Υλοποίηση Συστήματος Αποστολέα-Δέκτη σε Verilog με χρήση Πρωτοκόλλου UART και προβολή σε Οθόνη Τεσσάρων LED 7-Τμημάτων

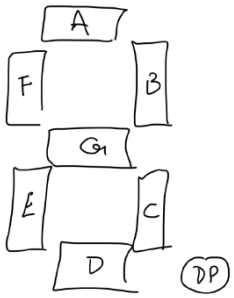
Κακανδρής Ιάσονας 10003 kiasonas@ece.auth.gr

Καμπουρίδου Ηλιάνα 10169 iliakamp@ece.auth.gr

Περιεχόμενα

Μέρος Α: 7 Segment Display Driver

Στόχος είναι η υλοποίηση ενός οδηγού-ελεγκτή 4 LED 7 τμημάτων, έτσι ώστε να προβάλλονται μια σειρά αλφαριθμητικών στις οθόνες.



Απεικονίσεις τιμών

Η οθόνη για την απεικόνιση του κάθε ψηφίου αποτελείται από 7 συν 1 σήματα. Τα σήματα αυτά είναι A,B,C,D,E,F,G και DP και αντιστοιχούν στα αντίστοιχα τμήματα του LED όπως φαίνεται στην διπλανή εικόνα. Για το σβηστό τμήμα χρησιμοποιούμε την τιμή 1, ενώ για να ανάψει την τιμή 0.

Παρακάτω, παρουσιάζουμε τον τρόπο εμφάνισης των συμβόλων μας σε 7 Segment Display Driver:

	A	**B**	**C**	**D**	**E**	**F**	**G**	**DP**
0	0	0	0	0	0	0	1	1
1	1	0	0	1	1	1	1	1
2	0	0	1	0	0	1	0	1
3	0	0	0	0	1	1	0	1
4	1	0	0	1	1	0	0	1
5	0	1	0	0	1	0	0	1
6	0	1	0	0	0	0	0	1
7	0	0	0	1	1	1	1	1
8	0	0	0	0	0	0	0	1
9	0	0	0	0	1	0	0	1
-	1	1	1	1	1	1	0	1
F	0	1	1	1	0	0	0	1
(κενό)	1	1	1	1	1	1	1	1

Τα αντίστοιχα σύμβολα παρουσιάζονται και με μια δυαδική τιμή για να γίνονται αντιληπτά από το σύστημα μας. Η απεικόνιση τους σε δυαδική τιμή φαίνεται στον διπλανό πίνακα.

	0	**1**	**2**	**3**
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
-	1	0	1	0
F	1	0	1	1
(κενό)	1	1	0	0

Module LEDcoder (υλοποίηση αποκωδικοποιητή 7τμημάτων)

Εδώ γίνεται η μετάφραση των δυαδικών τιμών των συμβόλων μας, στα σήματα του LED οδηγού για την προβολή τους. Το σύστημα παίρνει σαν είσοδο τη μεταβλητή char (4 bit δεδομένα), και επιστρέφει την μεταβλητή HEX (8 bit).

Η υλοποίηση επιτυγχάνεται με μια δομή case, η οποία λειτουργεί με βάση τις προαναφερθέντες αντιστοιχίες, όποτε εισάγεται μια νέα τιμή στο module.

Οδήγηση 4 ψηφίων

Η οδήγηση των 4 ψηφίων γίνεται εναλλάξ. Κάθε ψηφίο, προβάλλεται ανά 320ns. Για να επιτευχθεί αυτό, χρησιμοποιούμε ένα ρολόι 50MHz (κύκλος ρολογιού 20ns). Έτσι κάθε ψηφίο προβάλλεται ανά 16 κύκλους.

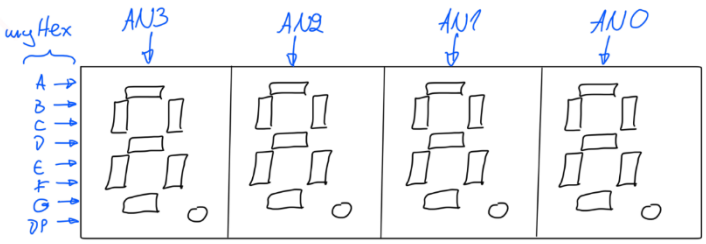
Υλοποίηση ρολογιού

Ορίζοντας στο κύκλωμα μας μονάδα χρόνου το 1ns, υλοποιούμε το ρολόι με τον απλό τρόπο που φαίνεται δίπλα. Ξεκινώντας από το 1, η τιμή του θα εναλλάσσεται κάθε 10 ns μεταξύ 0 και 1, δημιουργώντας έτσι περίοδο 20ns

```
initial begin
  forever begin
    clk<=1;
    #10
    clk<=0;
    #10
    clk <=1;
  end
end
```

Υλοποίηση για 4 ψηφία

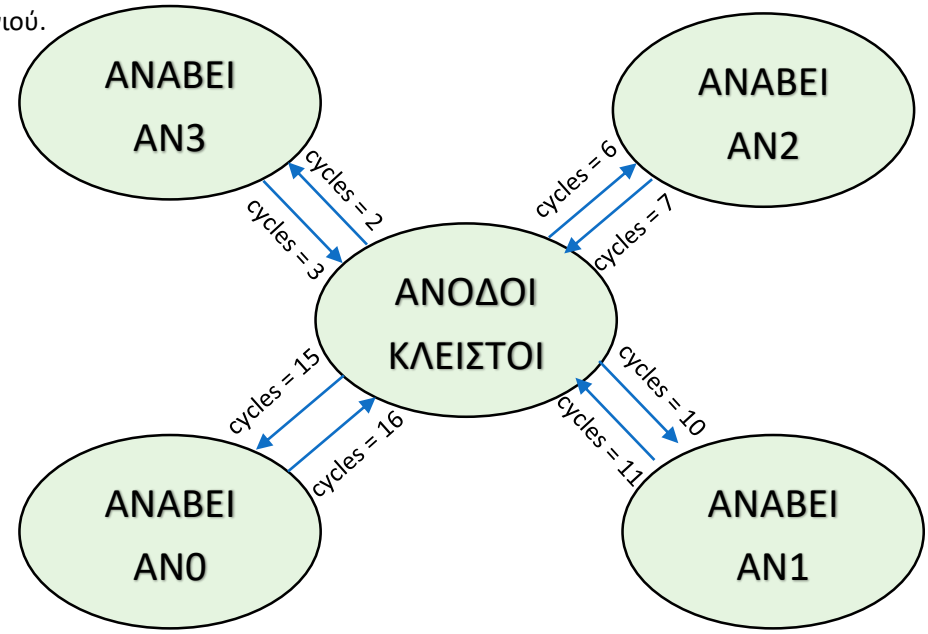
Ορίζουμε 3 μεταβλητές (char , HEX , AN) για καθένα από τα 4 ψηφία μου. Αυτές αντιστοιχούν στη δυαδική τιμή, την τιμή των οδηγών των LED, και τα σήματα των διόδων. Η τιμή κάθε οθόνης προκύπτει από το δικό της LEDcoder.



Τα HEX χρησιμοποιούν την μεταβλητή myHEX (8 bit) για την φόρτωση των τιμών τους. Κάθε φορά που μια από τις μεταβλητές AN βρίσκεται στην τιμή 0, το αντίστοιχο HEX παίρνει την τιμή του myHEX. Επειδή κάθε LED προβάλλει διαφορετική τιμή, πρέπει να εφαρμοστεί η κατάλληλη πολυπλεξία των LED, και αντίστοιχα το myHEX να αλλάζει κατάλληλα την τιμή του.

Module anodoi (υλοποίηση οδήγησης 4 ανόδων)

Περιγραφή της πολυπλεξίας των ανόδων. Παίρνει σαν εισόδους το ρολόι και τα HEX και δίνεις ως εξόδους τα σήματα AN και το myHex. Η λειτουργία βασίζεται σε ένα μετρητή 16 κύκλων του ρολογιού.



Παραπάνω παρουσιάζεται το FSM, δηλαδή το μοντέλο που περιγράφει μαθηματικά την οδήγηση των τεσσάρων ανόδων. Αποτελείται από 5 καταστάσεις ανάλογα με το ποιες δίοδοι είναι αναμμένες, οι οποίες μεταβάλλονται με συγκεκριμένες τιμές του μετρητή.

Τιμή μετρητή	AN3	AN2	AN1	AN0	Λειτουργία
0	1	1	1	1	Φόρτωση HEX3
1	1	1	1	1	
2	0	1	1	1	AN3⇩
3	1	1	1	1	AN3⇧
4	1	1	1	1	Φόρτωση HEX2
5	1	1	1	1	
6	1	0	1	1	AN2⇩
7	1	1	1	1	AN2⇧
8	1	1	1	1	Φόρτωση HEX1
9	1	1	1	1	
A	1	1	0	1	AN1⇩
B	1	1	1	1	AN1⇧
C	1	1	1	1	Φόρτωση HEX0
D	1	1	1	1	
E	1	1	1	0	AN0⇩
F	1	1	1	1	AN0⇧

Ο μετρητής είναι 4 bits για αυξάνεται κατά ένα σε κάθε κύκλο ρολογιού. Όταν φτάσει στα 16 μηδενίζει και ξεκινάει ξανά. Κάθε σήμα ανάβει κυκλικά για ένα κύκλο και σβήνει στο επόμενο. Ανάμεσα στην οδήγηση 2 διαδοχικών συμβόλων αφήνονται 2 κύκλοι κενοί στους οποίους γίνεται η αλλαγή στα δεδομένα των ενδείξεων ώστε να υπάρχει ο απαραίτητος χρόνος να γίνει ολοκληρωμένα το setup. Η ακριβής ενέργειες που γίνονται για κάθε τιμή του μετρητή φαίνονται στον διπλανό πίνακα.

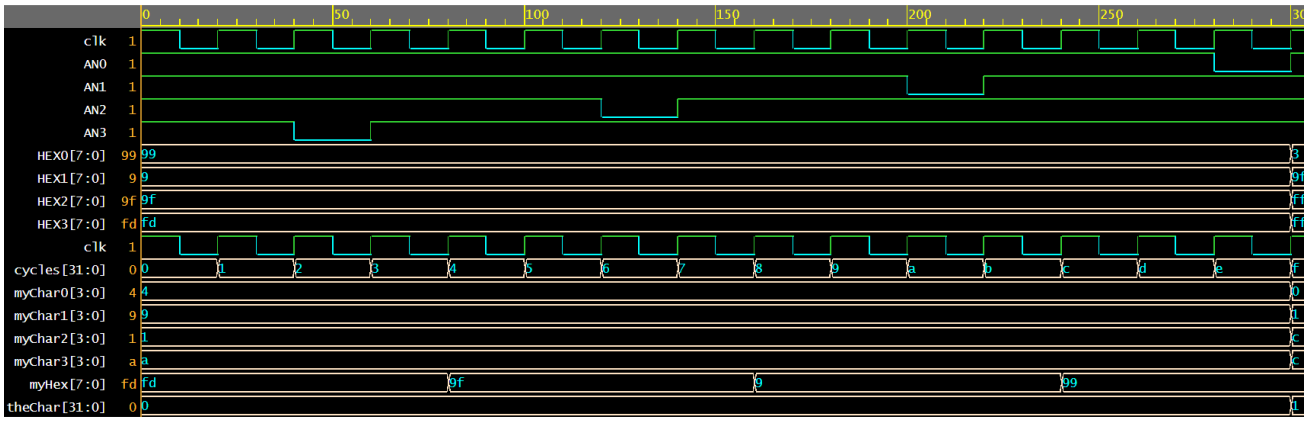
Κύκλωμα ελέγχου (test bench)

Το κύκλωμα ελέγχου χρησιμοποιείται για να πιστοποιήσει την εύρωστη λειτουργία του κυκλώματος ή του συστήματος. Χρησιμοποιείται για να προσομοιώσει την συμπεριφορά του, δίνοντας τις τιμές εισόδου και ελέγχοντας τα αποτελέσματα από τις εξόδους.

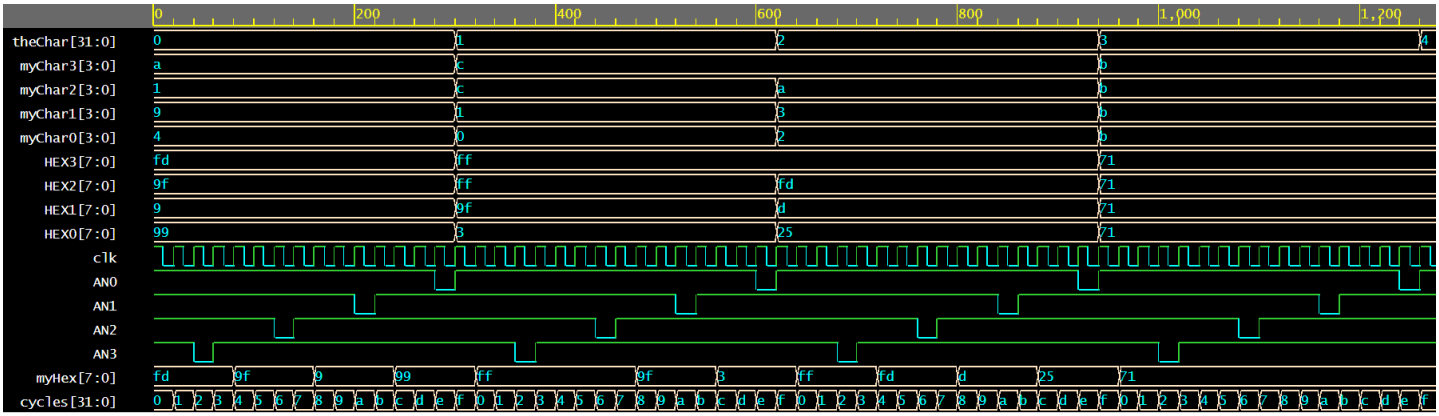
Δημιουργούνται όλα τα απαραίτητα σήματα εισόδου που αναφέρθηκαν πάνω για την λειτουργία καθώς και το ρολόι. Καλούνται 4 LEDcoder, ένα για κάθε LED. Οι έξοδοι αυτών τοποθετούνται με καλώδια (wire) στις αντίστοιχες εισόδους του module anodoi. Από αυτό παίρνω έτοιμα τα σήματα ανόδων και το σήμα των δεδομένων των ενδείξεων.

Τέλος τοποθετείται και εδώ ένας μετρητής 16 κύκλων ρολογιού ώστε να αλλάζει τα δεδομένα. Το κύκλωμα μας ελέγχεται για 4 διαφορετικά σετ τιμών που αλλάζουν διαδοχικά. Η μεταβλητή theChar δείχνει σε ποιο σετ βρίσκομαι κάθε φορά.

Αποτελέσματα προσομοίωσης



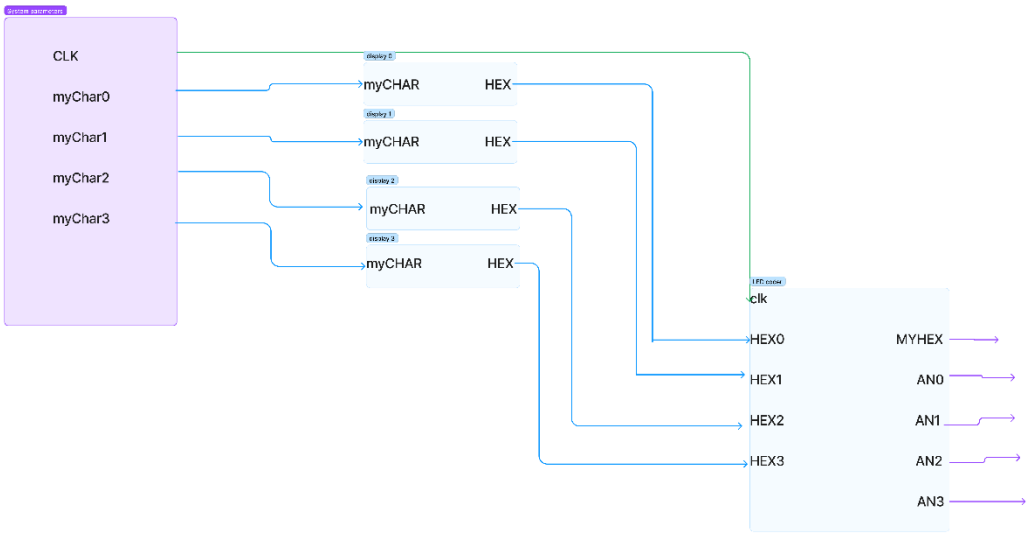
Βλέπουμε τον τρόπο λειτουργίας για έναν κύκλο δεδομένων. Τα σήματα των ανόδων κατεβαίνουν για έναν κύκλο ρολογιού και ταυτόχρονα τα δεδομένα τους υπάρχουν φορτωμένα στο myHex για εμφάνιση στην οθόνη LED. Οι τιμές των Hex φορτώνονται διαδοχικά στο myHex για 4 κύκλου ρολογιού όπου στον 3ο εξ'αυτών πέφτει το σήμα και φορτώνονται στην αντίστοιχη οθόνη LED. Η πολυπλεξία των σημάτων γίνεται με τον επιθυμητό τρόπο. Επιπλέον εξασφαλίζεται η εμφάνιση των σωστών δεδομένων ακόμα και αν υπάρξουν μικροκαθυστερήσεις.



Βλεπουμε ότι κάθε 16 κύκλους του ρολογιού αλλάζει η τιμή του theChar και ανανεώνονται τα δεδομένα των myChar και εν συνεχεία και η μεταφρασή τους σε HEX. Οπότε το σύστημα μπορεί να τρέχει ξανά και ξανά διαδοχικα με διαφορετικά δεδομένα. Εμείς λειτουργούμε το σύστημα για τα αλφαριθμητικά, '-194', '__10','_-32' και 'FFFF', τα ψηφία αφήνονται να προβληθούν μια φορά και περνάμε στα επόμενα

Όλα φαίνεται να παρουσιαζούν την σωστή συμπεριφορά και λειτουργία.

Σχηματικό Διάγραμμα



Μέρος Β: Υλοποίηση Γενικού Ασύρματου Δέκτη Αποστολές (UART)

Στόχος αυτού του μέρους είναι η υλοποίηση ενός συστήματος σειριακής επικοινωνίας μεταξύ αποστολέα και δέκτη με χρήση UART. Το UART είναι ένα πρωτόκολλο ασύγχρονης επικοινωνίας που επιτρέπει την μεταφορά δεδομένων μεταξύ συσκευών με ανεξάρτητα και ασυσχέτιστα ρολόγια

Module baud_controller (ελεγκτής Baud Rate)

Ο ελεγκτής χρησιμοποιείται εσωτερικά σε αποστολέα και δέκτη. Παράγει ένα νέο ρολόι ειδικά σχεδιασμένο για να κάνει αποστολή και λήψη σήματος σε κατάλληλες χρονικά στιγμές.

Σαν είσοδο παίρνει τα σήματα reset, clk καθώς και το baud_select, που περιέχει τον κωδικό για την ταχύτητα επικοινωνίας του UART που έχει προσυμφωνηθεί μεταξύ πομπού και δέκτη. Σαν έξοδο δίνει το νέο προσαρμοσμένο ρολόι sample_ENABLE.

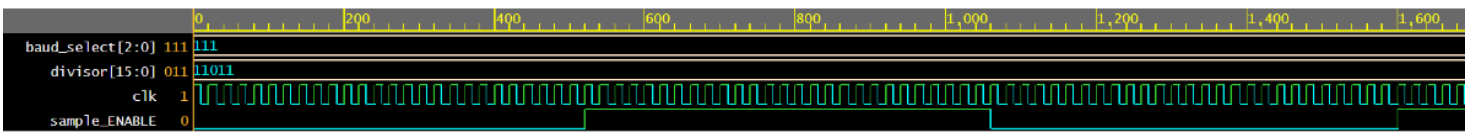
Ανάλογα με την επιθυμητή ταχύτητα, βρίσκουμε αυτή σε πόσους κύκλους του ρολογιού μας αντιστοιχεί. Η τιμή αυτή περνάει στην μεταβλητή μας divisor. Κάθε divisor κύκλους του ρολογιού η τιμή του sample_ENABLE αντιστρέφεται. Έτσι προκύπτει το σήμα της ταχύτητας επικοινωνίας.

Η τιμή του divisor προκύπτει, υπολογίζοντας αρχικά την περίοδο του ρολογιού Baud $T_{sc} = \frac{1}{16*Baud\ rate}$, τον κύκλο του επιλεγμένου ρολογιού $T_{clock} = 20\ \text{ns}$ και διαιρώντας αυτά τα 2 μεγέθη $\frac{T_{sc}}{T_{clock}}$. Η τιμή που θα προκύψει στρογγυλοποιείται στον πλησιέστερο ακέραιο ώστε να είναι υλοποιήσιμο. Από την στρογγυλοποίηση προκύπτει ένα σχετικό σφάλμα το οποίο δεν υπερβαίνει σε καμία περίπτωση το 0.5%, οπότε θεωρείται αμελητέου μεγέθους. Όλη αυτή η διεργασία έχει γίνει θεωρητικά όπως παρουσιάζεται στον παρακάτω πίνακα και στο πρόγραμμά μας απλά αντιστοιχήθηκε η τιμή του select με την ακέραια τιμή τον κύκλων ρολογιού με χρήση μιας δομής case.

Select	Baud rate (bits/sec)	Tsc (ns)	Divisor	Divisor (approx.)	Relative Error %
000	300	208330	10416.67	10417	0.00319
001	1200	52080	2604.16	2604	0.00614
010	4800	13020	651.04	651	0.0064
011	9600	6510	325.52	326	0.147
100	19200	3250	162.76	163	0.147
101	38400	1620	81.38	81	0.467
110	57600	1080	54.25	54	0.467
111	115200	540	27.12	27	0.467

Προσομοίωση Baud

Στην παρακάτω εικόνα βλέπουμε την υλοποίηση αυτού για ένα δεδομένο baud select.



Module uart_transmitter (υλοποίηση UART αποστολέα)

Παίρνει και αποστέλλει τα δεδομένα

Υλοποίηση

Εντός του Module έχουμε τις εξής εισόδους :

- Reset :bit επαναφοράς του module
- Clk : το ρολόι
- Tx_Data : τα 8 bit δεδομένα
- baud_select : το baud rate
- Tx_WR : bit που ενημερώνει πως έχουμε παραλαβή δεδομένων προς αποστολή
- Tx_EN : σήμα ενεργοποίησης του αποστολέα

Οι έξοδοί μας είναι :

- TxD : το bit αποστολής δεδομένων
- Tx_BUSY : bit που ενημερώνει το σύστημα για την λειτουργία του αποστολέα

Χρησιμοποιούμε επιπλέον τις εξής μεταβλητές :

- tx_counter : μεταβλητή η οποία χρησιμοποιείται τόσο ως μετρητής των bit που στείλαμε, όσο και σαν δείκτης της μεταβλητής Tx_Data
- parity_bit : μετράει τον αριθμό λογικών 1 που στέλνουμε. Αν είναι ζυγός τότε parity bit = 0 , αλλιώς , parity bit =1
- current_state, next_state : για το fsm
- Tx_sample_ENABLE : το clock από το baud_rate που θα χρησιμοποιήσουμε

Αρχικά χρησιμοποιούμε το **baud_controller Module** προκειμένου να εφαρμόσουμε το σωστό ρυθμό αποστολής δεδομένων. Για να γίνει αυτό περνάμε σαν έξοδο του baud_controller τη μεταβλητή Tx_sample_ENABLE , της οποίας θα χρησιμοποιούμε την άνοδο της (posedge) για να αποστέλλουμε τα bit.

Έπειτα αρχικοποιούμε τις μεταβλητές μας με τις σωστές τιμές (Tx_BUSY , tx_counter , parity_bit = 0, TxD = 1 και current_state , next_state = tx_off). Ενώ και όταν η μεταβλητή Tx_WR μεταβεί σε λογικό 1 (έχουμε παραλαβή δεδομένων), οι ίδιες αρχικές τιμές επανατοποθετούνται στις μεταβλητές μας, προκειμένου να μπορεί να ξεκινήσει η αποστολή νέων δεδομένων.

Για κάθε θετική ακμή του Tx_sample_ENABLE κάνουμε τις εξής διεργασίες : Αρχικά αν έχουμε θετικό reset επαναφέρουμε τις μεταβλητές μας στις αρχικές τους τιμές. Αν όχι θέτουμε την τιμή του current_state ίση με του next_state. Στην συνέχεια ελέγχουμε αν η μεταβλητή Tx_EN είναι θετική, δηλαδή ο αποστολέα βρίσκεται σε λειτουργία και αντίστοιχα θέτουμε την μεταβλητή Tx_BUSY = 1 για να δείξουμε ότι συντελείται μεταφορά δεδομένων.

Έπειτα ελέγχουμε την τιμή του counter:

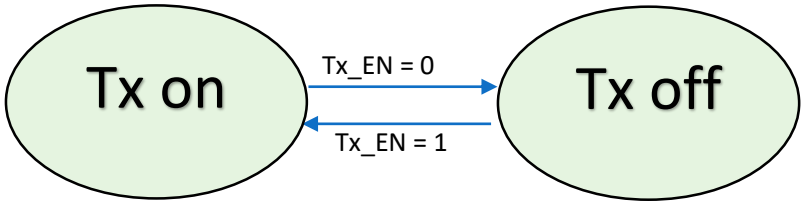
- Counter = 0: είμαστε στο αρχικό Bit, οπότε αποστέλλουμε μέσω του TxD bit, το λογικό 0
- Counter = 9: έχουμε το parity bit.
- Counter = 10 : Στέλνουμε το stop bit το οποίο ισούται με 1
- Στις ενδιάμεσες τιμές (1 με 8) θέτουμε το TxD ίσο με τα bit πληροφορίας ενώ ταυτόχρονα ανάλογα με την τιμή αυτού του bit προσαρμόζεται και το parity bit

Συνολικά στέλνονται 11 bits (8 δεδομένων, 1 αρχής, 1 τέλος και 1 ισοτιμίας)

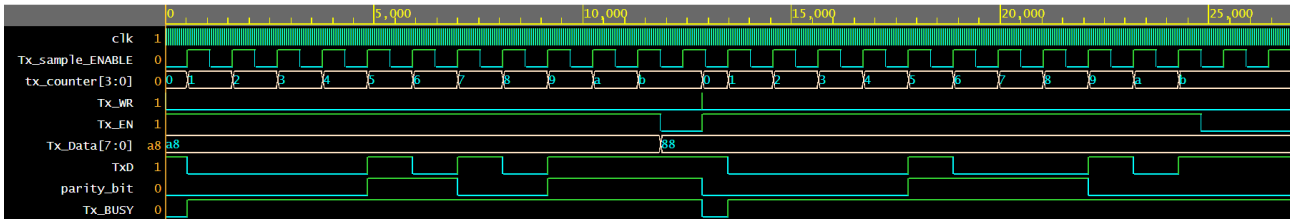
Στο τέλος κάθε κύκλου αυξάνουμε τον Counter κατά 1. Αν ο counter είναι ίσος με 11 τότε έχουμε στείλει όλα μας τα δεδομένα οπότε θέτουμε το Tx_BUSY = 0 (τέλος αποστολής δεδομένων) και θέτουμε το TxD = 1 που είναι η προκαθορισμένη τιμή.

FSM

Έχει 2 καταστάσεις που ρυθμίζονται από την τιμή του Tx_EN (Tx_EN = 1: next_state <= tx_on και Tx_EN = 0: next_state <= tx_off). Οι καταστάσεις αυτές δείχνουν αν ο αποστολέας βρίσκεται σε λειτουργία ή όχι. Το διάγραμμά του προκύπτει:



Προσομοίωση



Στέλνονται 2 σετ δεδομένων 8-bits το κάθε ένα. Το σήμα Tx_WR ανάβει στιγμιαία όταν υπάρχουν νέα δεδομένα για αποστολή ενώ το Tx_EN μένει αναμμένο καθόλη την διάρκεια της επικοινωνίας. Το TxD στέλνει σειριακά 11 bits για κάθε πακέτο συχρονισμένα με το ρολόι baud. Το parity αλλάζει ανάλογα με τα δεδομένα που αποστέλονται μέχρι να ολοκληρωθεί η αποστολή των δεδομένων και να σταλεί και αυτό για να μπορεί να πραγματοποιηθεί έλεγχος σφαλμάτων. Όσο διαρκεί η αποστολή ανάβει το σήμα Tx_BUSY.

Module uart_receiver (υλοποίηση UART δέκτη)

Ο δέκτης λαμβάνει το σήμα των σειριακών δεδομένων.

Υλοποίηση

Εντός του module έχουμε τις εξής εισόδους :

- Reset :bit επαναφοράς του module
- Clk : το ρολόι
- baud_select : το baud rate
- Rx_EN: σήμα ενεργοποίησης του δέκτη
- RxD: bit λήψης των δεδομένων

Οι έξοδοι μας είναι :

- Rx_DATA: τα κωδικοποιημένα δεδομένα που λάβαμε
- Rx_FERROR: σήμα σφάλματος συχνότητας
- Rx_PERROR: σήμα σφάλματος parity bit
- Rx_VALID: σήμα εγκυρότητας των δεδομένων

Τέλος χρησιμοποιούμε και τις εξής μεταβλητές :

- Rx_sample_ENABLE: το clock από το baud_rate που θα χρησιμοποιήσουμε
- Rx_counter: μεταβλητή η οποία χρησιμοποιείται τόσο ως μετρητής των bit που δεχτήκαμε, όσο και σαν δείκτης της μεταβλητής Rx_POS_DATA

- Rx_parity_bit: Το parity bit που υπολογίζεται από τα δεδομένα που λάβαμε
- Tx_parity_bit: Το parity bit που έστειλες ο αποστολέας
- State: δείχνει αν έχει ξεκινήσει η διαδικασία της λήψης
- Rx_POS_DATA: τα δεδομένα που λάβαμε από τον αποστολέα (πριν ελεγχθούν)

Αρχικά βρίσκουμε τον ρυθμό με τον οποίο θα λειτουργεί το Rx_sample_ENABLE από το baud_rate module. Έπειτα αρχικοποιούμε τις μεταβλητές μας με τις κατάλληλες τιμές (Rx_counter , Rx_FERROR , Rx_PERROR, Rx_VALID , state, Rx_DATA, Tx_parity_bit, Rx_parity_bit <= 0 και current_state, next_state <= rx_off)

Στην συνέχεια χρησιμοποιούμε την αρνητική ακμή του RxD προκειμένου να κάνουμε τον έλεγχο για τον Rx_FERROR. Ελέγχουμε αν το Rx_sample_ENABLE έχει τιμή 0 τι στιγμή που αλλάζει το RxD, οπότε τα δεδομένα δεν μεταβάλλονται συγχρονισμένα με το ρολόι του δέκτη. Οπότε δέκτης και αποστολέας δεν λειτουργούν με την ίδια ταχύτητα επικοινωνίας. Αυτό σημαίνει πως έχουμε σφάλμα χρονισμού και ανοίγει το flag μας.

Σε κάθε θετική ακμή του Rx_EN σημαίνει πως είμαστε έτοιμοι για παραλαβή νέων δεδομένων οπότε θέτουμε τις αρχικές τιμές πάλι στις μεταβλητές μας.

Στον δέκτη χρησιμοποιούμε την αρνητική ακμή του Rx_sample_ENABLE για να πάρουμε τα δεδομένα μας, και να κάνουμε έλεγχο του reset. Χρησιμοποιούμε την αρνητική ακμή ώστε η δειγματοληψία να πραγματοποιείται στην προβλεπόμενη μέση του αποσταλμένου bit.

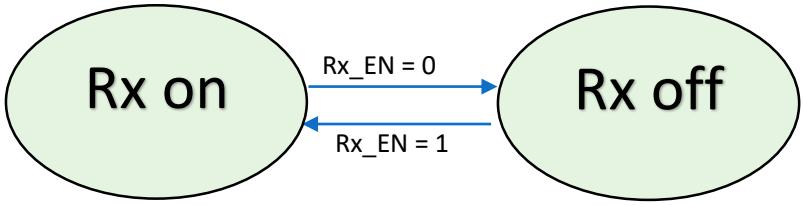
Όπως και στον αποστολέα έτσι και στον δέκτη αν το reset είναι θετικό, επαναφέρουμε τις αρχικές τιμές στα δεδομένα μας. Ειδάλλως κάνουμε τις εξής ενέργειες : Θέτουμε το νέο state ίσο με το επόμενο state. Ελέγχουμε την τιμή του Rx_EN και αν είναι θετική μπαίνουμε εντός του ελέγχου . Αν οι τιμές των RxD και state είναι 0 σημαίνει πως λάβαμε το start Bit και αλλάζουμε το State σε 1 , αυξάνουμε τον counter και θέτουμε την τιμή 0 στα Rx_VALID και Rx_parity_bit .

Δεδομένου τώρα πως το state είναι 1 έχει ξεκινήσει η λήψη των δεδομένων και βλέπουμε την τιμή του Rx_counter

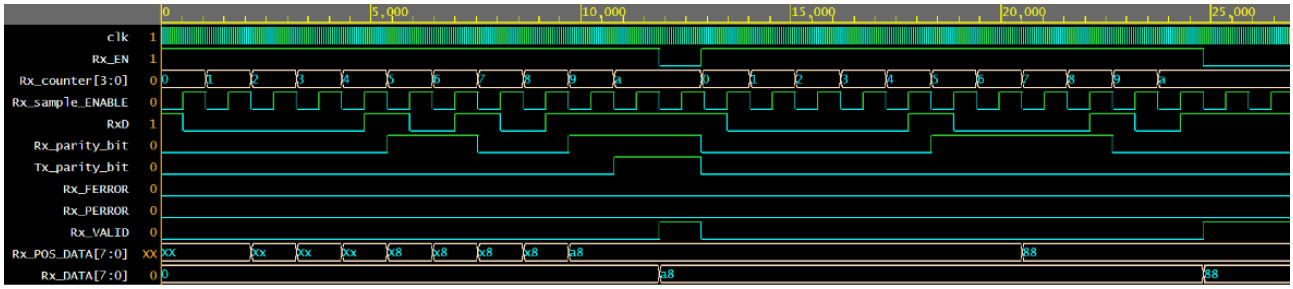
- Rx_counter < 9: βρισκόμαστε στην λήψη των 8 Bit δεδομένων μας. Τα αποθηκεύουμε στην μεταβλητή Rx_POS_DATA και αυξάνουμε τον Counter μας. Επιπλέον, αν ήρθε λογικό 1, ανανεώνουμε την τιμή του Rx_parity_bit.
- Rx_counter = 9: έρχεται το parity bit από τον δέκτη. Αν η τιμή του διαφέρει από αυτή του Rx_parity_bit, δηλαδή αυτού που υπολογίσαμε από τα bit που λάβαμε σηκώνεται το flag μας (Rx_PERROR), καθώς αυτό σημαίνει ότι τα δεδομένα που λάβαμε διαφέρουν από αυτά που στάλθηκαν.
- Rx_counter = 10: ελέγχουμε αν τα Rx_PERROR και Rx_FERROR έχουν τιμή 0. Αν ισχύει, τότε τα δεδομένα μας είναι σωστά, οπότε η τιμή Rx_VALID αλλάζει σε λογικό 1 και τα δεδομένα μας περνάνε και στην έξοδο Rx_DATA

FSM

Όπως και στον αποστολέα, έτσι και στον δέκτη έχουμε 2 καταστάσεις που δείχνουν αν βρίσκεται σε λειτουργία ή όχι ανάλογα με την τιμή του Rx_EN (Rx_EN = 1: next_state <= rx_on , Rx_EN = 1: next_state <= rx_off). Το διάγραμμα:



Προσομοίωση

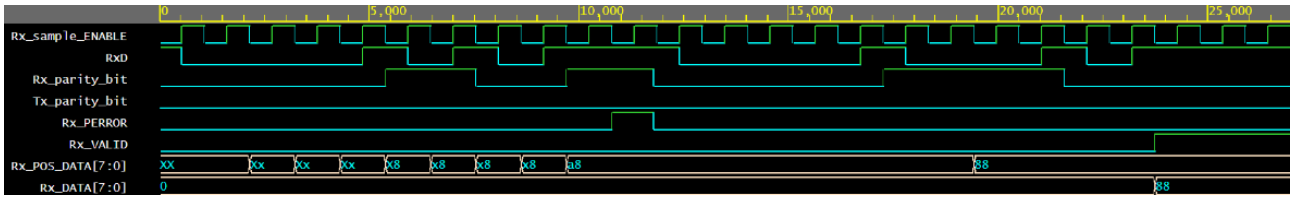


Βλέπουμε πως η σειριακή πληροφορία που μεταφέρεται από το RxD σε κάθε κάθοδο του ρολογιού περνιέται bit-bit στο Rx_POS_DATA. Όταν ολοκληρωθεί, γίνεται η σύγκριση των parity bit. Αν δεν έχουμε κάποιο σφάλμα (συγχρονισμού η parity), τα δεδομένα φορτώνονται στην έξοδο Rx_DATA. Επιπλέον, το Rx_VALID φορτώνεται με την κατάλληλη τιμή (0 αν υπήρξε σφάλμα αλλιώς 1) και εμφανίζονται στην παραπάνω προσομοίωση. Ομοίως φορτώνονται και τα σήματα Rx_PERROR και Rx_FERROR στην περίπτωση που έχουμε σφάλμα στο parity bit ή την ταχύτητα επικοινωνίας.

Καθόλη την διάρκεια της λειτουργίας το σήμα Rx_EN βρίσκεται στο λογικό 1 σηματοδοτόντας την διένεξη της διεργασίας.

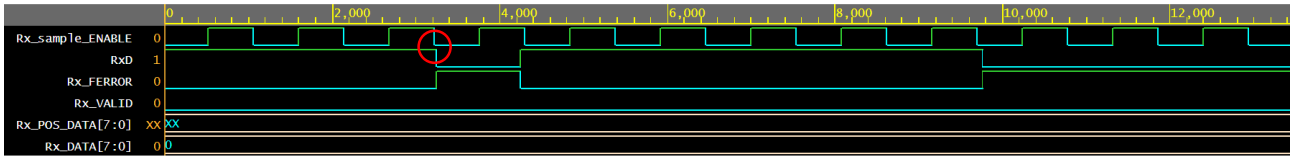
Περιπτωση σφαλματος στο parity bit

Έστω λόγω θορύβου το parity bit του Tx λαμβάνεται πάντα 0. Στο 1ο σετ προκύπτουν διαφορετικά parity bits οπότε αντιλαμβάνομαι πως υπάρχει σφάλμα η πληροφορία που έλαβα είναι λανθασμένη και ανάβει το σήμα ελέγχου parity Rx_PERROR για αυτό.



Περιπτωση σφαλματος στο χρονισμο

Έστω ότι δεν έγινε σωστά η προσυμφωνία ταχύτητας επικοινωνίας μεταξύ αποστολέα και δέκτη και λειτουργούν σε διαφορετικούς ρυθμούς. Ο δέκτης το ελέγχει και αντιλαμβάνεται το σφάλμα αυτό, οπότε ανάβει το σήμα του Rx_FERROR.

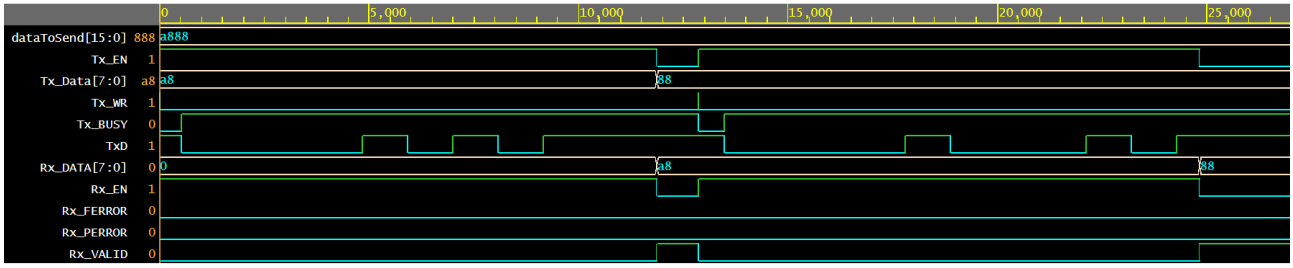


TestBench

Δημιουργούνται όλα τα απαραίτητα σήματα για την λειτουργία καθώς και ο ορισμός και η διασύνδεση των module. Το σήμα σειριακής εξόδου του αποστολέα τοποθετείται με καλώδιο στον δέκτη επιτυγχάνοντας έτσι την σειριακή επικοινωνία.

Τα σήματα εισόδου ελέγχου Tx_EN,Tx_WR και Rx_EN αρχικοποιούνται ώστε να ξεκινήσει η επικοινωνίας. Όταν ληφθεί σήμα για την λήψη, είτε επιτυχή είτε σφάλματος, τα σήματα τροποποιούνται για να σταματήσει η επικοινωνία. Αν υπάρχουν και άλλα διαθέσιμα δεδομένα προς αποστολή αυτά φορτώνονται και η επικοινωνία ξανά ξεκινά ενεργοποιώντας εκ νέου τα σήματα.

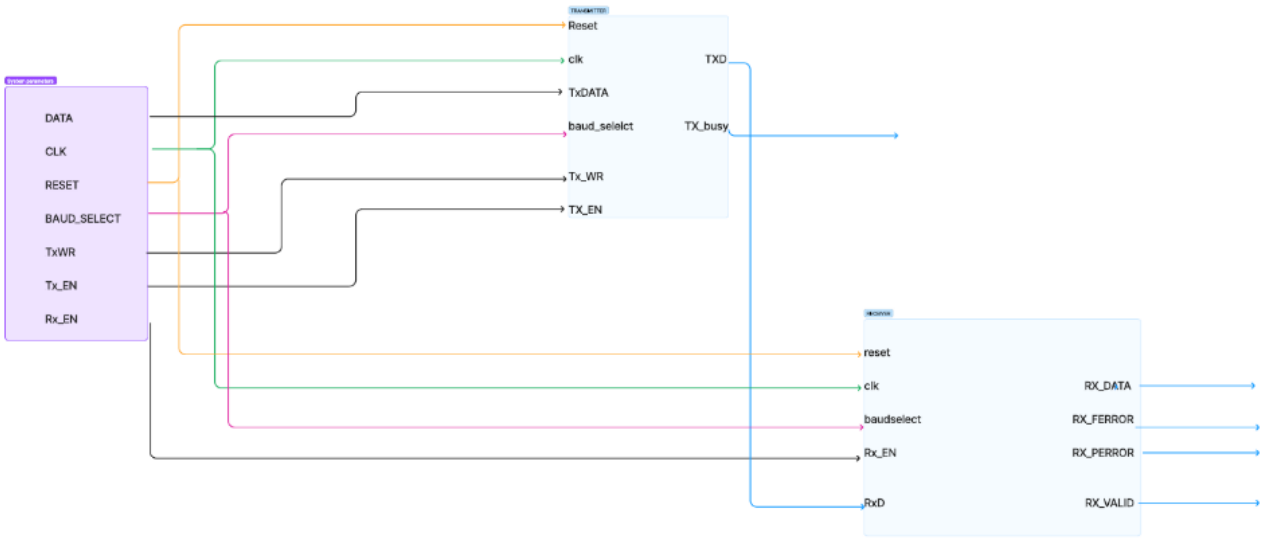
Προσομοίωση



Βλέπουμε την διαδικασία αποστολής και λήψης καθώς και τον τρόπο που μεταβάλλονται τα σήματα ελέγχου κατά τη διάρκεια αυτής. Δίνεται σήμα ενεργοποίησης αποστολέα – δέκτη καθώς και ενημέρωσης για ύπαρξη δεδομένων προς αποστολή. Αυτά γίνονται σειριακά και βγαίνουν ασύγχρονα στο σήμα TxD. Από εκεί ο δέκτης τα δειγματοληπτεί, τα περνάει σε πίνακα και βγάζει και τα αντίστοιχα σήματα.

Οπότε βλέπουμε πως τα δεδομένα που αποστέλλονται μετά από κάποιο χρονικό διάστημα τα παίρνουμε από τον πίνακα του δέκτη. Άρα η επικοινωνίας έχει επιτευχθεί.

Σχηματικό Διάγραμμα



<u>Σημείωση</u> το baud rate module δεν εμφανίζεται στο σχηματικό διάγραμμα καθώς συντελείται εσωτερικά στον αποστολέα και τον δέκτη.

Μέρος Γ: Σύστημα Δέκτη με προβολή μηνύματος στην οθόνη τεσσάρων LED 7-τμημάτων

Στόχος είναι η συνένωση του μέρους Α με το Β. Ουσιαστικά η έξοδος του μέρους Β προσαρμόζεται κατάλληλα και οδηγείται στην είσοδο του Α. Έτσι οι ενδείξεις που συλλέχτηκαν από τον αποστολέα, θα προβληθούν από το δέκτη.

Module messageDisp

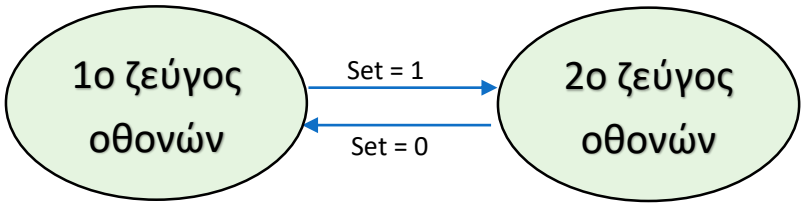
Η 8-bits έξοδος του δέκτη Rx_DATA σπάει σε 2 τετράδες και τοποθετείται στα 2 πρώτα myChar ή στα 2 τελευταία ανάλογα με το μέρος (1ο η 2ο) του αλφαριθμητικού που λαμβάνω. Παίρνει σαν είσοδο, την πληροφορία που παίρνω από το δέκτη Rx_DATA, τα σήματα του δέκτη Rx_FERROR, Rx_PERROR, Rx_VALID καθώς και το ρολόι. Δίνει σαν έξοδο τα 4 myChar για τις 4 οθόνες καθώς και 2 σήματα ελέγχου dispReady και setReady.

Λειτουργεί σε 2 φάσης που διαχωρίζονται από το σήμα set. Όταν αυτό έχει την τιμή 0 η πληροφορία του δέκτη περνάει στο ζευγάρι myChar3 και myChar2, ώστε να προβληθεί στις 2 πρώτες οθόνες, ενώ όταν γίνει 1 η πληροφορία περνάει στο myChar1 και myChar0, για να προβληθεί σε εκείνες τις οθόνες.

Μόλις η τιμή ενός από τα σήματα του δέκτη γίνει 1, περνάει η πληροφορία στο αντίστοιχο ζεύγος myChar όπως περιγράψαμε πριν, και κρατάμε σε μια μεταβλητή την πληροφορία Rx_VALID για να ξέρουμε αν έγινε σωστά η λήψη. Στο δεύτερο ζεύγος ελέγχουμε επιπλέων αν και οι δυο λήψεις έγιναν σωστά, αν έστω και μια από τις 2 δεν έγινε αλλάζουμε το μήνυμα στο μήνυμα σφάλματος ‘FFFF’ ώστε να προβληθεί αυτό στην οθόνη.

Αφού τελειώσει η διεργασία για το πρώτο ζεύγος το σήμα setReady γίνεται 1 ενώ για το δεύτερο αντίστοιχα το dispReady.

Οι 2 φάσεις λειτουργίας του αντιστοιχούν σε 2 καταστάσεις στο fsm



Σύνδεση κυκλώματος

Design

Στο Design έχουμε 6 module, 3 από το μέρος Β (baud_controller, uart_transimitter, uart_receiver),2 από το μέρος Α (LEDcoder, Anodoi) και ένα για την συνένωση messageDisp.

Η μόνη αλλαγή που έχει γίνει στα προηγούμενα module είναι η προσθήκη του σήματος dispReady στο module anodoi. Ουσιαστικά η πολυπλεξία των σημάτων για την προβολή στις οθόνες αρχίζει αφού το νέο σήμα ελέγχου πάρει την τιμή 1. Με αυτόν τον τρόπο η προβολή ξεκινάει αφού η πληροφορία ληφθεί και περαστεί αντίστοιχα σε όλες τις μεταβλητές.

Test bench

Δημιουργούνται όλα τα απαραίτητα σήματα εισόδου-εξόδου για τα 2 προηγούμενα μέρη καθώς και το ρολόι. Γίνεται αντιστοίχηση θυρών για όλα τα module.

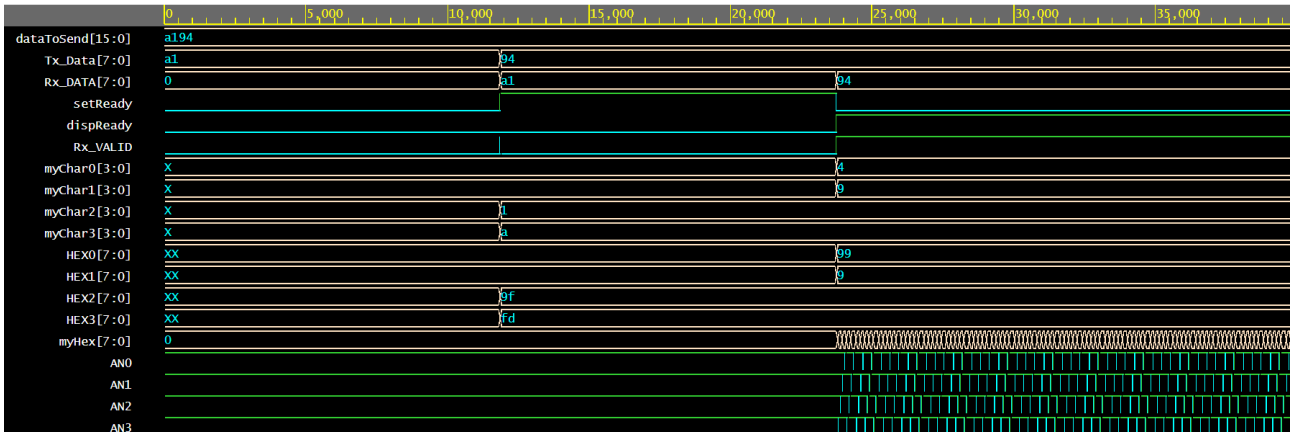
Προστίθενται και τα 2 νέα σήματα ελέγχου setReady και dispReady, που σηματοδοτούν ότι ολοκληρώθηκε η διαδικασία της λήψης και της προσαρμογής για το 1^{ο} και 2^{ο} ζεύγος αντίστοιχα. Όταν κάποιο από αυτά γίνει 1 η επικοινωνία μεταξύ αποστολέα και δέκτη κλείνει, με μηδενισμό

των αντίστοιχων σημάτων Tx_EN και Rx_EN. Ενώ το setReady σηματοδοτεί και την ανανέωση των δεδομένων και την επανέναρξη της διαδικασίας της επικοινωνίας ανοίγοντας και πάλι τα αντίστοιχα σήματα.

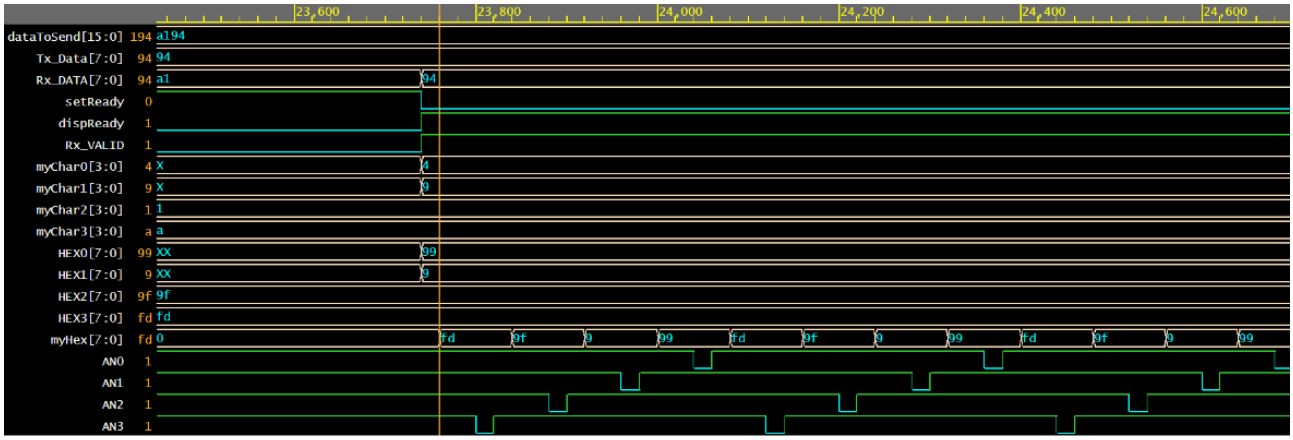
Προσομοίωση

Περίπτωση χωρίς σφάλματα

Αποστολή του μηνύματος -194 (σε δυαδικό 1010000110010100) με baud select 111.



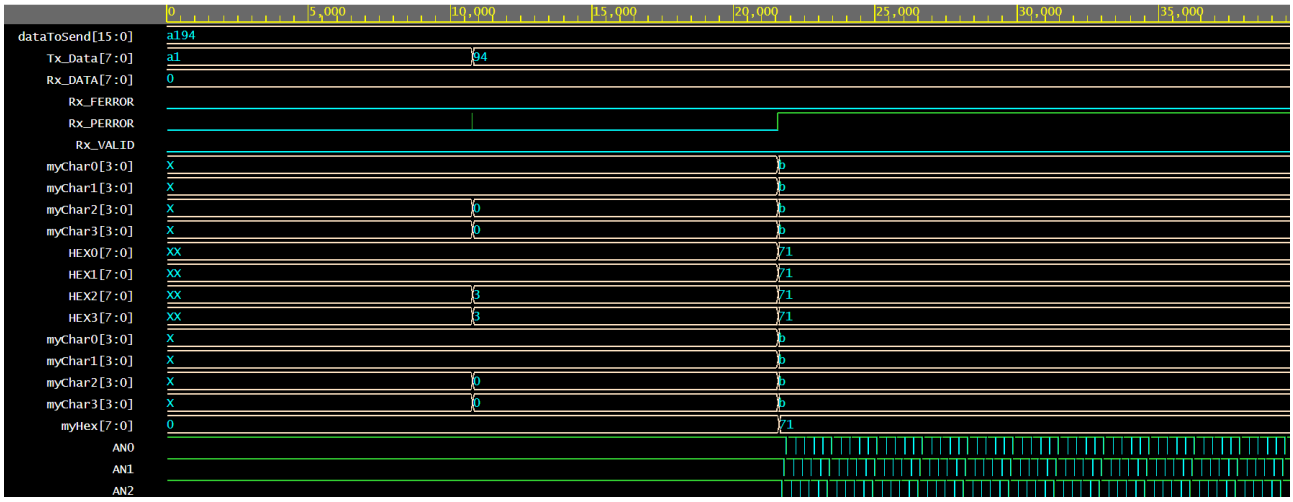
Βλέπουμε ότι το dataToSend περνάει τα δεδομένα του μισά μισά σε 2 χρονικές φάσεις στο TxData αυτά όπως τα υλοποιήσαμε στο δεύτερο μέρος αποστέλλονται και λαμβάνονται από το δέκτη. Κάθε σετ δεδομένων που λαμβάνεται περνάει στα αντίστοιχα myChar και μεταφράζεται στα HEX. Όταν συμπληρωθούν οι τιμές και των 2 σετ αρχίζει η προβολή τους στις οθόνες με την πολυπλεξία των σημάτων ανόδου που υλοποιήθηκε στο πρώτο μέρος. Η ολοκλήρωση κάθε σετ σηματοδοτείται από την άνοδο των σημάτων setReady και dispReady αντίστοιχα.



Το ίδιο διάγραμμα το βλέπουμε και εστιασμένο στην στιγμή έναρξης της πολυπλεξίας. Επιτυγχάνεται επιτυχώς η προβολή του αλφαριθμητικού που αποστάλθηκε.

Περίπτωση σφάλματος στο parity bit

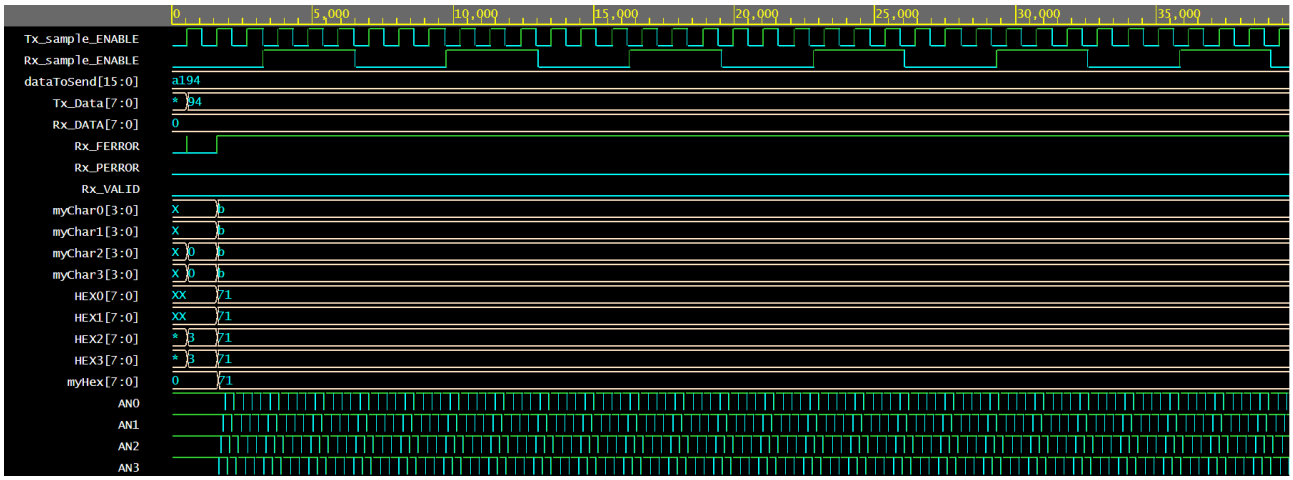
Στο ίδιο παράδειγμα με πριν αλλάζω τον κώδικα ώστε το parity που θα λάβει ο δέκτης αντί για την τιμή 1 να έχει την τιμή 0.



Το σύστημα μου εντοπίζει το σφάλμα κατά την λήψη ενεργοποιεί το σήμα Rx_PERROR. Ο έλεγχος για τα valid αποτυγχάνει λόγω του σφάλματος. Οπότε στις οθόνες μας προβάλλεται το μήνυμα σφάλματος ‘FFFF’.

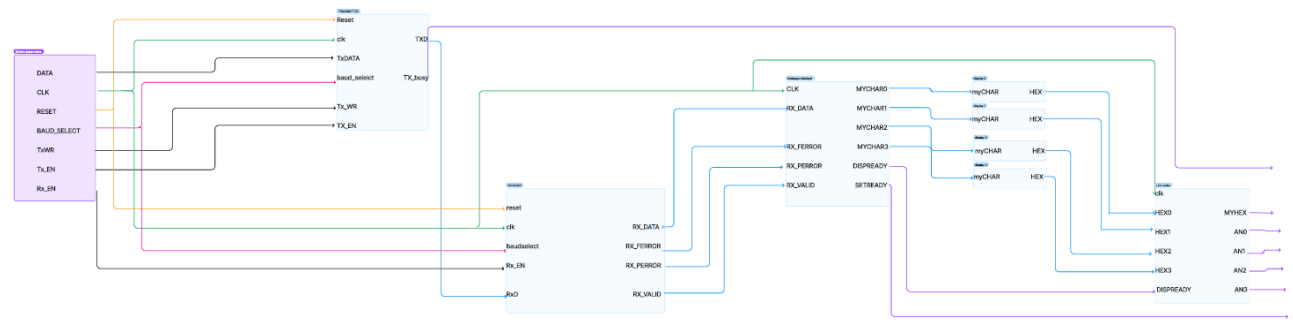
Περίπτωση σφάλματος στο χρονισμό – ταχύτητα επικοινωνίας

Πάνω στο ίδιο παράδειγμα τροποποιώ των κώδικα ώστε η επιλογή του baud rate να είναι διαφορετική μεταξύ αποστολέα και δέκτη οπότε προκύπτουν και διαφορετικά samples οι κυμματομορφές των οποίων φαίνονται στην προσομοίωση.



Λόγω των διαφορετικών baud rate η δειγματοληψία δεν γίνεται σωστά και ενεργοποιείται το σήμα Rx_FERROR και για τα 2 σετ. Οπότε κατά αντιστοιχία και πάλι στην οθόνη μας προβάλλεται το μήνυμα σφάλματος.

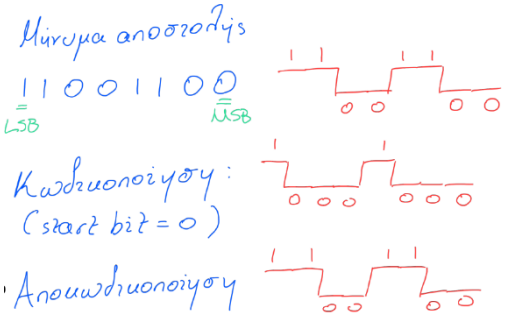
Σχηματικό Διάγραμμα



Μέρος Δ: Υλοποίηση Κωδικοποιητή-Αποκωδικοποιητή

Μέθοδος Κωδικοποίησης

Για την υλοποίηση του κωδικοποιητή και αποκωδικοποιητή χρησιμοποιήσαμε την μέθοδο Non Return to Zero Inverted. Είναι μέθοδος χαρτογράφησης δυαδικού σήματος σε φυσικό. Στην συγκεκριμένη μέθοδο, η αποστολή λογικού 1 σημαίνει μεταβολή του ψηφιακού σήματος (από 1 σε 0 και αντίστροφα) ενώ αποστολή λογικού 0 σημαίνει καμία μεταβολή στο σήμα.



Η λειτουργία φαίνεται μέσω ενός παραδείγματος στο διπλανό σχήμα.

Χρησιμοποιώντας τη συγκεκριμένη μέθοδο έχουμε προτερήματα όπως:

- Απλή υλοποίηση
- Η DC τάση δεν επηρεάζει το κωδικοποιημένο σήμα
- Ευκολία στον εντοπισμό σφαλμάτων κατά την μετάδοση

Η παρούσα μέθοδος εμφανίζει και κάποια μειονεκτήματα όπως:

- Ευαισθησία σε διαδοχική αποστολή πολυάριθμων 0
- Περιορισμένες δυνατότητες ανίχνευσης σφαλμάτων

Module nrz_i_coder (υλοποίηση κωδικοποιητή)

Παίρνει σαν είσοδο τα δεδομένα και το ρολόι και δίνει ως έξοδο την νέα αλληλουχία κωδικοποιημένων δεδομένων.

Εντός του Module χρησιμοποιούμε έναν counter μέσω του οποίου ελέγχουμε της τιμές του μηνύματος και φορτώνουμε τις αντίστοιχες τιμές στο κωδικοποιημένο σήμα. Αν το σήμα του μηνύματος είναι το λογικό 1, το κωδικοποιημένο bit παίρνει την αντίθετη τιμή του προηγούμενου κωδικοποιημένου bit, αλλιώς παίρνει την ίδια τιμή. Ιδιάζουσα περίπτωση είναι το 1ο Bit του οποίου η ανάθεση γίνεται με την ίδια συνθήκη αλλά με βάση το start bit, το οποίο γνωρίζουμε πως είναι 0. Έπειτα από 8 κύκλους, το μήνυμα έχει κωδικοποιηθεί και ο counter επιστρέφει στην τιμή 0.

Module nrz_i_decoder (υλοποίηση αποκωδικοποιητή)

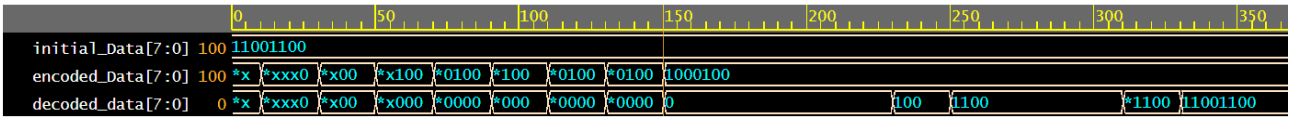
Παίρνει σαν είσοδο το ρολόι, τα κωδικοποιημένα δεδομένα καθώς και τα σήματα εξόδου ελέγχου του δέκτη. Ως έξοδο δίνει τα δεδομένα αποκωδικοποιημένα καθώς και ένα επιπλέον σήμα RxReady που δείχνει πότε έχει ολοκληρωθεί η διαδικασία της αποκωδικοποίησης.

Σε αυτόν χρησιμοποιούμε ξανά έναν counter για τον έλεγχο των τιμών του κωδικοποιημένου μηνύματος. Αν το Bit που ελέγχουμε είναι διαφορετικό από το προηγούμενο Bit του κωδικοποιημένου σήματος, στο αποκωδικοποιημένο bit παίρνει την τιμή 1, αλλιώς του ανατίθεται η τιμή 0. Ιδιάζουσα πάλι περίπτωση αποτελεί το 1ο bit, του οποίου ελέγχεται απευθείας η τιμή του με τα λογικά 0 και 1.

Η λειτουργία αυτού του module δίνεται όταν το σήμα start παίρνει την λογική τιμή 1. Το οποίο συμβαίνει όταν ένα από τα σήματα ελέγχου του δέκτη μεταβεί από το 0 στο 1. Με αυτόν τον τρόπο η αποκωδικοποίηση ξεκινάει μετά την ολοκλήρωση της λήψης.

Αυτόνομη Λειτουργία

Παρακάτω παρουσιάζουμε ένα παράδειγμα αυτόνομη λειτουργίας του συστήματος κωδικοποίησης και αποκωδικοποίησης. Initial_Data τα αρχικά μας δεδομένα, encoded_data τα κωδικοποιημένα Bit, και decoded_data τα αποκωδικοποιημένα bit. Παρατηρούμε ότι στο τέλος της διαδικασίας έχουμε ανακτήσει το αρχικό μας σήμα.



Σύνδεση κυκλώματος

Design

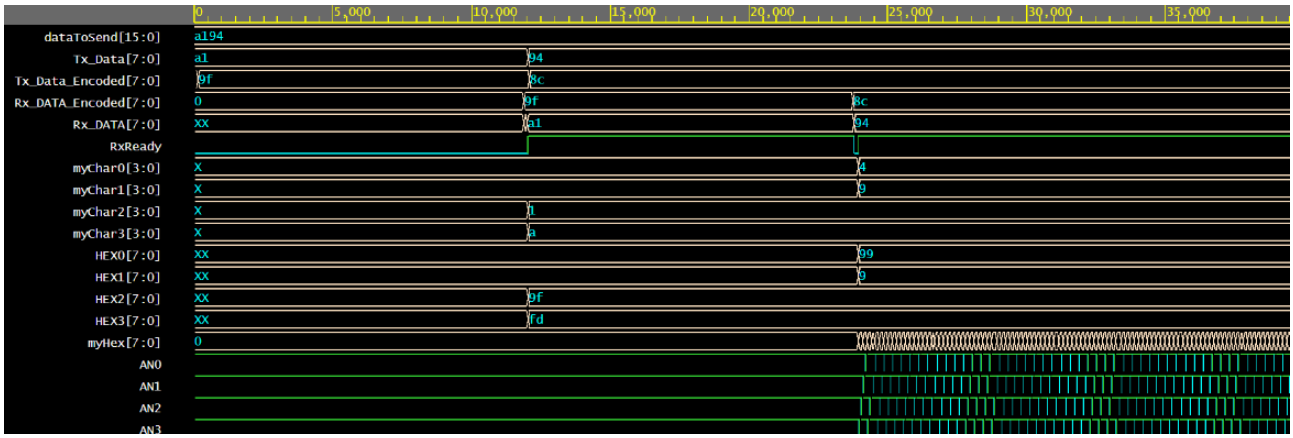
Στο Design έχουμε 8 module, 6 από το προηγούμενο μέρος (baud_controller, uart_transimitter, uart_receiver,LEDcoder, Anodoi, messageDisp) και προστίθενται 2 νέα για την κωδικοποίηση και αποκωδικοποίηση (nrz_i_coder, nrz_i_decoder).

Στο messageDisp έχει προστεθεί ως είσοδος το σήμα RxReady. Με αυτό επιτυγχάνω να μεταφέρεται η πληροφορία του δέκτη στα myChar μετά την ολοκλήρωση της αποκωδικοποίησης ώστε να μην προβάλλονται λανθασμένα και ανολοκλήρωτα μηνύματα στις οθόνες.

Test bench

Είναι ίδιο με αυτό του προηγούμενου μέρους με μόνη διαφορά τα καλώδια διασύνδεσης των module. Το Tx_DATA πάει στον κωδικοποιητή και μετά η κωδικοποιημένη πληροφορία μπαίνει στον αποστολέα. Ενώ η έξοδος του δέκτη μπαίνει στον αποκωδικοποιητή και από εκεί παίρνουμε την αποκωδικοποιημένη πληροφορία για να περαστεί κατάλληλα στα myChar.

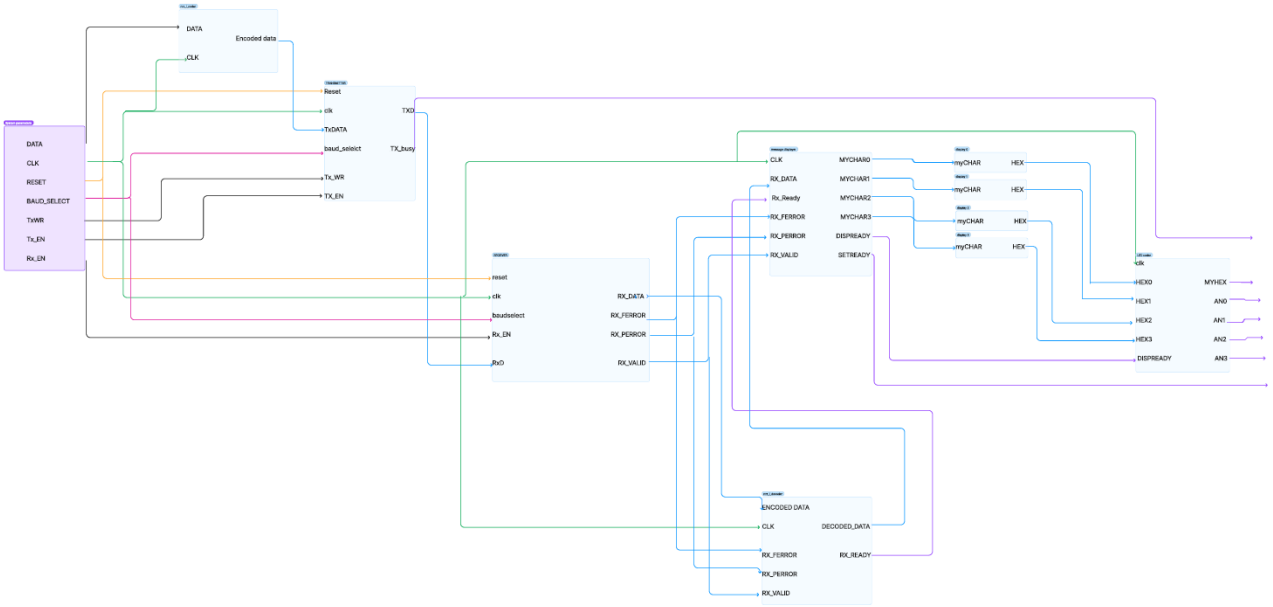
Προσομοίωση



Βλέπουμε την διαδικασία μέσα από το ολοκληρωμένο συστημά μας. Το σήμα ξεκινάει, κωδικοποιείται, αποστέλλεται από αποστολέα σε δέκτη και αποκωδικοποιείται. Το αποδικωποιημένο σήμα όταν το RxReady γίνει 1 μεταφέρεται στις μεταβλητές για τις οθόνες. Από εκεί μεταφράζεται και ξεκινάει η πολυπλεξία για την προβολή στις οθόνες.

Μέσα από όλη αυτή τη διαδικασία η πληροφορία (αν δεν υπάρξει κάποιο σφάλμα στην επικοινωνία λόγω θορύβου ή λανθασμένου χρονισμού) καταφέρνει να μεταφερθεί επιτυχώς και να προβληθεί στις οθόνες μας.

Σχηματικό Διάγραμμα



<u>Σημείωση</u> όλα τα σχηματικά διαγράμματα έχουν υλοποιηθεί με το εργαλείο figma λόγω προβλήματος με το questa.