Εργασία HARDWARE I

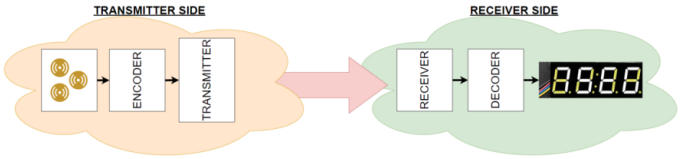
Αλεξόπουλος Δημήτριος 10091
aadimitri@ece.auth.gr
Μανώλης Δημήτριος 10104
dmanolis@ece.auth.gr

Περιεχόμενα

Κατάλογος Σχημάτων

1 Εισαγωγή

Στην παρούσα εργασία θα υλοποιήσουμε σε Verilog, ένα ψηφιακό σύστημα αποστολέα-δέκτη, όπως παρουσιάζεται στην παρακάτω εικόνα. Η επικοινωνία θα πραγματοποιηθεί με χρήση του πρωτοκόλλου UART, ένα σειριακό, ασύγχρονο πρωτόκολλο επικοινωνίας. Ο αποστολέας θα στέλνει ένα σύνολο από δεδομένα μέσω της σειριακής γραμμής και ο δέκτης θα εμφανίζει τις ενδείξεις που λαμβάνει σε μία οθόνη τεσσάρων LED 7-τμημάτων.



Σχήμα 1.1: Σύστημα Πομπού-Δέκτη *UART*

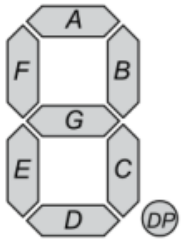
Η εργασία χωρίζεται σε τρία μέρη. Στο πρώτο μέρος θα υλοποιήσουμε έναν οδηγό ενδείξεων LED 7-τμημάτων, ενώ στο δεύτερο μέρος θα γίνει η υλοποίηση του συστήματος σειριακής επικοινωνίας UART με τον σχεδιασμό ενός ελεγκτή Baud Rate και του πομπού-δέκτη. Αυτά θα συνδεθούν στο τρίτο μέρος της εργασίας, όπου τα δεδομένα που στέλνει ο αποστολέας λαμβάνονται από τον δέκτη και προβάλλονται στην οθόνη LED.

Για την υλοποίηση των κυκλωμάτων σε Verilog χρησιμοποιήσαμε το περιβάλλον EDA Playground για την προσομοίωση και το πρόγραμμα Xilinx για την σύνθεση και την εξαγωγή των σχηματικών διαγραμμάτων. Επιπλέον, για την σχεδιαγραμματική απεικόνιση των FSM της εργασίας χρησιμοποιήσαμε το εργαλίο της ιστοσελίδας Excalidraw.

2 Μέρος Α

Στο πρώτο μέρος της εργασίας θα ξεκινήσουμε με την υλοποίηση του οδηγού της οθόνης LED. Στόχος μας είναι θέτοντας σαν είσοδο μία ακολουθία από 16 bits να προβληθεί στην οθόνη μία ακολουθία αλφαριθμητικών και ειδικών χαρακτήρων.

Η οθόνη 7 τμημάτων αποτελεί την απλούστερη μορφή αναπαράστασης ενός αλφαριθμητικού ψηφίου. Το κάθε ψηφίο αποτελείται από τα 7 συν 1 (δεκαδικό σημείο) σήματα A, B, C, D, E, F, G και DP, όπως φαίνεται στην Εικόνα.

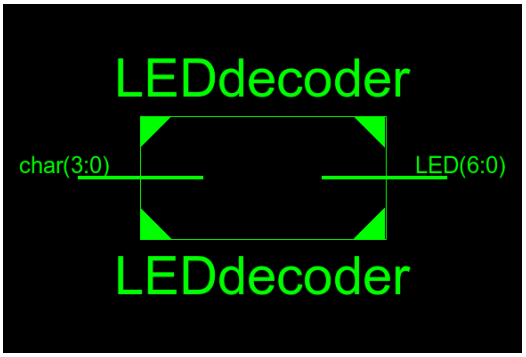


Σχήμα 2.1: Οθόνη *LED 7* τμημάτων

Για να μπορέσουμε να εμφανίσουμε ένα μήνυμα με περισσότερα από ένα ψηφία, χρησιμοποιούμε περισσότερες οθόνες 7-τμημάτων τη μία δίπλα στην άλλη. Για την αποφυγή πολλαπλών σημάτων, οι οθόνες χρησιμοποιούν τα ίδια σήματα A, B, C, D, E, F, G και DP, ωστόσο για να επισημανθεί ποιά οθόνη θα προβάλει το ψηφίο την εκάστοτε στιγμή, χρησιμοποιούνται οι άνοδοι AN, με την βοήθεια των οποίων οι προβαλλόμενες τιμές ανανεώνονται ασύγχρονα στην οθόνη με καθορισμένη περίοδο.

2.1 Αποκωδικοποιητής 7-τμημάτων

Υλοποιούμε, αρχικά, ένα συνδυαστικό αποκωδικοποιητή, ο οποίος θα έχει ως είσοδο το ψηφίο ή χαρακτήρα που θέλουμε να εμφανιστεί στην οθόνη, και ως έξοδο τις τιμές οδήγησης των 7 τμημάτων. Το σχηματικό διάγραμμα του κυκλώματος φαίνεται παρακάτω:



Σχήμα 2.2: Σχηματικό Αποκωδικοποιητή 7-τμημάτων

Για τον έλεγχο της σωστής λειτουργίας του κυκλώματος σχεδιάσαμε ένα κύκλωμα ελέγχου testbench και τρέξαμε την προσομοίωση θέτοντας κάποιες δοκιμαστικές τιμές char σαν είσοδο και επιβεβαιώντας ότι οι τιμές LED της εξόδου πράγματι ανταποκρίνονται στην πραγματικότητα και οδηγούν σωστά τις ενδείξεις της οθόνης 7-τμημάτων.

Σχήμα 2.3: Αποτελέσματα προσομοιώσεων *testbench* του *LED Decoder*

Τα αποτελέσματα φαίνονται παραπάνω και αποδεικνύουν την σωστή λειτουργία του κυκλώματος αποκωδικοποίησης. Οι τιμές του LED μεταφράζονται σε ενεργές περιοχές της οθόνης 7-τμημάτων όπως φάινονται στο Σχήμα 2.1.

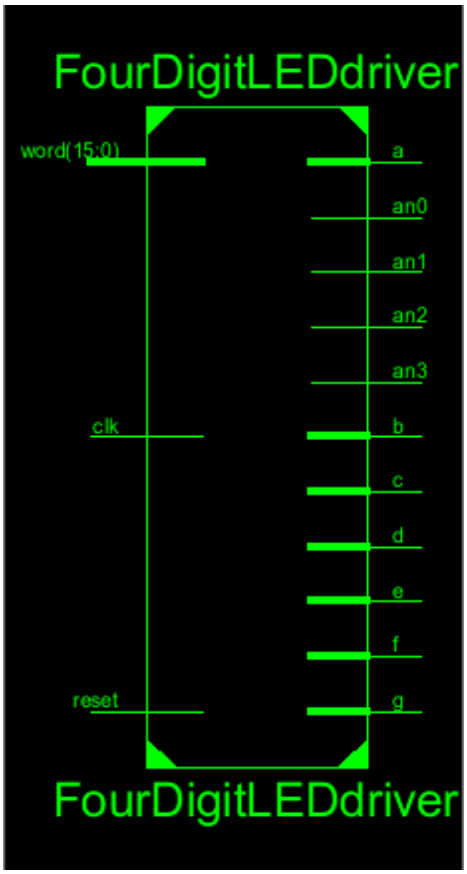
2.2 Οδήγηση των Ανόδων

Όπως είπαμε και παραπάνω, για τη χρήση των τεσσάρων ψηφίων απαιτείται η εναλλάξ οδήγηση του κάθε ψηφίου με κάποια προκαθορισμένη ελάχιστη καθυστέρηση φόρτισης. Διαιρώντας το ρολόι κατάλληλα οδηγούμε τα σήματα των ανόδων όπως στην παρακάτω εικόνα:

Τιμή μετρητή	AN3	AN2	AN1	AN0
1111	1	1	1	1
1110	0	1	1	1
1101	1	1	1	1
1100	1	1	1	1
1011	1	1	1	1
1010	1	0	1	1
1001	1	1	1	1
1000	1	1	1	1
0111	1	1	1	1
0110	1	1	0	1
0101	1	1	1	1
0100	1	1	1	1
0011	1	1	1	1
0010	1	1	1	0
0001	1	1	1	1
0000	1	1	1	1

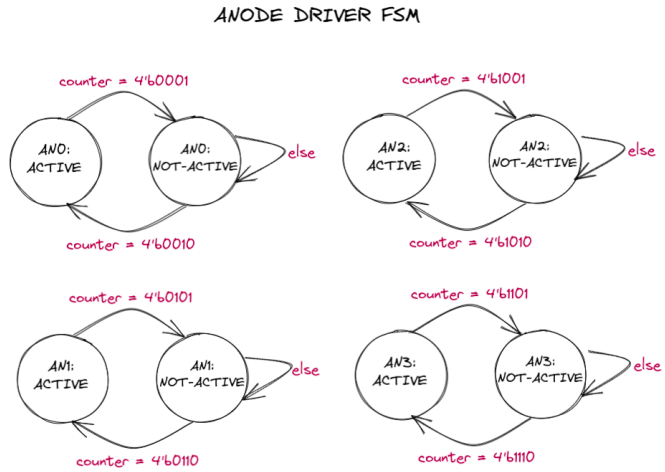
Σχήμα 2.4: Μετρητής για την οδήγηση των ανόδων

Η κάθε άνοδος ενεργοποιείται όταν η τιμή που της αντιστοιχεί μηδενίζεται, δηλαδή οι άνοδοι είναι active low. Ο περιστροφικός αυτός μετρητής, λοιπόν, επιτυγχάνει την μη-επικαλυπτόμενη οδήγηση των ανόδων. Το σχηματικό διάγραμμα του κυκλώματος φαίνεται παρακάτω:



Σχήμα 2.5: Σχηματικό οδηγού των ανόδων

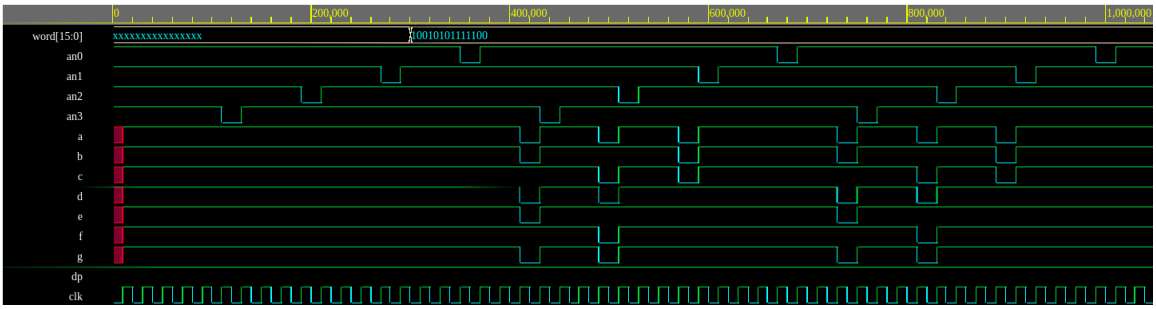
Στην παρακάτω εικόνα παρατίθεται το σχηματικό του FSM του οδηγού των ανόδων. Σε αυτό φαίνεται η εναλλαγή των καταστάσεων του κυκλώματος ανάλογα με τις εκάστοτε συνθήκες:



Σχήμα 2.6: Σχηματικό του *FSM* του οδηγού των ανόδων

Και πάλι για τον έλεγχο της σωστής λειτουργίας του κυκλώματος σχεδιάσαμε ένα κύκλωμα

ελέγχου testbench και τρέξαμε την προσομοίωση θέτοντας κάποιες δοκιμαστικές τιμές λέξεων των 16bit σαν είσοδο και επιβεβαιώντας ότι οι τιμές των ανόδων της εξόδου πράγματι ανταποκρίνονται στην πραγματικότητα και οδηγούν σωστά την κάθε μία από τις τέσσερεις LED οθόνες που θα χρησιμοποιηθούν για την απεικόνιση των τεσσάρων αλφαριθμητικών και ειδικών χαρακτήρων.



Σχήμα 2.7: Αποτελέσματα προσομοιώσεων *testbench* του *Anode Driver*

Τα αποτελέσματα φαίνονται παραπάνω και αποδεικνύουν την σωστή λειτουργία του κυκλώματος οδήγησης των ανόδων. Οι τιμές του word μεταφράζονται διαδοχικά και μοιράζονται στις επιμέρους οθόνες LED, όπως ορίστηκε από το Σχήμα 2.4.

3 Μέρος Β

Σε αυτό το Μέρος της εργασίας υλοποιήσαμε ένα σύστημα σειριακής επικοινωνίας, το οποίο θα χρησιμοποιεί το πρωτόκολλο UART (Universal Asynchronous Receiver Transmitter - Γενικού Ασύγχρονου Δέκτη Αποστολέα). Το σύστημα αποτελείται από έναν UART Αποστολέα και έναν UART Δέκτη, οι οποίοι μεταφέρουν δεδομένα στη μια κατεύθυνση, από τον Αποστολέα προς στον Δέκτη, μέσω μιας σειριακής σύνδεσης ενός σήματος.

3.1 Ελεγκτής **Baud Rate**

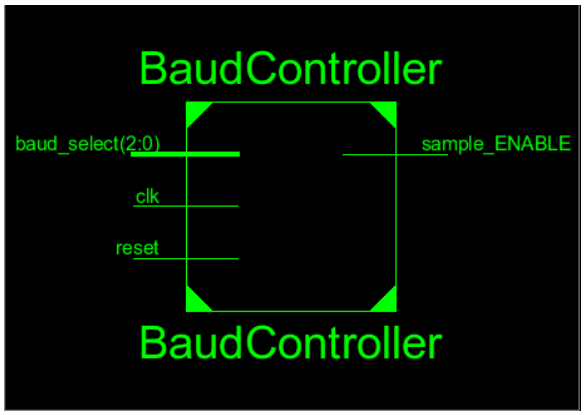
Για τον ορθό ρυθμό δειγματοληψίας, έτσι ώστε να μην χάνονται ή να πολλαπλασιάζονται δεδομένα στο πρωτόκολλο UART, ο Αποστολέας και ο Δέκτης πρέπει να προσυμφωνούν στην ταχύτητα της μεταξύ τους επικοινωνίας σε μονάδες Baud (bits/sec). Η προσυμφωνία αυτή δεν είναι μέρος του πρωτοκόλλου επικοινωνίας, αλλά επιλέγεται από τον παρακάτω πίνακα στον οποίο φαίνεται κι η σχετική κωδικοποίησή της από το 3-bit σήμα, το οποίο και την επιλέγει.

BAUD_SEL	**Baud Rate**
000	300 bits/sec
001	1200 bits/sec
010	4800 bits/sec
011	9600 bits/sec
100	19200 bits/sec
101	38400 bits/sec
110	57600 bits/sec
111	115200 bits/sec

Σχήμα 3.1: *Baud Rates*

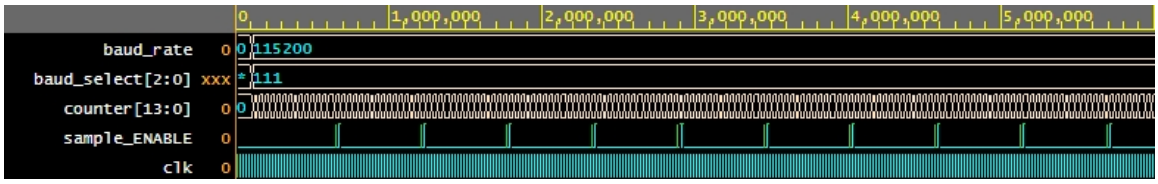
Στόχος του ελεγκτή Baud Rate είναι να παρέχει το κατάλληλο σήμα δειγματοληψίας, ανάλογα με τον επιλεγμένο Rate. Για τον Ελεγκτή Βαυδ Ρατε, χρησιμοποιούμε ρολόι 50 ΜΗζ, ενώ ταυτόχρονα παράγουμε το σήμα δειγματοληψίας μέσω ενός μετρητή κύκλων ρολογιού και μετράμε, με το μικρότερο δυνατό ποσοστό λάθους μέσω σωστών στρογγυλοποιήσεων, την ποσότητα Tsc = 1/(16×BaudRate), η οποία αντιστοιχεί στην περίοδο δειγματοληψίας, σε αριθμό κύκλων.

Στην παρακάτω εικόνα φαίνονται τα inputs/outputs του Baud Controller. Την εξάγαμε μετά από Synthesis στο πρόγραμμα Xilinx. Το component αυτό αποτελείται από τρεις εισόδους, δύο εκ των οποίων είναι το ρολόι και το σήμα reset ενώ η τρίτη είναι η επιλογή της συχνότητας. Η μία έξοδος είναι το περιοδικό σήμα Sample_ENABLE το οποίο θα χρησιμοποιούν τόσο ο αποστολέας όσο κι ο δέκτης.



Σχήμα 3.2: Σχηματικό του *component*

Τρέξαμε την προσομοίωση επιλέγοντας ρυθμό 115200bits/sec. Στο σχήμα 3.3 τοποθετήσαμε τα baud_rate και baud_select ώστε να αποδεικνύεται ότι η επιλογή του ρυθμού από το testbench λειτούργησε σωστά. Επίσης φαίνονται το ρολόι, ο μετρητής και το αποτέλεσμα του sample_ENABLE, αποτέλεσμα το οποίο βγαίνει στρογγυλοποιώντας την σχέση: $\frac{10^9}{16 \times 20 \times baud\_rate}$ και χρησιμοποιώντας το ως μέγιστη τιμή του μετρητή, μετά από την οποία αυτός μηδενίζεται.



Σχήμα 3.3: Αποτέλεσμα προσομοίωσης

Το αποτέλεσμα είναι το αναμενόμενο και το simulation έτρεξε χωρίς προβλήματα.

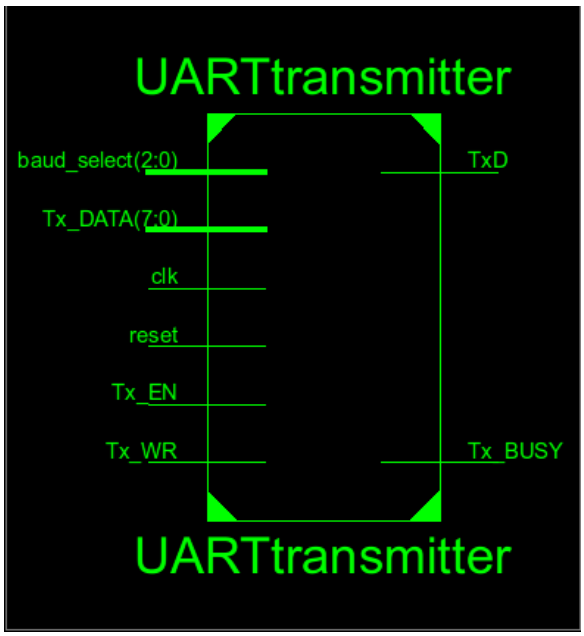
3.2 Αποστολέας **UART**

Έπειτα, υλοποιήσαμε το FSM του Αποστολέα ακολουθώντας τις αναλυτικές οδηγίες της εκφώνησης για το κάθε ένα απο τα σήματα εισόδου κι εξόδου αλλά και για την δομή της λογικής πίσω από τον κώδικα. Αρχικά ορίσαμε πέντε διαφορετικές καταστάσεις μηχανής, τις:

- TxIDLE, για την αδρανοποίηση της μηχανής.
- TxSTART, για την αρχικοποίηση.
- TxDATA, κατάσταση στην οποία βρίσκεται όσο εκπέμπει τα δεδομένα.
- TxPARITY, κατάσταση για το πρώτο βήμα του parity check.
- TxSTOP, τελική κατάσταση για την ορθή κατάληξη, πρωτού επιστρέψει στην αδρανοποίηση.

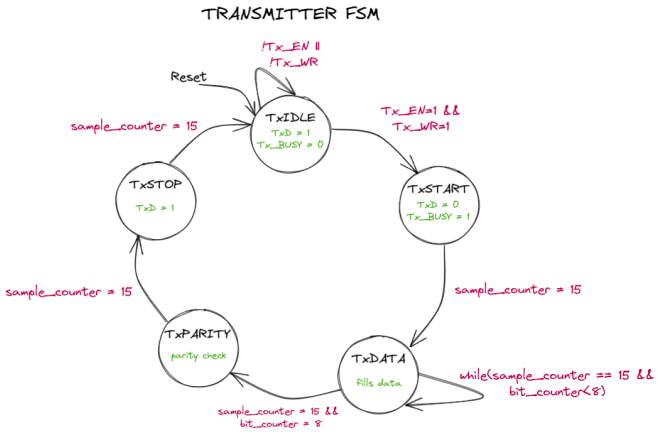
Έτσι, καταφέραμε να ενεργοποιήσουμε τον αποστολέα, να στείλουμε σειριακά, μέσω της εξόδου TxD, τα κατάλληλα bits σε κάθε 16 κύκλους του sample_ENABLE και τέλος να τον απενεργοποιήσουμε ξανά.

Στην παρακάτω εικόνα φαίνεται το σ0χηματικό του κυκλώματος με τα inputs/outputs του. Έχουμε έξι εισόδους, τρεις εκ των οποίων είναι το ρολόι, το σήμα reset και η επιλογή της συχνότητας Baud. Επίσης έχω τα σήματα που χρησιμοποιούνται για την κατάλληλη ενεργοποίηση και γενικότερη λειτουργία του αποστολέα, τα Tx_EN, Tx_WR. Τέλος έχω τα δεδομένα που έρχονται παράλληλα σε έναν καταχωρητή στη μορφή Tx_DATA. Οι έξοδοι είναι το σήμα Tx_BUSY που δείχνει πως ο αποστολέας είναι απασχολημένος με άλλη διαδικασία και το καλώδιο TxD που βγάζει σειριακά τα βιτς της εξόδου.



Σχήμα 3.4: Σχηματικό του Αποστολέα

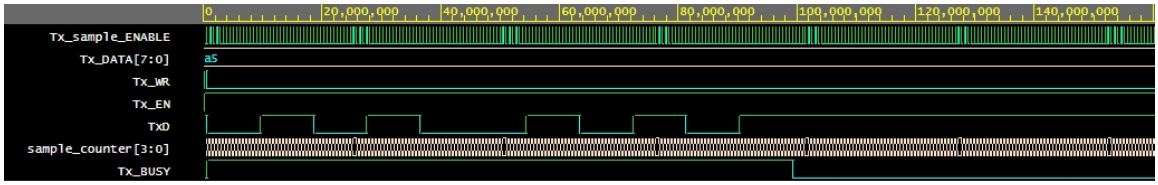
Στην παρακάτω εικόνα παρατίθεται το σχηματικό του FSM του αποστολέα. Σε αυτό φαίνεται η εναλλαγή των καταστάσεων του κυκλώματος. Στις παρακάτω δύο εικόνες φαίνεται το



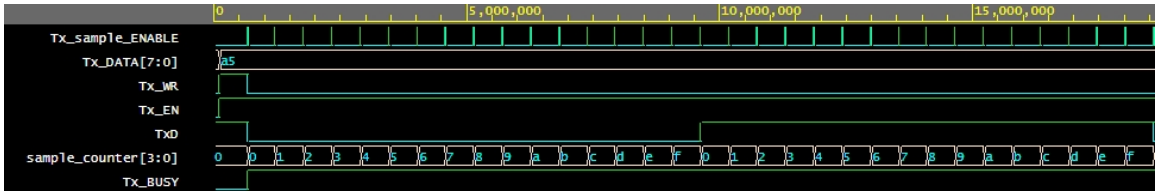
Σχήμα 3.5: *FSM* του Αποστολέα

αποτέλεσμα της προσομοίωσης μέσω του κατάλληλα προσαρμοσμένου testbench. Επιλέξαμε να

δείξουμε τα δεδομένα που δέχεται σαν είσοδο ο αποστολέας ((a5)hex = (10100101)b), τα σήματα εισόδου κι εξόδου που συμβάλουν στην ορθή λειτουργία καθώς και τα bits εξόδου. Στη μεγενθυμένη εικόνα φαίνεται πως η τιμή του TxD διατηρείται κάθε 16 μετρήσεις του μετρητή που μετράει τους παλμούς sample_ENABLE.



Σχήμα 3.6: Αποτέλεσμα προσομοίωσης



Σχήμα 3.7: Αποτέλεσμα προσομοίωσης *zoomed*

3.3 Δέκτης **UART**

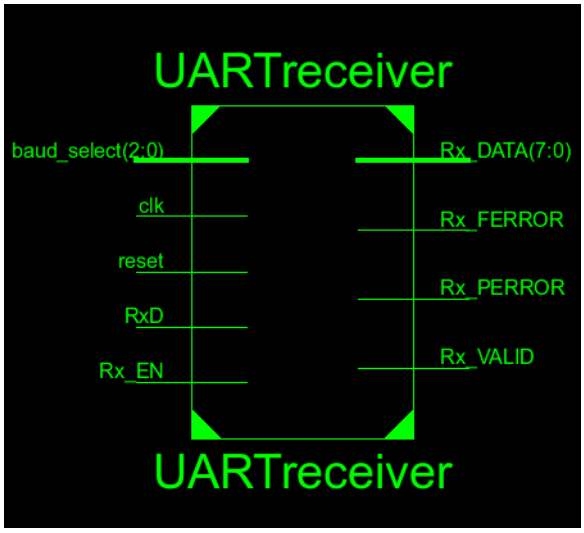
Επόμενο ζητούμενο ήταν η υλοποίηση του FSM του Δέκτη. Ξανά, ακολουθήσαμε αναλυτικά τις οδηγίες της εκφώνησης για την κατάλληλη διαχείρηση των εισόδων κι εξόδων αλλά και για την δομή της λογικής πίσω από τον κώδικα. Χρειάστηκαν για τον δέκτη, πέντε διαφορετικές καταστάσεις μηχανής όπως ακριβώς και στον αποστολέα, οι:

- RxIDLE, για την αδρανοποίηση της μηχανής μετά το πέρας των δεδομένων.
- RxSTART, για την αρχικοποίηση και αναμονή για τα δεδομένα που εκπέμπει ο αποστολέας.
- RxDATA, κατάσταση στην οποία βρίσκεται όσο λαμβάνει τα δεδομένα, αποθηκεύοντάς τα σε έναν προσωρινό καταχωρητή κι όχι στον καταχωρητή εξόδου Rx_DATA.
- RxPARITY, κατάσταση για το δεύτερο βήμα του parity check. Αν το check δεν πετύχει, σηκώνεται σημαία σφάλματος.
- RxSTOP, τελική κατάσταση για την ορθή κατάληξη, πρωτού επιστρέψει στην αδρανοποίηση. Σε αυτή τη κατάσταση γίνεται κι ο framing έλεγχος.

Έτσι, καταφέραμε να ενεργοποιήσουμε τον δέκτη, να λάβουμε σειριακά, μέσω της εισόδου RxD, τα κατάλληλα bits σε κάθε 16 κύκλους του sample_ENABLE, να ελέγξουμε για λάθη στην μεταφορά των βιτς και τέλος να τον απενεργοποιήσουμε ξανά.

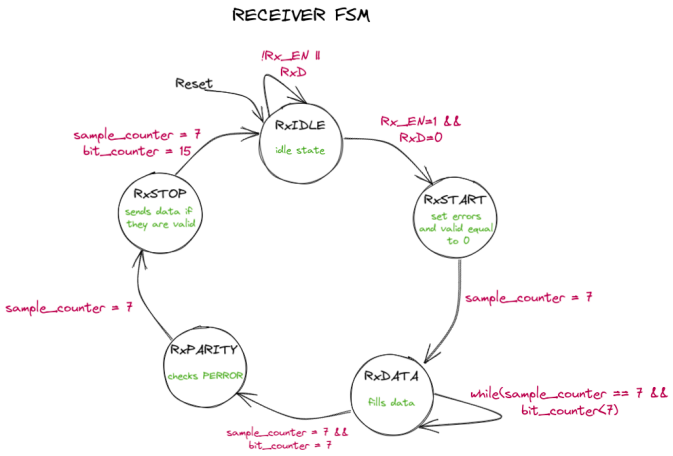
Στην παρακάτω εικόνα φαίνεται το σχηματικό του κυκλώματος με τα inputs/outputs του. Έχουμε πέντε εισόδους, τρεις εκ των οποίων είναι το ρολόι, το σήμα reset και η επιλογή της συχνότητας Baud. Επίσης έχω ένα σήμα που χρησιμοποιείται για την κατάλληλη ενεργοποίηση

και γενικότερη λειτουργία του δέκτη, το Rx_EN. Τέλος έχω τα δεδομένα που έρχονται σε σειρά μέσω του καλωδίου RxD. Οι έξοδοι είναι τα σήματα ένδειξης ορθής ή μη λειτουργίας, Rx_FERROR, Rx_PERROR, Rx_VALID, και ο καταχωρητής Rx_DATA που βγάζει τα βιτς της εξόδου.



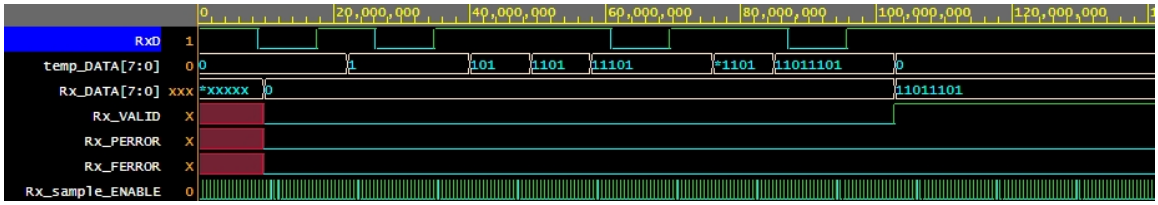
Σχήμα 3.8: Σχηματικό του Δέκτη

Στην παρακάτω εικόνα παρατίθεται το σχηματικό του FSM του δέκτη. Σε αυτό φαίνεται η εναλλαγή των καταστάσεων του κυκλώματος.

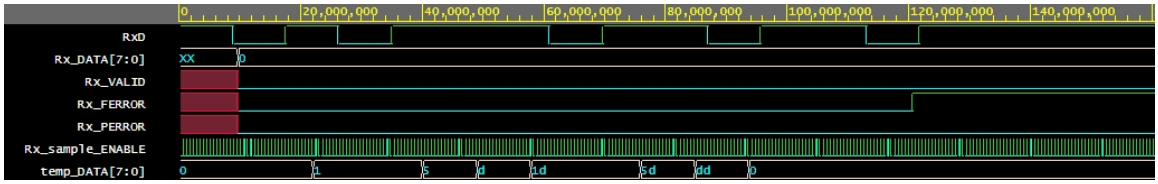


Σχήμα 3.9: *FSM* του Δέκτη

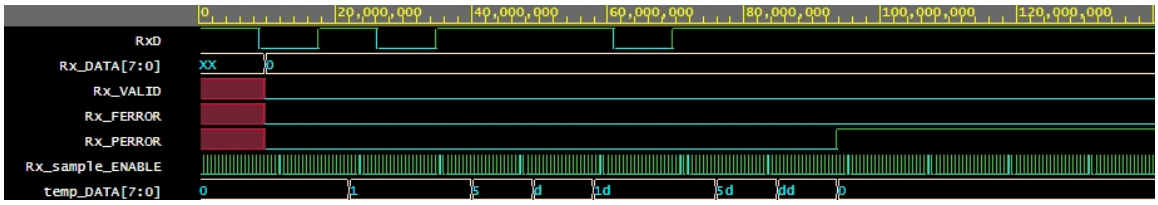
Στις παρακάτω τρεις εικόνες φαίνεται το αποτέλεσμα της προσομοίωσης μέσω του κατάλληλα προσαρμοσμένου testbench, τέτοιο ώστε να βγάζει ένα αποτέλεσμα ορθής λήψης, ένα με PERROR κι ένα με FERROR. Επιλέξαμε να δείξουμε τα δεδομένα που δέχεται σαν είσοδο σειριακά ο δέκτης, τα σήματα εξόδου που αναδεικνύουν την ορθή ή μη λειτουργία του, καθώς και τον προσωρινό καταχωρητή και τον καταχωρητή εξόδου. Βλέπουμε πως όταν σηκώνεται σήμα σφάλματος, δεν σηκώνεται το VALID και δεν περνάνε τα δεδομένα του προσωρινού καταχωρητή σε αυτόν της εξόδου. Αντιθέτως, όταν δεν υπάρχει σφάλμα, σηκώνεται το σήμα VALID και περνάνε τα δεδομένα εξόδου στον Rx_DATA.



Σχήμα 3.10: Αποτέλεσμα προσομοίωσης



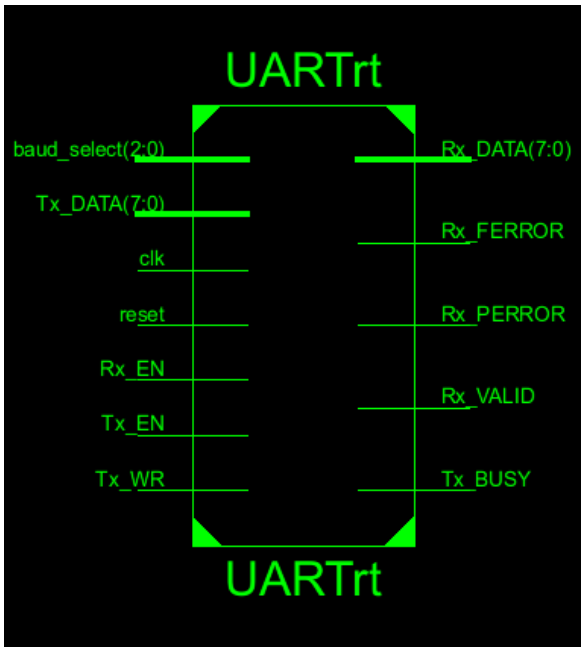
Σχήμα 3.11: Αποτέλεσμα προσομοίωσης με *FERROR*



Σχήμα 3.12: Αποτέλεσμα προσομοίωσης με *PERROR*

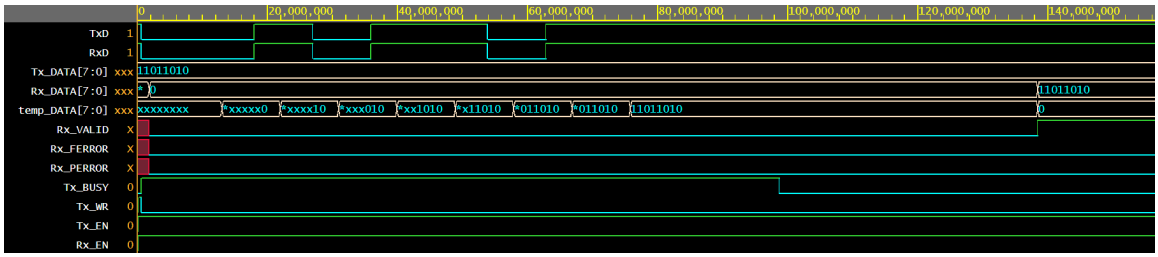
3.4 Συνένωση Αποστολέα-Δέκτη

Έχοντας υλοποιήσει τον Αποστολέα και Δέκτη, ο στόχος μας τωρα είναι να συνενωθούν σε ένα πλήρες κανάλι UART. Υλοποιούμε, λοιπόν, το ζεύγος Αποστολέα-Δέκτη, και το κατάλληλο πλαίσιο δοκιμής για να ελέγξουμε ότι η μεταφορά των τεσσάρων διαδοχικών λέξεων γίνεται σωστά, και ότι τα απεσταλμένα δεδομένα παραλαμβάνονται σωστά στην πλευρά του Δέκτη. Η επικοινωνία του πλαισίου δοκιμής με τους Αποστολέα και Δέκτη, βασίζεται αποκλειστικά στο πρωτόκολλο των σχετικών σημάτων, στα σήματα ελέγχου Rx_VALID, Tx_BUSY. Το σχηματικό διάγραμμα του κυκλώματος φαίνεται παρακάτω:



Σχήμα 3.13: Σχηματικό του Συστήματος Αποστολέα-Δέκτη

Για τον έλεγχο της σωστής λειτουργίας του κυκλώματος σχεδιάσαμε ένα κύκλωμα ελέγχου testbench και τρέξαμε την προσομοίωση θέτοντας κάποιες δοκιμαστικές τιμές TxDATA σαν είσοδο. Περιμένουμε ο αποστολέας να στείλει τα δεδομένα και ο δέκτης να τα λάβει και στην περίπτωση που αυτά είναι σωστά να τα αποθηκεύσει σε έναν καταχωρητή RxDATA. Σε περίπτωση σφάλματος, όπως αναφέρθηκε παραπάνω, ο δέκτης δεν περνάει τα δεδομένα στο καταχωρητή, αλλά θέτει το Rx_VALID ίσο με μηδέν.



Σχήμα 3.14: Αποτελέσματα προσομοιώσεων *testbench* του *Transceiver*

Πράγματι παρατηρούμε ότι τα δεδομένα στάλθηκαν και λήφθηκαν σωστά.

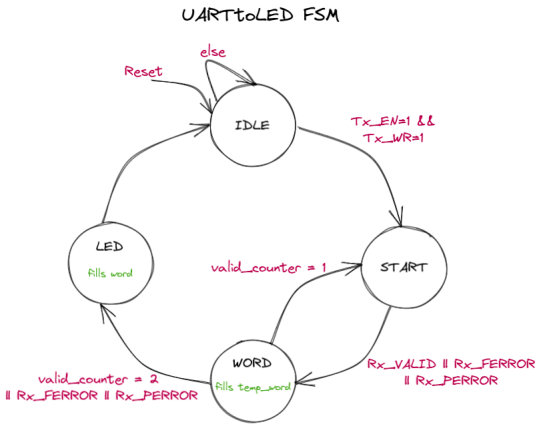
4 Μέρος Γ

Στο τελευταίο μέρος της εργασίας υλοποιήσαμε τη συνένωση του Μέρους Α και του Μέρους Β. Συνδέσαμε κατάλληλα την έξοδο του Δέκτη με την είσοδο την οθόνης τεσσάρων LED 7-τμημάτων. Με τον τρόπο αυτό οι ενδείξεις που συλλέχθηκαν από τον Αποστολέα, προβάλονται από το Δέκτη. Ο Αποστολέας έχει διαχωρίσει την κάθε ένδειξη σε δύο πακέτα των 8-bit, οπότε όταν ο Δέκτης παραλαμβάνει τα 2 πακέτα, αποθηκεύει τα 16 bit, τα προβάλει στην οθόνη. Στην περίπτωση που ο Δέκτης εντοπίσει κάποιο σφάλμα (FERROR ή PERROR) εμφανίζει στην οθόνη το μήνυμα σφάλματος $FFFF$.

Χρειάστηκαν τέσσερεις διαφορετικές καταστάσεις μηχανής για τον σωστό σχεδιασμό του FSM, οι:

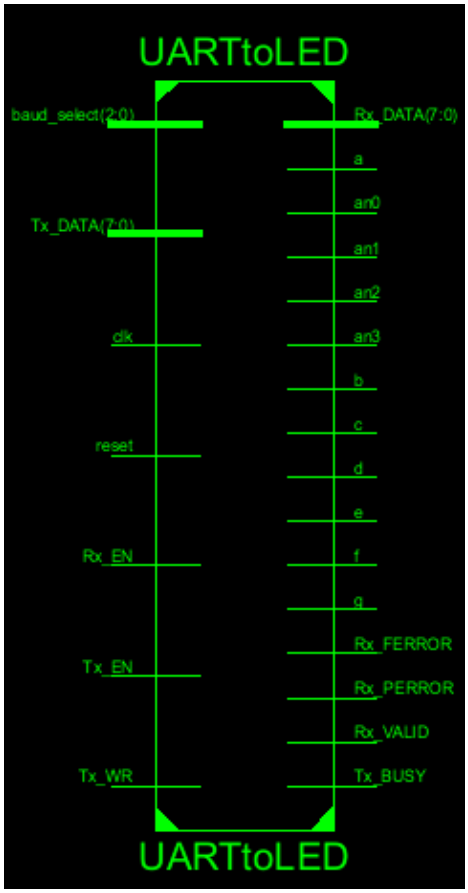
- IDLE, για την αδρανή κατάσταση της μηχανής, εώς ότου δωθεί το κατάλληλο σήμα για start.
- START, για την αρχικοποίηση και αναμονή εώς ότου τελείωσει την εκπομπή και τη λήψη ο transceiver.
- WORD, για την εκχώρηση δεδομένων σε έναν προσωρινό καταχωρητή και έλεγχο αν έχει φτάσει ένα σήμα σφάλματος ή δύο σήματα VALID.
- LED, εδώ γεμίζει ο καταχωρητής word και έτσι ανάβουν τα κατάλληλα LEDs.

Στην παρακάτω εικόνα φαίνεται το σχεδιάγραμμα FSM



Σχήμα 4.1: *FSM* του *UARTtoLED*

Η μηχανή θεωρητικά δουλεύει ορθά και το component το οποίο συνέθεσε το πρόγραμμα Xilinx φαίνεται στην παρακάτω εικόνα. Παρόλα αυτά, επειδή το online πρόγραμμα EDA_PLAYGROUND δεν τρέχει για τον απαραίτητο χρόνο κάναμε κάποιες μικρές αλλαγές και χωρίσαμε το τελευταίο benchtest σε δύο διαφορετικές προσομοιώσεις ώστε να αποδειχθεί η ορθή λειτουργία.



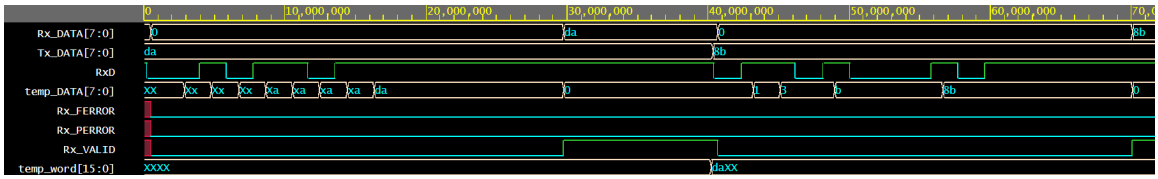
Σχήμα 4.2: Σχηματικό της Μηχανής *UARTtoLED*

Στα παρακάτω χρονοδιαγράμματα φαίνονται τα αποτελέσματα των δύο διαφορετικών προσομοιώσεων. Μία βασική αλλαγή που έκανε τις προσομοιώσεις μας δυνατές ήταν η αλλαγή του Baud Rate από 115200 σε 6×115200 ώστε να γίνονται όλα τα βήματα σε γρηγορότερο ρυθμό και να «χωράνε» τα αποτελέσματα στην οθόνη.

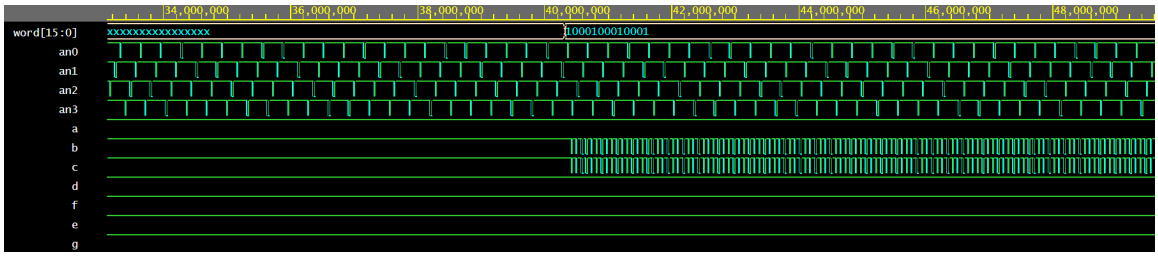
Στην πρώτη, βλέπει κανείς πως ο transceiver καταφέρνει πράγματι να στείλει δύο διαφορετικά πακέτα των 8 bits και αυτά να περνάνε τόσο στην έξοδο Rx_DATA όσο και στον προσωρινό καταχωρητή temp_word ο οποίος θεωρητικά με τον κατάλληλο χρόνο, θα περνούσε τα δεδομένα στον καταχωρητή word κι αυτός με τη σειρά του στα LEDs. Επίσης φαίνονται τα δύο σήματα για τα πιθανά σφάλματα καθώς και το σήμα VALID το οποίο σηκώνεται δύο φορές στο τέλος του κάθε 8bit πακέτου.

Στη δευτέρη, αφού στείλαμε μονάχα ένα πακέτο των 8bits, περάσαμε το αποτέλεσμα, δύο φορές, στον καταχωρητή word ώστε να ελέγξουμε την σωστή εναλλαγή της κατάστασης και το άναμα των LEDs. Η λέξη που προσομοιώσαμε είναι η 1111 και η εικόνα αποδεικνύει την ορθή

λειτουργία του κυκλώματος.



Σχήμα 4.3: Πρώτη προσομοίωση του *UARTtoLED*



Σχήμα 4.4: Δεύτερη προσομοίωση του *UARTtoLED*

5 Μέρος Δ - Σκιαγράφημα

5.1 Κωδικοποιητής-Αποκωδικοποιητής

Μετά από έρευνα πάνω στην κρυπτογραφία σε επίπεδο **Hardware**, ενδιαφέρουσα προσέγγιση μας φάνηκε αυτή που παρατίθεται σε αυτή την δημοσίευση. Οι ερευνητές εδώ, ασύμμετρα, κρυπτογράφησαν δεδομένα σε έναν «αποστολέα» και τα αποκρυπτογράφησαν στον «δέκτη» χωρίς την χρήση εξωτερικής **third party** παρέμβασης στη διαδικασία. Η μέθοδος παρατίθεται αναλυτικότερα παρακάτω.

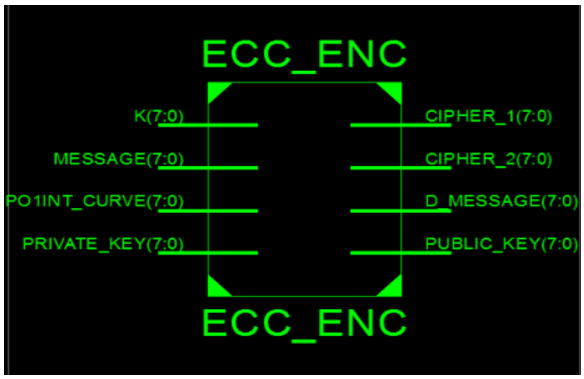
5.2 Κρυπτογραφία ελλειπτικών καμπυλών - **ECC**

Η κρυπτογραφία ελλειπτικής καμπύλης (ECC) χρησιμοποιεί τις μαθηματικές ιδιότητες των ελλειπτικών καμπυλών για την παραγωγή κρυπτογραφικών συστημάτων δημόσιου κλειδιού. Όπως όλες οι κρυπτογραφίες δημόσιου κλειδιού, το ECC βασίζεται σε μαθηματικές συναρτήσεις που είναι απλές στον υπολογισμό σε μία κατεύθυνση, αλλά είναι πολύ δύσκολο να αντιστραφεί. Στην περίπτωση του ECC, αυτή η δυσκολία έγκειται στην αδυναμία υπολογισμού του διακριτού λογάριθμου ενός τυχαίου ελλειπτικού στοιχείου καμπύλης σε σχέση με ένα ευρέως γνωστό σημείο βάσης. Στην εργασία που παρατέθηκε παραπάνω χρησιμοποιείται ένα μέρος μονάχα αυτής της μεθόδου, το οποίο αναλύεται αλγοριθμικά και είναι συνθέσιμο.

Ο αλγόριθμος αυτός συνοπτικά περιγράφεται ως εξής:

Έστω x το ιδιωτικό κλειδί του πομπού και y το ιδιωτικό κλειδί του δέκτη. Στην κρυπτογράφηση θεωρουμε το M ένα μήνυμα ή σήμα εισόδου ή απλό κείμενο. Το P είναι δημόσιο κλειδί, ενώ η δημόσια τιμή του δέκτη είναι $(y*p)$. Τώρα, το x πολλαπλασιάζεται με το $(y*p)$ και δίνει αποτέλεσμα $x*y*p$. Το μήνυμα κρυπτογραφείται κάνοντας χρήση του τύπου $M+(x*y*p)$. Κατά τη διαδικασία αποκρυπτογράφησης στη συνέχεια, το x πολλαπλασιάζεται με p για να δώσει τη δημόσια τιμή του πομπού που είναι $x*p$. Τώρα πολλαπλασιάζεται με το μυστικό κλειδί του παραλήπτη, δηλαδή, το y και δίνει το αποτέλεσμα $(y*x*p)$. Το μήνυμα αποκρυπτογραφείται και πάλι αφαιρώντας την τιμή από το ληφθέν μήνυμα και από την τιμή του πομπού $(M+(x*y*p))-(x*y*p)$ που ισούται με το M, δηλαδή το αρχικό μήνυμα M.

Παρατίθεται, επίσης, ένα σχηματικό του κυκλώματος κωδικοποιήσης/αποκωδικοποίησης



Σχήμα 5.1: *Encoder/Decoder*

6 Σχόλια - Παρατηρήσεις

Οι κύριες δυσκολίες που παρουσιάστηκαν κατά την διάρκεια της υλοποίησης της εργασίας αφορούσαν τις ιδιαιτερότητες της γλώσσας Verilog, καθώς είναι μία γλώσσα στην οποία οτιδήποτε γράφεται στον κώδικα θα πρέπει να είναι συνθέσιμο και να αποτυπωθεί εν τέλει «στο πυρίτιο». Είναι, δηλαδή, μια γλώσσα με αρκετά διαφορετική λογική από τις συνήθεις γλώσσες προγραμματισμού με τις οποίες έχουμε ασχοληθεί σε άλλα μαθήματα. Μετά από ένα μεταβατικό στάδιο, ωστόσο, εξοικείωσης με την γλώσσα και την λογική των μηχανών πεπερασμένων καταστάσεων η υλοποίηση έγινε πολύ πιο κατανοητή.

Σε πιο πρακτικά ζητήματα, μια σοβαρή δυσκολία μας υπήρξε η χρήση των προγραμμάτων σύνθεσης και προσομοίωσης. Το πρόγραμμα Quartus που υπήρξε η πρώτη απόπειρα μας αποδείχθηκε εξαιρετικά δύσχρηστο κι έτσι μεταβήκαμε στο online ide της EDA Playground, το οποίο ήταν πολύ πιο εύχρηστο. Εκεί, ωστόσο, αντιμετωπίσαμε δύο άλλα προβλήματα.

Πρώτον, το εργαλείο αυτό δεν έκανε σύνθεση επομένως ήταν αδύνατο να ελέξγουμε εάν ο κώδικας που γράψαμε ήταν συνθέσιμος στο υλικό. Το πρόβλημα αυτό αντιμετωπίστηκε με την χρήση του προγράμματος Xilinx, στο οποίο πραγματοποιήσαμε την σύνθεση του κώδικά μας και την εξαγωγή των σχηματικών διαγραμμάτων RTL. Λόγω εξοικείωσης, όμως, με το εργαλείο προσομοίωσης του EDA Playground αποφασίσαμε να συνεχίσουμε τις προσομοιώσεις μας σε αυτό.

Έτσι, ερχόμαστε στο δεύτερο βασικό πρακτικό πρόβλημά μας, το οποίο ήταν το γεγονός ότι το εργαλείο EDA Playground είχε συγκεκριμένο χρόνο προσομοίωσης κι επομένως για κάποιες τιμές Baud Rate, καθώς και για την τελική προσομοίωση του συνδυασμένου κυκλώματος, ήταν αδύνατη η πλήρης προσομοίωση λόγο χρονικού timeout. Η μία λύση σε αυτό θα ήταν να επιχειρήσουμε ξανά την προσομοίωση στο πρόγραμμα Xilinx, ωστόσο αποφασίσαμε για λόγους συνέπειας και χρηστικότητας να κανονικοποιήσουμε απλά την συχνότητα Baud Rate, όπως αναφέρθηκε παραπάνω.

Σαν παρατήρηση, επίσης, αναφέρουμε ότι οι υλοποιήσεις μας δουλεύουν με ακριβώς τον ίδιο τρόπο για διαφορετικά Baud Rates με μόνη διαφοροποίηση την ταχύτητα αποστολής των δεδομένων. Για λόγους παρουσίασης, ωστόσο, και εξαιτίας του προβήματος που αναφέραμε παραπάνω επιλέχθηκε σε κάθε περίπτωση η μεγαλύτερη συχνότητα Baud Rate χωρίς βλάβη της γενικότητας.

Τέλος, όσο αναφορά το προαιρετικό μέρος Δ της εργασίας, παρατίθεται ένα κομμάτι της έρευνας που πραγματοποιήσαμε καθώς και του αλγορίθμου που θα ακολουθούσαμε για την κατάλληλη υλοποίηση του κωδικοποιητή/αποκωδικοποιητή, αλλά, δυστυχώς, λόγω έλλειψης χρόνου δεν έχουμε κάποιο χρονοδιαγραμματικό και πρακτικό αποτέλεσμα.

Τέλος Αναφοράς.