Reed Solomon Decoder with UART

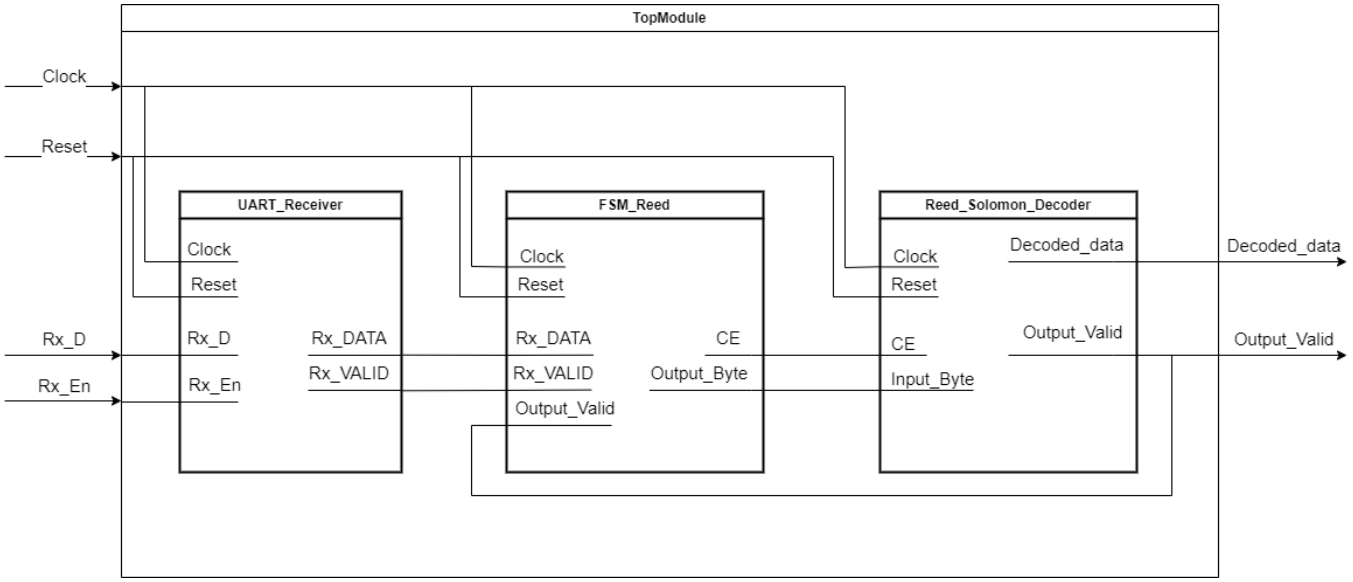
Δημήτρης Τσαλαπάτας - Γιάννης Ρείνος

10 Ιουλίου 2023

Περιεχόμενα

1 General Description of Design

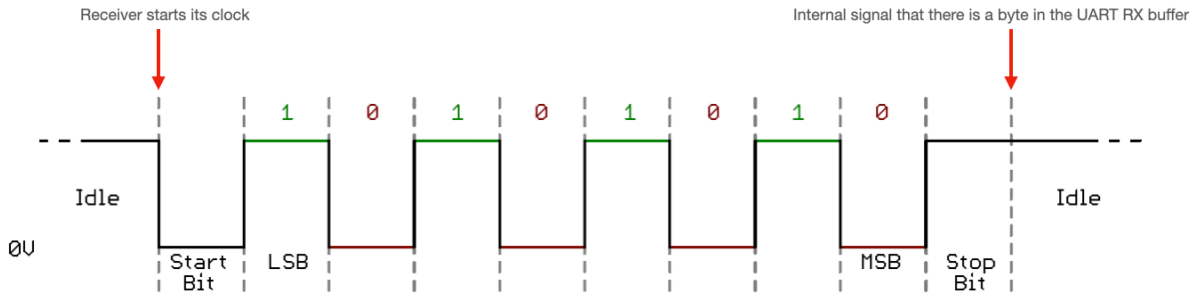
Το design που επιλέξαμε να υλοποιήσουμε είναι ένας `Reed Solomon Decoder` ο οποίος λαμβάνει δεδομένα μέσω ενός UART Receiver. Θεωρούμε ότι λαμβανουμε τα δεδομένα από έναν αντίστοιχο encoder ο οποίος τα στέλνει μέσω UART Transmitter. Τα δύο modules σύνδεονται μέσω μίας FSM που υλοποιήσαμε, η οποία συλλέγει μέσω του Receiver τα δεδομένα και κάθε φορά που ένα μήνυμα είναι έτοιμο το προωθεί στον Decoder για να γίνει η αποκωδικοποίηση.
Παρατίθεται παρακάτω το top level Dataflow για το design



Σχήμα 1: Top view Dataflow

2 UART Receiver

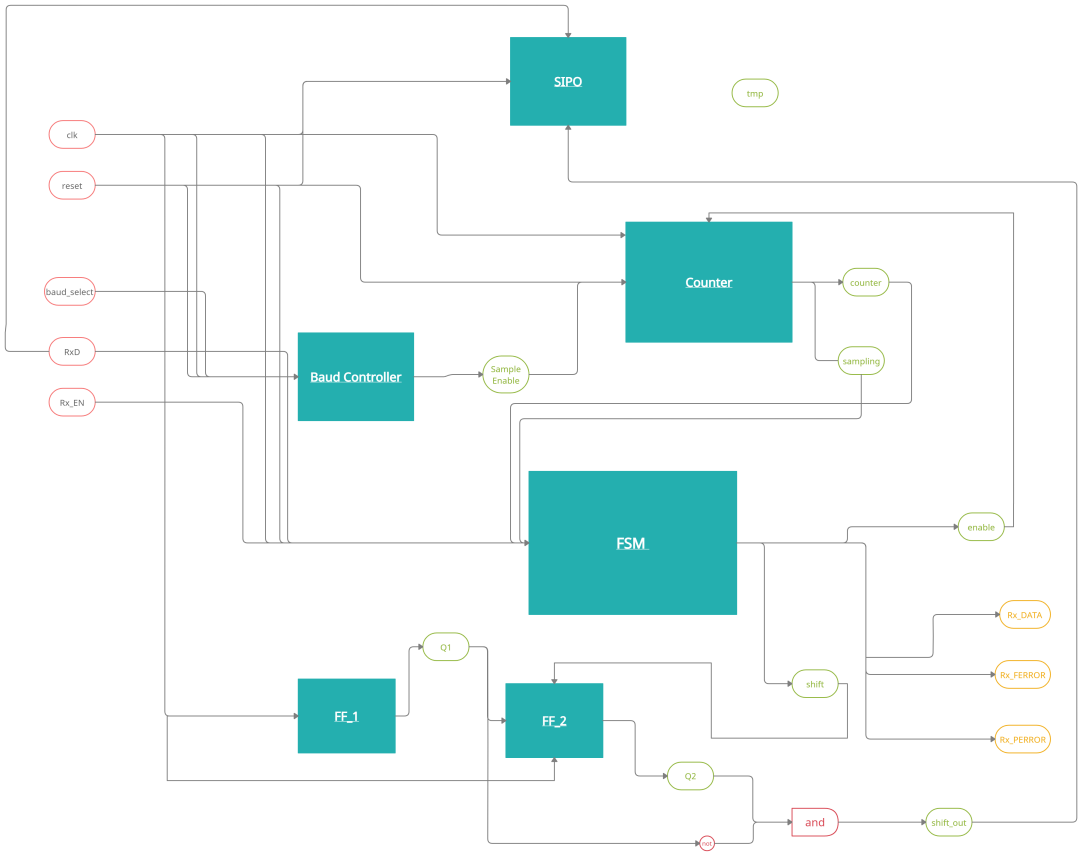
Αρχικά, χρησιμοποιείται στο design ένας `UART` Receiver δικιάς μας υλοποίησης, ο οποίος λαμβάνει πακέτα των 8 bits σε όλες τις πιθανές ταχύτητες που υποστηρίζονται από το πρωτόκολλο του UART. Ο receiver λαμβάνει σειριακά 11 bits από τον Transmitter. Αρχικά ο transmitter στέλνει το `start bit`, ώστε να ξεκινήσει η επικοινωνία, στην συνέχεια ακολουθούν τα 8 bits δεδομένα, ένα parity bit και τέλος ένα stop bit όπου σηματοδοτεί την λήξη της επικοινωνίας. Μέσω της FSM ο receiver δειγματολειπτεί στην μέση του κάθε bit που στέλνει ο Transmitter και εφόσον λάβει και τα 8 bits ελέγχει το parity του πακέτου και εάν είναι ίδιο με αυτό που έλαβε σημαίνει ότι τα δεδομένα στάλθηκαν σωστά. Όταν ληφθεί το stop bit τελειώνει η αποστολή του συγκεκριμένου πακέτου. Τέλος, έχουν ληφθεί τα δεδομένα, τα στέλνει παράλληλα σαν output μαζί με ένα σήμα `Rx_VALID` για να δηλώσει ότι τα δεδομένα είναι έγκυρα.



Σχήμα 2: UART Communication example

2.1 UART dataflow

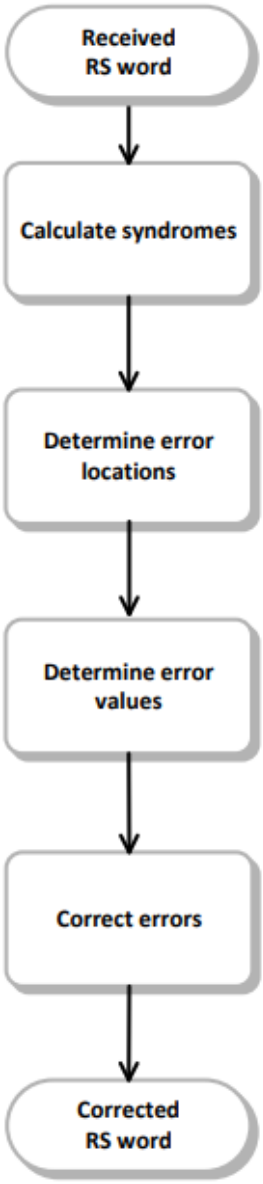
Παρακάτω φαίνεται το dataflow για το UART στο οποίο περιέχεται το module του baud controller, λειτουργία του οποίου είναι να παρέχει το σήμα για δειγματοληψία στον Receiver ανάλογα με την εκάστοτε ταχύτητα. Επίσης, υπάρχει μία FSM η οποία ελέγχει όλη την λειτουργία του κυκλώματος με την χρήση ενός counter, και τέλος χρησιμοποιείται και ένα SIPO (serial to pararel shifter) για να μετατρέπονται τα δεδομένα από σειριακά σε παράλληλα.



Σχήμα 3: UART dataflow

3 Reed Solomon Decoder

Ο Reed-Solomon είναι ένας αλγόριθμος κωδικοποίησης που χρησιμοποιείται για τη διόρθωση σφαλμάτων στη μετάδοση και αποθήκευση δεδομένων. Ο κώδικας υλοποιείται με τη δημιουργία ενός συνόλου πολυωνύμων που χρησιμοποιούνται για την κωδικοποίηση των δεδομένων. Περιλαμβάνει κάποια parity bits, τα οποία του επιτρέπουν να ανιχνεύει και να διορθώνει σφάλματα με τη χρήση μαθηματικών αλγορίθμων. Στο design μας, χρησιμοποιούμε τον Reed Solomon Decoder, ο οποίος λαμβάνει ένα `codeword` (κωδικοποιημένο από έναν encoder) μήκους 204 bytes και αφού τα αποκωδικοποιήση βγάζει output ένα `codeword` μήκους 188 bytes.

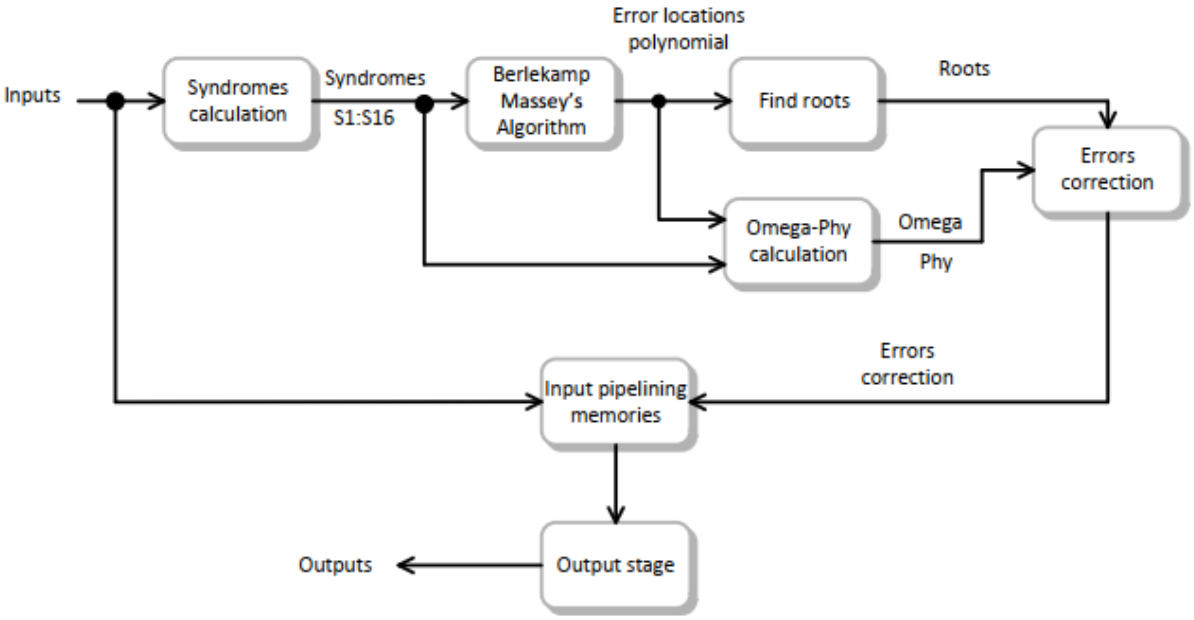


Σχήμα 4: Reed Solomon Decoder Flowchart

3.1 Steps of Operation

Για κάθε εισερχόμενο codeword, υπολογίζονται τα syndromes και το εισερχόμενο codeword αποθηκεύεται στα input pipelining memories.

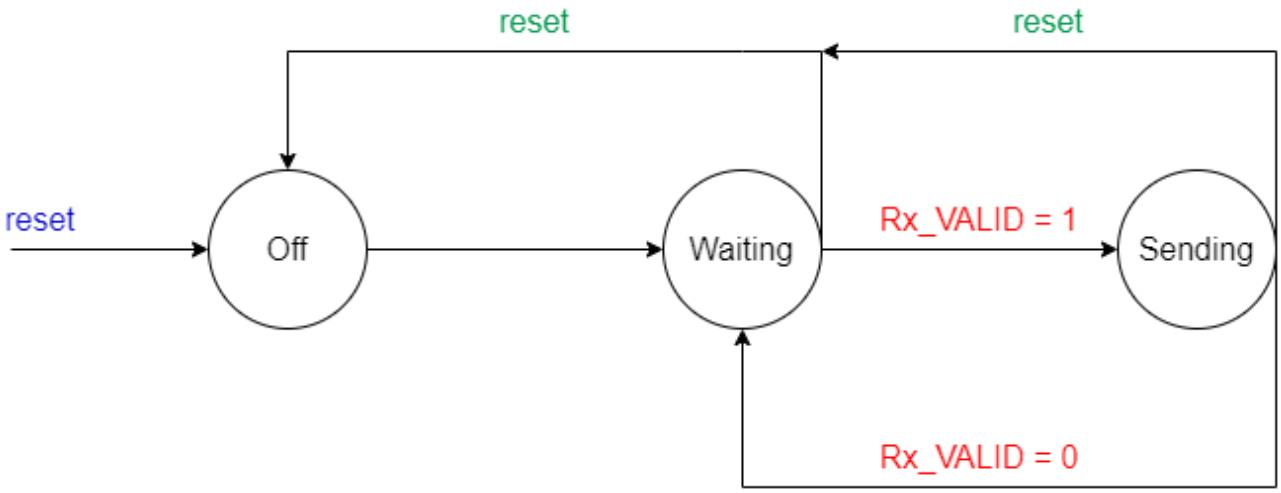
- Τα syndromes θα χρησιμοποιηθούν για να πάρουμε το πολυώνυμο εντοπισμού σφάλματος (Lambda) χρησιμοποιώντας τον αλγόριθμο Berlekamp-Massey. Οι συντελεστές θα προέρχονται από το L1 έως το L8 που θα αντιπροσωπεύουν το Lambda στην ακόλουθη μορφή: $Lambda = 1 + L1 * X + L2 * X^2 + L3 * X^3 + ... + L8 * X^8$
- Το πολυώνυμο Lambda θα εισαχθεί σε δύο μπλοκ που λειτουργούν ταυτόχρονα, αυτά τα δύο μπλοκ είναι:
  - ❖ Εύρεση ριζών: εύρεση ριζών του πολυωνύμου Lambda (έως 8 ρίζες).
  - ❖ Υπολογισμός Omega-Phy: τα syndromes είναι επίσης είσοδος σε αυτό το μπλοκ για να υπολογίσουν δύο πολυώνυμα.
    - ✓ $Omega = Lambda * syndromes$
    - ✓ $Phy = derivative(Lambda)$
- Η μονάδα διόρθωσης σφαλμάτων υπολογίζει τις θέσεις και τις τιμές των σφαλμάτων, και στη συνέχεια γράφει τις διορθωμένες τιμές στα pipelining memories, όπως φαίνεται στο σχήμα 5
- Το output stage block απελευθερώνει τη διορθωμένη έξοδο από τα pipelining memories και επίσης παράγει handshaking flags.



Σχήμα 5: Block diagram of Reed Solomon Decoder

4 FSM Υλοποίηση

Υλοποιήσαμε μια Moore FSM με 3 state, για να συνδέσουμε τα δύο design μεταξύ τους. Κάθε φορά που ο receiver λαμβάνει δεδομένα και τα οδηγεί προς το output, το σήμα `Rx_VALID` γίνεται 1 και η FSM προωθεί τα δεδομένα στον Reed-Solomon Decoder και ανεβάζει το σήμα `CE` στο 1 για ένα κύκλο ρολογιού. Ουσιαστικά, η FSM στέλνει συνεχώς δεδομένα από τον Receiver προς τον Decoder και οταν εκείνος λάβει 204 bytes ξεκινάει την αποκωδικοποίηση. Η ροή των δεδομένων προς τον decoder γίνεται συνεχώς, καθώς η αποκωδικοποίηση γίνεται αρκετά πιο γρήγορα από την λήψη των δεδομένων από τον Receiver.

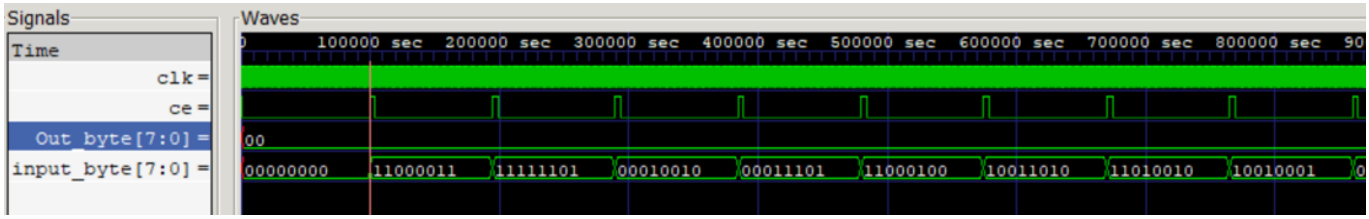


Σχήμα 6: Moore FSM for connection

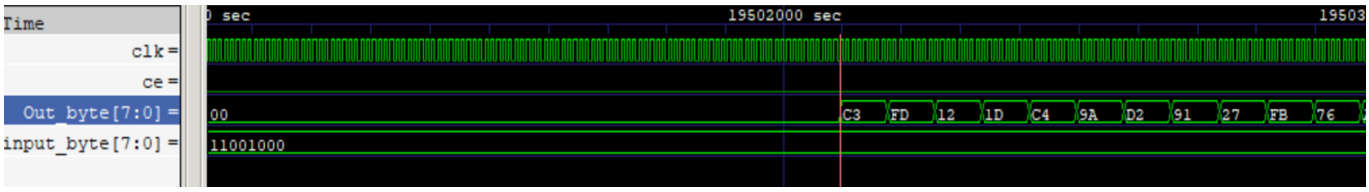
5 Testbench and Simulation

Για το Simulation τροποποιήσαμε το Testbench του Reed-Solomon Decoder με σκοπό να προσομοιώσουμε την συνεχή ροή δεδομένων από ένα UART Transmitter προς τον UART Receiver του design μας. Το Testbench ελέγχει 100 ακολουθίες αριθμών των 204 bytes η καθεμιά, τα οποία διαβάζει από ένα αρχείο και συγκρίνει τα αποτελέσματα του Decoder μέσω ενός άλλου αρχείου που περιέχει τα σωστά αποτελέσματα. Οπότε, το testbench κάνει αυτόματα τον έλεγχο και εάν βρεθεί κάποιο λάθος αναφέρει σε ποια γραμμή έγινε.
Παράλληλα, μαζί με το design του decoder δόθηκε και ένα απλό script σε `Matlab` όπου κάνει generate random περιπτώσεις 100 τέτοιων ακολουθιών και ταυτόχρονα παράγει και το αποτέλεσμα μετά την αποκωδικοποίηση. Οπότε μπορούν να δημιουργηθούν αμέτρητα tests για το συγκεκριμένο design. Εφόσον το input 204 bytes τα test που μπορουν να παραχθούν είναι άπειρα, οπότε για να επιβεβαιωθεί η σωστή λειτουργία του design θα έπρεπε να τεσταριστούν οριακές περιπτώσεις με βάση τον μαθηματικό αλγόριθμο που ακολουθεί το Reed Solomon.
Παρακάτω φαίνονται κάποιες εικόνες από τις κυματομορφές του simulation, όπου στην πρώτη απεικονίζεται η μετάδοση των δεδομένων από τον Receiver στο Decoder. Ενώ,

στην δεύτερη παρατηρούμε την διαδικασία αποκωδικοποίησης των δεδομένων από τον decoder.



Σχήμα 7: Data Transmission Waveforms



Σχήμα 8: Decoded Data Output Waveforms

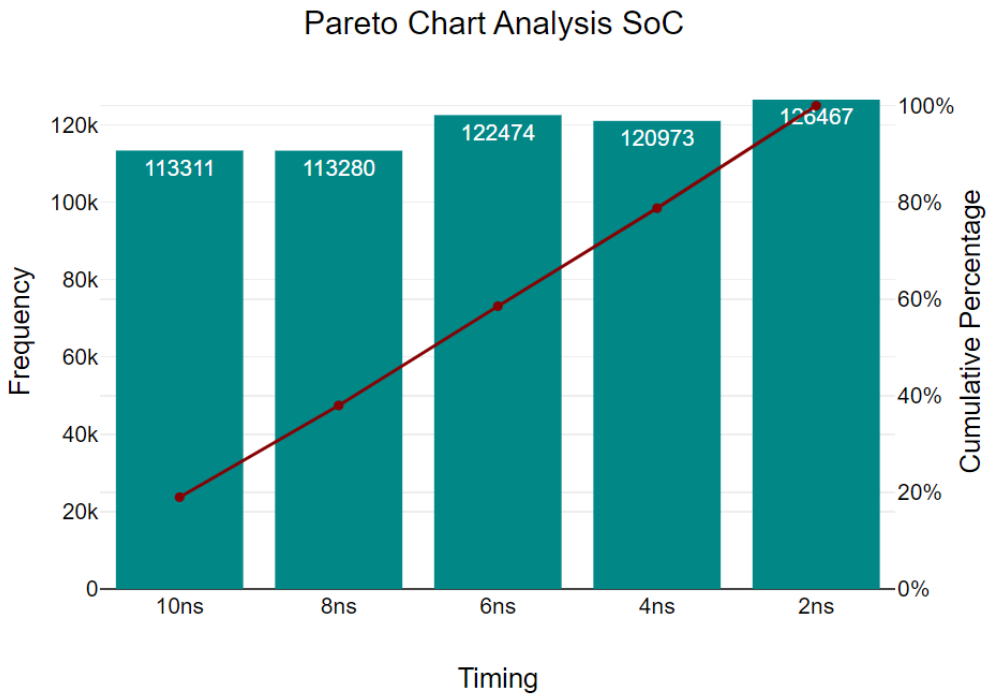
6 Real World Application

Ο κώδικας Reed-Solomon χρησιμοποιείται συνήθως σε διάφορες εφαρμογές, όπως δορυφορικές επικοινωνίες και ψηφιακές συσκευές αποθήκευσης. Είναι ιδιαίτερα χρήσιμος σε περιπτώσεις όπου υπάρχει μεγάλη πιθανότητα εμφάνισης σφαλμάτων κατά τη μετάδοση ή την αποθήκευση δεδομένων, όπως σε θορυβώδη κανάλια επικοινωνίας ή σε περιβάλλοντα με υψηλές ηλεκτρομαγνητικές παρεμβολές.
Οπότε το συγκεκριμένο design βρίσκει εφαρμογή σε όλες τις μεταφορές δεδομένων οι οποίες είναι ευάλωτες σε αλλοίωση των δεδομένων, δεδομένου ότι το transmission γίνεται με το πρωτόκολλο UART.

7 Synthesis Process

7.1 Top-Down view

Για την διαδικασία της σύνθεσης χρειάστηκε να δημιουργήσουμε ένα script TCL συμβατό με τις εντολές του `Genus` της `Cadence`. Γενικότερα, η μέθοδος του top down στην σύνθεση είναι ότι παρέχεις στο εργαλείο όλα τα αρχεία της verilog και υλοποιεί την σύνθεση για όλο το κύκλωμα με βάση την ιεραρχία.
Ξεκινήσαμε να δοκιμάζουμε για διάφορα ρολόγια σύνθεση, ξεκινώντας από τα 10ns έως τα 2ns όπου εκεί παραβιάζονταν το **slack** (δηλαδή μικρότερο του 50ps το οποίο είναι απαγορευτικό διότι θα προστεθούν και οι καθυστερήσεις των καλωδίων) οπότε ως optimal θεωρήθηκε το design με ρολόι 4ns περίοδο.
Παρακάτω φαίνεται η καμπύλη pareto με την περίοδο σε σχέση με το area



Σχήμα 9: Pareto chart

8 Post-Synthesis Simulation

Μετά την σύνθεση πήραμε το gate level design σε verilog και το sdf αρχείο που παράγει το genus με τις καθυστερήσεις της κάθε πύλης και έπρεπε να το προσωμοιώσουμε με το testbench, που είχαμε ήδη υλοποιημένο, στο Incisive.
Εν τέλη, λόγω του όγκου του design η προσομοίωση του gate level στο Incisive ήταν αδύνατη, οπότε προσωμοιώσαμε το gate level στο Icarus χωρίς το sdf θεωρώντας πως οι καθυστερήσεις δεν θα επηρεάσουν την λειτουργία του.
??εικονα??

9 Placement Process

Για την διαδικασία του placement χρειάστηκε να υλοποιήσουμε ένα tcl script συμβατό με τις εντολές του `Innovus` της `Cadence`. Ξεκινήσαμε την διαδικασία του placement με τις default ρυθμίσεις που περιγράφονταν στο αντίστοιχο pdf. Ωστόσο, παρατηρήσαμε ότι στα timing reports είχαμε violations σε πολλά paths, κάποια από τα οποία δεν διορθώθηκαν μετά το optimisation.
Για να το διορθώσουμε αυξήσαμε το core utilization έως το 80% και τα layers από 6 σε 7 αλλά δεν υπήρξε ουσιαστικό αποτέλεσμα. Τελικά, αλλάξαμε χειροκίνητα τα dimensions του design από $380x380$ σε $450x450$ και με τις υπόλοιπες ρυθμίσεις ως έχει και ενώ είχαμε αρνητικό slack μετά το optimisation το slack έγινε 0 ενώ επίσης και τα violating paths έγιναν 0 ενώ το density αυξήθηκε στο 61.18%. Οπότε θεωρήσαμε ότι εφόσον δεν έχει violations είναι ιδανικό για να συνεχίσει η διαδικασία.

Στην συνέχεια δημιουργήσαμε το clock tree synthesis ορίζοντας τους buffers και τους inverters που χρειάζονται για το CTS και κάναμε και optimize το αποτέλεσμα.

	slack	violating paths	power	density
Timing Report before opt	-73.96	8452/23636	-	-
Timing Report after opt	0	0/23636	total: 24.99	61.18%

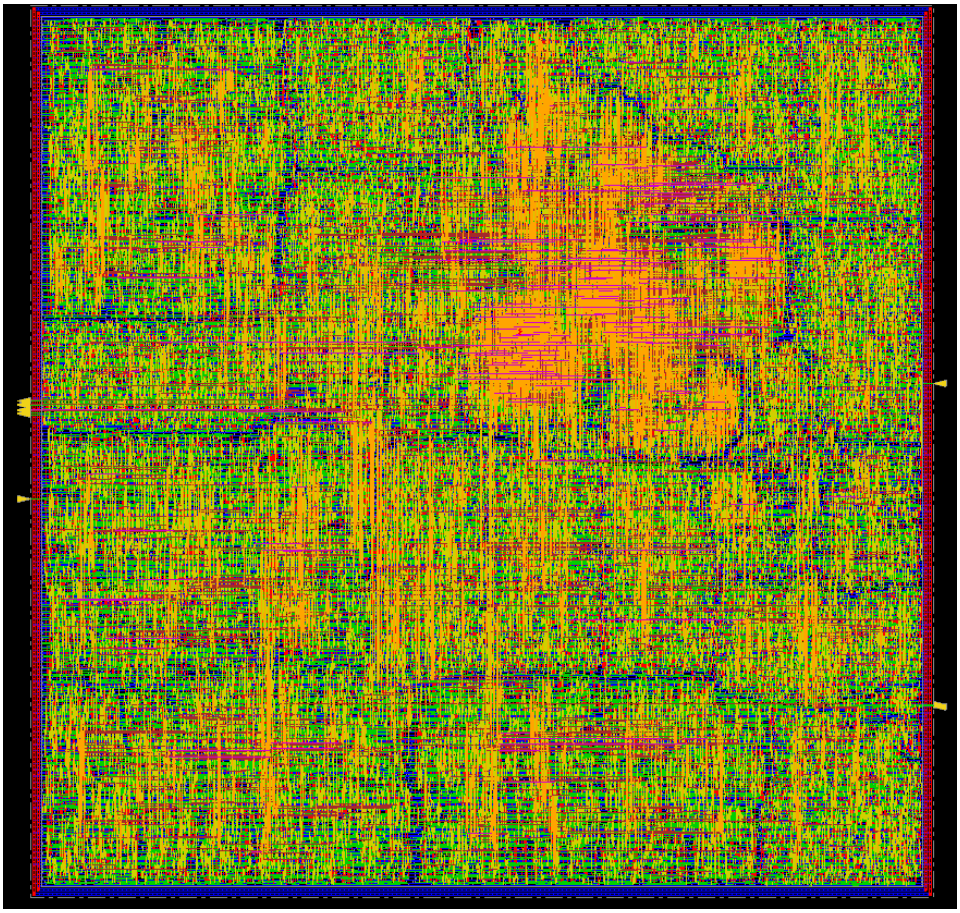
Σχήμα 10: Timing Comparison

Ακολούθησε η υπόλοιπη διαδικασία που περιλαμβάνει το nano-routing και το optimize αυτού και λάβαμε τα τελικά reports που επιβεβαιώνουν την λειτουργικότητα του κυκλώματος και το power analysis το οποίο φαίνεται αναλυτικά στην εικόνα που ακολουθεί.

Total Internal Power:	14.95751339	59.86%
Total Switching Power:	8.39434638	33.59%
Total Leakage Power:	1.63629881	6.55%
Total Power:	24.98815868	

Σχήμα 11: Power Results

Τέλος, ακολούθησε η διαδικασία του verification όπου ελέγχθηκε το geomtry και το connectivity όπου κανένα από τα δύο δεν εμφάνισε ούτε errors ούτε warnings. Ακολουθεί εικόνα από το placement όλων των cells και των καλωδίων όπως φαίνεται στο Innovus.
Παρακάτω φαίνονται τα συνολικά αποτελέσματα του placement πριν και μετά από το optimization



Σχήμα 12: Final Design