



Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Ψηφιακά Συστήματα HW σε Χαμηλά Επίπεδα Λογικής I

7^ο Εξάμηνο

Στεφανίδης Ιωάννης 9587

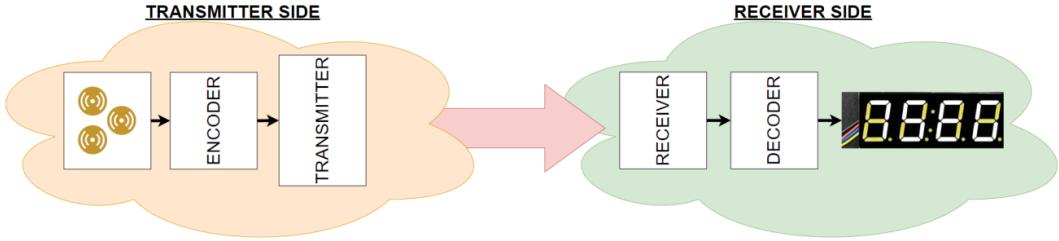
30 Ιανουαρίου 2023

Περιεχόμενα

1	Εισαγωγή	2
2	7-Segment Display Driver	2
2.1	LedDecoder module	2
2.2	LedDriver4 module	3
3	UART	5
3.1	BaudController module	5
3.2	Transmitter module	7
3.3	Receiver module	9
4	Τελική σύνδεση	11

1 Εισαγωγή

Στην εργασία αυτή έπρεπε να υλοποιήσουμε σε Verilog, ένα ψηφιακό σύστημα αποστολέα-δέκτη, όπως παρουσιάζεται στο Σχήμα 1.1. Η επικοινωνία θα πραγματοποιείται με χρήση του πρωτοκόλλου UART. Ο αποστολέας θα λαμβάνει από το περιβάλλον του τις ενδείξεις κάποιον αισθητήρων και ο δέκτης θα εμφανίζει τις ενδείξεις που λαμβάνει σε μία οθόνη τεσσάρων LED 7-τμημάτων.



Σχήμα 1.1: Προεπισκόπηση της συνολικής υλοποίησης.

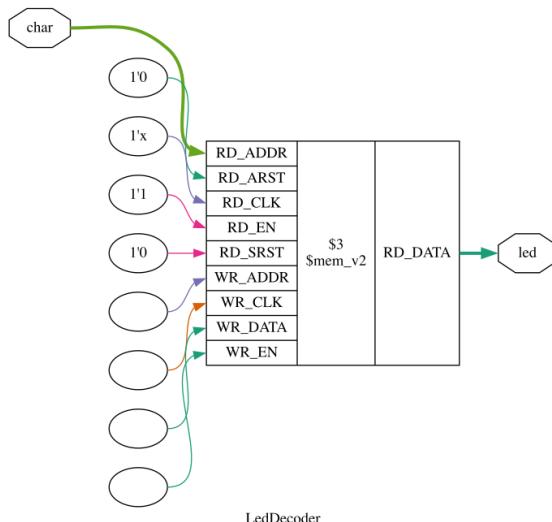
2 7-Segment Display Driver

2.1 LedDecoder module

Στόχος αυτού του module ήταν να δέχεται ως είσοδο ένα σύμβολο 4 bits και να βγάζει ως έξοδο 6 bits με τα οποία θα ελέγχεται μια οθόνη 7 τμημάτων (με το 7^o bit *DP* να είναι πάντα 1).

Υλοποίηση

Η υλοποίηση ήταν πολύ απλή με την χρήση ενός always block για την δημιουργία μιας ασύγχρονης read-only μνήμης. Αυτό φαίνεται και στο Σχήμα 2.1 που έκανε σύνθεση το εργαλείο yosys.



Σχήμα 2.1: Σύνθεση του LedDecoder με yosys.

Επαλήθευση

Για την επαλήθευση της σωστής λειτουργίας δημιουργήθηκε ένα testbench που δίνονται οι τιμές από 0 έως 13 ως char και λαμβάνουμε τις αναμενόμενες τιμές για τα 6 bits των led.



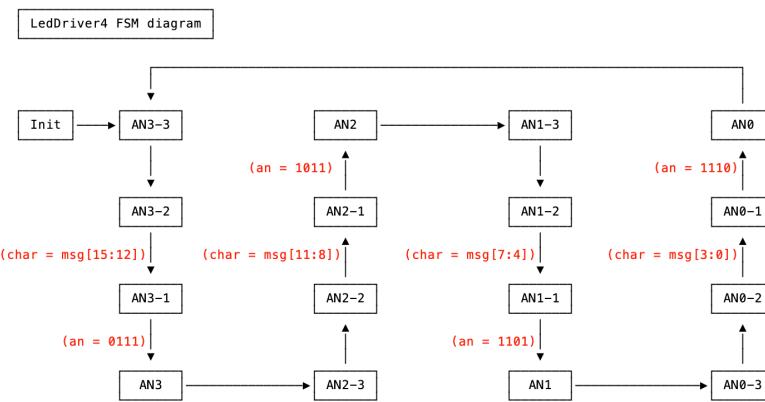
Σχήμα 2.2: LedDecoder testbench

2.2 LedDriver4 module

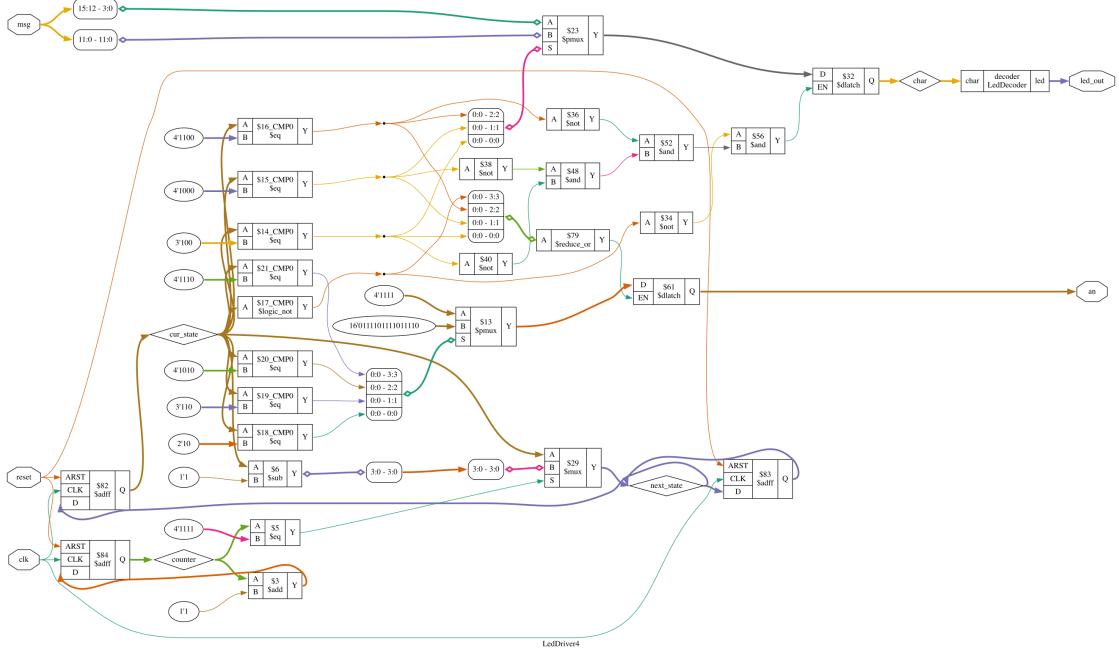
Στόχος αυτού το module ήταν να οδηγεί 4 οθόνες 7 τμημάτων με την χρήση ενός *LedDecoder*. Δηλαδή θα πρέπει να δέχεται ένα μήνυμα τεσσάρων συμβόλων άρα 16 bits και με τους κατάλληλους χρονισμούς να δείχνει τα πρώτα 4 bit στην πρώτη οθόνη, τα επόμενα 4 στην δεύτερη κτλ.. Έχοντας λοιπόν ως εξόδους 4 bits για τις ανόδους της κάθε οθόνης και 6 bits τα οποία θα είναι κοινά και θα οδηγούν τα led των οθονών, μπορούμε να φορτώνουμε το σύμβολο που θέλουμε να εμφανιστεί και μετά να οδηγούμε στην άνοδο της οθόνης που θέλουμε να ανάψουμε στο 0. Κάνοντας την παραπάνω διαδικασία κυκλικά με μεγάλη συχνότητα θα φαίνεται ότι όλες οι οθόνες είναι αναμμένες.

Υλοποίηση

Η υλοποίηση έγινε με την χρήση ενός FSM με 16 καταστάσεις που φαίνεται στο Σχήμα 2.3, πηγαίνοντας στην επόμενη κατάσταση κάθε 16 κύκλους τους 50MHz ρολογιού. Στο Σχήμα 2.4 φαίνεται η σύνθεση του εργαλείου yosys για αυτό το module, το οποίο εμπεριέχει και το *LedDecoder* module.



Σχήμα 2.3: FSM του LedDriver4.

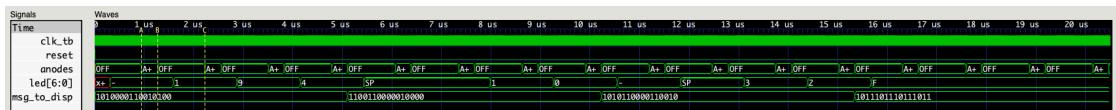


Σχήμα 2.4: Σύνθεση του LedDriver4 με yosys.

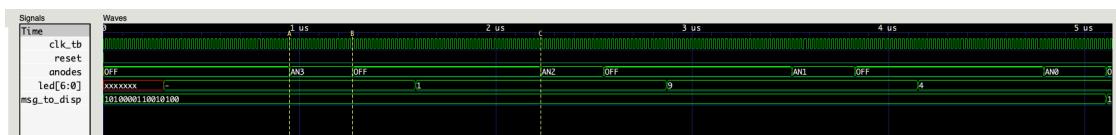
Επαλήθευση

Για την επαλήθευση της σωστής λειτουργίας δημιουργήθηκε ένα testbench που δίνονται ως είσοδοι τα μηνύματα "-194", "10", "-32" και "FFFF". Στο Σχήμα 2.5 φαίνεται πως τα μηνύματα μεταφράζονται σωστά ανά 4 bits και πως κάθε άνοδος άγει για κάποιο χρονικό διάστημα.

Μεγεθύνοντας στην διάρκεια μεταδόσεις ενός μηνύματος (Σχήμα 2.6) μπορούμε να μετρήσουμε την διάρκεια που άγει η κάθε άνοδος, δηλαδή την διάρκεια από το marker A έως B. Η διάρκεια αυτή είναι 320ns όπως ήταν αναμενόμενο αφού θέλαμε να άγει για 16 κύκλους του ρολογιού $16 \times 20 = 320\text{ns}$. Επίσης η διάρκεια που δεν άγει καμία άνοδος είναι σωστά $3 \times 320 = 960\text{ns}$ (από B έως C marker).



Σχήμα 2.5: LedDriver4 testbench.

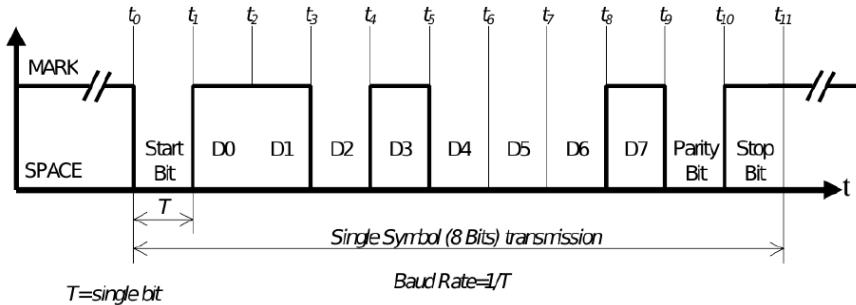


Σχήμα 2.6: LedDriver4 testbench μεγέθυνση.

3 UART

Ο στόχος για την εργασία είναι η υλοποίηση ενός συστήματος σειριακής επικοινωνίας, το οποίο θα χρησιμοποιεί το πρωτόκολλο UART (Universal Asynchronous Receiver Transmitter - Γενικού Ασύγχρονου Δέκτη Αποστολέα). Το σύστημα θα αποτελείται από έναν UART Αποστολέα και έναν UART Δέκτη, οι οποίοι μεταφέρουν δεδομένα στη μια κατεύθυνση, από τον Αποστολέα στον Δέκτη, μέσω μιας σειριακής σύνδεσης ενός σήματος.

Το UART που θα υλοποιηθεί, θα χρησιμοποιηθεί για τη σειριακή μεταφορά τουλάχιστον μιας αλληλουχίας τεσσάρων διαφορετικών συμβόλων συνήθως των 8-bit, από τον Αποστολέα στον Δέκτη. Για τα πλαίσια της εργασίας οι ενδείξεις θα αποστέλλονται από τον Αποστολέα στον Δέκτη σε 2 πακέτα των 8-bit. Πιο συγκεκριμένα αν για παράδειγμα η ένδειξη του αισθητήρα είναι '-888' τότε τα δύο πακέτα των 8-bit θα είναι '-8' και '88'. Ανάλογα με την κωδικοποίηση που κάνατε τα δύο πακέτα θα πρέπει να ερμηνευτούν σε συμβολοσειρές από 0 και 1. Π.χ. αν 8 -> 1000 και '-' -> 1010 τότε το πρώτο πακέτο των 8-bit θα είναι το 10101000 ('-8') και το δεύτερο το 10001000 ('88').



Σχήμα 3.1: Χρονοδιάγραμμα επικοινωνίας ενός συμβόλου.

3.1 BaudController module

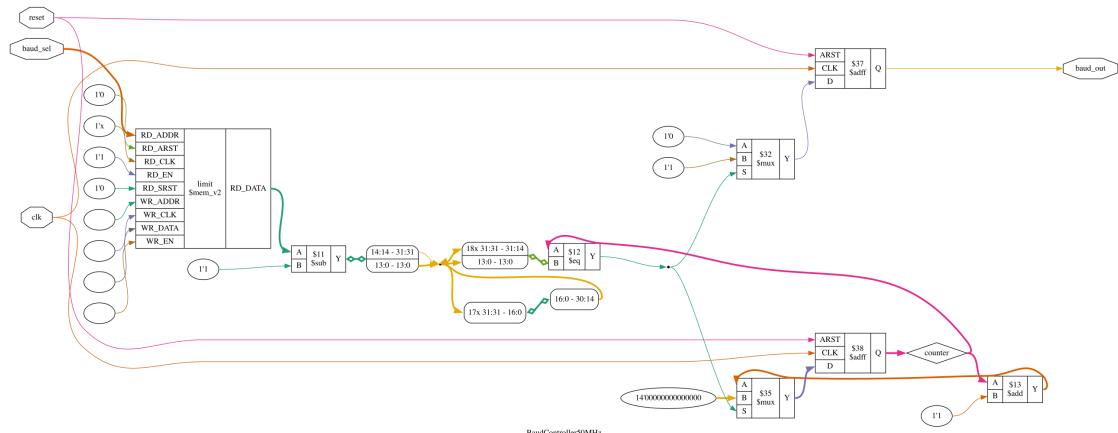
Στόχος αυτού του module είναι παροχή στον αποστολέα και δέκτη ενός sampling signal για να αποστολή και την λήψη bit αντίστοιχα ανάλογα με το επιλεγμένο Baud Rate. Για παράδειγμα επιλέγοντας Baud Rate 115200 bits/sec θα έχουμε $T = 1/(16 \times BR) = 543ns$, δηλαδή θα πρέπει το sampling signal να γίνεται '1' κάθε 543ns. Επειδή όμως το σύστημα μας τρέχει με ρολόι 50MHz ($T = 20ns$) μπορούμε να έχουμε $T = 540ns$ άρα το Baud Rate $BR = 1/(16 \times T) \simeq 115740.74$ bits/sec καταλήγοντας σε ένα error 0.47%. Οπότε σκοπός είναι να φτιάξουμε τον baud rate controller το οποίο θα έχει το μικρότερο δυνατό error για λειτουργία με ρολόι 50MHz.

Πίνακας 1: Error για κάθε Baud Rate με ρολόι 50MHz.

Desired Baud Rate	Actual Baud Rate	Count limit	Error
300	299.99	10,417	-0.003
1,200	1200.08	2,604	0.006
4,800	4800.31	651	0.006
9,600	9585.89	326	-0.147
19,200	19171.78	163	-0.147
38,400	38580.25	81	0.469
57,600	57870.37	54	0.469
115,200	115740.74	27	0.469

Υλοποίηση

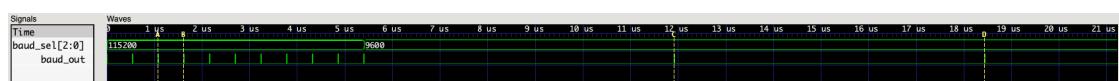
Η υλοποίηση ήταν πολύ απλή με την χρήση ενός initial block για την δημιουργία μιας ασύγχρονης read-only μνήμης στην οποία έχουμε αποθηκευμένα τα count limits για κάθε Baud Rate που υπολογίσαμε στον Πίνακα 1. Έπειτα σε ένα always block μετράμε τους χτύπους του ρολογιού μέχρι να φτάσουμε το όριο για το επιλεγμένο Baud Rate και να κάνουμε το sampling signal '1' για ένα κύκλο του ρολογιού.



Σχήμα 3.2: Σύνθεση του BaudController50MHz με yosys.

Επαλήθευση

Για την επαλήθευση της σωστής λειτουργίας δημιουργήθηκε ένα testbench που αρχικά για 10 sampling signals έχουμε επιλεγμένο Baud Rate 115200 bits/sec και μετά το αλλάζουμε σε 9600 bits/sec. Στο Σχήμα 3.3 μετρώντας τους χρόνους AB και CD βρίσκουμε αντίστοιχα τις σωστές τιμές $27 \times 20 = 540ns$ $326 \times 20 = 6520ns$.



Σχήμα 3.3: BaudController50MHz testbench.

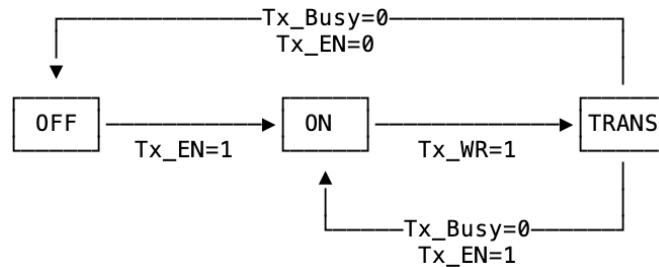
3.2 Transmitter module

Στόχος αυτού του module είναι να δέχεται ως είσοδο 8 bits και ανάλογα το επιλεγμένο Baud Rate να τα μεταδίδει μέσω της 1 bit εξόδου του. Θα πρέπει

- κάθε επικοινωνία να ξεκινάει με το start bit '0'.
- μετά από την διάδοση των data bits να μεταδίδεται το parity bit.
- μετά από το parity bit να μεταδίδεται ένα stop bit '1'.
- $T = 1/(16BaudRate)$
- Όταν δεν μεταδίδονται δεδομένα η έξοδος TxD να είναι στο '1'.

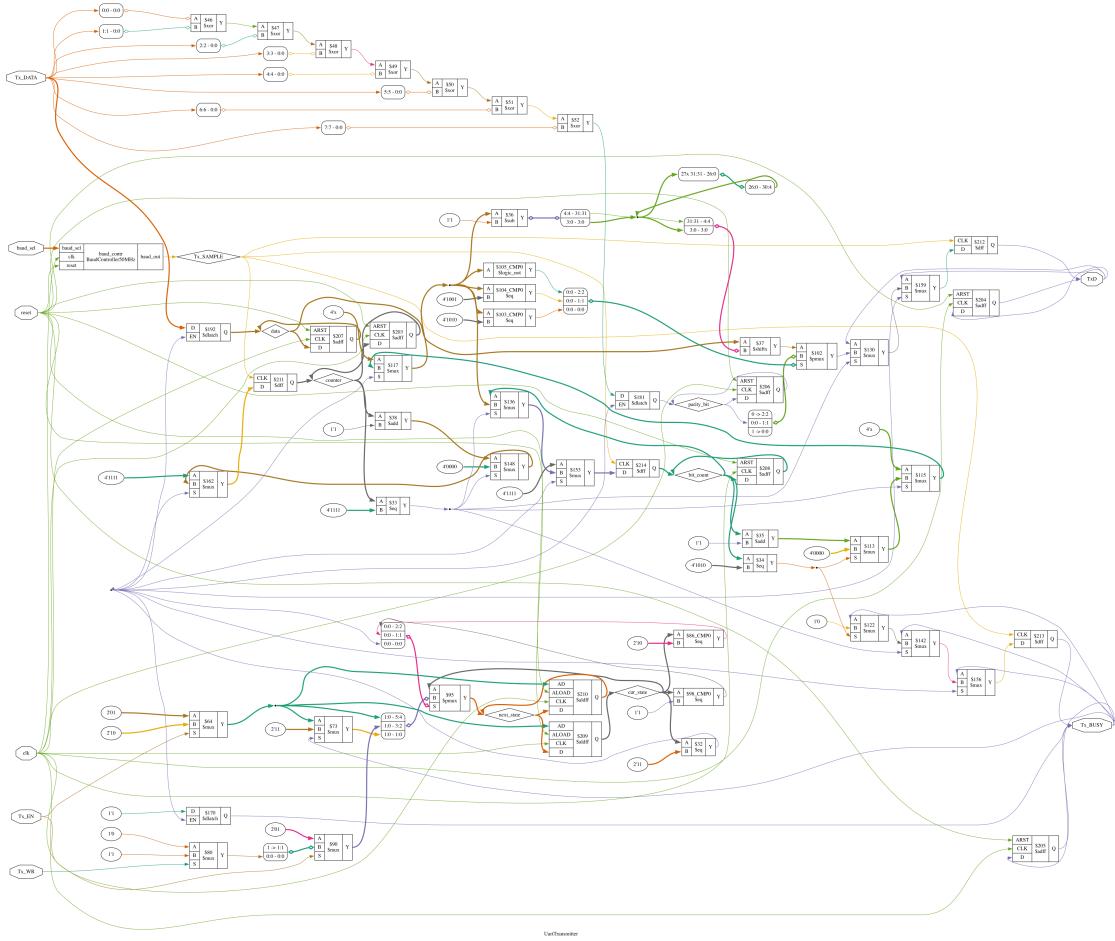
Υλοποίηση

Η υλοποίηση έγινε με την χρήση ενός FSM με 3 καταστάσεις (Σχήμα 3.4). Όταν στην κατάσταση ON το Tx_WR γίνει '1' τότε διαβάζονται τα δεδομένα από την είσοδο σε έναν εσωτερικό register, υπολογίζεται το parity bit και αλλάζει η κατάσταση σε TRANS (Transmitting) κάνοντας την έξοδο Tx_BUSY '1'.



Σχήμα 3.4: FSM UART Transmitter

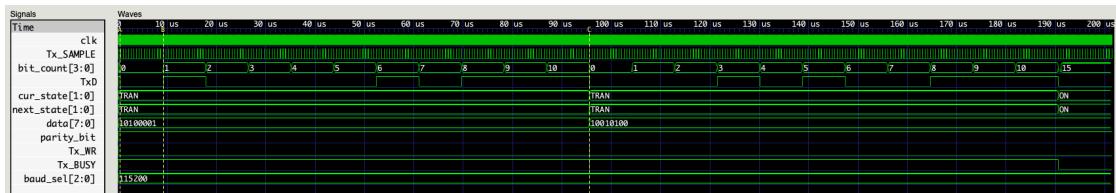
Όσο βρισκόμαστε στην κατάσταση TRANS ανά 16 δείγματα που δεχόμαστε από τον BaudController ανεβάζουμε έναν register *bit_count* κατά 1 έως ότου να φτάσουμε στο 11^o bit (1 start + 8 data + 1 parity + 1 stop). Έτσι ανάλογα την τιμή του *bit_count* αλλάζει η τιμή της εξόδου TxD.



Σχήμα 3.5: Σύνθεση του UART Transmitter με yosys.

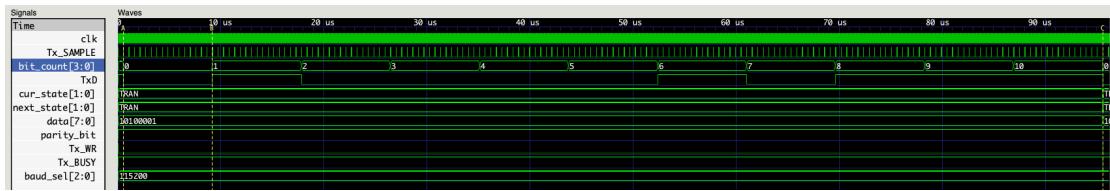
Επαλήθευση

Για την επαλήθευση της σωστής λειτουργίας δημιουργήθηκε ένα testbench που μεταδίδεται το μήνυμα "-194". Όπως φαίνεται στο Σχήμα 3.6 κι στα δύο πακέτα '-1' και '94' το parity bit υπολογίζεται σωστά '1'. Στέλνονται συνολικά 11 bits για κάθε πακέτο με το start κι stop bit να είναι σωστά και η έξοδος `Tx_BUSY` βρίσκεται στο '1'.



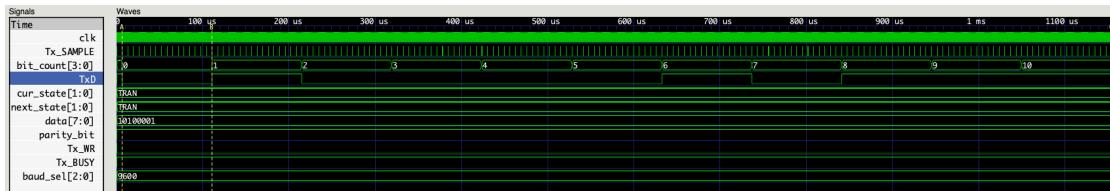
Σχήμα 3.6: UART transmitter testbench για μετάδοση του '-194' (BR=115200).

Εστιάζοντας μόνο στην μετάδοση του '-1' (Σχήμα 3.7) μπορούμε να μετρήσουμε την διάρκεια που το κάθε bit ήταν ενεργό στην έξοδο `TxD`, δηλαδή την διάρκεια από τον marker A έως B η οποία είναι $16 \times 540 = 8640\text{ns}$. Επίσης η συνολική διάρκεια μετάδοσης ενός πακέτου marker A έως C είναι $8640 \times 11 = 95040\text{ns}$. Και οι δύο παραπάνω τιμές είναι οι αναμενόμενες για το επιλεγμένο Baud Rate 115200 bits/sec.



Σχήμα 3.7: UART transmitter testbench για μετάδοση του '-1' (BR=115200).

Μεταδίδοντας το ίδιο πακέτο '-1' για Baud Rate 9600 bits/sec (Σχήμα 3.8), μετράμε πάλι σωστή περίοδο μετάδοσης ενός bit $6520 \times 16 = 104320\text{ns}$.



Σχήμα 3.8: UART transmitter testbench για μετάδοση του '-1' (BR=9600).

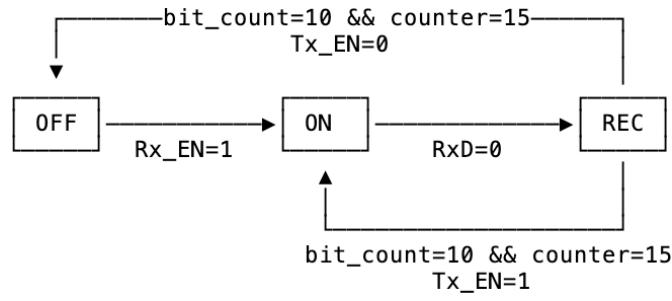
3.3 Receiver module

Στόχος αυτού του module είναι η λήψη των 8 data bits ένα ένα από τον αποστολέα μέσω του καλωδίου RxD. Ο δέκτης θα πρέπει:

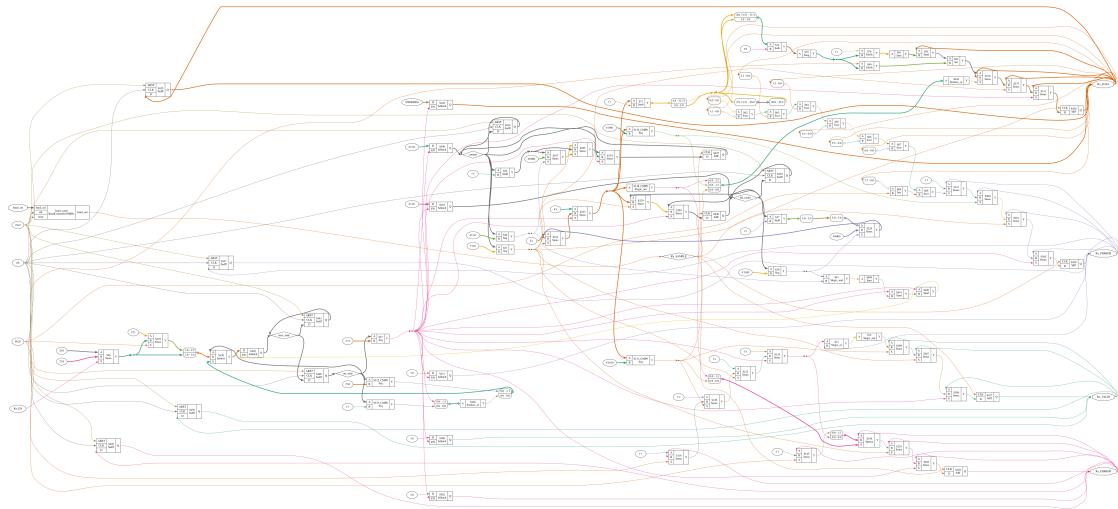
- Να ξεκινάει την δειγματοληψία μετά τον εντοπισμό του start bit.
- Να ευθυγραμμίσει την δειγματοληψία του RxD στην μέση της διάρκειας αποστολής. Δηλαδή εφόσον ο αποστολέας στέλνει ένα bit 16 "φορές" ο δέκτης να διαβάσει το RxD την 8^η φορά.
- Να υπολογίσει το parity bit για τα δεδομένα που έλαβε και αν δεν είναι ίδιο με το parity bit του αποστολέα να πάει στο '1' την έξοδο Tx_PERROR.
- Αν την στιγμή που περίμενε το stop bit δεν το λάβει τότε να πάει στο '1' την έξοδο Tx_FERROR.
- Αν όλα πάνε καλά κι διαβάσει 8 bits τα παρέχει στην έξοδο Tx_DATA κάνοντας και την έξοδο Tx_VALID '1'.

Υλοποίηση

Η υλοποίηση έγινε με την χρήση ενός FSM με 3 καταστάσεις (Σχήμα 3.9). Όταν στην κατάσταση ON διαβαστεί RxD '0' δηλαδή το start bit, ο δέκτης μπαίνει στην κατάσταση REC (Receiving).



Σχήμα 3.9: FSM UART Receiver.

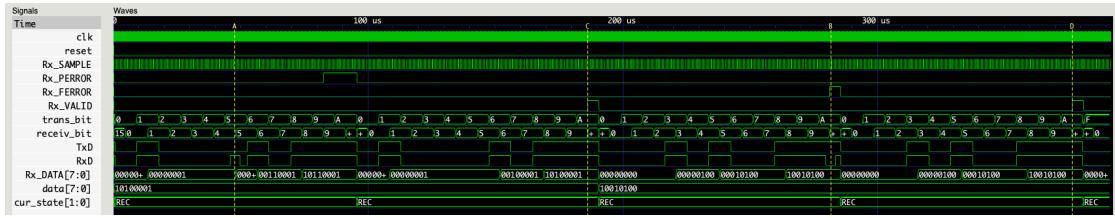


Σχήμα 3.10: Σύνθεση του UART Receiver με yosys.

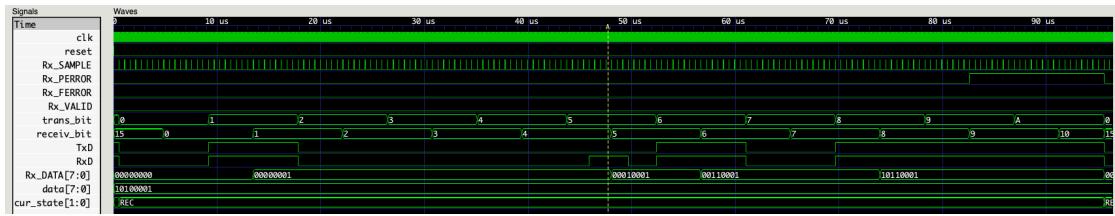
Επαλήθευση

Για την επαλήθευση της σωστής λειτουργίας δημιουργήθηκε ένα testbench στο οποίο έχει συνδεθεί ο αποστολέας με τον δέκτη, και πρώτα στέλνεται το πακέτο '-1' δύο φόρες την πρώτη φορά με parity error και την δεύτερη σωστά. Έπειτα στέλνεται το πακέτο '94' δύο φορές, την πρώτη με frame error και την δεύτερη σωστά (Σχήμα 3.11).

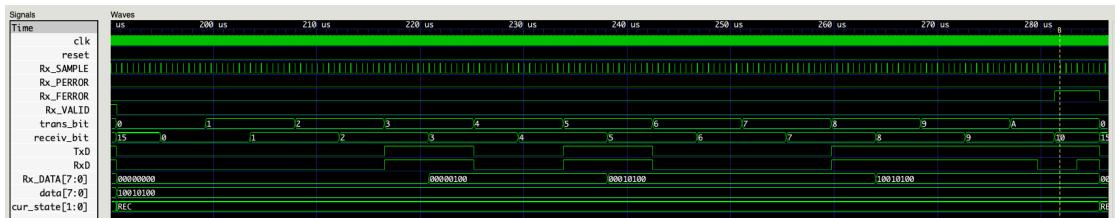
Το parity και frame error γίνεται με την προσθήκη θορύβου κοντά στην στιγμή δειγματοληψίας του δέκτη όπως φαίνεται στα markers A και B. Ενώ στα markers C και D φαίνεται ότι μετά την λήψη και του stop bit εφόσον δεν υπήρχε κάποιο error το Tx_VALID πάει στο '1'.



Σχήμα 3.11: UART Receiver testbench για λήψη του '-194' (BR=115200).



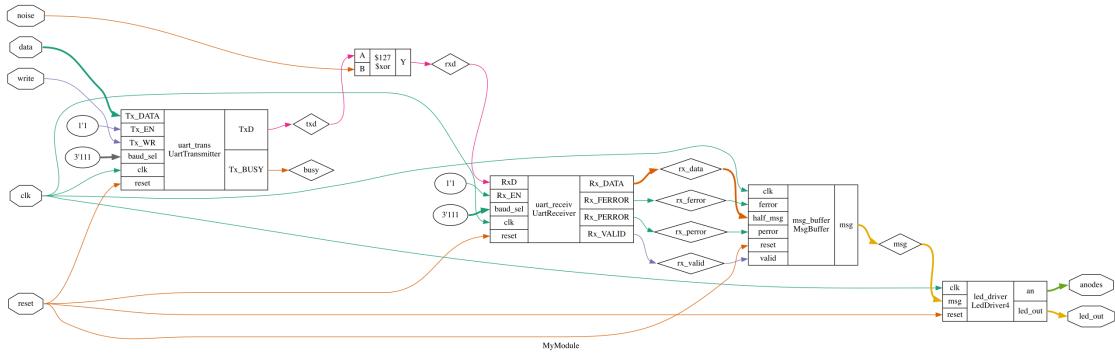
Σχήμα 3.12: Μεγέθυνση στην λήψη με parity error. Την χρονική στιγμή Α δέκτης διαβάζει 1 ενώ στέλνεται 0.



Σχήμα 3.13: Μεγέθυνση στην λήψη με frame error. Την χρονική στιγμή Β δέκτης διαβάζει 0 ενώ περίμενε το stop bit 1.

4 Τελική σύνδεση

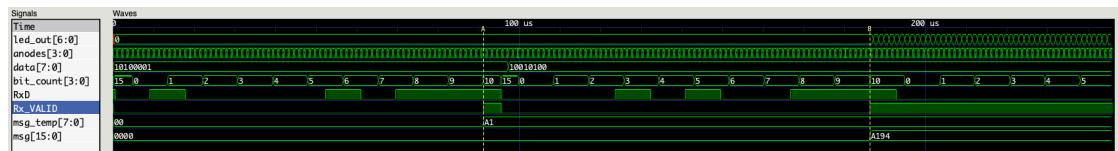
Τέλος για να συνδεθούν όλα τα modules μαζί φτιάχτηκε ένα νέο module *MyModule* το οποίο κάνει τις απαραίτητες συνδέσεις. Το *MyModule* έχει ως εισόδους τα 7 bits data, 1 bit write και noise και ως εξόδους τις 4 ανόδους και τα 6 led. Το *MyModule* ήταν απαραίτητο ώστε να το yosys να θεωρήσει όλα τα άλλα modules ως blackbox.



Σχήμα 4.1: Τελική συνδεσμολογία όλων των modules.

Όπως φαίνεται στο Σχήμα 4.1 το `noise` περνάει μέσα από μια XOR μαζί με το `TxD` και παράγουν το `RxD` ώστε θέτοντας το `noise` σε '1' να αντιστρέφουμε το σήμα που λαμβάνει ο δέκτης και έτσι να προσομοιάσουμε θόρυβο.

Επίσης ανάμεσα από το `UartReceiver` και `LedDriver4` υπάρχει άλλο ένα module το οποίο ήταν απαραίτητο για την ολοκλήρωση της εργασίας. Επειδή το μήνυμα που θέλουμε να δείξουμε στην οθόνη είναι 16 bits αλλά ο αποστολέας κι ο δέκτης επικοινωνούν 8 bits την φορά αυτό σημαίνει ότι θα χρειαστούμε 2 λήψεις ώστε να έχουμε ένα πλήρες μήνυμα. Το `MsgBuffer` module λοιπόν αυτό που κάνει είναι να εξάγει 16 bits `msg` μετά από 2 `valid` `half_msg` που έχει λάβει από τον `UartReceiver` και αν εντοπίσει είτε `perror` είτε `ferror` κάνει το `msg` 'FFFF'.



Σχήμα 4.2: Αποστολή '-194' και εμφάνιση στις οθόνες.

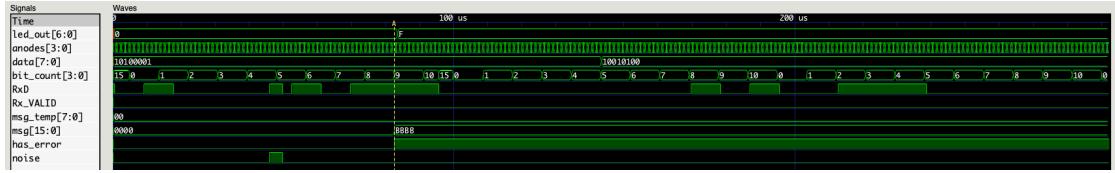
Στο Σχήμα 4.2 την χρονική στιγμή Α φτάνει το τελευταίο bit του πρώτου μηνύματος και το μήνυμα υπολογίζεται ως `valid`. Εκείνη την στιγμή το 8 bits μήνυμα αποθηκεύεται στο `msg_tmp` register του `MsgBuffer`. Την χρονική στιγμή Β φτάνει 2^o `VALID` μήνυμα και τότε το `MsgBuffer` το ενώνει με το άλλο μισό που έλαβε προηγουμένως και το περνάει στο `LedDriver4`.

Μεγεθύνοντας στην χρονική περίοδο Β και μετά, δηλαδή αφού έχει ληφθεί το πλήρες μήνυμα (Σχήμα 4.3), βλέπουμε ότι τιμή του `led_out` είναι σωστή την κατάλληλη χρονική στιγμή. Δηλαδή την στιγμή που η άνοδος 3 είναι στο 0 στο `led_out` έχουμε τιμή '-' άρα στην πρώτη οθόνη θα εμφανιστεί ο χαρακτήρας '-', αντίστοιχα για τις άλλες ανόδους καταλαβαίνουμε ότι το μήνυμα που θα δούμε στις 4 οθόνες είναι '-194'.



Σχήμα 4.3: Αποστολή '-194' και εμφάνιση στις οθόνες αφού έχει ληφθεί το πλήρες μήνυμα.

Τέλος στο Σχήμα 4.4 βλέπου την περίπτωση όπου το *MsgBuffer* εντοπίζει parity error από το πρώτο κιόλας πακέτο άρα κάνει την έξοδο του msg 'FFFF'.



Σχήμα 4.4: Αποστολή '-194' με parity error.

Κώδικας και εικόνες

Όλες οι εικόνες, τα waveforms και ο κώδικας των modules είναι διαθέσιμα στο github repo github.com/johnstef99/hw1-auth.