ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

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

Εργασία στο Μάθημα :

«Ψηφιακά Συστήματα HW σε Χαμηλά Επίπεδα Λογικής Ι»

Καρβουνάρης Παναγιώτης (ΑΕΜ: 10193)

Μποσινάκη Κωνσταντίνα (ΑΕΜ: 10082)

(Ομάδα 2)

Επιβλέπων καθηγητής:

Σκετόπουλος Νικόλαος

Θεσσαλονίκη, Ιανουάριος 2023

Περιεχόμενα

Εισαγωγή

Στην παρούσα εργασία, μας ζητήθηκε να υλοποιήσουμε ένα σύστημα Αποστολέα-Δέκτη με χρήση του πρωτόκολλο UART και έναν οδηγό ενδείξεων 7-τμημάτων τεσσάρων LED για να προβληθεί μία ακολουθία αλφαριθμητικών και ειδικών χαρακτήρων.

Αρχικά, μας ζητήθηκε η μεμονωμένη υλοποιηση των επιμέρους τμημάτων και στη συνέχεια η αρμονική συνένωσή τους. Το πρώτο στάδιο ήταν η υλοποίηση ενός αποκωδικοποιητή 7 τμημάτων για την εμφάνιση 4 ψηφίων σε LED οθόνες . Το δεύτερο στάδιο ήταν η επικοινωνία ενός Αποστολέα και ενός Δέκτη για τη μεταφορά με UART πρωτόκολλο μίας ακολουθίας bit. Και το τελικό στάδιο ήταν η ένωσή τους ώστε μία ακολουθία από bit να σταλεί από τον Αποστολέα στον Δέκτη έγκυρα και στη συνέχεια να περάσει από τον τελευταίο στο στάδιο της αποκωδικοποίησης και τελικά να εμφανιστούν στις σωστές οθόνες τα αντίστοιχα ψηφία που επιλέχθηκαν κατά την είσοδο στον Αποστολέα.

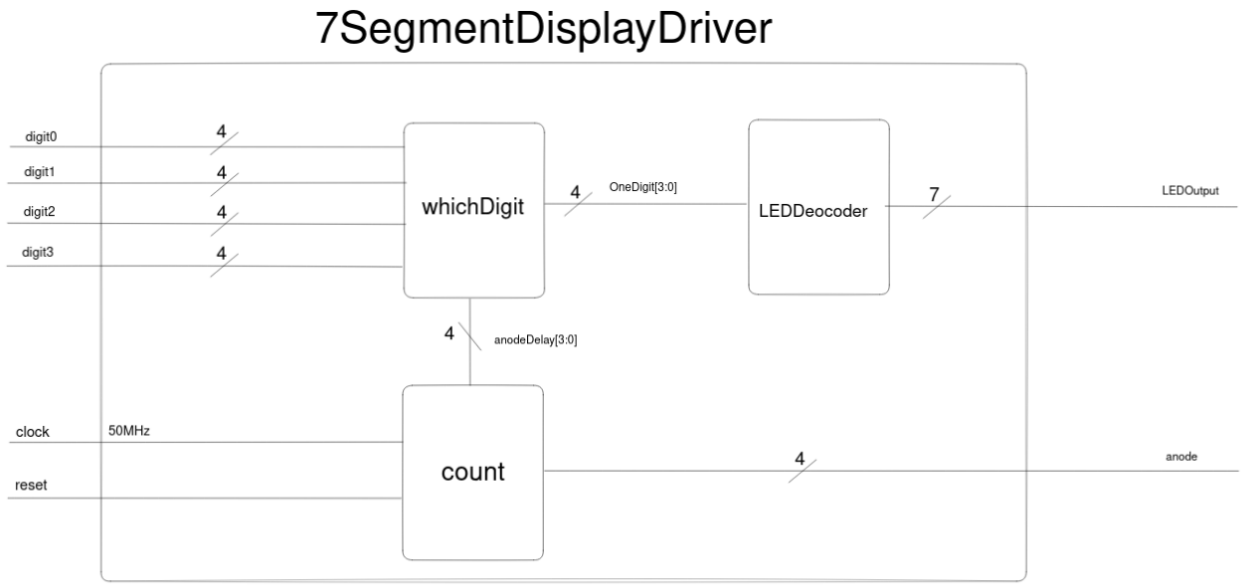
Στόχος της εργασίας ήταν η εξοικείωση με την γλώσσα verilog η οποία είναι γλώσσα περιγραφής υλικού (VHDL).

Για την επίτευξη της εργασίας χρησιμοποιήσαμε το excalidraw ως εργαλείο δημιουργίας σχηματικών διαγραμμάτων και των FSM, ενώ για την γραφή του κώδικα και τις προσομοιώσεις χρησιμοποιήσαμε το online εργαλείο EDA Playground. Για την εξαγωγή των κυματομορφών χρησιμοποιήσαμε επίσης το εργαλείο Icarus Verilog for Windows.

Α' Μέρος

7 Segment Display Driver

1. Σχηματικό διάγραμμα του κυκλώματος.



Στο παραπάνω σχήμα φαίνονται τα διαγράμματα των 3 module (*whichDigit*, *count, LEDDecoder*), καθώς και του module ολόκληρου του κυκλώματος (*7SegmentDisplayDriver*).

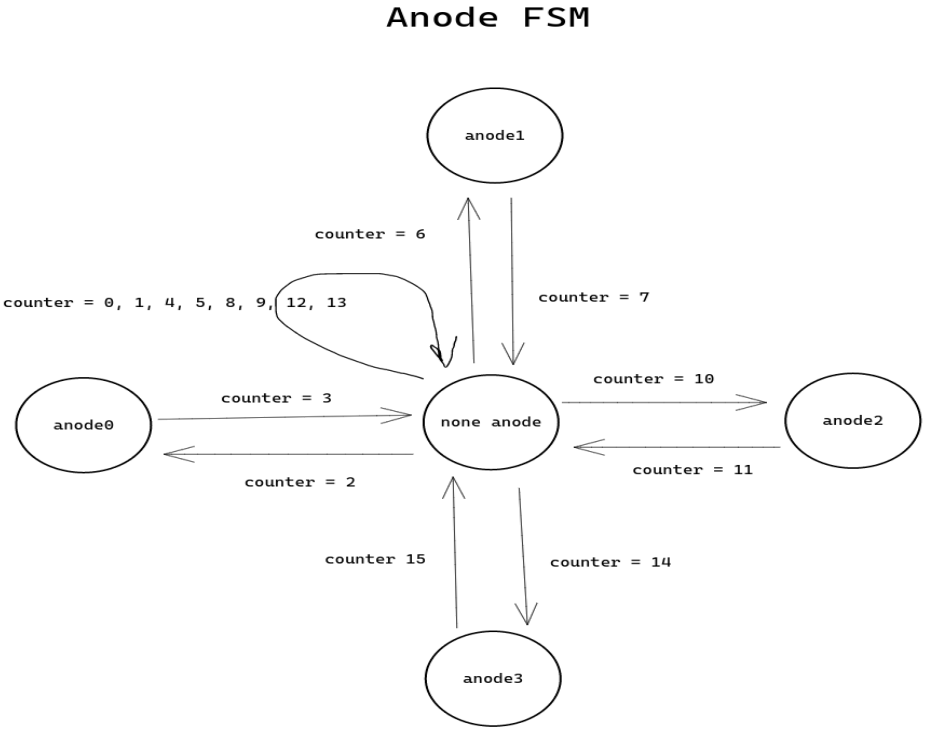
- Το *top module* έχει 6 **εισόδους**, μία για κάθε ένα από τα **4 ψηφία** που θέλουμε να αναπαραστήσουμε (digit0, digit1, digit2, digit3) και 2 εισόδους που χρησιμοποιούνται ως clock και reset. Επιπλέον, διαθέτει 2 **εξόδους**, ένα 7-bit σήμα (**LEDOutput**) που διαθέτει την πληροφορία για το ποια LED θα ανάψουν και ενα 4-bit (**anode**) που οδηγεί την παραπάνω πληροφορία στην σωστή οθόνη.
- Το *count module* παίρνει σαν εισόδους τα clock και reset με βάση τα οποία ανανεώνεται η μεταβλητή counter (από 0 μέχρι 15). Το σήμα counter όπως αναφέρεται στην εκφώνηση της εργασίας ανανεώνεται κάθε 16 κύκλους του ρολογιού (0,32μsec). Για το λόγο αυτό και επειδή έχει

χρησιμοποιηθεί το posetive edge του clock στη sensitivity list του always block για την ανανέωση του counter, χρησιμοποιήθηκε άλλο ένα σήμα, το count, το οποίο μετράει τους κύκλους του ρολογιού και κάθε 16 κύκλους αυξάνει το counter κατά 1. Ανάλογα με την τιμή του counter το module παράγει δύο εξόδους των 4 bit, τις anodeDelay και anode. Η μεταβλητή **anodeDelay** χρησιμοποιείται στη συνέχεια ως είσοδος στο ***whichDigit*** *module* και **ενεργοποιεί** ανάλογα με την τιμή της το **αντίστοιχο** από τα **4 ψηφία** που θέλουμε να εμφανίσουμε στην οθόνη. Η μεταβλητή **anode** βγαίνει σαν έξοδος ολόκληρου του κυκλώματος και ανάλογα με την τιμή της **ενεργοποιεί** την **αντίστοιχη οθόνη LED** ώστε να εμφανιστεί το εισαγμένο ψηφίο με τη μορφή της 7-bit μεταβλητής. LED

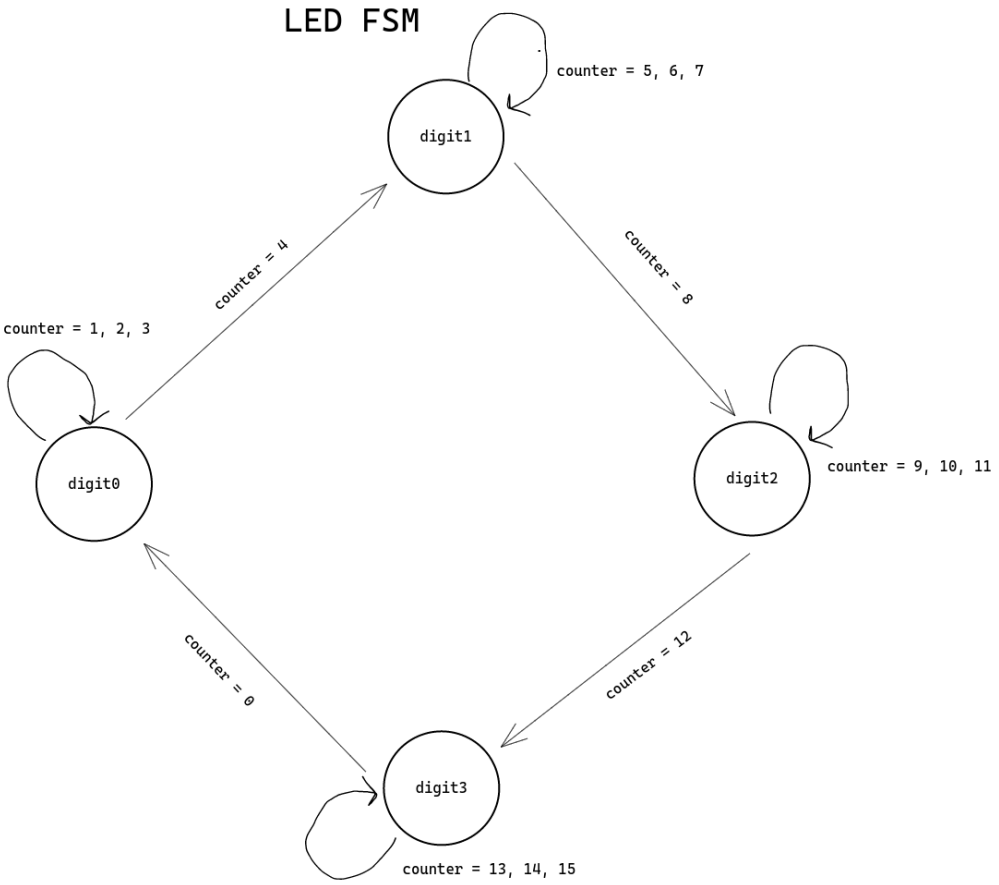
- Το *whichDigit module* έχει σαν εισόδους τα **4 ψηφία** που πρέπει να εμφανιστούν στις οθόνες, καθώς και το σήμα 4-bit (**anodeDelay**) το οποίο δίνει την πληροφορία ποιό από τα τέσσερα ψηφία πρέπει να ενεργοποιηθεί κάθε φόρα. Ως έξοδο έχει το 4-bit ψηφίο που επιλέγεται με την παραπάνω διαδικασία.
- Το *LEDDecoder module* **αποκωδικοποιεί** το 4-bit ψηφίο που παίρνει σαν είσοδο σε ένα LED ψηφίο των 7 bit.

2. Σχηματικά των υλοποιημένων FSM.

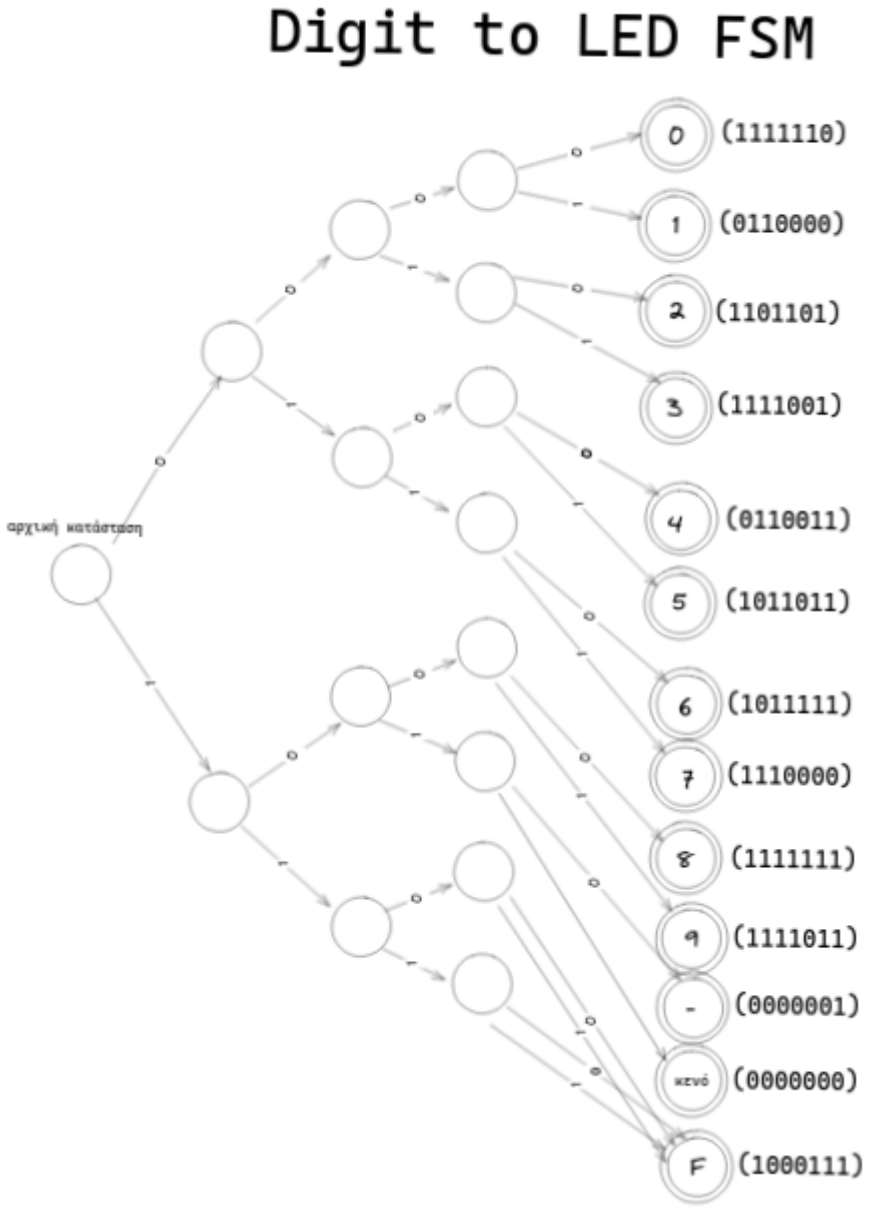
Σχηματικό των **καταστάσεων της ανόδου** στο οποίο φαίνεται ποια άνοδος είναι ενεργή ανάλογα με την τιμή του counter. Σε ορισμένες τιμές του counter καμία άνοδος δεν είναι ενεργοποιημένη.



Σχηματικό των **καταστάσεων του ψηφίου** που φαίνονται στην οθόνη στο οποίο φαίνεται ποιο ψηφίο εμφανίζεται στην οθόνη για κάθε τιμή του counter. Ο counter είναι ο ίδιος που αναφέρεται και στα παραπάνω FSM. Λαμβάνουμε υπόψη ότι η εναλλαγές των ψηφίων είναι συνεχείς και καμία στιγμή η οθόνη δεν θα μείνει χωρίς αναμένω ψηφίο.

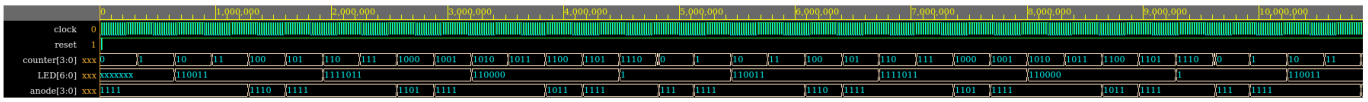


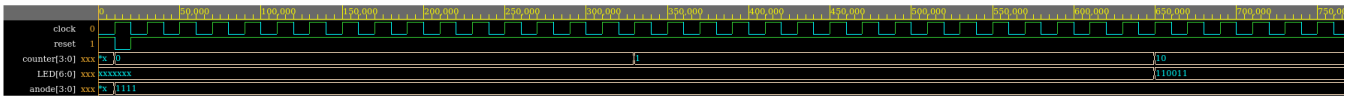
Σχηματικό των **καταστάσεων των LED** που φαίνονται στην οθόνη στο οποίο φαίνεται ποιο ψηφίο εμφανίζεται (δηλαδή ποια LED φώτα ανάβουν) στην οθόνη για κάθε τιμή του 4bit εισερχόμενου ψηφίου. Κάθε ψηφίο από 0 έως και 11 αντιστοιχεί σε μία ξεχωριστή κατάσταση, ενώ τα ψηφία 12 -> 15 αντιστοιχούν στην ίδια κατάσταση. Με κάθε νέα είσοδο ψηφίου η διαδικασία επαναλαμβάνεται από την αρχική κατάσταση.



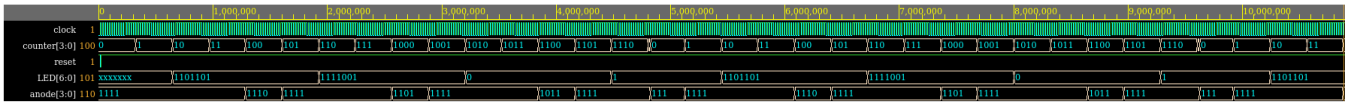
3. Κυματομορφές

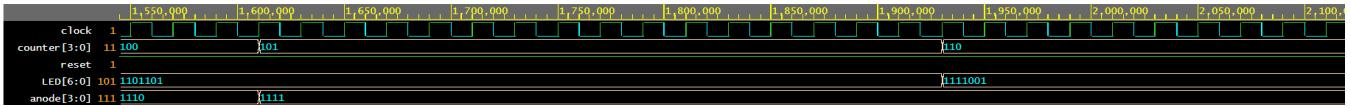
Εμφάνιση στην οθόνη των χαρακτήρων “-194”.





Εμφάνιση στην οθόνη των χαρακτήρων “- 32”.





Παρατηρούμε ότι το σήμα **LED** βρίσκεται στην έξοδο **δύο κύκλους ρολογιού πριν** από το σωστό σήμα **ανόδου**. Όμως σε αυτήν την διάρκεια η άνοδος έχει την τιμή 1111 οπότε το σήμα LED δεν αντιστοιχείται σε καμία οθόνη, αλλά **περιμένει** μέχρι να **αλλάξει** το σήμα **anode** ώστε να γίνει σωστή οδήγηση. Όταν είναι **ενεργοποιημένη** κάποια **άνοδος**, δηλαδή υπάρχει κάποιο 0 στα 4 bit της μεταβλητή anode, τα 7 bit της μεταβλητής **LED** αντιστοιχούν στο **ψηφίο που πρέπει να εμφανιστεί στη συγκεκριμένη άνοδο.**

4. Περιγραφή

Για την υλοποίηση του κωδικοποιητή των 4 σε 7 bit και την εμφάνιση των ψηφίων στην αντίστοιχη οθόνη LED, ακολουθήσαμε μία λογική όπως περιγράφεται στο παραπάνω σχηματικό διάγραμμα. Παρακάτω αναφέρουμε κάποιες πιο ειδικές πληροφορίες σχετικά με τη λειτουργία του προγράμματος:

- Η **περίοδος** του **clock** έχει οριστεί στα 0,20 nsec.

- Το **reset** ενεργοποιείται στην αρχή και έπειτα παραμένει στο 1 για την υπόλοιπη διάρκεια της προσομοίωσης.
- Το **clock** είναι υπεύθυνο για τον χρονισμό του συστήματος καθώς οι αλλαγές συμβαίνουν στο **positive edge** του.
- Τα **anode** και **anodeDelay** είναι τα σήματα που είναι υπεύθυνα για τον **συγχρονισμό** των εξόδων προσθέτοντας μία καθυστέρηση στην προετοιμασία των δεδομένων.

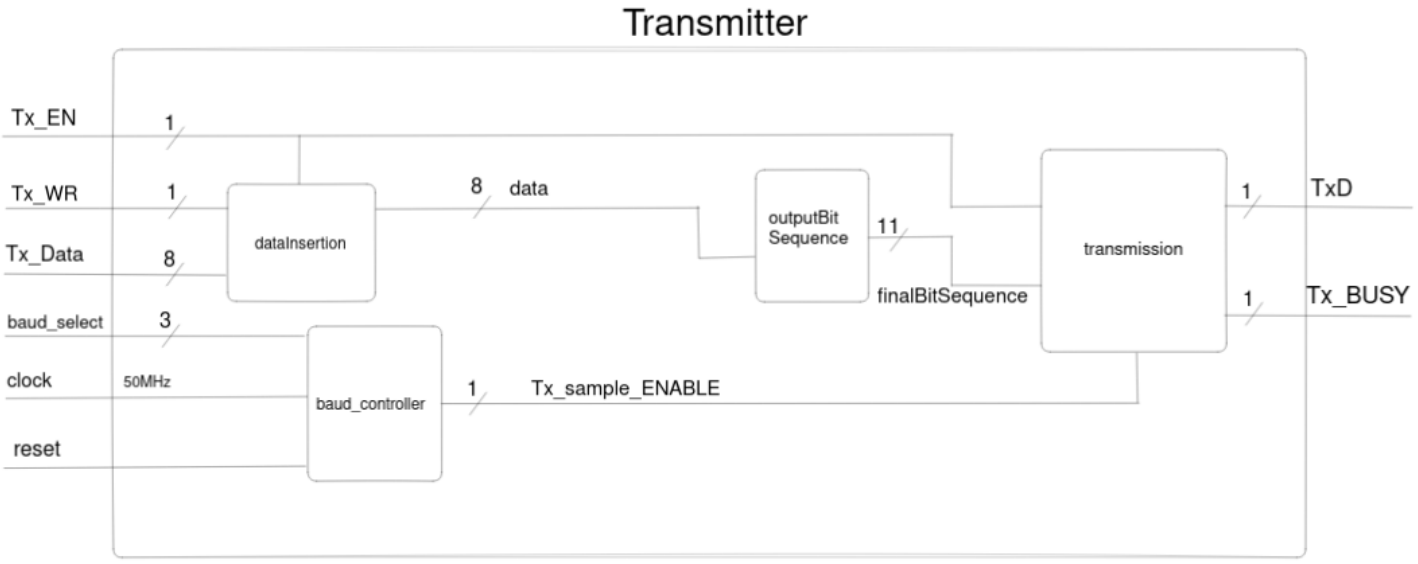
5. Προβλήματα

- Μία από τις δυσκολίες που αντιμετωπίσαμε αφορά τον **συγχρονισμό** της ενεργοποίησης της σωστής ανόδου όταν η τιμή της μεταβλητής LED ταυτίζεται με το ψηφίο που πρέπει να εμφανιστεί. Όπως αναφέρεται και στην εκφώνηση της εργασίας, το σήμα που θα δοθεί με βάση την τιμή του counter στο whichDigit module ώστε να περάσει σαν είσοδος στο LEDDecoder module το σωστό 4-bit ψηφίο πρέπει να προηγείται κατά 2 τιμές του counter του σήματος για να ενεργοποιηθεί η αντιστοιχη οθόνη LED. Για να το πετύχουμε αυτό δημιουργήσαμε τα σήματα τα anode και anodeDelay, την λειτουργία των οποίων αναφέραμε παραπάνω.
- Δυσκολευτήκαμε να καταλάβουμε τη χρήση του **reset** και τη σχέση του με το counter και το clock.

Β' Μέρος

Transmitter

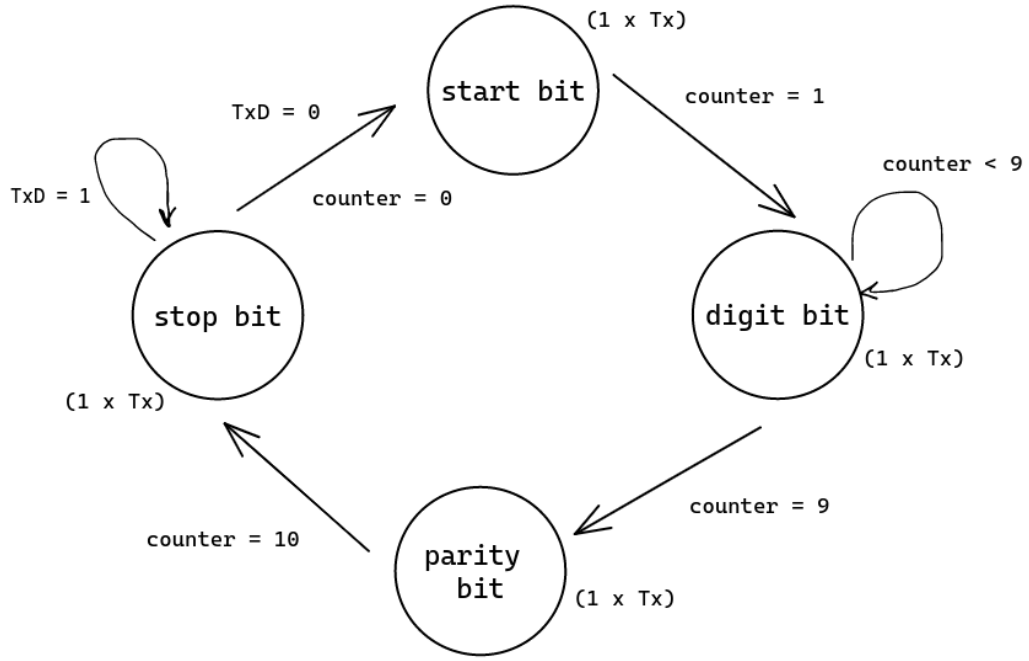
1. Σχηματικό διάγραμμα του κυκλώματος.



2. Σχηματικά των υλοποιημένων FSM.

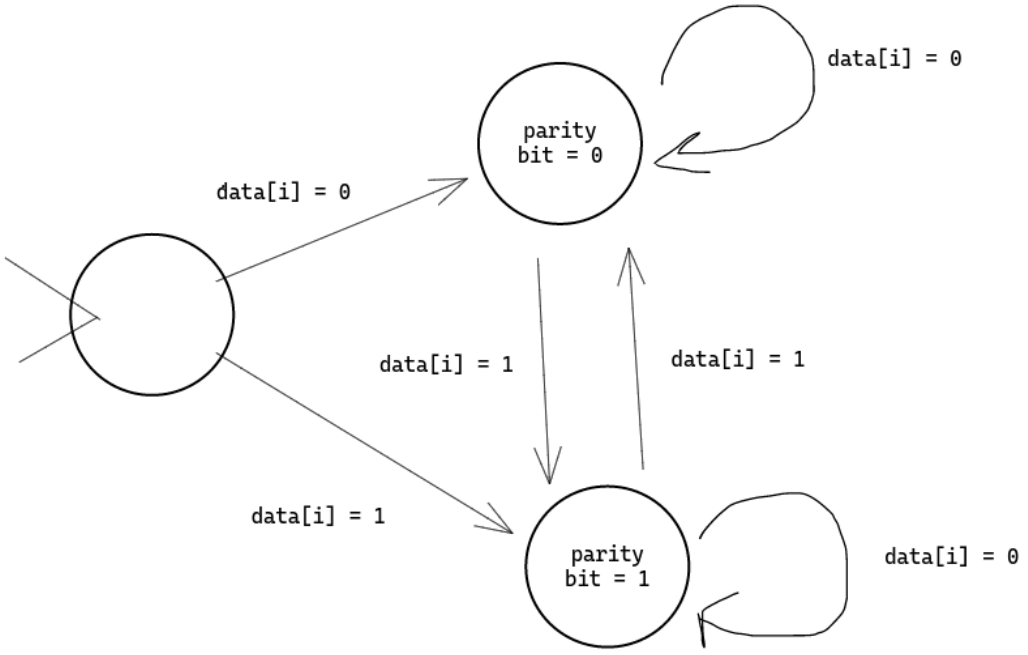
Σχηματικό διάγραμμα των **καταστάσεων της εξόδου του Transmitter** στο οποίο φαίνεται η αλληλουχία των λαμβανόμενων bit. Ο counter είναι ένα σήμα το οποίο μετράει τον αριθμό των bit που έχουν τεθεί προς αποστολή με αρχή (counter = 0) το start bit. Τα digit bit είναι τα bit του 8-bit ψηφίου της εισόδου.

Transmitter FSM



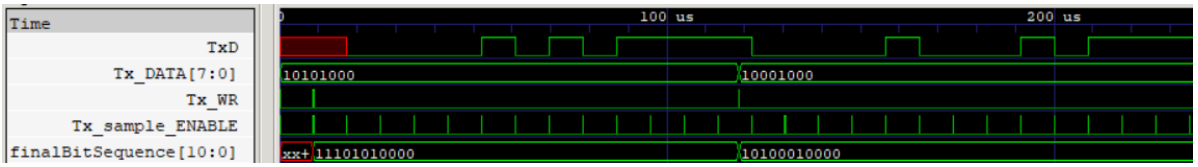
Σχηματικό διάγραμμα των **καταστάσεων της εξόδου του Transmitter** στο οποίο φαίνεται η αλληλουχία των λαμβανόμενων bit. Ο counter είναι ένα σήμα το οποίο μετράει τον αριθμό των bit που έχουν τεθεί προς αποστολή με αρχή (counter = 0) το start bit. Τα digit bit είναι τα bit του 8-bit ψηφίου της εισόδου.

Parity Bit Calculator FSM



3. Κυματομορφές

Στην παρακάτω κυματομορφή φαίνονται τα σήματα Tx_DATA, finalBitSequence και TxD. Το σήμα **finalBitSequence** αποτελείται από το σήμα Tx_DATA προσθέτοντας τα κατάλληλα start, stop και parity bit. Στις τιμές του TxD (1bit) εμφανίζονται **με τη σειρά** τα αντίστοιχα bit από την ακολουθία bit του σήματος finalBitSequence. Επίσης, φαίνεται τόσο η **ενεργοποίηση** του **Tx_WR** για μία περίοδο clock καθώς και η **περιοδική ενεργοποίηση του Tx_sample_ENABLE. Σε κάθε ενεργοποίηση του αλλάζει και η τιμή του TxD**.



4. Περιγραφή

Για την υλοποίηση του Transmitter χρησιμοποιήσαμε τα παρακάτω module:

- Το *baud_controller* module, στο οποίο υπολογίζεται η **περίοδος μετάδοσης ενός bit**. Η περίοδος αυτή εξαρτάται από το σήμα baud_select το οποίο όπως αναφέρεται στη εικόνα 9 της εκφώνησης της εργασίας αντιστοιχεί σε τιμές για το baud rate. Η έξοδος του module είναι το σήμα sample_ENABLE το οποίο παίρνει την τιμή 1 για ένα κύκλο του clock φορά που μεταδίδεται ένα bit στην έξοδο του transmitter. Η απόσταση μεταξύ 2 διαδοχικών τιμών 1 του sample_ENABLE ισούται με την περίοδο μετάδοσης bit του Transmitter. Ο υπολογισμός αυτής της περιόδου έγινε με βάση τον παρακάτω πίνακα.

Baud Rate	Tsc	Max Counter	Max Counter (Rounded)
300	0.003333333	166666.65	166666
1200	0.000833333	41666.65	41666
4800	0.000208333	10416.65	10416

Baud Rate	Tsc	Max Counter	Max Counter (Rounded)
9600	0.000104170	5208.5	5208
19200	0.000052083	2604.15	2604
38400	0.000026041	1302.05	1302
57600	0.000017361	868.05	868
115200	0.000008680	434	434

➔ Στην πρώτη στήλη του πίνακα βρίσκεται το **baud rate** που αντιστοιχεί στην εκάστοτε είσοδο **baud_select**.

➔ Στη δεύτερη στήλη έχουμε τις τιμές της **περιόδου μετάδοσης** του transmitter οι οποίες υπολογίζονται από τον εξής τύπο:
Tsc = 1/baud_rate

➔ Στην τρίτη στήλη έχουμε υπολογίσει έναν **max counter** ο οποίος αντιστοιχεί στο πλήθος των περιόδων του clock που “χωράνε” σε μία περίοδο του transmitter. Ο τύπος για τον υπολογισμό του max counter είναι ο εξής:
max_counter= Tsc/20 (όπου 20 η περίοδος του clock)
Καθώς κατά τη διαίρεση ο max counter συχνά έπαιρνε δεκαδικές τιμές, στην τελευταία στήλη του πίνακα βρίσκονται οι τιμές του **max counter στρογγυλοποιημένες** στο χαμηλότερο κοντινό ακέραιο αριθμό.

➔ Τέλος έχουμε χρησιμοποιήσει έναν **counter** ο οποίος αυξάνεται κατά 1 σε κάθε clock και μόλις φτάνει τη τιμή του max counter, το σήμα **sample_ENABLE** θα γίνεται **1**, και ο μετρητής θα επιστρέφει στο μηδέν. Με αυτόν τον τρόπο, κάθε περίοδο Tsc το σήμα sample_ENABLE θα γίνεται 1 για έναν κύκλο.

- Το *dataInsertion* module, το οποίο χρησιμοποιείται για την είσοδο των δεδομένων στο Transmitter module. Πρακτικά, κάθε φορά που το **Tx_WR**

ενεργοποιείται (δηλαδή έχει spike διάρκειας ενός clock του ρολογιού) και καθώς είναι ενεργό το σύστημα σύμφωνα με το σήμα **Tx_EN**, τα δεδομένα που βρίσκονται στο **Tx_Data εισέρχονται στον Transmitter** για επεξεργασία και αποστολή.

- Το *outputBitSequence* module, το οποίο χρησιμοποιείται για τον σχηματισμό της ακολουθίας των 11 bit. Παίρνει σαν **είσοδο** τα **8 bit που θα μεταδοθούν** και το **parity bit**. Το parity bit είναι ένα bit επαλήθευσης, που σχετίζεται με τον αριθμό των άσσων που βρίσκονται στο επιθυμητό 8_bit σήμα. Μέσα στο module ελέγχονται τα 8 bit του εισερχόμενου ψηφίου και μετράται το πλήθος των 1. Αν το πλήθος είναι άρτιος αριθμός το parity bit θα είναι 0, ενώ αν είναι περιττός το parity bit θα είναι 1.
Στην **έξοδο** βγαίνει ένα σήμα των **11 bit** στο οποίο το 1ο και το 11ο bit είναι πάντα το start bit (0) και stop bit (1), στα bit 1 έως 8 είναι τα 8 bit του μεταδιδόμενου ψηφίου και το 9ο bit (προτελευταίο) αντιστοιχεί στο parity bit .

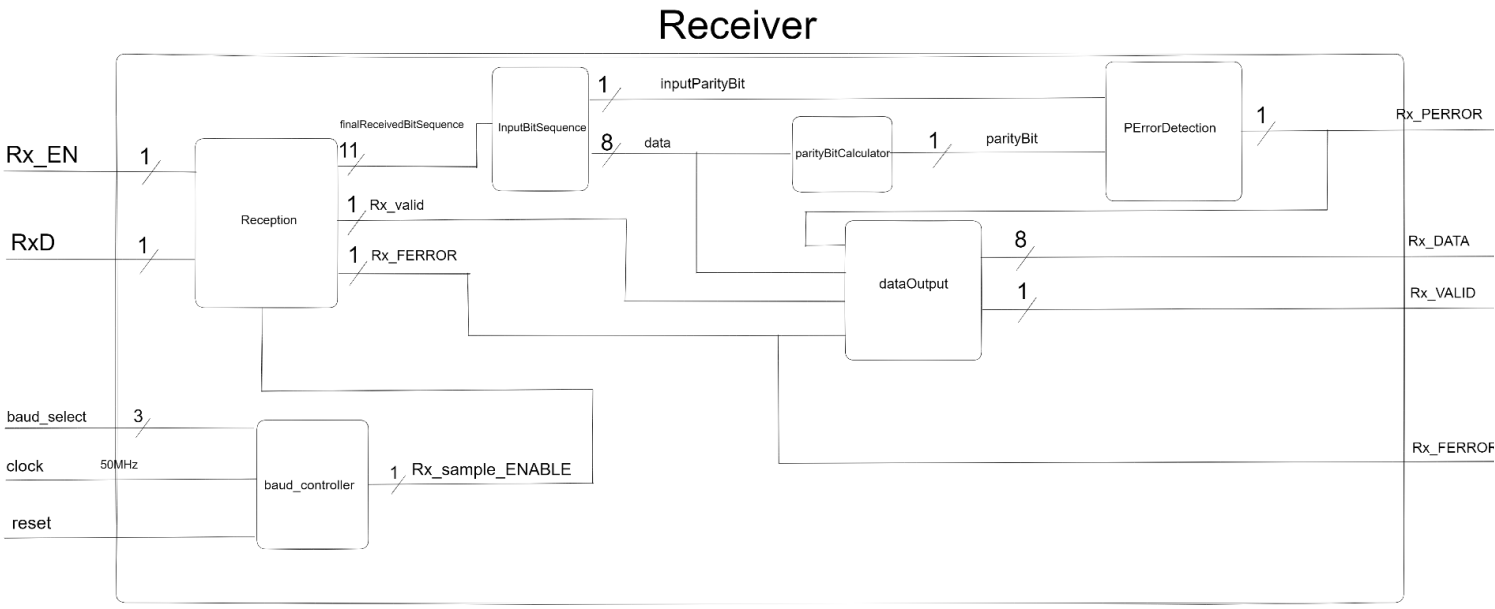
- Το *transmission* module, το οποίο :
  - ➔ παίρνει σαν **είσοδο** το σχηματισμένο σήμα των 11 bit (**finalBitSequence**) που θέλουμε να μεταδοθεί, το σήμα Tx_EN και το Tx_sample_ENABLE. Κάθε φορά που το σήμα Tx_sample_ENABLE γίνεται 1, το σήμα εξόδου TxD παίρνει την τιμή ενός bit από το σήμα finalBitSequence. Η σειρά του bit που θα περάσει στο TxD καθορίζεται από έναν **μετρητή (i)**.
  - ➔ Το **i** κάθε φορά που μπαίνει ένα καινούργιο σήμα finalBitSequence (δηλαδή κάθε φορά που εισέρχεται ψηφίο προς μετάδοση), αρχικοποιείται στην τιμή -1. Στη συνέχεια, κάθε φορά που το σήμα Tx_sample_ENABLE γίνεται 1, το i αυξάνεται κατά 1 και δείχνει άρα στο αμέσως επόμενο στοιχείο του finalBitSequence. Ο λόγος που το i αρχικοποιείται στο -1 είναι για να ξεκινήσει κατα την μετάδοση του πρώτου bit του finalBitSequence από το 0 και όχι από το 1 (αν είχε αρχικοποιηθεί στο 0).
  - ➔ Το σήμα **Tx_BUSY** είναι συνεχώς στο 1 (υποδηλώνοντας ότι ότι ο Αποστολέας βρίσκεται σε διαδικασία μετάδοσης και δεν μπορεί

να λάβει το επόμενο σύμβολο) και γίνεται 0 μόνο όταν το i έχει την τιμή 11 (δηλαδή έχουν περάσει όλα τα bit του finalBitSequence στο TxD).

➔ Τέλος, όταν το σήμα **Tx_WR** γίνεται 1 (μέσω του testbench) το i γίνεται 0 ώστε να αρχίσει η μεταφορά των bit του finalBitSequence στο TxD.

Receiver

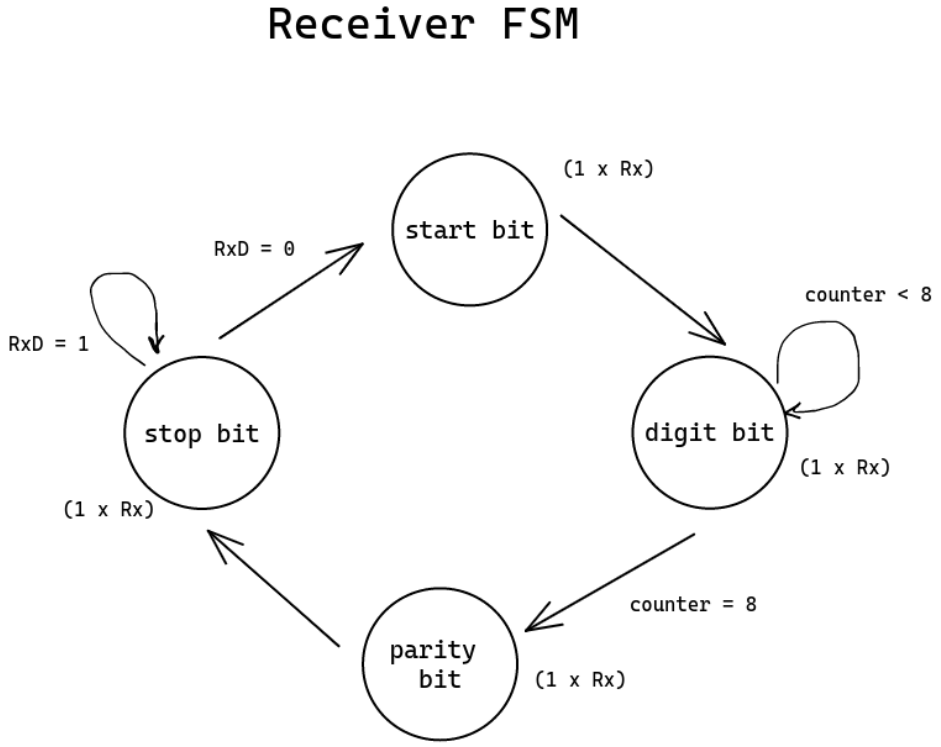
1. Σχηματικό διάγραμμα του κυκλώματος.



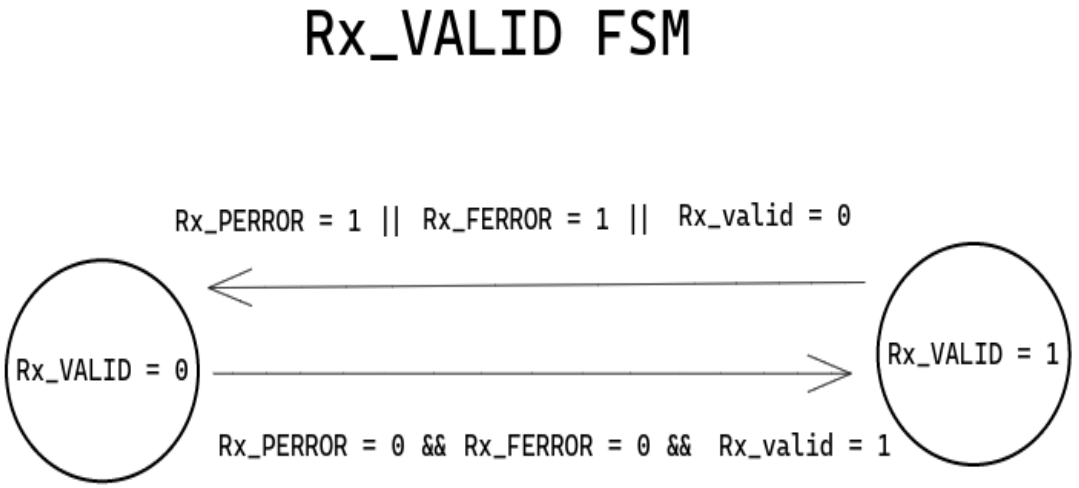
2. Σχηματικά των υλοποιημένων FSM.

Σχηματικό διάγραμμα των **καταστάσεων της εισόδου του Transmitter** στο οποίο φαίνεται η αλληλουχία των λαμβανόμενων bit. Ο counter είναι ένα σήμα το οποίο μετράει τον αριθμό των bit που έχουν ληφθεί, με αρχή (counter = 0) το start bit. Τα digit bit είναι τα bit του 8-bit ψηφίου που θα μεταδοθούν στην συνέχεια στην έξοδο. Όταν η είσοδος του Receiver

βρίσκεται στην κατάσταση stop bit, παραμένει εκεί μέχρι να λάβει start bit (RxD = 0)

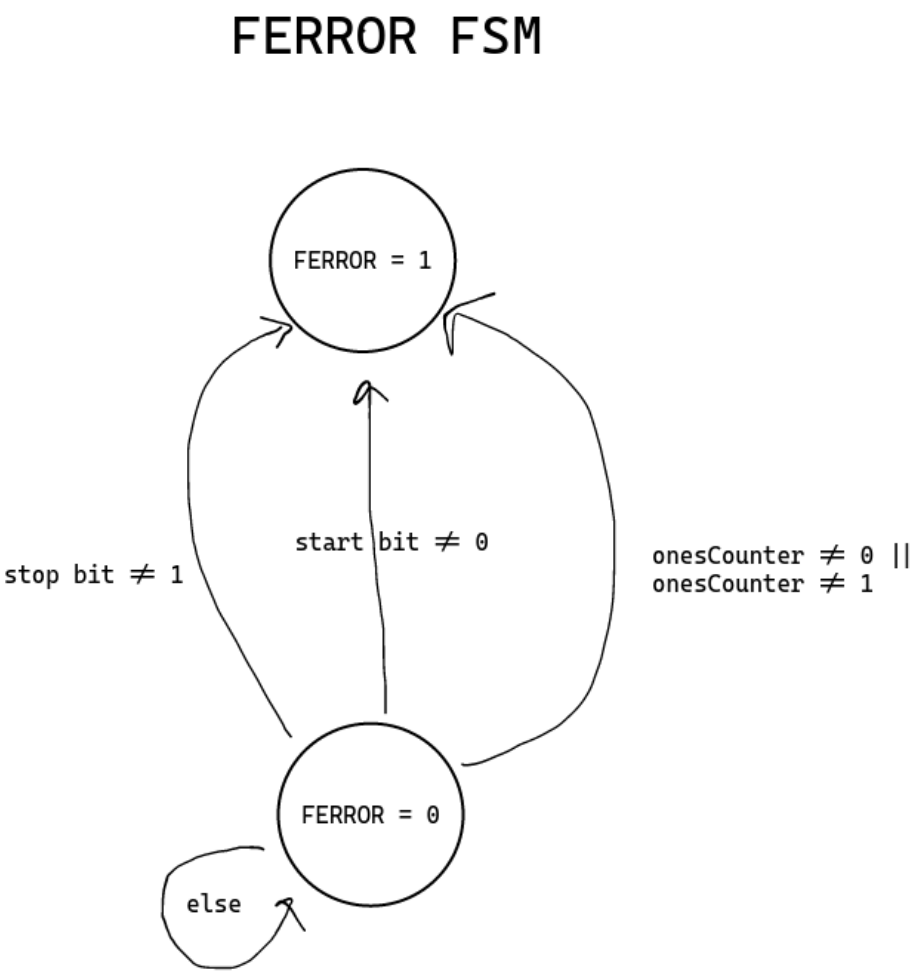


Σχηματικό διάγραμμα των **καταστάσεων του σήματος Rx_VALID του Receiver** στο οποίο. Φαίνεται η εξάρτηση του σήματος από τα σήματα Rx_FERROR, Rx_PERROR και Rx_valid για την μετάβαση από την τιμή 0 στην 1 και αντιστρόφως.



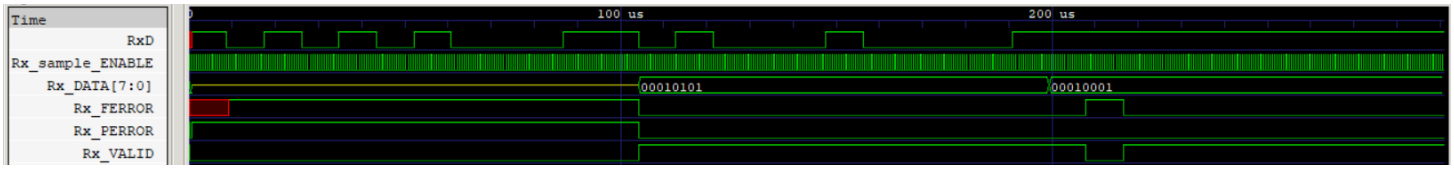
Σχηματικό διάγραμμα των **καταστάσεων του σήματος Rx_FERROR** του Δέκτη στο οποίο φαίνονται οι τιμές που λαμβάνει το σήμα. Υπάρχουν 3 περιπτώσεις ώστε να γίνει το σήμα ίσο με 1 (δηλαδή να έχουμε σφάλμα):

- το **stop** bit να μην είναι ίσο με 1
- το **start** bit να μην είναι ίσο με 0
- εφόσον κάνουμε 15 **δειγματοληψίες** για κάθε λαμβανόμενο bit, κρατάμε σε έναν counter τον αριθμό των δειγματοληψιών που είναι ίσες με 1 και αν αυτός ο counter έχει τιμή διαφορετική από 0 ή 15 (που σημαίνει ότι τουλάχιστον μία δειγματοληψία έχει τιμή διαφορετική από τις υπόλοιπες) τότε ενεργοποιείται το σφάλμα.



3. Κυματομορφές που αποδεικνύουν την ορθή λειτουργία του κυκλώματος.

Στην παρακάτω κυματομορφή βλέπουμε πως τα bit του σήματος RxD μεταφέρονται στο σήμα (8bit) Rx_DATA αφού αφαιρεθούν τα start, stop και parity bits. Επίσης, φαίνονται και τα σήματα λάθους και VALID. Η τιμή του τελευταίου όπως φαίνεται είναι ίση με 1 κατά την περίοδο που λαμβάνουμε το σήμα Rx_DATA.



Στην παρακάτω κυματομορφή απεικονίζεται η σχέση των σημάτων RxD και Rx_sample_ENABLE. Κάθε 16 παλμούς του Rx_sample_ENABLE αλλάζει η τιμή του RxD και λαμβάνεται το επόμενο bit από τον Δεκτη.

4. Περιγραφή

Για την υλοποίηση του Receiver χρησιμοποιήσαμε τα παρακάτω module:

- Το *baud_controller* module, στο οποίο υπολογίζεται η **περίοδος λήψης ενός bit** . Η περίοδος αυτή εξαρτάται από το σήμα baud_select το οποίο όπως αναφέρεται στη εικόνα 9 της εκφώνησης της εργασίας αντιστοιχεί σε τιμές για το baud rate. Η έξοδος του module είναι το σήμα sample_ENABLE το οποίο παίρνει την τιμή 1 για ένα κύκλο του clock 16 φορές κατά τη διάρκεια που μεταδίδεται ένα bit στην έξοδο του transmitter. Η απόσταση μεταξύ 2 διαδοχικών τιμών 1 του sample_ENABLE ισούται με την περίοδο δειγματοληψίας του Receiver. Ο υπολογισμός αυτής της περιόδου έγινε με βάση τον παρακάτω πίνακα.

Baud Rate	Tsc	Max Counter	MaxCounter (Rounded)
300	0.00020833	10416.5	10416
1200	0.00005208	2604	2604
4800	0.00001302	651	651
9600	0.00000651	325.5	325
19200	0.00000326	163	163
38400	0.00000163	81.5	81
57600	0.00000109	54.5	54
115200	0.00000054	27	27

➔ Στην πρώτη στήλη του πίνακα βρίσκεται το **baud rate** που αντιστοιχεί στην εκάστοτε είσοδο **baud_select**.

➔ Στη δεύτερη στήλη έχουμε τις τιμές της **περιόδου δειγματοληψίας** του **Receiver** οι οποίες υπολογίζονται από τον εξής τύπο:

Tsc = 1/16*baud_rate

➔ Στην τρίτη στήλη έχουμε υπολογίσει έναν **max counter** ο οποίος αντιστοιχεί στο πλήθος των περιόδων του clock που “χωράνε" σε μία περίοδο του receiver. Ο τύπος για τον υπολογισμό του max counter είναι ο εξής:

max_counter= Tsc/20 (όπου 20 η περίοδος του clock)

Καθώς κατά τη διαίρεση ο max counter συχνά έπαιρνε δεκαδικές τιμές, στην τελευταία στήλη του πίνακα βρίσκονται οι τιμές του **max counter στρογγυλοποιημένες** στο χαμηλότερο κοντινό ακέραιο αριθμό.

➔ Τέλος έχουμε χρησιμοποιήσει έναν counter ο οποίος αυξάνεται κατά 1 σε κάθε clock και μόλις φτάνει τη τιμή του max counter, το σήμα **sample_ENABLE** θα γίνεται **1**, και ο μετρητής θα επιστρέφει στο μηδέν. Με αυτόν τον τρόπο, κάθε περίοδο Tsc το σήμα sample_ENABLE θα γίνεται 1 για έναν κύκλο.

Σε σχέση με την περίοδο του transmitter, η περίοδος του receiver είναι 16 φορές μικρότερη.

- Το *reception* module, το οποίο είναι υπεύθυνο για την "**παραλαβή**" των bit που στέλνει ο transmitter, την **ένωσή** τους για τη **δημιουργία** του **11-bit σήματος** που στάλθηκε και την δημιουργία ορισμένων σημάτων ελέγχου.
    - ➔ Αρχικά, έχουμε ως είσοδο το **Rx_EN** που αποτελεί σήμα ενεργοποίησης του receiver. Χρησιμοποιούμε το σήμα Rx_sample_ENABLE, που είναι 16 φορές πιο γρήγορο από το Tx_sample_ENABLE, ως ενεργούς κύκλους του receiver. Κάθε φορά που το **Rx_sample_ENABLE** γίνεται **1 δειγματοληπτούμε την τιμή του RxD,** έτσι θα πάρουμε **16 τιμές του ίδιου RxD.**
    - ➔ Άρα αξιοποιώντας τον κανόνα της **πλειοψηφίας** (μέσω του σήματος **onesCounter** που μετράει το **πλήθος των 1** που δειγματοληπτήθηκαν) καταλήγουμε να πάρουμε το σωστό bit και να το αποθηκεύσουμε, αντιμετωπίζοντας λάθη που συμβαίνουν στο κανάλι είτε λόγω κάποιας καθυστέρησης είτε λόγω θορύβου, και αντίστοιχα προχωράμε στα επόμενα bit.
    - ➔ Όμως, επειδή τα Rx_sample_ENABLE και Tx_sample_ENABLE **δεν είναι πλήρως συγχρονισμένα** δηλαδή κάθε ένα Tx_sample_ENABLE δεν είναι ακριβώς 16 * Rx_sample_ENABLE, δημιουργείται κάποιο **σφάλμα** στην πλαισίωση των δεδομένων λήψης. Παρατηρήσαμε ότι υπάρχει μία **απόκλιση 2 περιόδων clock** στο συγχρονισμό του transmitter και του receiver. Οπότε, ενώ αρχικά και οι 16 δειγματοληψίες βρίσκονται μέσα στο εύρος μετάδοσης του bit και άρα έχουν την ίδια τιμή, μετά την μετάδοση ορισμένων bit (αθροίζοντας τις 2 περιόδους clock απόκλισης για όλα τα προηγούμενα bit) προκύπτει σφάλμα. Δηλαδή, **μία από τις 16 δειγματοληψίες** (και συγκεκριμένα η πρώτη) **βρίσκεται μέσα στο εύρος περιόδου του προηγούμενου bit και άρα έχει διαφορετική τιμή**.
    - ➔ Ένας τρόπος αύξησης του χρόνου μέχρι την δημιουργία αυτού του σφάλματος είναι η **χρήση των 15 από τις 16 συνολικά**

δειγματοληψίες, δηλαδή να μην λαμβάνουμε υπόψη το αποτέλεσμα από την πρώτη από αυτές.

➔ Φυσικά, στη συνέχεια, μετά από τον ίδιο αριθμό bit θα προκύψει **εκ νέου το σφάλμα**, καθώς πλέον και η δεύτερη δειγματοληψία θα βρίσκεται εκτός των ορίων. Ωστόσο, υπολογίσαμε πως αυτό θα γίνει ξανά **μετά από 13 bit** (540 = περίοδος του Rx_sample_ENABLE για baud select = 111 / 40, όπου 40ns = 2 περίοδοι clock), και εφόσον εμείς ενδιαφερόμαστε για τη μετάδοση άλλων 9, δεν θα μας απασχολήσει.

➔ Όταν ολοκληρωθεί η λήψη των δεδομένων δημιουργούμε το σήμα **Rx_valid** που μας πληροφορεί ότι **έχουμε λάβει stop bit**, άρα τα δεδομένα έχουμε ληφθεί με **επιτυχία**.

➔ Το σήμα **Rx_ferror** είναι το σφάλμα που σχετίζεται με την **πλαισίωση** των δεδομένων και εμφανίζεται όταν τουλάχιστον **μία από τις δειγματοληψίες που λαμβάνουμε δεν συνάδει με τις υπόλοιπες ή όταν στις θέσεις που περιμένουμε start bit και stop bit δεν παίρνουμε τα ανάλογα σήματα**. Αν εμφανιστεί σφάλμα και το Rx_ferror πάρει την τιμή 1 θα διατηρήσει αυτή την τιμή για όλη τη διάρκεια λήψης των υπόλοιπων bit. Το σήμα **Rx_FERROR** είναι το σήμα σφάλματος που εμφανίζεται στην έξοδο και λαμβάνεται υπόψη για την εγκυρότητα των δεδομένων, παίρνει την τιμή του **Rx_ferror** στο τέλος του λαμβανόμενου 11-bit σήματος και διατηρεί αυτή την τιμή μέχρι το τέλος του επόμενου 11-bit σήματος. Με αυτόν τον τρόπο έχουμε τα δεδομένα του πρώτου σήματος ενεργά στην έξοδο γνωρίζοντας την εγκυρότητα του για αρκετό χρόνο ώστε να τα επεξεργαστούμε.

➔ Αντίστοιχη διαδικασία χρησιμοποιούμε και με το finalReceivedBitSequence και **receivedBitSequence**, δηλαδή κατά την διάρκεια λήψης των δεδομένων χρησιμοποιούμε το receivedBitSequence, στο οποίο **αποθηκεύουμε ένα ένα τα bit** μέχρι να ολοκληρωθεί η μεταφορά τους και έπειτα τα **μεταφέρουμε στο finalReceivedBitSequence** για να αρχίσει η επεξεργασία τους και να πάρουμε το επιθυμητό 8-bit σήμα στην έξοδο. Με αυτόν τον τρόπο, όσο χρόνο στέλνουμε ένα δεύτερο 11-bit σήμα από τον Transmmitter, στην έξοδο του receiver θα έχουμε διαθέσιμο το

πρώτο 8-bit σήμα σε συνδυασμό με το σωστό σήμα Rx_VALID.

- Το *inputBitSequence* module, το οποίο χρησιμοποιείται για να **χωρίσει** το **8-bit signal** της πληροφορίας και το **1-bit parity bit**, που θα χρησιμοποιηθούν στην συνέχεια.

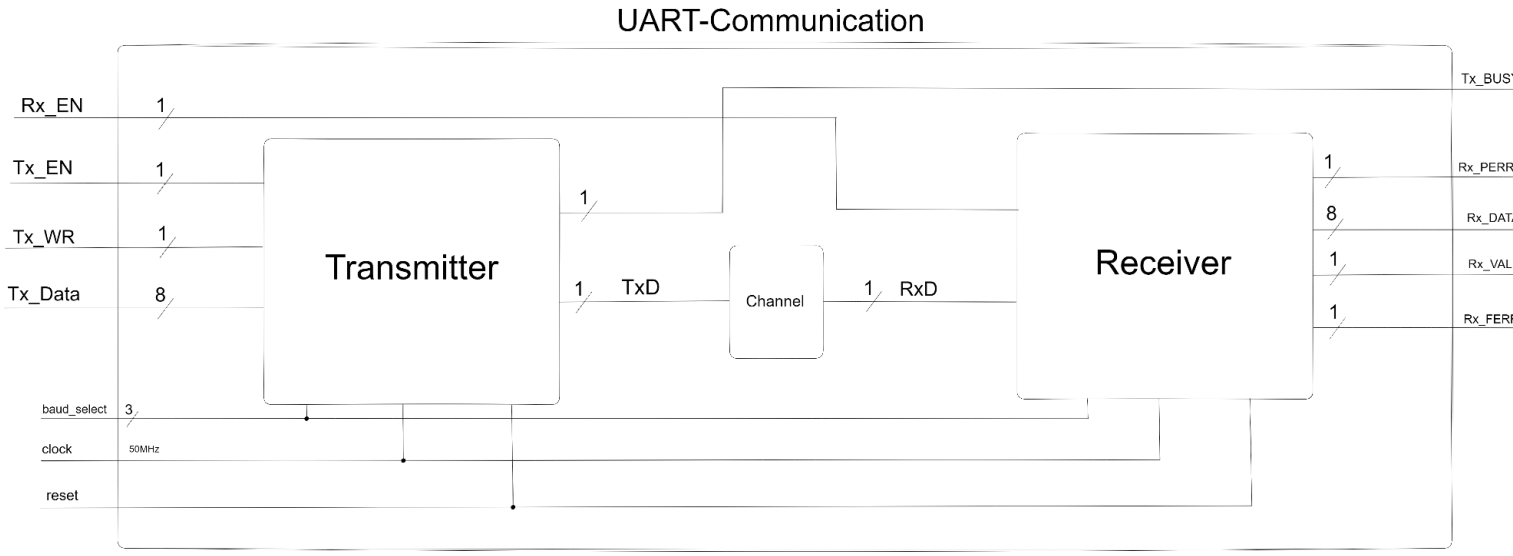
- Το *parityBitcalculator* module, το οποίο λειτουργεί όμοια με το αντίστοιχο module του transmitter **υπολογίζοντας** το **parity bit** για το εισερχόμενο σήμα των 8 bit.

- Το *PErrorDetection* module, το οποίο είναι υπεύθυνο για τον υπολογισμό του **Px_PERROR** σήματος. **Συγκρίνει** τα δύο σήματα εισόδου του. Το πρώτο σήμα είναι το parity bit (parityBit) που υπολογίζεται από το *parityBitcalculator* module ενώ το δεύτερο είναι το parity bit (inputParityBit) που διαβάζεται από το προτελευταίο bit του 11-bit σήματος (finalReceivedBitSequence) που λαμβάνεται

- Το *dataOutput* module, το οποίο αποτελεί το τελευταίο στάδιο επεξεργασίας, καθώς **στέλνει το επιθυμητό 8-bit signal στην έξοδο του receiver** και **ελέγχει** όλες τις πληροφορίες από τα διάφορα **errors**, ώστε να βγάλει το σήμα **Rx_VALID** που μας ενημερώνει σχετικά με την εγκυρότητα των δεδομένων.

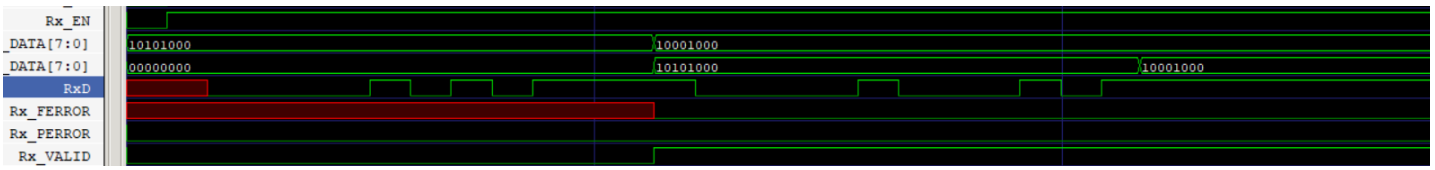
Ένωση Transmitter - Receiver

1. Σχηματικό διάγραμμα του κυκλώματος.



2. Κυματομορφές

Παρακάτω φαίνονται οι κυματομορφές που απεικονίζουν την **μεταφορά** των **δεδομένων** από τον **Αποστολέα** στον **Δέκτη**. Επίσης, παρατηρούμε ότι ο **συγχρονισμός** των δύο πλευρών λειτουργεί σωστά χωρίς να παράγει σήματα λάθους με το σήμα Rx_VALID να έχει τιμή 1.



3. Περιγραφή

Για την υλοποίηση του Συστήματος UART Αποστολέα Δέκτη για Σειριακή Μεταφορά Δεδομένων χρησιμοποιήσαμε τα παρακάτω module:

- Όλα τα module που χρησιμοποιήθηκαν για τον Transmitter καθώς και το module *uart_transmitter* που είναι το αντίστοιχο του **design** που χρησιμοποιήθηκε στον **Transmitter**.

- Όλα τα module που χρησιμοποιήθηκαν για τον Receiver καθώς και το module *uart_receiver* που είναι το αντίστοιχο του **design** που χρησιμοποιήθηκε στον **Receiver**.

- Το *channel* module, το οποίο χρησιμοποιείται για να **μοντελοποιήσει** το κανάλι της επικοινωνίας. Στην δική μας περίπτωση δεν έχει κάποια λειτουργικότητα, αλλά θα μπορούσαμε να το χρησιμοποιήσουμε για να προσθέσουμε θόρυβο ή καθυστερήσεις.

4. Προβλήματα

- Αντιμετωπίσαμε **πρόβλημα** στο **συγχρονισμό** του Αποστολέα και Δέκτη, καθώς οι περίοδοι πρακτικά **δεν είναι ανάλογες**, όπως έχει αναφερθεί και παραπάνω, με αποτέλεσμα να ενεργοποιούνται τα σήματα σφαλμάτων και να μην έχουμε έγκυρα δεδομένα στην έξοδο. Για παράδειγμα, όταν χρησιμοποιούμε ως baud_select το 111, που είναι η πιο γρήγορη μεταφορά δεδομένων στο uart πρωτόκολλο επικοινωνίας, η περίοδος του sample_ENABLE του Receiver δεν είναι ακριβώς 16 φορές μεγαλύτερη από την αντίστοιχο περίοδο του sample_ENABLE του Transmitter, αλλά είναι περισσότερο από 16 φορές **μεγαλύτερη κατά 40 ns**, που αποτελούν δύο περιόδους του clock. Εξαιτίας αυτής της διαφοράς δημιουργείται **σφάλμα μετά από 13 bit.**

- Σκεφτήκαμε τρεις τρόπους επίλυσης του παραπάνω προβλήματος.
    1. Αρχικά, προσπαθήσαμε να κάνουμε **δειγματοληψία** στο **κέντρο** κάθε καινούργιου bit που εισέρχεται στον transmitter αλλά τα αποτελέσματα ήταν **αναξιόπιστα**, επειδή το σύστημα είναι πολύ επιρρεπές σε θόρυβο.
    2. Για να λύσουμε το πρόβλημα που δημιουργήθηκε παραπάνω αποφασίσαμε να κάνουμε **δειγματοληψία 16 φορές για κάθε**

bit και να χρησιμοποιήσουμε την **πλειοψηφία** αυτών των bit ώστε να καταλήξουμε στο σωστό αποτέλεσμα. Με αυτόν τον τρόπο το σύστημα δεν ήταν πλέον ευαίσθητο στον θόρυβο, αλλά έπειτα από λίγο χρόνο **εμφανιζόταν σφάλμα** που αφορούσε την πλαισίωση των δεδομένων, αν και το bit που λαμβάναμε από την πλειοψηφία ήταν σωστό.

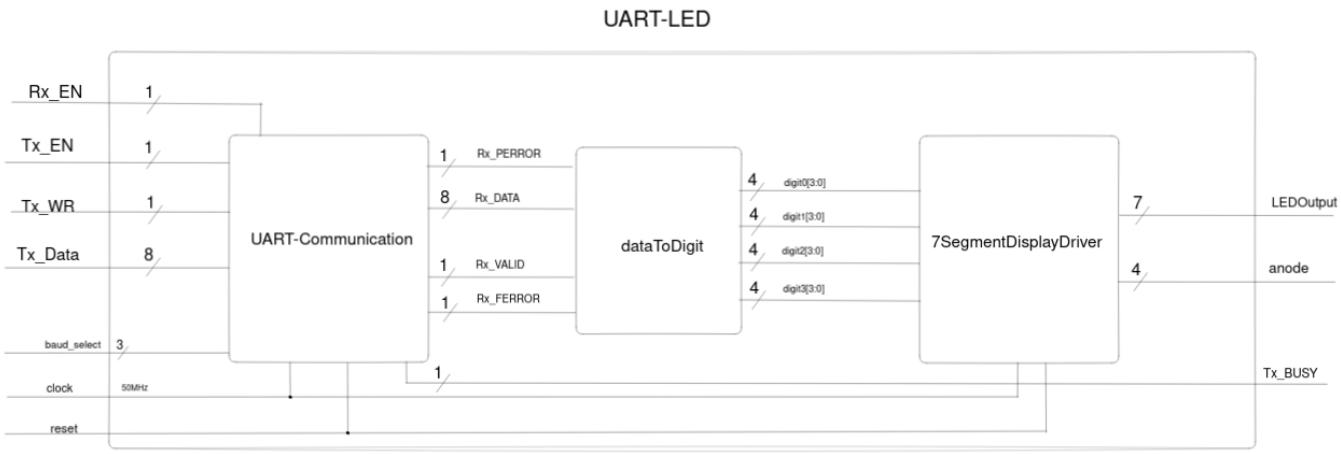
3. Τέλος, στην προσπάθεια μας να αντιμετωπίσουμε και αυτήν την ατέλεια στον τρόπο λήψης των δεδομένων καταλήξαμε στο συμπέρασμα ότι μπορούμε **να μην λαμβάνουμε υπόψη την πρώτη δειγματοληψία κάθε bit**, ώστε να αποφεύγουμε την δημιουργία του error για κάποιο χρονικό διάστημα (θα εμφανιστεί εν τέλη αλλά θα χρειαστεί διπλάσιο χρόνο) και παράλληλα θα έχουμε αρκετές δειγματοληψίες (15) ώστε να λειτουργήσει σωστά ο κανόνας της πλειοψηφίας και να μην επηρεάζεται το σύστημα από θόρυβο.

- Ένα δεύτερο πρόβλημα που προέκυψε ήταν η **δημιουργία καλών testbench**, καθώς πολλές φορές η **αρχικοποίηση** και οι **καθυστερήσεις** αποτελούσαν **πρόβλημα** στην ορθή λειτουργία του συστήματος. Αυτό το πρόβλημα κυρίως αντιμετωπίστηκε με χρήση της μεθόδου trial and error, αλλά και με την εμπειρία που αποκτήσαμε στην διάρκεια της ενασχόλησής μας στα πλαίσια αυτής της εργασίας. Τα testbench που έχουμε παραδώσει στους κώδικες verilog της εργασίας είναι φτιαγμένα έτσι ώστε το σύστημα να λειτουργεί σωστά σε κάθε περίπτωση και να **αποφεύγονται** τα **σφάλματα**. Αν γίνουν οι κατάλληλες αλλαγές, μπορούμε να δημιουργήσουμε **τεχνητά σφάλματα** και να δούμε ότι τα **σήματα ελέγχου λειτουργούν σωστά.**

Γ’ Μέρος

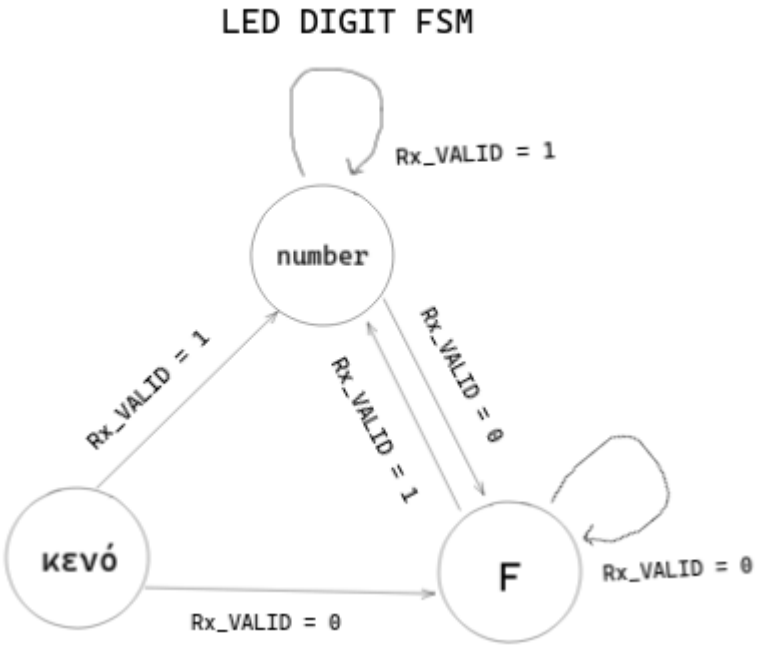
Σύστημα Δέκτη με προβολή μηνύματος σε οθόνη τεσσάρων LED 7-τμημάτων

1. Σχηματικό διάγραμμα του κυκλώματος.



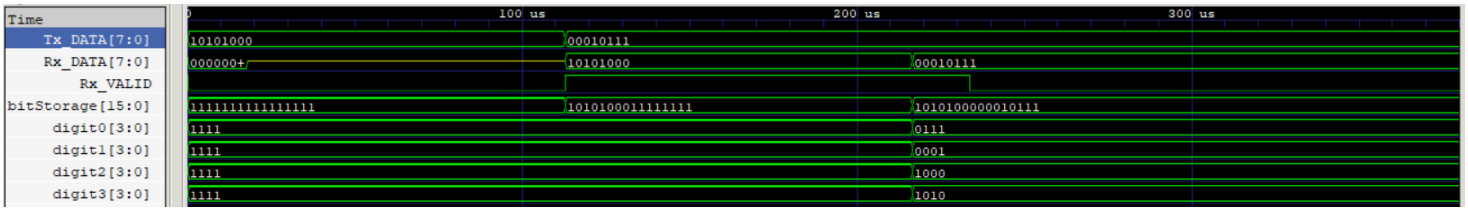
2. Σχηματικό των υλοποιημένων FSM

Σχηματικό διάγραμμα των **καταστάσεων των τιμών που φαίνονται στην κάθε οθόνη LED.** Το διάγραμμα απεικονίζει τις καταστάσεις για ένα από τα ψηφία που φαίνονται στις οθόνες, αλλά ισχύουν και για τα 4. Η αρχική κατάσταση έχει οριστεί να είναι το κενό ψηφίο με όλα τα LED φώτα σβηστά. Στη συνέχεια ελέγχεται το σήμα **Rx_VALID** και όταν είναι ίσο με **ένα** το εμφανίζεται στην οθόνη το αντίστοιχο ψηφίο (**number**) που έχει σταλεί από τον Transmitter. Στην κατάσταση number περιλαμβάνονται όλες οι δυνατές τιμές ενός ψηφίου (0,1,...,9,-,κενό) εκτός της F. Αν το Rx_VALID είναι **0** τότε εμφανίζεται το **F**.

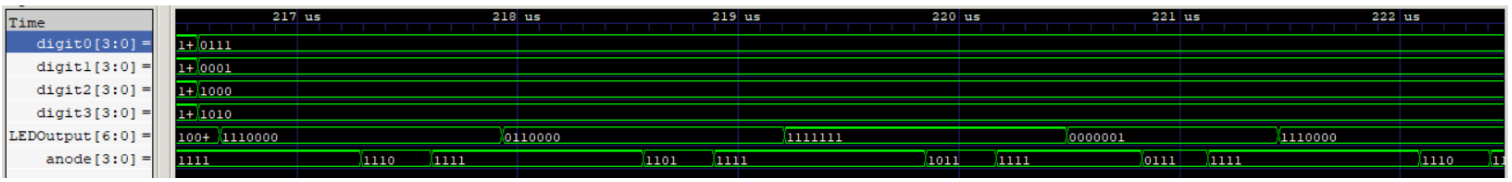


3. Κυματομορφές

Παρακάτω φαίνονται οι κυματομορφές για ψηφία εισόδου “-817”. Στην πρώτη κυματομορφή φαίνονται τα σήματα Tx_DATA, Rx_DATA, Rx_VALID, bitStorage και digit0->3. Η **τιμή** που εισάγουμε μέσω του testbench στο **Tx_DATA** βλέπουμε να **μεταφέρεται** αρχικά στο **Rx_DATA**. Στη συνέχεια 2 σήματα Tx_DATA των **8bit αποθηκεύονται** προσωρινά στο **bitStorage** και “**χωρίζονται**” στα **4 digit.**



Στη δεύτερη κυματομορφή απεικονίζεται η έξοδος της **οθόνης LED** με βάση τα ψηφία που προκύπτουν από την επικοινωνία Αποστολέα και Δέκτη. Παρατηρούμε ότι ο **συγχρονισμός** της εξόδου LED και της



αναμμένης οθόνης (βάση των τιμών του σήματος anode) είναι ο επιθυμητός.

4. Περιγραφή

Για την υλοποίηση του Συστήματος Δέκτη με προβολή του μηνύματος στην οθόνη τεσσάρων LED 7 τμημάτων χρησιμοποιήσαμε τα παρακάτω module:

- Όλα τα module που χρησιμοποιήθηκαν στο Μέρος Β' για την επικοινωνία Transmitter-receiver καθώς και το module *uart_communication* που είναι το αντίστοιχο του **design** που χρησιμοποιήθηκε στον **Μέρος Β'**

- Όλα τα module που χρησιμοποιήθηκαν στο Μέρος Α' για την υλοποίηση του αποκωδικοποιητή 7-τμημάτων τεσσάρων LED καθώς και το module *7SegmentDisplayDriver* που είναι το αντίστοιχο του **design** που χρησιμοποιήθηκε στον **Μέρος Α'**

- Το *dataToDigit* module, το οποίο δέχεται σαν **είσοδο** τα **8bit** σήματα που προκύπτουν από την επικοινωνία του Αποστολέα και του Δέκτη, καθώς και τα σήματα **valid** και **λάθους**. Από αυτό το module **προκύπτουν** τα **4 4-bit ψηφία** που δέχεται σαν είσοδο το *7SegmentDisplayDriver* module μέσα από το οποίο θα προκύψουν τα 4 ψηφία που θα **εμφανιστούν** στις 4 **οθόνες**.
    - ➔ Κάθε φορά που αλλάζει η έξοδος του *uart_communication,* δηλαδή έρχεται **καινούργιο 8bit** σήμα τότε, ελέγχοντας αρχικά την τιμή του Rx_VALID για την εγκυρότητα των δεδομένων, το σήμα περνάει σε ένα στον πίνακα **bitStorage**.
    - ➔ Ο πίνακας αυτός είναι έχει περισσότερο βοηθητική παρά πρακτική σημασία, καθώς ο ρόλος του είναι να **δέχεται** τα **δύο** 8 bit σήματα που προκύπτουν από την επικοινωνία και να δημιουργεί ένα **16 bit** σήμα **ενώνοντάς** τα. Ουσιαστικά το 1ο σήμα "μπαίνει" στις πρώτες 8 από αριστερά θέσεις του **bitStorage** ενώ το 2ο στις υπόλοιπες 8 προς τα δεξιά.

➔ Στη συνέχεια το **bitStorage χωρίζεται** στα **4** σήματα **digit0->3** με βάση την σειρά προβολής τους στις οθόνες (δηλαδή 3->2->1->0).

➔ Αν προκύψει σήμα **λάθους** από το uart_communication, τότε το bitStorage παίρνει και στις 16 θέσεις του την **τιμή 1** (για να προκύψει το **F** πρέπει η τιμή του 4 bit σήματος στην είσοδο να είναι σύμφωνη με την αντιστοίχιση που έχουμε κάνει στο module LEDDecoder) έτσι ώστε να προκύψει για όλα τα ψηφία στις οθόνες η ένδειξη F.

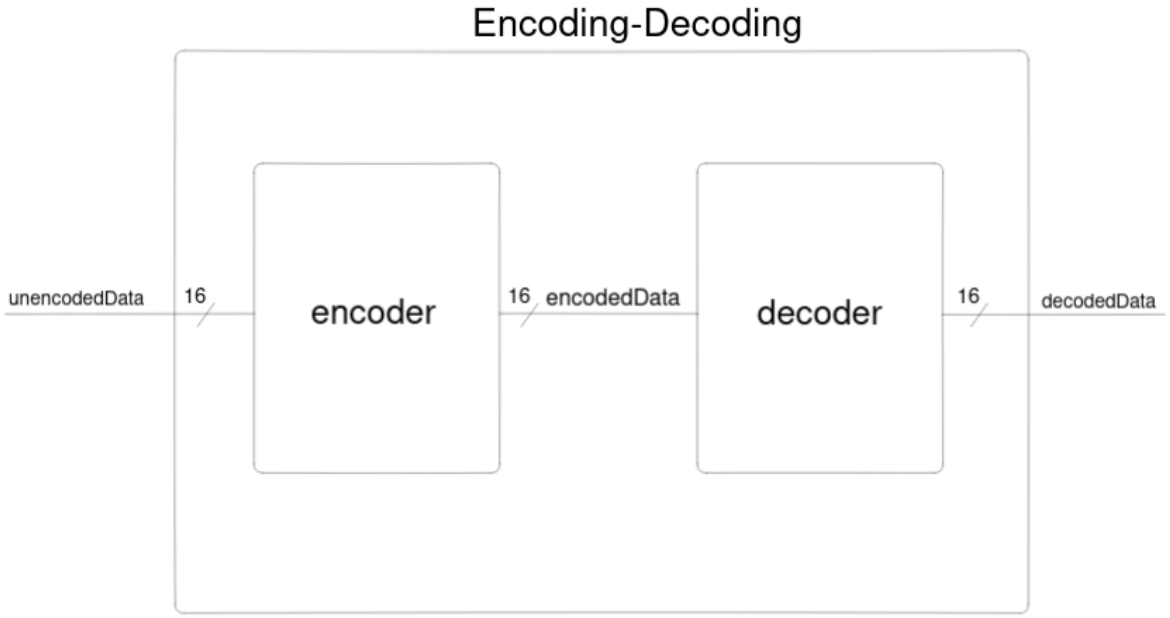
5. Προβλήματα

- Δυσκολευτήκαμε να δημιουργήσουμε ένα σωστό **16 bit register**, το οποίο προέρχεται από τα δύο 8 bit που λαμβάνουμε μέσω της UART επικοινωνίας εξαιτίας ενός θέματος **συγχρονισμού** με το **Rx_VALID**. Καταφέραμε να λύσουμε αυτό το πρόβλημα εισάγοντας ένα άλλο register delay, το οποίο πρακτικά λειτουργεί ως μεσάζοντας.

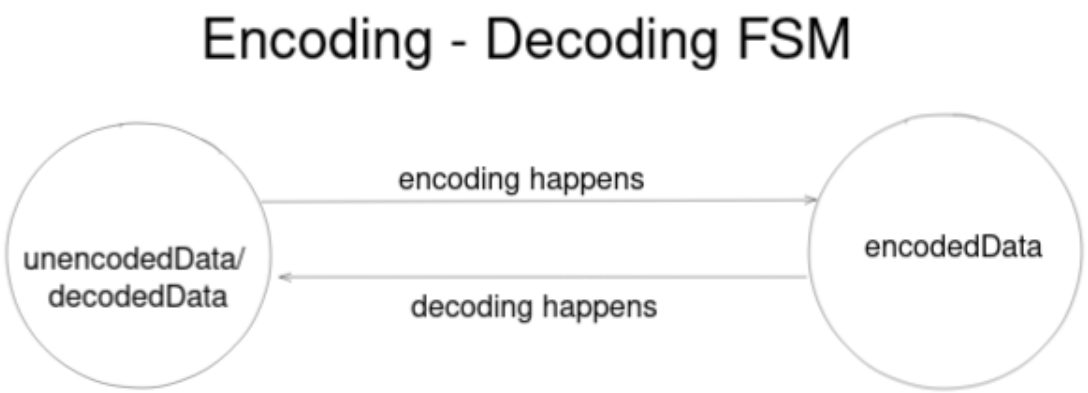
Δ' Μέρος

Απλή Υλοποίηση Κωδικοποίησης - Αποκωδικοποίησης

1. Σχηματικό διάγραμμα του κυκλώματος.

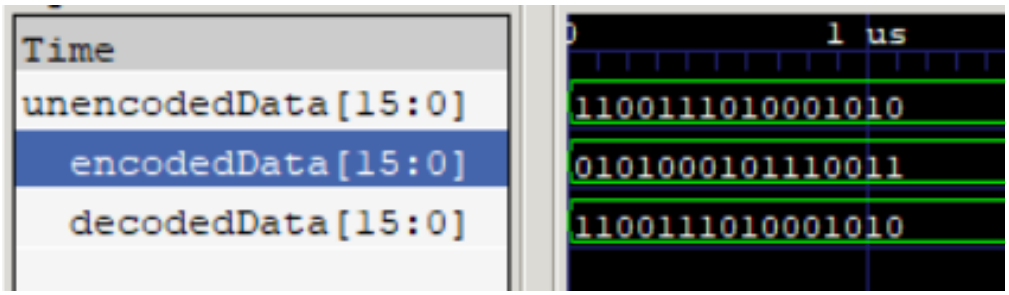


2. Σχηματικό των υλοποιημένων FSM



3. Κυματομορφές

Στην παρακάτω κυματομορφή φαίνονται 3 σήματα των 16 bit. Βλέπουμε ότι τα 2 σήματα της ακολουθίας bit πριν (**unencoded**) και μετά (**decoded**) την κωδικοποίηση και αποκωδικοποίηση αντίστοιχα, είναι όμοια. Το **κωδικοποιημένο** σήμα διαφέρει με βάση την αποκωδικοποίηση που έχουμε επιλέξει.



4. Περιγραφή

Για την υλοποίηση του Κωδικοποιητή και Αποκωδικοποιητή χρησιμοποιήσαμε τα εξής module:

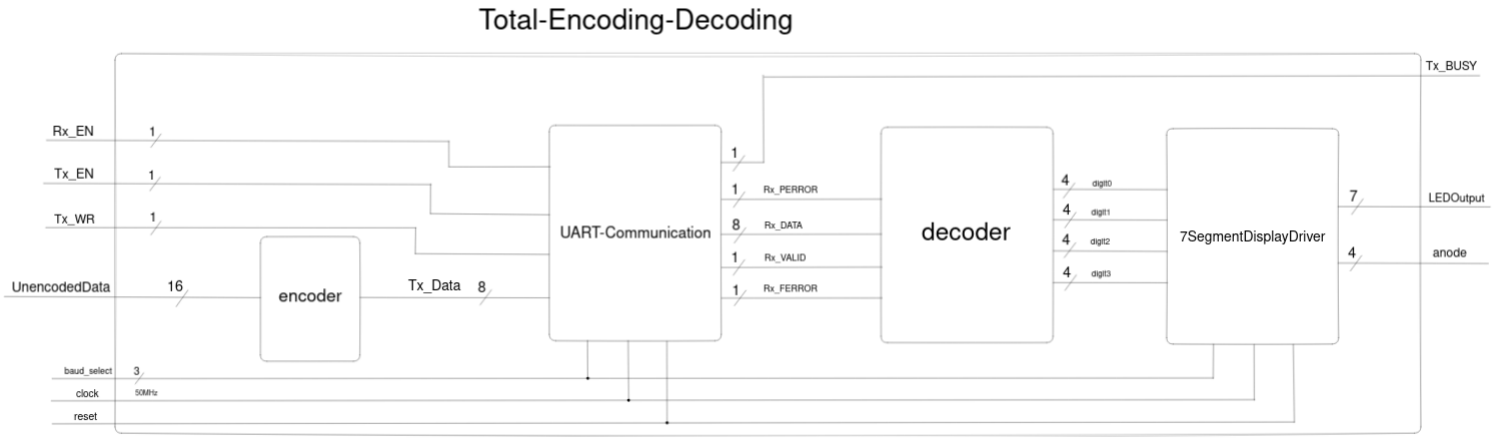
- Το *encoder* module το οποίο είναι υπεύθυνο για την **κωδικοποίηση** μίας ακολουθίας bit. Σαν είσοδο παίρνει ένα σήμα των 16 bit και βγάζει στην έξοδο το αντίστοιχο **κωδικοποιημένο** σήμα 16 bit. Επιλέξαμε ως τεχνική κωδικοποίησης την **αντιστροφή** της ακολουθίας **των bit.** Δηλαδή σε αυτό το module γίνεται μία ανταλλαγή των bit του εισαγόμενου σήματος (πρώτο με τελευταίο, δεύτερο με προτελευταίο κ.ο.κ.). Παρακάτω παραθέτουμε ένα **παράδειγμα** της κωδικοποίησης:
  σήμα χωρίς κωδικοποίηση : 0 1 0 0 1 1 0 0 0 1 1 0 1 1 1
  σήμα μετά την κωδικοποίηση : 1 1 1 0 1 1 0 0 0 1 1 0 0 1 0
  Ουσιαστικά έχουμε επιλέξει για την κωδικοποίηση και αποκωδικοποίηση n = 16 bit εισόδου που αντιστοιχίζονται μονοσήμαντα σε m = n = 16 bit εξόδου.

- Το *decoder* module το οποίο είναι υπεύθυνο για την **αποκωδικοποίηση** μίας ακολουθίας bit. Σαν είσοδο παίρνει ένα σήμα των 16 bit και βγάζει στην έξοδο το αντίστοιχο

αποκωδικοποιημένο σήμα 16 bit. Συμβαίνει η ανάποδη διαδικασία από αυτή που περιγράφηκε στον encoder.

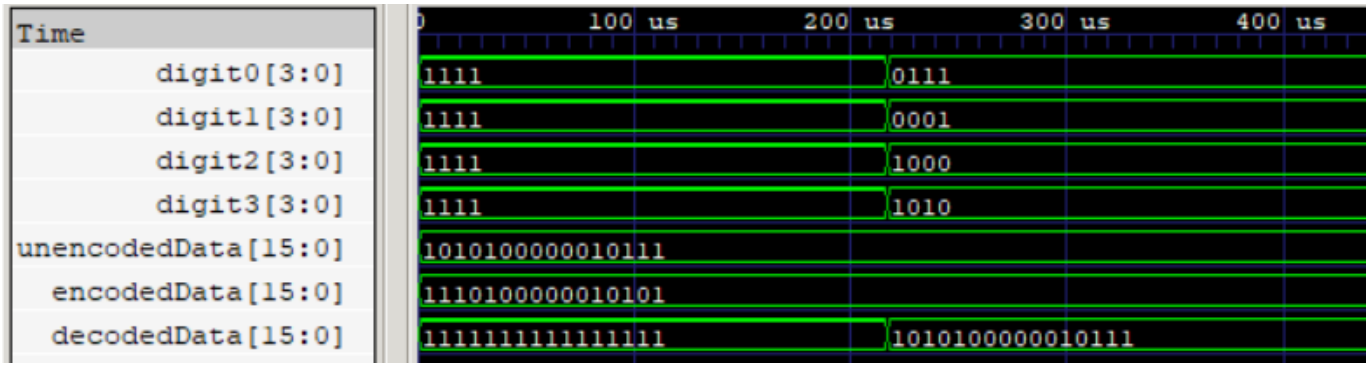
Υλοποίηση Κωδικοποίησης - Αποκωδικοποίησης σε συνδυασμό με το Γ' Μέρος

1. Σχηματικό διάγραμμα του κυκλώματος.

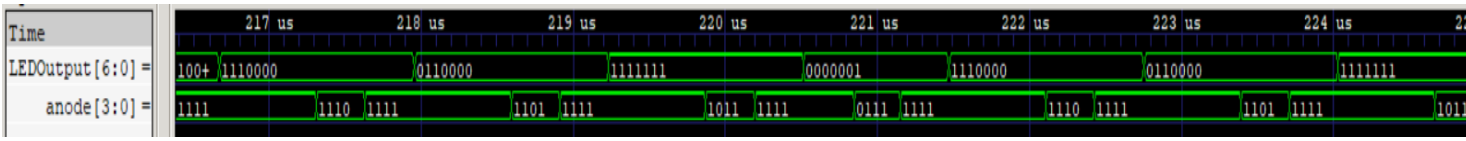


2. Κυματομορφές

Στην παρακάτω κυματομορφή φαίνονται τα σήματα που επιβεβαιώνουν ότι η **κωδικοποίηση-αποκωδικοποίηση** των επιθυτών σημάτων πέτυχε. Βλέπουμε ότι τα 2 σήματα της ακολουθίας bit πριν (**unencoded**) και μετά (**decoded**) την κωδικοποίηση και αποκωδικοποίηση αντίστοιχα, είναι **όμοια**. Το **κωδικοποιημένο** σήμα **διαφέρει** με βάση την κωδικοποίηση που έχουμε επιλέξει. Τέλος, τα **τέσσερα** σήματα των 4 bit εμφανίζονται ορθά μέσα στο σύστημα ώστε να αντιστοιχηθούν τελικά στα ανάλογα **7 bit σήματα εξόδου.**



Στη δεύτερη κυματομορφή απεικονίζεται η έξοδος της **οθόνης LED** με βάση τα ψηφία που προκύπτουν από την επικοινωνία Αποστολέα και Δέκτη. Παρατηρούμε ότι ο **συγχρονισμός** της εξόδου LED και της αναμμένης οθόνης (βάση των τιμών του σήματος anode) είναι ο επιθυμητός.



3. Περιγραφή

Για την υλοποίηση του κωδικοποιητή-αποκωδικοποιητή σε συνδυασμό με το σύστημα Αποστολέα-Δέκτη που υλοποιήσαμε στο Μέρος Γ' χρησιμοποιήσαμε τα παρακάτω module:

- Όλα τα module που χρησιμοποιήθηκαν στο Μέρος Γ' για την υλοποίηση του Συστήματος Δέκτη με προβολή του μηνύματος στην οθόνη τεσσάρων LED 7 τμημάτων.

- **Τροποποιήσαμε** και μετονομάσαμε το module *dataToDigit* σε *decoder*. Το module αυτό ουσιαστικά πραγματοποιεί την ίδια λειτουργία με το *dataToDigit* module του Γ' Μέρους, δηλαδή

υπολογίζει τα **4 σήματα** που θα χρησιμοποιηθούν στα **ψηφία** της οθόνης LED. Η προσθήκη που έγινε σχετίζεται με τη διαδικασία της **αποκωδικοποίησης**. Συγκεκριμένα, το 16 bit σήμα bitStorage που αποθηκεύεται στον decoder υφίσταται μία αποκωδικοποίηση πριν περάσουν οι κατάλληλες τιμές στα σήματα digit0->3.

- Το *encoder* module στο οποίο γίνεται η **κωδικοποίηση**. Συγκεκριμένα, εισάγεται ένα 16 bit σήμα μέσω του testbench το οποίο υφίσταται κωδικοποίηση (με την αντιστροφή των bit όπως αναφέρθηκε παραπάνω) και προκύπτει ένα κωδικοποιημένο 16bit σήμα στην έξοδο.

4. Προβλήματα

Κατά την **κωδικοποίηση** επιλέξαμε να δουλεύουμε με σήματα των **16 bit** ενώ ο **Αποστολέας** δέχεται σήματα των **8 bit** προς μετάδοση. Για αυτό το λόγο χρειάστηκε να βρεθεί ένας τρόπος να "**χωρίζεται**" το σήμα των 16 bit σε 2 σήματα των 8 bit τα οποία θα μεταφέρονται με κατάλληλο **συγχρονισμό** στο σήμα εισόδου (Tx_DATA) του Αποστολέα. Χρησιμοποιήσαμε, λοιπόν, το σήμα **Tx_WR** για αυτό το σκοπό. Συγκεκριμένα, το σήμα αυτό μπαίνει σαν **είσοδος** στο **encoder** module. Εφόσον η μεταφορά των δεδομένων Tx_DATA γίνεται με την ενεργοποίηση του σήματος Tx_WR (που διαρκεί 1 clock). Οπότε μας ενδιαφέρει μόνο για εκείνη την χρονική περίοδο (1 clock) να είναι έχει το σήμα Tx_DATA τα σωστά bit.
Οπότε στο encoder module σε κάθε **negative edge** του Tx_WR (αφού δηλαδή έχουν περάσει τα δεδομένα που θέλουμε από το Tx_DATA προς επεξεργασία) , στο **Tx_DATA μεταφέρονται** τα **επόμενα** 8 bit από το 16 bit σήμα **encodedData**. Αυτή η τιμή **διατηρείται μέχρι** το **επόμενο positive edge** του Tx_WR στο οπίο και θα χρησιμοποιηθεί, ενώ αλλάζει με το που γίνει το Tx_WR ίσο με 0.