδει τα μέγιστα από άποψης χρόνου αλλά και ασφάλειας.

Αξίζει επίσης να αναφερθούμε στα αποτελέσματα που λήφθηκαν μέσω του time analyzer για το ιστόγραμμα των slack. Στο ιστόγραμμα της εικόνας 5.25 απεικονίζονται τα αποτελέσματα.



Εικ. 5.25: Ιστόγραμμα αποτελεσμάτων slack με χρήση 50 διαστημάτων για το μοντέλο slow $1200mV$,$0C$

Το slack που απεικονίζεται στην εικόνα 5.25 έχει τα εξής χαρακτηριστικά:

- Τα διαστήματα που απεικονίζονται με το μπλε χρώμα αφορούν το κατά πόσο έχει καταφέρει η σχεδίαση να ικανοποιήσει τις χρονικές προδιαγραφές βάση της συχνότητας του ρολογιού.
- Σε περίπτωση που κάποιο διάστημα του slack έχει αρνητική τιμή απεικονίζεται με κόκκινο χρώμα και αναφέρεται στο για πόσο χρόνο δεν ικανοποιείται η χρονική προδιαγραφή βάση της συχνότητας λειτουργίας.

5.4 Πειραματικά αποτελέσματα γεννήτριας

5.4.1 Πειραματικά αποτελέσματα γεννήτριας στα $10MHz$

Η εξαγωγή των πειραματικών έγινε με την ενσωμάτωση του κώδικα VHDL στο αναπτυξιακό FPGA της Intel που αναφέρθηκε παραπάνω.

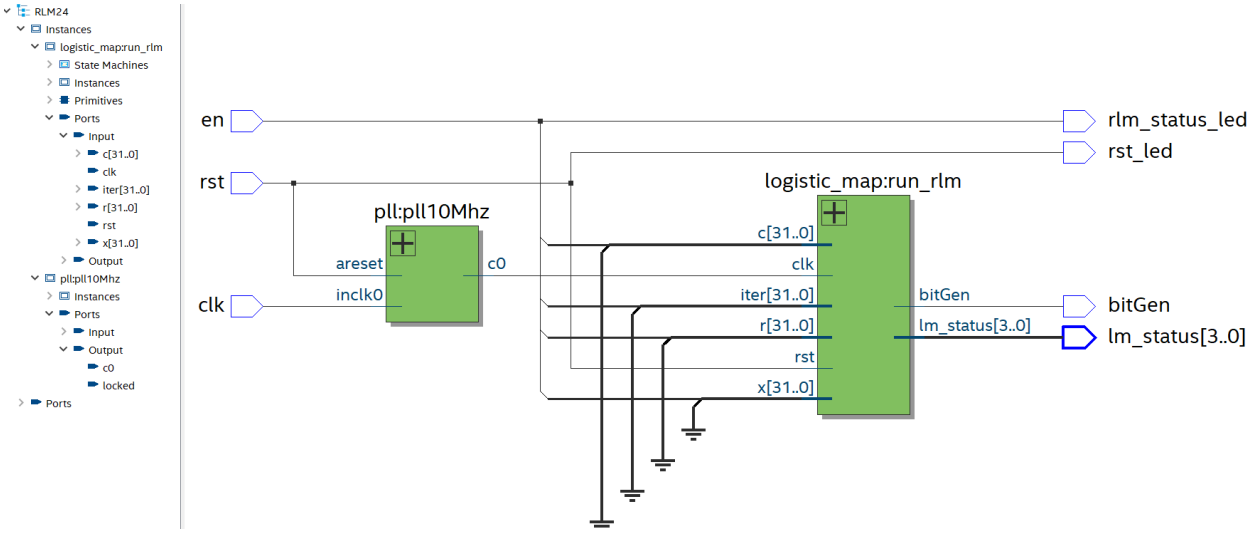
Ο κορμός της εφαρμογής βασίζεται στο γεγονός ότι ένα ρολόι συγχρονίζει όλες τις επιμέρους διατάξεις. Το αναπτυξιακό της terasic διαθέτει ρολόι στα $50Mhz$[30]. Για τον υποβιβασμό της συχνότητας χρησιμοποιήθηκε το κύκλωμα κλειδώματος φάσης PLL(Phased Lock Loop). Η διάταξη αυτή αποτελεί πνευματική ιδιοκτησία IP(intellectual property) της altera(intel). Με τον τρόπο αυτό έγινε εφικτή η επίτευξη ρολογιού χρονισμένου στα $10Mhz$.

Στην εικόνα 5.26 απεικονίζεται η RTL οπτική της διάταξης rlm24.

Για το πειραματικό μέρος της υλοποίησης και για λόγους αποσφαλμάτωσης, χρησιμοποιήθηκαν ορισμένα I/O με μοναδικό σκοπό την οπτική αποσφαλμάτωση.

- 2 I/O που ενεργοποιούνται από το $en(rlm\_status\_led)$ και το $rst(rst\_led)$.
- 4 I/O με σκοπό την αποσφαλμάτωση του state machine (lm_status).

Το επόμενο βήμα για την επίτευξη του προγραμματισμού είναι η ανάθεση του κάθε σήματος στο εκάστοτε pin του αναπτυξιακού. Αυτό επιτυγχάνεται μέσω του pin planner της quartus. Ο πίνακας με τις αναθέσεις των σημάτων απεικονίζεται παρακάτω.



Εικ. 5.26: RTL οπτική της διάταξης RLM που χρονίζεται με χρήση PLL

bitGen	Output	PIN_AA2	3	B3_N0	PIN_AA2	2.5 V		12mA ...ault)	2 (default)	
clk	Input	PIN_P11	3	B3_N0	PIN_P11	2.5 V		12mA ...ault)		
en	Input	PIN_F15	7	B7_N0	PIN_F15	2.5 V		12mA ...ault)		
lm_status[3]	Output	PIN_B11	7	B7_N0	PIN_B11	2.5 V		12mA ...ault)	2 (default)	
lm_status[2]	Output	PIN_A11	7	B7_N0	PIN_A11	2.5 V		12mA ...ault)	2 (default)	
lm_status[1]	Output	PIN_B10	7	B7_N0	PIN_B10	2.5 V		12mA ...ault)	2 (default)	
lm_status[0]	Output	PIN_A10	7	B7_N0	PIN_A10	2.5 V		12mA ...ault)	2 (default)	
rlm_status_led	Output	PIN_A8	7	B7_N0	PIN_A8	2.5 V		12mA ...ault)	2 (default)	
rst	Input	PIN_B14	7	B7_N0	PIN_B14	2.5 V		12mA ...ault)		
rst_led	Output	PIN_A9	7	B7_N0	PIN_A9	2.5 V		12mA ...ault)	2 (default)	

Εικ. 5.27: Αναθέσεις σημάτων στο pin planner

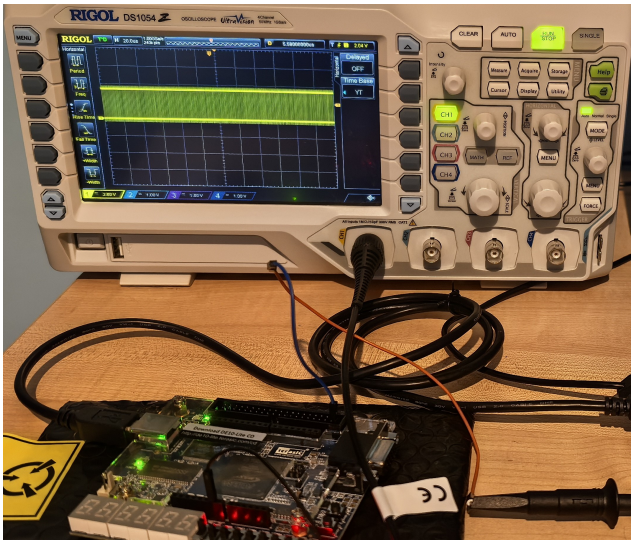
Με την ολοκλήρωση των assignments έγινε και ο προγραμματισμός του αναπτυξιακού. Οι μετρήσεις της γεννήτριας τυχαίων αριθμών έγιναν με την βοήθεια παλμογράφου της rigol ds1054. Πρόκειται για παλμογράφο με δυνατότητα δειγματοληψίας στα $50Mhz$ και 4 κανάλια δειγματοληψίας. Το setup του παλμογράφου απεικονίζεται στην εικόνα 5.28.

Για την εξακρίβωση των αποτελεσμάτων χρησιμοποιήθηκε το trigger του παλμογράφου για να πάρουμε μια περίοδο του σήματος.

Στην εικόνα 5.29 παρατηρούνται τα πρώτα $12uS$ της κρυπτογεννήτριας. Ο διακόπτης που ενεργοποιεί την γεννήτρια ενεργοποιήθηκε μετά από $800nS$. Παρατηρώντας την χρονική διάρκεια των παλμών υπάρχει ομοιομορφία σε σχέση με την προσομοίωση του αρχείου vwf.

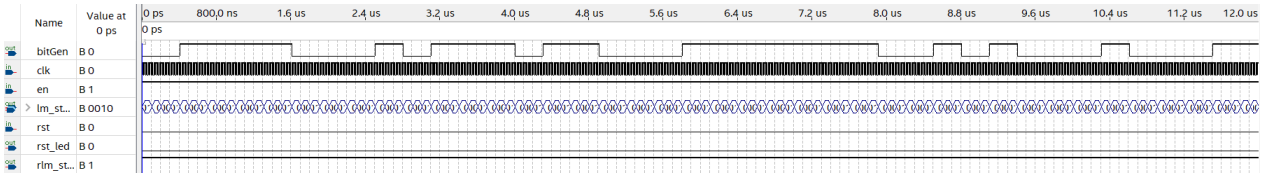
Οι εικόνες που παρουσιάζονται αποτελούν διαπιστευτήρια της ορθής λειτουργίας της διάταξης που αναλύθηκε, μελετήθηκε και σχεδιάστηκε στο αναπτυξιακό FPGA της intel. Στην εικόνα 5.30 απεικονίζεται και η προσομοίωση της λειτουργίας με χρήση αρχείο vwf. Παρατηρώντας τους πρώτους παλμούς τόσο στην εικόνα 5.30, όσο και στην 5.29 θα διαπιστώσουμε ότι τόσο τα bit με τιμή '1' όσο και αυτά με τιμή '0' έχουν ίδια διάρκεια.

Ένα απλός τρόπος μελέτης της αποτελεσματικότητας της κρυπτογράφησης, είναι η μελέτη του μήκους του κλειδιού(key space), μιας και αποτελεί και τον αριθμό των προσπαθειών που πρέπει να εξαντλήσει



Εικ. 5.28: Δειγματοληψία prpg με χρήση παλμογράφου

Εικ. 5.29: Πρώτα $12us$ παραγωγής τυχαίων αριθμών μετά την ενεργοποίηση του enable



Εικ. 5.30: Προσομοίωση με χρήση vwf

ένας αλγόριθμος brute force για να σπάσει την κρυπτογράφηση. Στην περίπτωση του RLM με μήκος διαύλου(bus width) $32bit$ για τις αρχικές συνθήκες x_0, r υπολογίζουμε το μήκος του κλειδιού ως:

$$key\_space = 2^{32} * 2^{32} = 2^{64} \tag{5.28}$$

Για τον υπολογισμό τον nist για το fpga ελέγχθηκαν μόνο 5 bitstreams των 1000000 δειγμάτων για την επιβεβαίωση των αποτελεσμάτων. Τα αποτελέσματα απεικονίζονται στον πίνακα 5.4.

Ο πίνακας με την δέσμευση πόρων αλλά και την χρονική απόκριση του RLM απεικονίζεται παρακάτω.

PROPORTION	STATISTICAL TEST
5/5	Frequency
5/5	BlockFrequency
5/5	CumulativeSums
5/5	Runs
5/5	LongestRun
5/5	Rank
5/5	FFT
5/5	NonOverlappingTemplate
5/5	OverlappingTemplate
5/5	Universal
5/5	ApproximateEntropy
5/5	RandomExcursions
5/5	RandomExcursionsVariant
5/5	Serial
5/5	LinearComplexity

Πίνακας 5.4: Αποτελέσματα ελέγχου rlm ψευδογεννήτριας

LU	LUT	Reg	PLL	IO	Max delay Path	FMAX	Power estimate
1436	0	62	1	10	40ns	24.90Mhz	105mW

Πίνακας 5.5: Πίνακας δέσμευσης πόρων για την υλοποίηση της διάταξης RLM

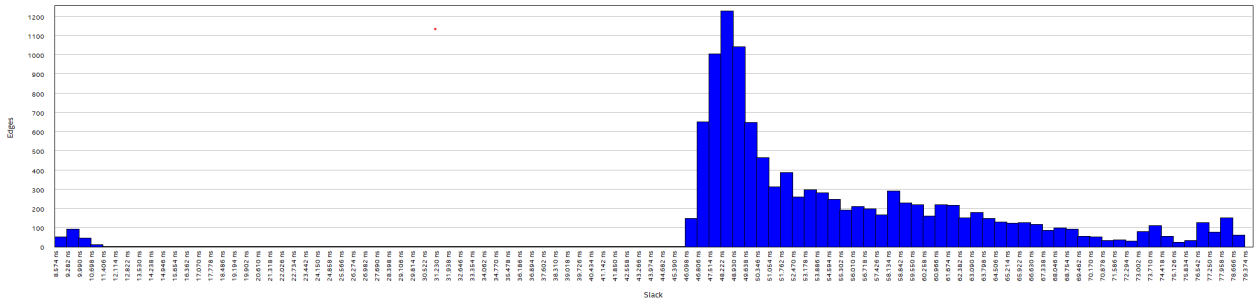
5.4.2 Πειραματικά αποτελέσματα γεννήτριας στα $12.5MHz$

Προκειμένου η σχεδίαση να υπερβεί την υπολογιστική της ισχύ και δεδομένου ότι η θεωρητικά ανώτερη συχνότητα ρολογιού υπολογίστηκε στα $25Mhz$, αυξήθηκε η συχνότητα λειτουργίας αλλάζοντας την ρύθμιση των πολλαπλασιαστών και των διαιρετών του PLL. Η νέα συχνότητα ρολογιού που επιλέχθηκε είναι τα $12.5Mhz$. Με την νέα συχνότητα ρολογιού το καινούργιο throughput που προκύπτει δίνεται από την εξίσωση 5.29.

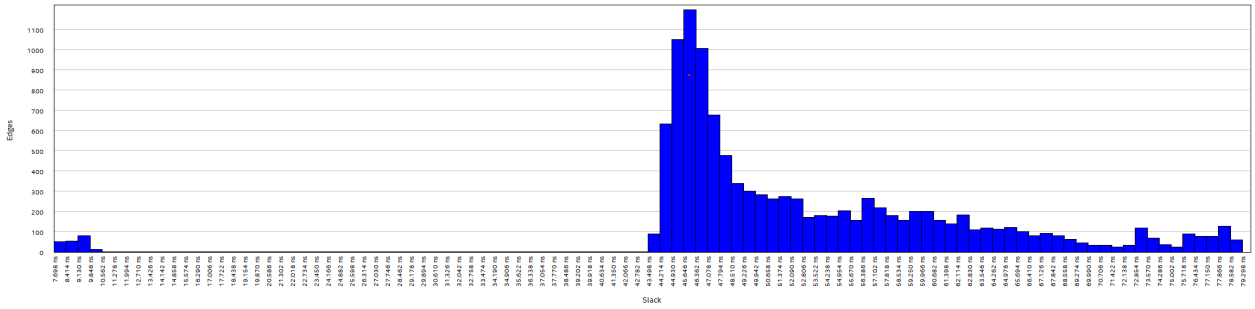
$$\begin{aligned} T &= \frac{1}{3MC \cdot \frac{1}{f}} \Rightarrow \\ T &= 4.166Mbps \end{aligned} \tag{5.29}$$

Με την εφαρμογή υψηλότερης συχνότητας ρολογιού επιτυγχάνεται η παραγωγή ενός bit ανά 3 κύκλους μηχανής. Για την αξιολόγηση της γεννήτρια RLM με υψηλότερη ταχύτητα ρολογιού χρησιμοποιήθηκε εκ νέου time analyzer. Στην εικόνα 5.31 απεικονίζονται τα slack του RLM με χρήση του μοντέλου slow 1200mV 85°C.

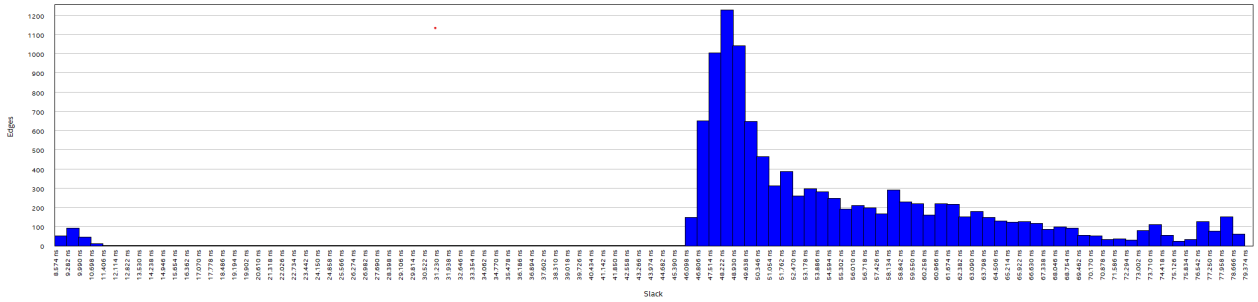
Στα ιστογράμματα που απεικονίζονται παραπάνω, παρατηρούμε και τους φυσικούς περιορισμούς που έχει η σχεδίαση με αθροιστές RCA. Όπως παρατηρούμε στα ιστογράμματα υπάρχει πλήθος σημάτων που οριακά καταφέρνουν να ικανοποιήσουν την χρονική απαίτηση της διάταξης βάση της συχνότητας ρολογιού. Τα σήματα αυτά είναι τα σήματα που βρίσκονται κοντά στο 0. Αυτό έχει σαν αποτέλεσμα, παρόλο που οι



Slow $1200mV$ $0°C$



Slow $1200mV$ $85°C$



Fast $1200mV$ $85°C$

Εικ. 5.31: Ιστόγραμμα slack για τα τρια μοντέλα προσομοίωσης

προσομοιώσεις έδειξαν ότι διάταξη έχει $f_{max} \approx 25MHz$, η διάταξη να μην δύναται να λειτουργήσει σε μεγαλύτερες συχνότητες των $12.5MHz$.

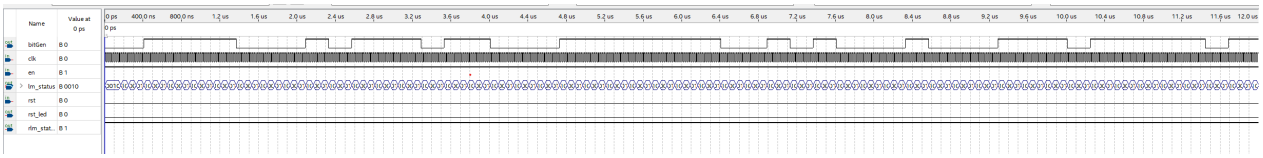
Παρόλο που το slack από μόνο του δεν είναι ικανό να αλλοιώσει την λειτουργία της γεννήτριας, είναι τροχοπέδη στην παραγωγή bit ανά κύκλο μηχανής, καθότι αθροιστικά οι καθυστερήσεις στα σήματα θα μπορούσαν να επιφέρουν και καθυστέρηση στην παραγωγή των bit στην έξοδο. Εφόσον στην σχεδίαση της γεννήτριας δεν υπάρχει κάποια διάταξη για να δημιουργεί μια καθυστέρηση, αρά και να συγχρονίζει την έξοδο βάση αυτής, θεωρούμε ότι το καλύτερο δυνατό αποτέλεσμα είναι η γεννήτρια να χρονιστεί στα $12.5MHz$, όπου τα σήματα δεν παρουσιάζουν αρνητικό slack.

Στην υλοποίηση με συχνότητα ρολογιού $12.5MHz$ η εκτιμώμενη συνολική κατανάλωση ισχύος που

προέκυψε από την χρήση του εργαλείου Power Analyzer είναι:

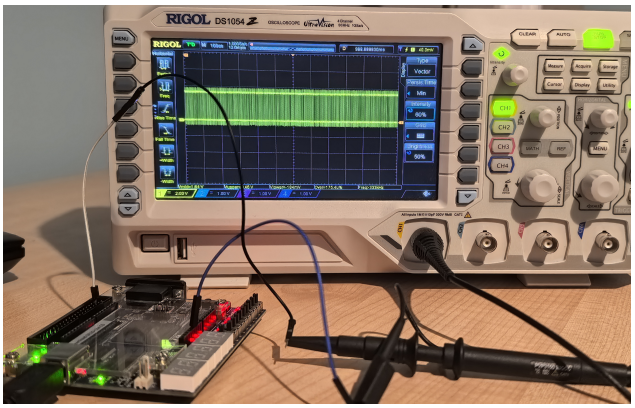
$$P_{tot} = 118.9mW \tag{5.30}$$

Παρατηρούμε μια αύξηση της τάξης των 10mV για μια αύξηση σε επεξεργαστική ισχύ της τάξης του 33%. Ένα αρκετά ικανοποιητικό αποτέλεσμα. Στην εικόνα 5.32 απεικονίζεται και η προσομοίωση της λειτουργίας της διάταξης με την χρήση vwf.



Εικ. 5.32: Προσομοίωση λειτουργίας RLM στα $12.5MHz$

Συγκρίνοντας τα αποτελέσματα της εικόνας 5.32 με της εικόνας 5.30 παρατηρούμε ότι η αλληλουχία των bit στην έξοδο είναι όμοια με την διαφορά ότι στον ίδιο χρόνο ($12\mu s$), έχουμε μεγαλύτερο throughput.



α)Υλοποίηση γεννήτριας RLM στα $12.5MHz$ με χρήση FPGA

β)Δειγματοληψία τυχαίας περιόδου της γεννήτριας

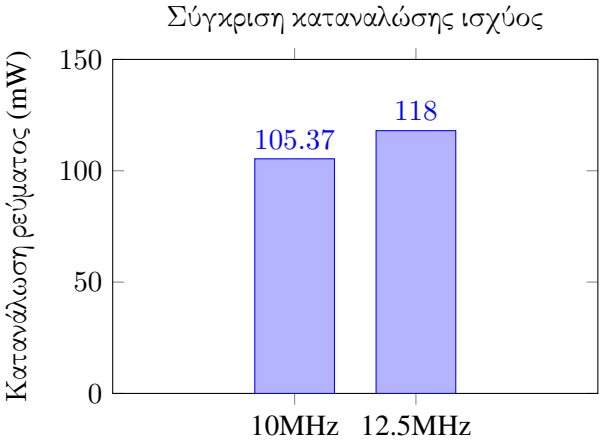
Εικ. 5.33: Σύλληψη δεδομένων με χρήση παλμογράφου

Όπως παρατηρούμε στην εικόνα 5.33β το ένα bit έχει διάρκεια $0.24\mu s$ ενώ οι παλμοί που έχουν μεγαλύτερη

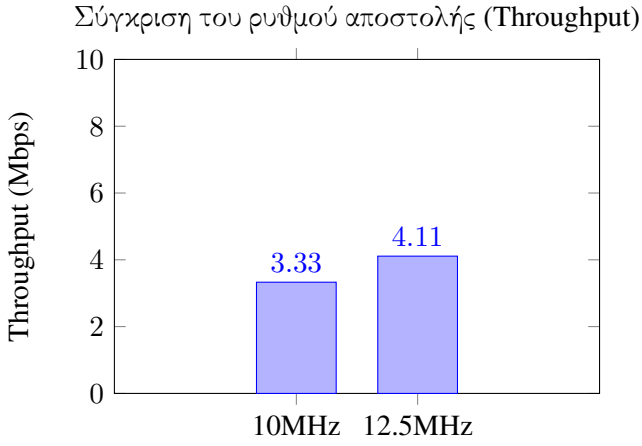
διάρκεια είναι διαδοχικά ’0’ ή ’1’. Η υλοποίηση της γεννήτριας στα $12.5MHz$ φαίνεται να είναι εξίσου αξιόπιστη με καλύτερα αποτελέσματα σε ότι έχει να κάνει με την ταχύτητα παραγωγής δεδομένων.

5.4.3 Συγκριτικά αποτελέσματα γεννήτριας σε **FPGA**

Στην εικόνα 5.35 απεικονίζεται ένα γράφημα για την σύγκριση του ρυθμού αποστολής(throughput) της διάταξης RLM με διαφορετικές συχνότητες ρολογιού, αλλά και ένα διάγραμμα με της κατανάλωσης ισχύος του αναπτυξιακού, βάση της εκτίμησης που εξήγαμε από το power analyzer της quartus Με γνώμονα ότι



Εικ. 5.34: Διάγραμμα κατανάλωσης ισχύος



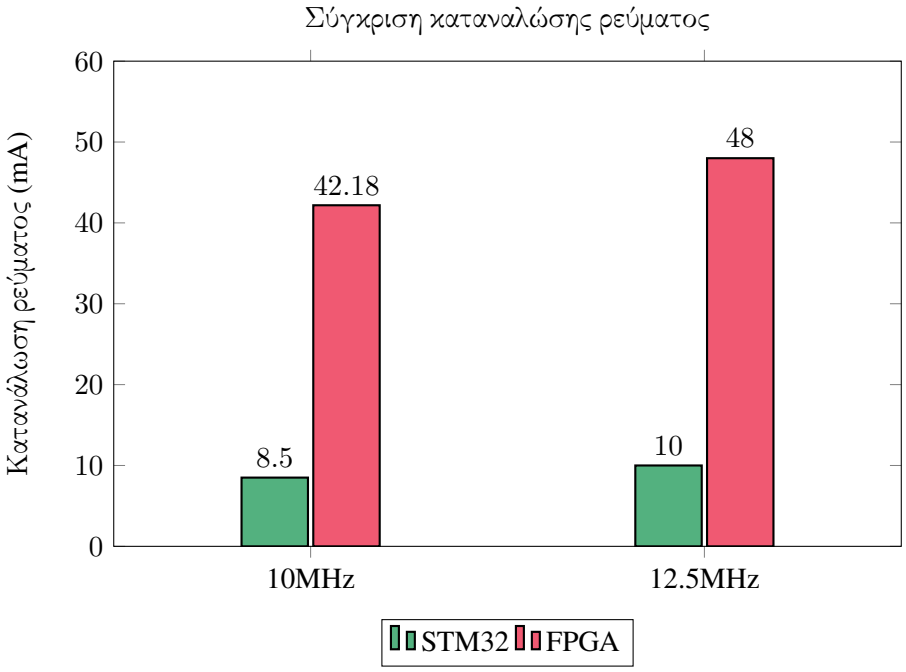
Εικ. 5.35: Throughput for Different Configurations

όλα τα io αλλά και ο πυρήνας του FPGA λειτουργούν στα 2.5V, οι αντίστοιχες καταναλώσεις ρεύματος υπολογίζονται σε:

$$I_{10MHz} = \frac{P}{V} \Rightarrow I_{10MHz} \approx 42.18mA$$
$$I_{12.5MHz} = \frac{P}{V} \Rightarrow I_{12.5MHz} \approx 48mA \tag{5.31}$$

5.4.4 Σύγκριση αποτελεσμάτων **FPGA/STM32**

Έχοντας ολοκληρώσει το κομμάτι των πειραματικών μελετών με χρήση τόσο συστημάτων μικροελεγκτή, όσο και με fpga, έχει ερευνητικό ενδιαφέρων να παρουσιαστούν οι δομικές τους διαφορές. Στην εικόνα 5.36 απεικονίζεται η διαφορά των συστημάτων σε ότι έχει να κάνει με την κατανάλωση ρεύματος. Παρατηρούμε



Εικ. 5.36: Διάγραμμα κατανάλωσης ρεύματος

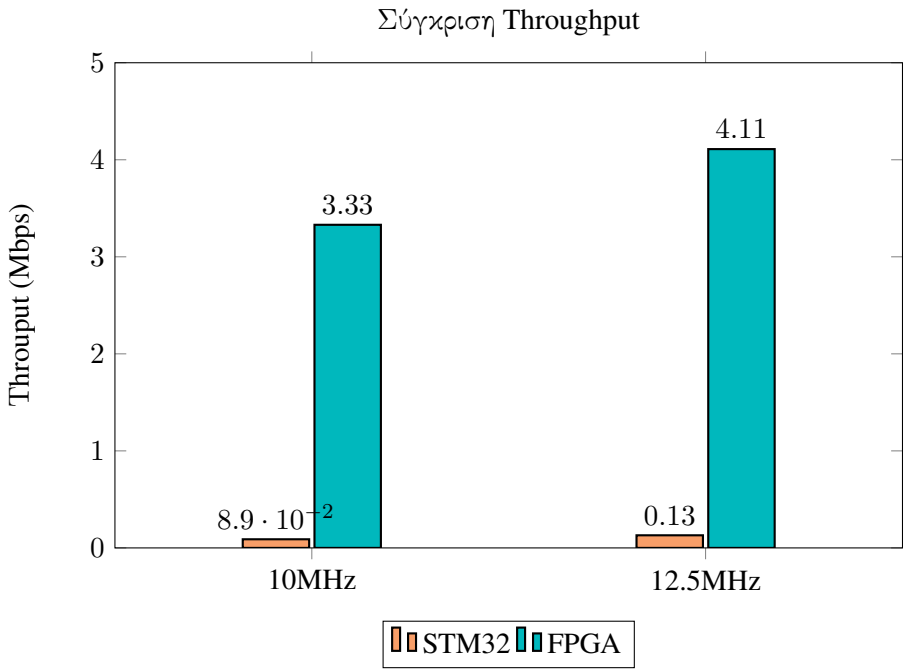
με μεγάλη βεβαιότητα ότι η υλοποίηση στο FPGA της γεννήτριας καθότι περιέχει πολλές λογικές διατάξεις έχει πολύ μεγαλύτερη κατανάλωση συγκεκριμένα:

$$\begin{aligned} I_{FPGA\_10MHz} &= 4.96 I_{STM32\_10MHz} \\ I_{FPGA\_12.5MHz} &= 4.8 I_{STM32\_10MHz} \end{aligned} \tag{5.32}$$

Η σύγκριση για τον ρυθμό αποστολής(throughput) δίνεται στην εικόνα 5.37. Εκεί που η σχεδίαση στο FPGA υπερβαίνει εκθετικά αυτήν στον μικροελεγκτή είναι στο throughput. Όπως παρατηρούμε οι λόγοι των throughput μεταξύ των δύο συστημάτων δίνονται από την σχέση:

$$\begin{aligned} T_{FPGA\_10MHz} &\approx 37.5 \cdot T_{STM32\_10MHz} \\ T_{FPGA\_12.5MHz} &\approx 31.61 \cdot T_{STM32\_12.5MHz} \end{aligned} \tag{5.33}$$

Αν και παρατηρούμε ότι στο κομμάτι της κατανάλωσης ισχύος στιγμιαία υπερισχύει ο μικροελεγκτής, πρέπει να λογαριάσουμε την μέση ισχύ για την κρυπτογράφηση ενός πακέτου για να μπορέσουμε να εξάγουμε με ασφάλεια συμπεράσματα. Για την κρυπτογράφηση ενός πακέτου ,για παράδειγμα, των $256bit$



Εικ. 5.37: Διάγραμμα σύγκρισης throughput

η γεννήτρια RLM στο FPGA θα χρειαζόταν:

$$\begin{aligned} t_{crypto} &= \frac{payload}{throughput} \Rightarrow \\ t_{crypto} &= \frac{256}{3.33 * 10^6} \Rightarrow \\ t_{crypto} &\approx 77 * 10^{-6} \approx 77uSec \end{aligned} \tag{5.34}$$

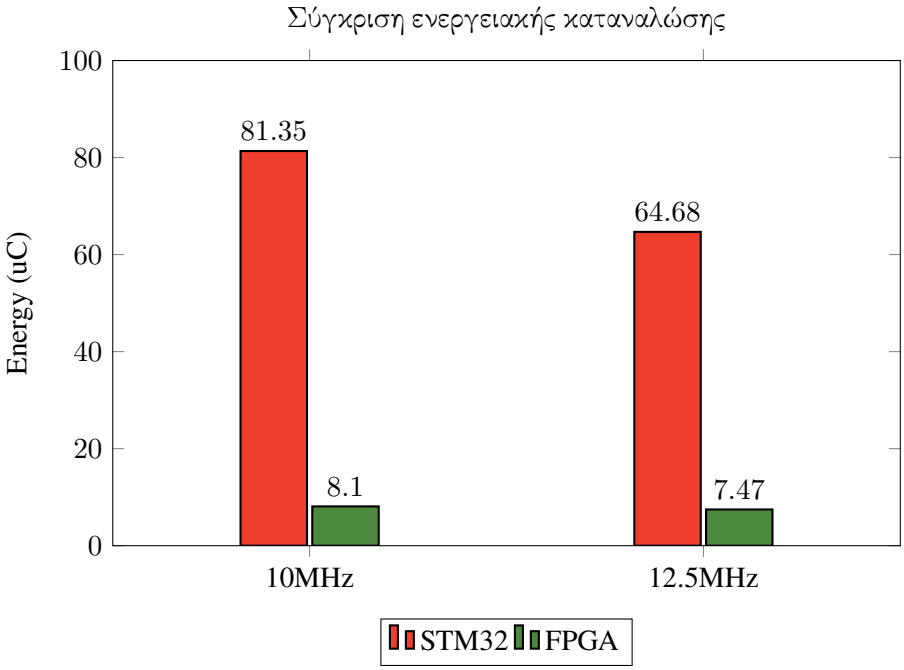
Αντίστοιχα η γεννήτρια του stm32 θα εκτελούσε την κρυπτογράφηση σε:

$$\begin{aligned} t_{crypto} &= \frac{payload}{throughput} \Rightarrow \\ t_{crypto} &= \frac{256}{89 * 10^3} \Rightarrow \\ t_{crypto} &\approx 2.9 * 10^{-3} \approx 2.9msec \end{aligned} \tag{5.35}$$

Άρα αν θέλαμε να μετρήσουμε την ενέργεια που καταναλώνουν οι δύο διατάξεις για το παράδειγμα αυτό στα $10MHz$, το αποτέλεσμα θα δινόταν από την παρακάτω σχέση:

$$\begin{aligned} E &= P \cdot t \Rightarrow \\ E &= V \cdot i \cdot t \Rightarrow \\ E_{stm32} &= 81.345 \cdot 10^{-6}C \approx 81uC \\ E_{fpga} &= 8.1175 \cdot 10^{-6}C \approx 8.1uC \end{aligned} \tag{5.36}$$

Φαίνεται λοιπόν πως παρότι το ρεύμα που καταναλώνει ο μικροελεγκτής είναι πολύ μικρότερο, παρόλα αυτά η ενέργεια που καταναλώνει για την κρυπτογράφηση κάθε πακέτου είναι 10 φορές μεγαλύτερη από αυτή του fpga. Τα συνολικά αποτελέσματα για την κατανάλωση ενέργειας στα δύο συστήματα παρουσιάζονται στην εικόνα 5.38.



Εικ. 5.38: Διάγραμμα σύγκρισης ενεργειακής κατανάλωσης

5.4.5 Συγκριτικά αποτελέσματα γνωστών επιθέσεων

Ολοκληρώνοντας την μελέτη για την γεννήτρια κρυπτογράφησης πρέπει να αναφέρουμε έστω και συνοπτικά την απόδοση των δύο υλοποιήσεων σε FPGA/stm32, έναντι σε γνωστές επιθέσεις. Η ανάλυση είναι καθαρά θεωρητική και βασίζεται στις πειραματικές μετρήσεις που εξήγαμε και από τα δυο συστήματα. Συγκεκριμένα οι επιθέσεις για τις οποίες μπορούμε να έχουμε μια καλή εκτίμηση του πως θα συμπεριφέρεται κάθε υλοποίηση είναι αυτές των πλευρικών καναλιών(side channel attacks).

Οι επιθέσεις πλευρικών καναλιών (Side-Channel Attacks - SCA) εκμεταλλεύονται πληροφορίες που διαρρέουν από την υλοποίηση κρυπτογραφικών αλγορίθμων, όπως η κατανάλωση ισχύος ή ο χρόνος εκτέλεσης. Οι κύριες κατηγορίες αυτών των επιθέσεων περιλαμβάνουν:

- Απλή Ανάλυση Ισχύος **(Simple Power Analysis - SPA)**:
  Αυτή η μέθοδος περιλαμβάνει την άμεση παρατήρηση και ερμηνεία των μετρήσεων κατανάλωσης ισχύος κατά τη διάρκεια της εκτέλεσης κρυπτογραφικών λειτουργιών. Μέσω της SPA, ένας επιτιθέμενος μπορεί να αναγνωρίσει μοτίβα που σχετίζονται με συγκεκριμένες λειτουργίες ή κρυπτογραφικά κλειδιά. [17]

- Διαφορική Ανάλυση Ισχύος **(Differential Power Analysis - DPA)**:

Η DPA χρησιμοποιεί στατιστικές μεθόδους για την ανάλυση πολλαπλών μετρήσεων κατανάλωσης ισχύος, με στόχο την εξαγωγή μυστικών πληροφοριών, όπως τα κρυπτογραφικά κλειδιά. Αυτή η τεχνική είναι πιο ισχυρή από την SPA και μπορεί να αποκαλύψει πληροφορίες ακόμα και παρουσία θορύβου. [17]

- Συσχέτιση Ανάλυσης Ισχύος **(Correlation Power Analysis - CPA)**:
  Η CPA αποτελεί εξέλιξη της DPA, όπου δημιουργείται ένα μοντέλο της κατανάλωσης ισχύος και συσχετίζεται με τις πραγματικές μετρήσεις, προκειμένου να ανακτηθούν μυστικά κλειδιά. Αυτή η μέθοδος απαιτεί ακριβή μοντελοποίηση της συσκευής-στόχου. [2]

- Επιθέσεις Χρονισμού **(Timing Attacks)**:
  Οι επιθέσεις χρονισμού εκμεταλλεύονται τις διαφορές στον χρόνο εκτέλεσης κρυπτογραφικών αλγορίθμων, οι οποίες μπορεί να αποκαλύψουν πληροφορίες σχετικά με τα δεδομένα εισόδου ή τα κρυπτογραφικά κλειδιά. [18]

Στον Πίνακα 5.6, παρουσιάζεται η σύγκριση της ευπάθειας μεταξύ υλοποιήσεων PRPG σε **STM32** και **FPGA**, λαμβάνοντας υπόψη τον χρόνο εκτέλεσης και την κατανάλωση ενέργειας.

Υλοποίηση **RLM24**	Χρόνος Κρυπτογράφησης **(256 bytes)**	Κατανάλωση Ρεύματος **(mA)**	Ευπάθεια σε **SPA/DPA/CPA**	Κίνδυνος Χρονικών Επιθέσεων
STM32 (10MHz)	2.9ms	8.5mA	Χαμηλή - Λιγότερη διαρροή ισχύος	Μέτρια - Πιθανή προβλεψιμότητα
FPGA (10MHz)	77μs	42.18mA	Υψηλή - Μεγαλύτερη κατανάλωση ισχύος	Χαμηλή - Εκτελείται πολύ γρήγορα

Πίνακας 5.6: Σύγκριση υλοποιήσεων PRPG ως προς κατανάλωση ενέργειας και ανθεκτικότητα σε επιθέσεις πλευρικών καναλιών.

5.5 Συμπεράσματα

Έχοντας παρουσιάσει όλα τα αποτελέσματα της γεννήτριας RLMIC που υλοποιήθηκε τόσο σε αναπτυξιακό FPGA, όσο και σε μικροελεγκτή, οφείλουν να αναλυθούν τα συμπεράσματα τόσο σε επίπεδο μαθηματικού μοντέλου, όσο και σε επίπεδο υλοποίησης σε hardware. Συνοπτικά η γεννήτρια παρουσιάζει:

- Πολύ καλά αποτελέσματα τυχαιότητας τόσο σε επίπεδο προσομοίωσης, όσο και σε επίπεδο σχεδίασης, καθότι πλήρη όλα τα κριτήρια των test του nist 802.22.
- Κατά συνέπεια πρόκειται για μια ανθεκτική σχεδίαση σε επιθέσεις καναλιού, καθότι έχει πολύ χαμηλούς δείκτες κατανάλωσης ισχύος τόσο σε FPGA, όσο και σε μικροελεγκτή.
- Ωστόσο έχει μικρό μήκος κλειδιού(key space), κάτι που ενισχύει την ευπάθεια της σε αλγορίθμους brute force.
- Όπως αναφέρθηκε και στο πρώτο κεφάλαιο η γεννήτρια που σχεδιάστηκε δεν είχε σκοπό το μέγιστο δυνατό throughput και για αυτό παρατηρείται μεγάλη απόκλιση σε σχέση με άλλες εργασίες που χρησιμοποιούν dsp για την μεγιστοποίηση της ταχύτητας.

Αναπτυξιακό	Συχνότητα Λειτουργίας(MHz)	Κατανάλωση Ισχύς(mW)	Κατανάλωση ρεύματος**(mA)**	Κατανάλωση Ενέργειας**(uC)**	**Throughput(Mbps)**
STM32H7	10	28.05	8.5	81.35	0.089
STM32H7	12.5	33.3	10	64.68	0.13
DE10-LITE	10	105.37	42.18	8.1	3.33
DE10-LITE	12.5	118	48	7.47	4.1

Πίνακας 5.7: Συγκεντρωτικός πίνακας αποτελεσμάτων

Στον πίνακα 5.7 απεικονίζονται τα συνολικά αποτελέσματα της διπλωματικής για κάθε αναπτυξιακό, καθώς και για κάθε συχνότητα λειτουργίας που δοκιμάστηκε. Συνολικά η υλοποίηση στο FPGA μας ωφελεί στη περίπτωση που θέλουμε μεγάλο throughput καθώς και χαμηλή πιθανότητα κινδύνου από χρονικές επιθέσεις. Η χρήση του ωστόσο αυξάνει αισθητά της πιθανότητες για επιθέσεις ανάλυσης ισχύος, λόγω της αυξημένης κατανάλωσης ισχύος. Ο μικροελεγκτής από την άλλη μπορεί να χρησιμοποιηθεί σε εφαρμογές που δεν έχουν μεγάλη απαίτηση σε throughput

Για την μεγιστοποίηση του throughput, καθότι και για την "οχύρωση" του αλγορίθμου από brute force, αποτελεί επόμενο βήμα η ανασχεδίαση των επιμέρους κυκλωμάτων του fpga που είναι υπεύθυνα για τους υπολογισμούς καθότι δεν υπάρχει κανένα επίπεδο βελτιστοποίησης(optimization), αλλά και η χρήση νέων χαρτών που είναι χαοτικοί για οποιαδήποτε αρχική συνθήκη αλλά και με μεγαλύτερο key space. Οι απαραίτητες αλλαγές για την επίτευξη αυτών, θα αναλυθεί παρακάτω στο κεφάλαιο της μελλοντικής έρευνας.

5.6 Επίλογος

Σε αυτό το κεφάλαιο αναλύθηκε σε βάθος η θεωρητική μελέτη που ήταν απαραίτητη για την σύλληψη της ιδέας της κρυπτογεννήτριας. Εν συνεχεία παρουσιάστηκαν τα αποτελέσματα της προσομοίωσης είτε με χρήση RTL, είτε με χρήση VWF αρχείων. Αναλύθηκαν οι αδυναμίες(μεγάλο propagation delay,σχετικά μικρή μέγιστη συχνότητα επεξεργασίας), αλλά και τα ισχυρά σημεία της διάταξης(μικρή κατανάλωση

πόρων, σχετικά μικρή κατανάλωση ενέργειας). Παρουσιάστηκαν τα αποτελέσματα από την υλοποίηση της διάταξης σε αναπτυξιακό FPGA της terasic με πυρήνα fpga της intel.Τέλος παρουσιάστηκαν τα συγκριτικά αποτελέσματα σε ότι έχει να κάνει με τις πειραματικές μετρήσεις των δύο ενσωματωμένων συστημάτων (throughput,κατανάλωση ρεύματος,ισχύος,ενέργειας), καθώς επίσης αναφέρθηκαν οι αδυναμίες του κάθε συστήματος από άποψη γνωστών αλγορίθμων επίθεσης στην κρυπτογεννήτρια. Το κεφάλαιο αυτό αποδεικνύει την πιστότητα των θεωρητικών αποτελεσμάτων της κρυπτογεννήτριας καθώς και την υλοποίηση τους σε ολοκληρωμένο ψηφιακό κύκλωμα.

6 Συγκριτικά αποτελέσματα με **state-of-art** βιβλιογραφία

Σε αυτό το κεφάλαιο παρουσιάζεται μια μελέτη της βιβλιογραφίας και ορισμένες συγκρίσεις με συστήματα παραγωγής τυχαίων αριθμών για κρυπτογράφηση σημάτων, με υλοποιήσεις που βασίζονται σε παραλλαγές του λογιστικού χάρτη που παρουσιάζεται σε αυτήν την εργασία.

LU	LUT	Reg	PLL	DSP	FMAX	Power estimation	Throughput(Mbps)	Work
1436	1	62	0	1	24.90Mhz	105mW	4.11	Διπλωματική
0	309	28	-	3	58.358MHz	-	-	[15]
648	-	126	1	14	-	-	-	[32]
571	-	32	-	8	50MHz	-	-	[19]
852	-	288	-	14	34.59MHz	-	-	[24]
2363	-	-	-	-	15	-	565	[13]
1186	-	-	-	-	36.78MHz	0.101W	1179.07	[10]
3160	-	-	-	13	32.41MHz	0.152W	1037.27	[9]

Πίνακας 6.8: Πίνακας συγκρίσεων απόδοσης/πόρων υλοποιήσεων, ψευδοτυχαίας γεννήτριας

Στον πίνακα 6.8 απεικονίζονται οι πόροι που καταναλώθηκαν από την κάθε ερευνητική εργασία με σκοπό την κρυπτογράφηση δεδομένων. Στο πίνακα απεικονίζονται μόνο οι πόροι που απαιτούνται για την παραγωγή τυχαίων αριθμών. Οι τρεις εργασίες που αναφέρονται στο παραπάνω πίνακα ασχολούνται κατά κύριο λόγο με την κρυπτογράφηση εικόνας. Ο σκοπός τους είναι ο ελάχιστος δυνατός χρόνος κρυπτογράφησης και μετάδοσης. Για αυτό τον λόγο δεν υπάρχει μια καθαρή σύγκριση.

Παρόλα αυτά παρατηρούμε ότι όλες οι υλοποιήσεις περιέχουν χρήση DSP για την πραγματοποίηση των αλγεβρικών υπολογισμών. Ένα καλό μέτρο σύγκρισης αποτελεί η εργασία του Samar[15]et al, όπου ο συγγραφέας έχει πετύχει την παραγωγή τυχαίων αριθμών μόνο σε επίπεδο προσομοίωσης. Ακόμα και μετά την χρήση DSP block με μήκος Bus 20bit σε αντίθεση με τα 32 που χρησιμοποιούνται στην παρούσα εργασία, ο Samar επιτυγχάνει διπλάσια δυνατή συχνότητα λειτουργίας. Στις εργασίες [10, 9, 13] που παρατηρείται χαόδης διαφορά στο throughput σε σχέση με αυτό της διπλωματικής δεν αναφέρονται στο throughput που προκύπτει από την παραγωγή bit αλλά σε αυτό της επικοινωνίας. Παρόλο που διαφορά σίγουρα είναι τάξεις μεγέθους πάνω λόγο της υψηλής συχνότητας ρολογιού των dsp που χρησιμοποιούνται.

Καθώς όλες οι εργασίες έχουν επιτύχει στα NIST test, ένα κριτήριο που μπορεί να μελετηθεί είναι το **key space** της κάθε εργασίας.

Key Space	Work
2^{64}	Διπλωματική
2^{60}	[15]
2^{276}	[32]
2^{60}	[24]
2^{60}	[19]

Πίνακας 6.9: Πίνακας συγκρίσεων key space

Παρατηρούμαι ότι λόγω του ότι οι αρχικές μας συνθήκες είναι $32bit$ αποκτάμε λίγο ασφαλέστερο key

space από ότι οι kotaki et al,και Mohamed et al[19, 24].

Συνοπτικά με την υλοποίηση του RLM επιτυγχάνεται ένα απλό μοντέλο κρυπτογράφησης μέσω παραγωγής τυχαίων αριθμών, ικανό να αποτελεί πυρήνα υλοποιήσεων με απαίτηση ασφαλή κρυπτογράφηση σε χαμηλό επίπεδο.

7 Προτάσεις Βελτιστοποίησης

Όπως αναφέρθηκε στο κεφάλαιο της υλοποίησης της γεννήτριας τυχαίων αριθμών η σχεδίαση αν και επιτυγχάνει τον σκοπό της σε επίπεδο proof-of-concept, έχει ενσωματωμένες στον πυρήνα της "παθογένειες" που ξεκινούν από τη σύλληψη της ιδέας της υλοποίησης.

- Μεγάλο t_{pd} που οφείλεται στο γεγονός ότι το κρατούμενο bit διαδίδεται από αθροιστή σε αθροιστή μειώνοντας την απόδοση τόσου του αθροιστή όσο και του MAC.
- Μικρό f_{max} που οφείλεται και στο t_{pd}.
- Μικρό key space λόγω της εξάρτησης του συστήματος μόνο από τις δυο αρχικές συνθήκες.
- Χαμηλό throughput λόγω σχεδίασης: Στην περίπτωση του throughput θα μπορούσε να έχει επιλεχθεί ένα πολύ μεγαλύτερο, παρ' όλα αυτά το $1Bit/4MC$ δεσμεύει την ταχύτητα παραγωγής bit.

Για όλους τους παραπάνω λόγους, μερικές από τις προτάσεις για την βελτιστοποίηση αποτελούν:

- Αλλαγή των αθροιστών από ripple-carry σε CLA(carry look ahead). Οι CLA έχουν το προτέρημα του πολύ μικρού t_{pd} ανεξαρτήτως μήκους διαύλου άλλα έχουν μεγαλύτερη απαίτηση σε υλικό, για αυτό και η υλοποίηση τους συστήνεται συνδυαστικά με κάποια άλλη μορφή αθροιστή.
- Χρήση DSP για τους πολλαπλασιασμούς του συστήματος ή χρήση πολλαπλασιαστών vedic.
- Υλοποίηση πιο σύνθετων χαοτικών χαρτών για αύξηση του key space αλλά και χρήση πραγματικών αρχικών συνθηκών για δημιουργία PUF(Physical unclonable function).

Ένα ακόμη σημαντικό βήμα για την βελτιστοποίηση της υλοποίησης της γεννήτριας τυχαίων αριθμών είναι η χρήση της τεχνικής του pipelining. Το pipelining είναι μία από τις πιο δημοφιλείς τεχνικές βελτιστοποίησης για εφαρμογές που υλοποιούνται σε FPGA, καθώς επιτρέπει τη μεγιστοποίηση της ταχύτητας εκτέλεσης των εργασιών μέσω της παράλληλης εκτέλεσης των σταδίων ενός υπολογιστικού κύκλου. Η τεχνική αυτή βασίζεται στην ιδέα ότι οι διάφορες φάσεις μιας εργασίας μπορούν να εκτελούνται ταυτόχρονα, μειώνοντας έτσι τον χρόνο που απαιτείται για την ολοκλήρωση ενός πλήρους κύκλου υπολογισμού.

Στην περίπτωση της γεννήτριας τυχαίων αριθμών, η εφαρμογή pipelining θα μπορούσε να έχει σημαντική επίδραση στην επίδοση του συστήματος. Αντί να εκτελούνται οι διάφορες λειτουργίες διαδοχικά, η κάθε λειτουργία (π.χ. αθροιστής, MAC, πολλαπλασιαστές, χαοτικοί χάρτες) θα μπορούσε να ξεκινήσει όταν το προηγούμενο στάδιο ολοκληρώνεται, επιτρέποντας την ταυτόχρονη εκτέλεση πολλών υπολογισμών.

Επιπτώσεις στην Εφαρμογή FPGA:

Αύξηση Ταχύτητας Εκτέλεσης **(Throughput)**: Με την εφαρμογή του pipelining, μπορεί να επιτευχθεί σημαντική αύξηση στην ταχύτητα παραγωγής δεδομένων, καθώς τα στάδια της διαδικασίας μπορούν να εκτελούνται παράλληλα. Αυτό θα οδηγήσει σε μεγαλύτερο throughput, κάτι το οποίο είναι κρίσιμο για συστήματα που απαιτούν γρήγορη παραγωγή τυχαίων αριθμών.

Μειωμένο **Tpd (Propagation Delay)**: Το pipelining βοηθά στην εξομάλυνση της κατανομής των καθυστερήσεων στους αθροιστές και άλλες μονάδες, καθώς τα δεδομένα μπορούν να "ρέουν" μέσα από τα στάδια χωρίς να περιμένουν την ολοκλήρωση του προηγούμενου υπολογισμού. Επομένως, το tpd μπορεί να μειωθεί, βελτιώνοντας την απόδοση του συστήματος.

Χρησιμοποίηση Πόρων **FPGA**: Αν και το pipelining επιτρέπει τη βελτιστοποίηση της ταχύτητας, έχει το κόστος της αύξησης της κατανάλωσης πόρων του FPGA, καθώς απαιτεί περισσότερα flip-flops και άλλα στοιχεία για την υλοποίηση των διάφορων σταδίων. Ωστόσο, αυτό το κόστος μπορεί να μειωθεί με τη χρήση αποδοτικών τεχνικών σχεδίασης και την επιλογή κατάλληλων συνδυασμένων αθροιστών (π.χ. CLA) που είναι αποδοτικοί σε επίπεδο υλικού.

Επιτάχυνση Λειτουργιών και Συνολική Απόδοση: Ο συνδυασμός pipelining με άλλες βελτιστοποιήσεις, όπως οι CLA αθροιστές και οι DSP πολλαπλασιαστές, μπορεί να επιφέρει σημαντική αύξηση στην ταχύτητα του συστήματος, επιτρέποντας ταχύτερη παραγωγή τυχαίων αριθμών και συνεπώς βελτιώνοντας τη γενική απόδοση της εφαρμογής.

Συμβατότητα με πιο σύνθετους Χάρτες: Στην περίπτωση των χαοτικών συστημάτων και χαρτών, η εφαρμογή pipelining θα μπορούσε να βοηθήσει στην ταυτόχρονη επεξεργασία διαφόρων σημείων των χαοτικών αλγορίθμων, επιτρέποντας τη γρήγορη σύγκλιση των αποτελεσμάτων και τη μείωση του χρόνου εκτέλεσης των πολύπλοκων υπολογισμών.

Με τις προτάσεις βελτιστοποίησης ολοκληρώνεται και η διπλωματική εργασία με σκοπό τη περαιτέρω επέκταση της στο μέλλον.

A Ευρετήρια

A.1 Βιβλιογραφία

Books

[5] John Crowe and Barrie Hayes-Gill. *Introduction to digital electronics*. Elsevier, 1998.

[26] Andrew Rukhin et al. *A statistical test suite for random and pseudorandom number generators for cryptographic applications*. Vol. 22. US Department of Commerce, Technology Administration, National Institute of . . ., 2001.

[28] Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[33] Μελετλίδου Ε. Μαάιτα Τ. Ειδικά Θέματα Μη Γραμμικής Δυναμικής. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις, 2023.

Papers

[1] Stefanos Boccaletti et al. "The control of chaos: theory and applications". In: *Physics reports* 329.3 (2000), pp. 103–197.

[3] Ricky Xiaofeng Chen. "A Brief Introduction to Shannon's Information Theory". In: *arXiv* (2021).

[7] Alfonso Delgado-Bonal and Alexander Marshak. "Approximate entropy and sample entropy: A comprehensive tutorial". In: *Entropy* 21.6 (2019), p. 541.

[8] Jonathan B Dingwell. "Lyapunov exponents". In: *Wiley encyclopedia of biomedical engineering* (2006).

[9] Fethi Dridi et al. "Design, hardware implementation on FPGA and performance analysis of three chaos-based stream ciphers". In: *Fractal and Fractional* 7.2 (2023), p. 197.

[10] Fethi Dridi et al. "The design and FPGA-based implementation of a stream cipher based on a secure chaotic generator". In: *Applied Sciences* 11.2 (2021), p. 625.

[11] William C Elmore. "The transient response of damped linear networks with particular regard to wideband amplifiers". In: *Journal of applied physics* 19.1 (1948), pp. 55–63.

[12] Ahlem Gasri et al. "A New Fractional-Order Map with Infinite Number of Equilibria and Its Encryption Application". In: *Complexity* 2022.1 (2022), p. 3592422.

[15] Samar M. Ismail et al. "Generalized fractional logistic map encryption system based on FPGA". In: *AEU - International Journal of Electronics and Communications* 80 (2017), pp. 114–126. DOI: 10.1016/j.aeue.2017.05.047.

[21] Edward N Lorenz. "Deterministic nonperiodic flow". In: *Journal of atmospheric sciences* 20.2 (1963), pp. 130–141.

[22] Yuling Luo et al. “Counteracting dynamical degradation of a class of digital chaotic systems via Unscented Kalman Filter and perturbation”. In: *Information Sciences* 556 (2021), pp. 49–66.

[23] Robert M May. “Simple mathematical models with very complicated dynamics”. In: *Nature* 261.5560 (1976), pp. 459–467.

[24] Sara M Mohamed et al. “FPGA realization of fractals based on a new generalized complex logistic map”. In: *Chaos, Solitons & Fractals* 160 (2022), p. 112215.

[25] Eric Peña and Mary Grace Legaspi. “Uart: A hardware communication protocol understanding universal asynchronous receiver/transmitter”. In: *Visit Analog* 54.4 (2020), pp. 1–5.

[29] Mao-Chyuan Tang et al. “Investigation and Modeling of Hot Carrier Effects on Performance of 45- and 55-nm NMOSFETs With RF Automatic Measurement”. In: *IEEE Transactions on Electron Devices* 55.6 (2008), pp. 1541–1546. DOI: 10.1109/TED.2008.921998.

[31] Shihong Wang et al. “Periodicity of chaotic trajectories in realizations of finite computer precisions and its implication in chaos communications”. In: *International journal of modern physics B* 18.17n19 (2004), pp. 2617–2622.

[32] Mohamed Yamni et al. “A Hardware-Accelerated Approach to Chaotic Image Encryption: LTB Map and FPGA Implementation”. In: *IEEE Access* 11 (2023). Senior Member, IEEE, pp. 7752–7770. DOI: 10.1109/ACCESS.2023.3239998.

Proceedings

[2] Eric Brier, Christophe Clavier, and Francis Olivier. “Correlation power analysis with a leakage model”. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2004, pp. 16–29.

[13] Guillaume Gautier et al. “Hardware implementation of lightweight chaos-based stream cipher”. In: *Proceedings of International Conference on Cyber-Technologies and Cyber-Systems, Porto, Portugal*. Vol. 22. 2019.

[14] Apostolos Iatropoulos et al. “Medical data encryption based on a modified sinusoidal 1d chaotic map and its microcontroller implementation”. In: *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE. 2021, pp. 1–4.

[17] Paul Kocher, Joshua Jaffe, and Benjamin Jun. “Differential power analysis”. In: *Annual International Cryptology Conference*. Springer. 1999, pp. 388–397.

[18] Paul C Kocher. “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems”. In: *Annual International Cryptology Conference*. Springer. 1996, pp. 104–113.

[19] Matheus M de A Kotaki and Maximiliam Luppe. “FPGA implementation of a pseudorandom number generator based on k–logistic map”. In: *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*. IEEE. 2020, pp. 1–4.

Datasheets

[4] Intel Corporation. *Intel® MAX® 10 FPGA Device Overview*. Accessed: 2023-09-16. 2023. URL: https://www.intel.com/content/www/us/en/products/programmable/fpga/max-10.html.

[20] Silicon Labs. *CP2102 USB-to-UART Bridge Controller Datasheet*. Accessed: February 12, 2025. 2023. URL: https://www.silabs.com/documents/public/data-sheets/CP2102-9.pdf.

[27] STMicroelectronics. *STM32 32-bit ARM Cortex MCUs*. https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html. Accessed: 2024-07-13.

[30] Terasic Technologies Inc. *DE10-Lite User Manual*. Accessed: 2024-09-23. 2017. URL: https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=China&No=1021&FID=a13a2782811152b477e60203d34b1baa.

A.2 Ευρετήριο κώδικα: VHDL

Adder Package

```
--Generic 1bit full adder
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

package adder is
    component CLA is
        port(A, B   : in  std_logic;
             C_in   : in  std_logic;
             Sum    : out std_logic;
             C_out  : out std_logic);
    end component CLA;

    component CLA_4B is
        port(
        A, B   : in  std_logic_vector(3 downto 0);  -- Inputs
        C_in   : in  std_logic;  -- Carry-in
        Sum    : out std_logic_vector(3 downto 0);  -- Sum output
        C_out  : out std_logic   -- Carry-out
        );
    end component CLA_4B;

    component FA_1B is
        port(
            A                   : in  std_logic;
            B                   : in  std_logic;
            C_in                : in  std_logic;
            C_out               : out std_logic;
            S                   : out std_logic);
    end component FA_1B;

    component FS_1B is
        port(
          A                     : in  std_logic;
          B                     : in  std_logic;
          B_in                  : in  std_logic;
          B_out                 : out std_logic;
          D                     : out std_logic);
    end component FS_1B;

    component FS_32B is
        port(
```

```
            A        : in  std_logic_vector(31 downto 0);
            B        : in  std_logic_vector(31 downto 0);
            Result   : out std_logic_vector(31 downto 0);
            Borrow   : out std_logic);
    end component FS_32B;

    component FA_32B is
        port(
                A        : in  std_logic_vector(31 downto 0);
                B        : in  std_logic_vector(31 downto 0);
                C_in     : in  std_logic;
                Result   : out std_logic_vector(31 downto 0);
                Borrow   : out std_logic);
    end component FA_32B;
end package;
```

Carry Look-Ahead Adder (CLA) 1-Bit Implementation

```
-- carry look ahead adder 1bit implementation-------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity CLA is
    port (
        A, B   : in  std_logic;  -- Inputs
        C_in   : in  std_logic;  -- Carry-in
        Sum    : out std_logic;  -- Sum output
        C_out  : out std_logic   -- Carry-out
    );
end CLA;

architecture behavioral of CLA is
    signal P, G : std_logic;  -- Propagate and Generate signals
begin
    -- Propagate and Generate logic
    P <= A XOR B;  -- Propagate
    G <= A AND B;  -- Generate

    -- Sum and Carry-Out calculation
    Sum   <= P XOR C_in;
    C_out <= G OR (P AND C_in);
end behavioral;
```

Full Adder 1-Bit Implementation

```
-- full adder 1bit implementation-------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity FA_1B is
        port( A                 : in  std_logic;
                B                       : in  std_logic;
                C_in                    : in  std_logic;
                C_out                   : out std_logic;
                S                       : out std_logic);
end FA_1B;

architecture behavioral of FA_1B is
        begin
        C_out <=(A AND B) OR (A AND C_in) OR (B AND C_in);
        S       <= A XOR B XOR C_in;
end behavioral;
```

Full Subtractor 1-Bit Implementation

```
-- full subtractor 1bit implementation--
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity FS_1B is
        port(
                A                       : in  std_logic;
                B                       : in  std_logic;
                B_in                    : in  std_logic;
                B_out                   : out std_logic;
                D                       : out std_logic
                );
end FS_1B;

architecture behavioral of FS_1B is
begin
        D     <=  A XOR B XOR B_in;
        B_out <= (NOT A AND B) OR (NOT(A XOR B) AND B_in);
end behavioral;
```

Full Subtractor 32-Bit Implementation

```vhdl
-- full subtractor 1bit implementation--
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.adder.all;

entity FS_32B is
        port(
        A        : in  std_logic_vector(31 downto 0);
        B        : in  std_logic_vector(31 downto 0);
        Result   : out std_logic_vector(31 downto 0);
        Borrow   : out std_logic);
end FS_32B;

architecture full_sub_32b of FS_32B is
        signal B_internal       :std_logic_vector       (31 downto 0):=(others =>'0');
        signal D_internal       :std_logic_vector       (31 downto 0):=(others =>'0');
        signal doneReg          :std_logic_vector       (31 downto 0):=(others =>'0');

        begin

                FS0: FS_1B port map
                (
                        A=>A(0),
                        B=>B(0),
                        B_in=>'0',
                        B_out=>B_internal(0),
                        D=>D_internal(0)
                );

                gen_sub: for i in 1 to 31 generate
                        FS:FS_1B port map
                        (
                                A=>A(i),
                                B=>B(i),
                                B_in=>B_internal(i-1),
                                B_out=>B_internal(i),
                                D=>D_internal(i)
                        );
                end generate;

        Result<=D_internal;
        Borrow<=B_internal(31);
end full_sub_32b;
```

Multiplier Implementation

```
-- Generic array multiplier
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.adder.all;
--Ripple Carry Adder--
entity mult_64 is
    generic(n: integer :=32);
port(
        M       : in  std_logic_vector(n-1     downto 0);   --Multiplicand
        Q       : in  std_logic_vector(n-1     downto 0);   --Multiplier
        P       : out std_logic_vector((n*2)-1 downto 0));
end mult_64;

architecture behavioral of mult_64 is
    constant arr_size : integer := 32;

    subtype prod_arr is std_logic_vector(arr_size-1 downto 0);
    TYPE    p_arr is array(0 to arr_size) of prod_arr;

    signal  partial_prod, partial_sum, partial_c:p_arr;

    begin
           --First step is to and between multiplier and multiplicand
            par_gen0: for i in 0 to arr_size-1 generate
                par_gen1: for j in 0 to arr_size-1 generate
                    partial_prod(i)(j)<= M(j) AND Q(i);
                end generate;
            --Instantiate partial_c register with 0's
                partial_c(0)(i)<='0';
            end generate;

            partial_sum(0)<= partial_prod(0);
            P(0)<= partial_prod(0)(0);

            -- Second step is adding the partial sums
            add_reg: for i in 1 to arr_size-1 generate
                add_c  : for j in 0 to arr_size-2 generate

                    -- Positional Mapping --
                    PARTIAL_SUMS(i)<=partial_prod(i);
                    FINAL_STEP( P(34)=> P ); END;
```

Logistic Map Implementation

```
-- Logistic Map Implementation--
 library IEEE;
 use IEEE.std_logic_1164.all;
 use ieee.numeric_std.all;
 use work.adder.all;
 use work.multiplier.all;

package cmap is
   component logistic_map is
        generic(n:integer:=32);
        port(   lm_status:out std_logic_vector(3 downto 0);
                        iter:in std_logic_vector(n-1 downto 0);
                        clk,rst:in std_Logic;
                        r:in std_logic_vector(n-1 downto 0);
                        x:in std_logic_vector(n-1 downto 0);
                        c:in std_logic_vector(n-1 downto 0);
                        Y:out std_logic_vector((n*2) -1 downto 0);
                        bitGen:out std_logic);
        end component logistic_map;
end package;

library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;
use work.adder.all;
use work.multiplier.all;
entity logistic_map is
        generic(n:integer:=32);
        port(   lm_status:out std_logic_vector(3 downto 0);
                        iter:in std_logic_vector(n-1 downto 0);
                        clk,rst:in std_Logic;
                        r:in std_logic_vector(n-1 downto 0);
                        x:in std_logic_vector(n-1 downto 0);
                        c:in std_logic_vector(n-1 downto 0);
                        Y:out std_logic_vector((n*2) -1 downto 0);
                        bitGen:out std_logic);
end logistic_map;

-----------------------------------------|
-- Implementation of x(i+1) = rx(i)(1-x)--|
-----------------------------------------|
architecture dataflow of logistic_map is

constant        arr_size     : integer := 31;
```

```
-----------------------------------------------------|
--State machine variables-----------------------------|
-----------------------------------------------------|
type            fsm_state_e is (IDLE,PROCESSING,LOAD,DONE);
type            alu_state_e is (IDLE,MAC,DONE);
-- signal       fsm_state : state_e;
signal      alu_state : alu_state_e;
-----------------------------------------------------|
--State machine variables-----------------------------|
-----------------------------------------------------|

---------------------------------------|
--Logistic map intermidiate registers--|
---------------------------------------|
signal          prpg                            : std_logic:='0';
signal          updCnt                          : std_logic:='0';
signal          iterReg                   : integer:=0;
signal          sBREG                           : std_logic:='0';
signal          cREG, xREG ,rREG          : std_logic_vector(arr_size downto 0);
signal          sDREG                           : std_logic_vector(arr_size downto 0);
signal          mRXREG                          : std_logic_vector((arr_size*2)+1 downto
signal      yREG                                : std_logic_vector((arr_size*2)+1 downto
---------------------------------------|
--Logistic map intermidiate registers--|
---------------------------------------|

begin
        --Generate (1-x)
    SUB_1_X:     FS_32B
                                port map (     A               => cREG,
                                               B                  => xREG,
                                               Result  => sDREG,
                                               Borrow  => sBREG);
        -- Generate r*x(i) --
        MULTIPLY_RX: mult_64
                                port map( M => rREG,
                                               Q => xREG,
                                               P => mRXREG);

        -- Compute output rx(1-x) --
        MULTIPLY_Y: mult_64
                               port map(  M => mRXREG(31 downto 0),
                                               Q => sDREG,
                                               P => yREG);
        --Assign to output y--
```

```
Y<=yreg;
bitGen<=prpg;

lm_status(0)<='1' when rst = '1'                else '0';
lm_status(1)<='1' when alu_state = IDLE else '0';
lm_status(2)<='1' when alu_state = MAC  else '0';
lm_status(3)<='1' when alu_state = DONE else '0';

ALU_PROC: process(clk,rst,alu_state)
begin
        if rst ='1' then
                alu_state<=IDLE;
                rREG     <= (others=>'0');
                cREG     <= (others=>'0');
                xREG     <= (others=>'0');
                sBREG    <= '0';
                updCnt   <= '0';
        elsif (rising_edge(clk)) then
                case alu_state is
                when IDLE =>
                        --Start of logistic map iterations
                        if(iterReg = 0) then --iterReg controlled in counter proces
                                alu_state <= MAC;
                                cREG <= c ;
                                xREG <= x ;
                                rREG <= r ;
                        -- Each iteration that its number hasn't surpassed the give
                        elsif iterReg < to_integer(unsigned(iter))then --In the fir
                                alu_state <= MAC;
                                updCnt<='0';
                        --Iteration number reached.
                        else
                                alu_state <=IDLE;
                                cREG  <= (others=>'0');
                                xREG  <= (others=>'0');
                                rREG  <= (others=>'0');
                        end if;
                when MAC =>
                        alu_state <= DONE;
                        xREG      <= "00000000" & yREG(47 downto 24); --Take the ou
                when DONE =>
                        alu_state<=IDLE;
                        updCnt <='1';
                end case;
        end if;
```

```
        end process ALU_PROC;

        --Counter process uses the number of iteration given by the user and assigns it to
        --Using the iterReg the logistic map IC knows exactly how many times to iterate the
        counter :process(clk,rst,updCnt)
        begin
                if(rst = '1') then
                        iterReg<= 0;
                elsif(rising_edge(clk) and updCnt= '1') then
                        if(iterReg = to_integer(unsigned(iter))) then
                                iterReg<=0;
                        else
                                iterReg<=iterReg+1;
                        end if;
                end if;
        end process counter;

        BIT_GEN: process(clk,rst,updCnt,xReg(0))
        begin
                if(rst ='1') then
                        prpg <= '0';
                elsif(rising_edge(clk) and updCnt ='1' and xReg(0)='0')then
                        prpg <= '0';
                elsif(rising_edge(clk) and updCnt ='1' and xReg(0)='1')then
                        prpg <= '1';
                end if;
        end process BIT_GEN;
end dataflow;
```

Logistic Map Implementation

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;
LIBRARY altera_mf;
USE altera_mf.ALL; -- LIB For PLL
use work.adder.all;
use work.multiplier.all;
use work.cmap.all;

entity RLM24 is
    port(clk,rst,en : in     std_logic;
         bitGen     : out    std_logic;
         lm_status  : out    std_logic_vector(3 downto 0);
         rlm_status_led,rst_led : out    std_logic);
```

```
end RLM24;

architecture behavioral of RLM24 is
    constant r      :   std_logic_vector(31 downto 0)   :="00000000000000000000000000000100
    constant x0     :   std_logic_vector(31 downto 0)   :="00000000000110011001100110011001
    constant c      :   std_logic_vector(31 downto 0)   :="00000001000000000000000000000000
    constant iter   :   std_logic_vector(31 downto 0)   :="00000000000011110100001001000000

    component pll is
        PORT
        (
            areset              : IN STD_LOGIC  := '0';
            inclk0              : IN STD_LOGIC  := '0';
            c0              : OUT STD_LOGIC ;
            locked              : OUT STD_LOGIC
        );
    end component pll;

    signal rR,xR,cR,iR      :std_logic_vector(31 downto 0):=(others=>'0');
    signal yR               :std_logic_vector(63 downto 0):=(others=>'0');

    signal clk10Mhz         :std_logic;
    signal clkLocked        :std_logic;
    begin

    --Gen 10Mhz Clock using PLL--
        pll10Mhz: pll port map (areset=>rst,
                            inclk0=>clk,
                            c0=>clk10Mhz,
                            locked=>clkLocked);
    --Run logistic map
    run_rlm: logistic_map port map(
                                    lm_status=>lm_status,
                                    iter=>iR,
                                    clk=>clk10Mhz,
                                    rst=>rst,
                                    r=>rR,
                                    x=>xR,
                                    c=>cR,
                                    Y=>yR,
                                    bitGen=>bitGen
                                  );

        --Status Leds--
    rst_led        <= '1' when rst ='1' else '0';
```

```
    rlm_status_led <= '1' when en ='1'  else '0';

    rlm_proc:process(clk,rst,en)
    begin
        if(en = '1') then
            rR<=r;
            xR<=x0;
            cR<=c;
            iR<=iter;

        else
            rR<=(others=>'0');
            xR<=(others=>'0');
            cR<=(others=>'0');
            iR<=(others=>'0');

        end if;
    end process;
end behavioral;
```