Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Πανεπιστήμιο Θεσσαλίας

2η Εργαστηριακή Εργασία - Υλοποίηση Μονάδας Γενικού Ασύγχρονου Δέκτη Αποστολέα (UART)

ECE333 - Εργαστήριο Ψηφιακών Συστημάτων

Διδάσκων: Χ. Σωτηρίου Ημερομηνία: 03/12/2023

Συγγραφέας Εργασίας: Γεώργιος Καπάκος

ΑΕΜ: 03165

Περιεχόμενα

ΠΕΡΙΛΗΨΗ:

Στόχος της 2ης εργασίας είναι η υλοποίηση των δύο μονάδων που απαιτούνται για την σειριακή επικοινωνία μέσω του πρωτοκόλλου UART, ενός Αποστολέα (Tx) και ενός Δέκτη (Rx). Σε αυτό την εργαστηριακή αναφορά περιγράφεται ο κώδικας με τον οποίο υλοποιήθηκαν, ο τρόπος με τον οποίο συγχρονίστηκαν οι δύο μονάδες μεταξύ τους και τα αποτελέσματα των πλαισίων δοκιμής. Τελικός στόχος του project είναι η επιτυχής μεταφορά τεσσάρων συμβόλων (8-bits data) από τον Αποστολέα στο Δέκτη.

ΕΙΣΑΓΩΓΗ:

Οι στόχοι της εργασίας αυτής είναι η υλοποίηση της ενός UART πρωτόκολλου. Υλοποιούμε, δηλαδή ένα σειριακό, ασύγχρονο πρωτόκολλο επικοινωνίας. Ο αποστολέας αποστέλλει 8 bits δεδομένων, τα οποία λαμβάνει από το σύστημα και ο δέκτης τα δειγματοληπτεί και τα αποθηκεύει σε έναν 8-bit register, ανά bit.

Το συγκεκριμένο πρωτόκολλο επικοινωνίας, έχει την ιδιαιτερότητα ότι οι 2 ή περισσότερες συσκευές, οι οποίες το χρησιμοποιούν για επικοινωνία , μπορεί να έχουν άσχετα ή ασυγχρόνιστα ρολόγια. Η ασύγχρονη επικοινωνία του UART υλοποιείται μέσω μιας ενσύρματης σύνδεσης των δεδομένων που στέλνει ο αποστολέας (TxD) και αυτά τα οποία λαμβάνει ο δέκτης (RxD). Μέσω αυτού του πρωτοκόλλου δουλεύουμε ανεξάρτητα από τις συνθήκες που επικρατούν στα συστήματα εισόδου και εξόδου, οπότε για να μην χάνονται δεδομένα, επιλέγεται μία κοινή ταχύτητα (baud rate), με την οποία θα επικοινωνούν μέσω του UART ο Transmitter και ο Receiver. Ο Δέκτης θα λειτουργεί με αυτή την ταχύτητα, ενώ ο Αποστολέας θα εξετάζει κάθε μεταδιδόμενο ψηφίο 16 φορές πιο γρήγορα.

Επειδή δεν υπάρχει κάποια εγγύηση ως προς τη σχετική συχνότητα και τη φάση μεταξύ των ρολογιών του Αποστολέα και Δέκτη όντας η επικοινωνία ασύγχρονη, ο δέκτης πρέπει να γνωρίζει πότε ξεκινάει η μετάδοση ενός συμβόλου. Οι δύο μονάδες είναι συνδεδεμένες με ένα σήμα ενός bit, το οποίο όταν δεν γίνεται κάποια μετάδοση, έχει την τιμή 1. Το σήμα εκκίνησης (start bit) σηματοδοτεί την έναρξη της επικοινωνίας για τον Δέκτη, το οποίο για να το αναγνωρίσει έχει την τιμή 0. Αντίστοιχα, τη λήξη του συμβόλου και το πέρας της επικοινωνίας σηματοδοτείται από το bit παύσης (stop bit), στο οποίο δίνεται η τιμή 1, για την επιστροφή στην αρχική κατάσταση αδράνειας.

Για την επαλήθευση της ορθής επικοινωνίας των δεδομένων, αυτά συνοδεύονται από ένα bit ισοτιμίας (parity bit), το οποίο δείχνει εάν το σύμβολό μας περιέχει άρτιο ή περιττό αριθμό άσων. Σε περίπτωση που τα δεδομένα που έχουμε λάβει είναι διαφορετικά από αυτά τα οποία έχουμε στείλει θα φανεί στην ισοτιμία του parity bit και θα ενεργοποιηθεί το σήμα λάθους Rx_PERROR. Επιπλέον εάν η δειγματοληψία κάποιου bit αποτύχει να γίνει στο κέντρο τότε ενεργοποιείται το σήμα λάθους Rx_FERROR. Αν κάποιο από αυτά είναι ενεργοποιημένα τότε τα bits, τα οποία λάβαμε δεν τα παίρνουμε υπόψιν μας.

Το πρωτόκολλο UART δεν περιλαμβάνει έλεγχο ροής μεταξύ Δέκτη - Αποστολέα, δηλαδή ο Δέκτης δεν μπορεί να σταματήσει τον Αποστολέα, έτσι και οι δυο θα πρέπει

να είναι έτοιμοι για διαρκή επικοινωνία. Τέλος, για τη μετάδοση πολλών διαφορετικών συμβόλων απαιτείται η σωστή διαχείριση των σημάτων εισόδου και εξόδου. Όταν ξεκινάει μια μεταφορά ο Αποστολέας δεν δέχεται νέο σύμβολο προς μετάδοση μέχρι να ολοκληρωθεί η τρέχουσα. Από την άλλη, ο Δέκτης διατηρεί τις τιμές των εξόδων του μέχρι να πραγματοποιηθεί νέα επικοινωνία.

ΜΕΡΟΣ Α-Ελεγκτής Baud Rate:

Υλοποίηση:

Σε αυτό το μέρος υλοποιούμε έναν ελεγκτή Baud Rate, ο οποίος θα υλοποιηθεί εσωτερικά στα κυκλώματα του αποστολέα και του δέκτη. Ο ελεγκτής αυτός προσδίδει στον δέκτη και τον αποστολέα έναν κοινό ρυθμό δειγματοληψίας, δηλαδή προσυμφωνούν την ταχύτητα της μεταξύ τους επικοινωνίας σε μονάδες Baud (bits/sec), σύμφωνα με το παρακάτω πινακάκι:

BAUD_SEL	Baud Rate *(bits/sec)*
0 0 0	300
0 0 1	1200
0 1 0	4800
0 1 1	9600
1 0 0	19200
1 0 1	38400
1 1 0	57600
1 1 1	115200

Στον παραπάνω πίνακα παρουσιάζονται οι ταχύτητες επικοινωνίας του UART, και το σήμα επιλογής Baud_Sel, που χρησιμοποιείται για την επιλογή Baud Rate, που μεταδίδεται ένα σύμβολο. Η περίοδος μετάδοσης του κάθε ψηφίου από τον Αποστολέα, αναλογεί σε Tsample = 1/BaudRate .

Ωστόσο, η λειτουργία και δειγματοληψία του UART Δέκτη γίνεται 16 φορές γρηγορότερα της προσυμφωνημένης συχνότητας Baud Rate, από τον Δέκτη. Πρακτικά, ο Δέκτης λειτουργεί σε ταχύτητα BaudRate × 16, ενώ ο Αποστολέας σε ταχύτητα BaudRate. Επομένως, ο Ελεγκτής θα πρέπει να στέλνει κατάλληλο σήμα δειγματοληψίας, με βάση το οποίο θα λειτουργούν ο Αποστολέας και ο Δέκτης. Αυτό έχει ως αποτέλεσμα οι δύο μονάδες να λειτουργούν ανεξάρτητα από το ρολόγια τους, το οποίο είναι επιθυμητό, αφού δεν είναι απαραίτητο ότι θα έχουν κοινό ρολόι και σε αυτή την περίπτωση δεν θα απαιτείται να τα συγχρονίσουμε.

Υπολογίζουμε για κάθε Baud Rate πόσοι κύκλοι ρολογιού αντιστοιχούν στα αντίστοιχα bits/sec, μέσω του τύπου: max_cycles = fclk/ fsample = fclk/(16*Baud_Rate).

Χρησιμοποιώντας το ρολόι της πλακέτας, έχουμε fclk = 100MHz. Οπότε από τον παραπάνω πίνακα και τον τύπο προκύπτει:

Baud Select	Baud Rate	max_cycles	Προσέγγιση	Σχετικό Σφάλμα(%)
000	300	20833.3333	20833	0.0016
001	1200	5208.3333	5208	0.0064
010	4800	1302.0833	1302	0.0064
011	9600	651.0416	651	0.0639
100	19200	325.5208	326	0.1472
101	38400	162.7604	163	0.1472
110	57600	108.5069	109	0.4544
111	115200	54.2535	54	0.4673

Το σχετικό σφάλμα υπολογίζεται από τον τύπο:

|Χπροσέγγιση - Χπραγματική|/ Χπραγματική * 100%

Παρατηρούμε πως όσο αυξάνουμε το Baud Rate οι κύκλοι μειώνονται και το σχετικό σφάλμα αυξάνεται. Βέβαια το σφάλμα προσέγγισης είναι αρκετά μικρό οπότε στην υλοποίηση του UART, δεν θα παρατηρούνται επί των πλείστων λάθη.

Για την υλοποίηση του Baud Rate Ελεγκτή δημιουργήσαμε το παρακάτω module:

Baud Controller:

Αποτελείται από τις εισόδους:

- **reset**: Ασύγχρονο reset του κυκλώματός μας
- **clk**: Ρολόι, με fclk = 100MHz συχνότητα
- **baud_select**, με βάση την συγκεκριμένη είσοδο επιλέγουμε ποιο baud rate, θα έχουμε για την μετάδοση του σήματός μας, σύμφωνα με τον πίνακα αντιστοιχίας του baud select στο baud rate. Αυτό το υλοποιούμε σε ένα combinational always block, με τη λίστα ευαισθησίας να περιέχει μόνο το baud_select. Συνάμα θέτουμε μία μεταβλητή την οποία ονομάζουμε size, τον αριθμό των μέγιστων κύκλων, ανάλογα με την επιλογή του baud select. Επιπλέον με βάση το size του baud select, για να μπορούμε να συμπεριλάβουμε όλα τα σήματα θέτουμε έναν 15 bit counter. Ο counter έχει αυτό το μέγεθος επειδή: 2^14<20833<2^15, για να μπορεί να μετρήσει όλες τις τιμές μας, το 20833 είναι η μέγιστη τιμή μας.

```
always@ (baud_select) begin
    case(baud_select)
    3'b000: begin
            size <= 20_833;
            end
    3'b001: begin
            size <= 5_208;
            end
    3'b010: begin
            size <= 1_302;
            end
    3'b011: begin
            size <= 651;
            end
    3'b100: begin
            size <= 326;
            end
    3'b101: begin
            size <= 163;
            end
    3'b110: begin
            size <= 109;
            end
    3'b111: begin
            size <= 54;
            end
    default: begin
             size <= 20_833;
             end
    endcase
end
```

Αποτελείται από την έξοδο:

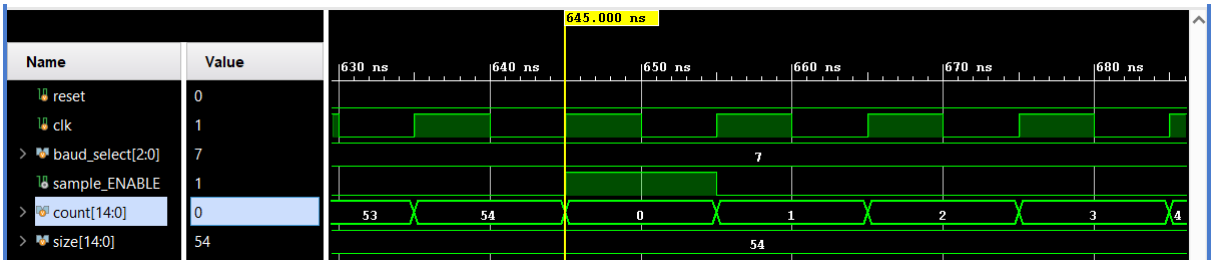
- **sample_ENABLE:** Έξοδος του ελεγκτή μας, η οποία ενεργοποιείται κάθε φορά για έναν κύκλο ρολογιού, όταν ο 15 bit counter, που έχουμε δημιουργήσει φτάσει στο επιλεγμένο size. Αλλιώς παραμένει ανενεργή και ο μετρητής αυξάνεται. Αυτός ο μετρητής υλοποιείται σε ένα sequential always block.

```
/*15 bit counter*/
always @ (posedge clk or posedge reset) begin
    if (reset) begin
        count <= 15'b000000000000000;
        sample_ENABLE <= 1'b0;
    end
    else if(count == size) begin
        sample_ENABLE <= 1'b1;
        count <=15'b000000000000000;
    end
    else begin
        count <= count + 15'b000000000000001;
        sample_ENABLE <= 1'b0;
    end
end
```

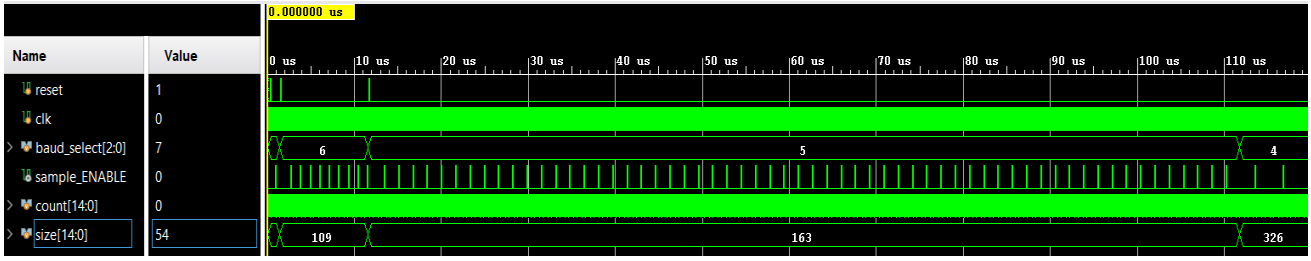
Επαλήθευση:

Για την επαλήθευση του κυκλώματός μας έχουμε δημιουργήσει ένα testbench, στο οποίο εξετάζουμε το σήμα εξόδου sample_ENABLE, για διαφορετικές τιμές του

baud_select. Πριν την κάθε αλλαγή στο baud_select, κάνω reset στο κύκλωμα, για την εκ νέου αρχικοποίησή του.



Στις παραπάνω κυματομορφές, παρατηρώ πως όταν ο μετρητής μας φτάσει την τιμή του size, δηλαδή του max_cycle, σύμφωνα με τον πίνακα που έχουμε δημιουργήσει, ενεργοποιεί την τιμή του sample_ENABLE, για έναν κύκλο ρολογιού. Αυτό επαναλαμβάνεται μέχρι το πέρας της προσομοίωσής μου.

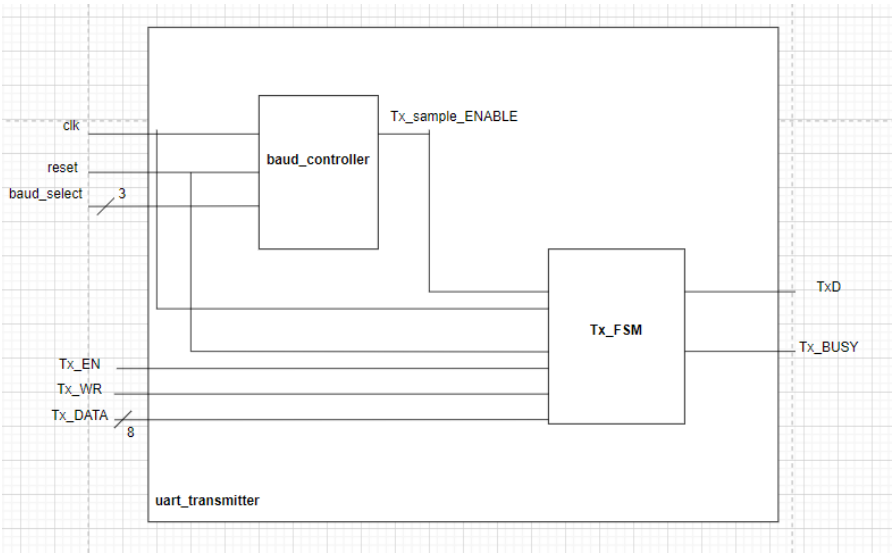


Στην παραπάνω κυματομορφές, βλέπουμε τη διαφορά συχνότητας, με την οποία αλλάζει το σήμα εξόδου ανάλογα με το Baud Rate, που έχουμε επιλέξει.

ΜΕΡΟΣ Β- Υλοποίηση UART Αποστολέα (Transmitter):

Υλοποίηση:

Η υλοποίηση του data flow του UART-Transmitter:



Σε αυτό το μέρος υλοποιούμε τον Αποστολέα UART, ως το top level module, uart_transmitter.

Το οποίο έχει ως εισόδους:

- **reset**: Ασύγχρονο σήμα αναίρεσης του κυκλώματος.
- **clk**: Ρολόι του κυκλώματος μας.
- **baud_select**: Σήμα επιλογής της ταχύτητας δειγματοληψίας.
- **Tx_DATA**: Αποτελεί το σύμβολο (8 bits), το οποίο θέλω να μεταδώσω.
- **Tx_EN**: Σήμα που ενεργοποιεί τον transmitter, για να αρχίσει την λειτουργία του, και παραμένει ανοιχτό μέχρι την ολοκλήρωση της μεταφοράς δεδομένων.
- **Tx_WR**: Σήμα που δηλώνει, ότι τα δεδομένα είναι έτοιμα προς αποστολή. Το σήμα παραμένει ενεργό για έναν κύκλο ρολογιού και γίνεται ξανά 1 όταν θέλουμε να στείλουμε καινούργιο σύμβολο.

Έχει ως εξόδους:

- **TxD:** Σήμα, το οποίο είναι τα ανά bit δεδομένα του μεταδίδουμε σειριακά στον δέκτη.
- **Tx_BUSY**: Σήμα, το οποίο δηλώνει πως ο αποστολέας βρίσκεται σε κατάσταση αποστολής και πως δεν μπορεί να δεχθεί νέα δεδομένα όσο βρίσκεται σε αυτήν την κατάσταση. Με την ολοκλήρωση της μετάδοσης του μηνύματος, το σήμα γίνεται 0 και επιτρέπει την ανάγνωση νέων δεδομένων.

Στο παραπάνω data flow φαίνεται ότι γίνονται instantiate τα παρακάτω modules:

- **baud_controller**: Αποτελεί τον ελεγκτή, τον οποίο έχουμε περιγράψει στο Α Μέρος της εργασίας μας. Παράγει ως έξοδό του το σήμα Tx_sample_ENABLE. Όταν το σήμα αυτό, είναι ανοιχτό επιτρέπει στο FSM, που έχουμε δημιουργήσει να προβεί σε αλλαγές των μεταβλητών του κυκλώματος.

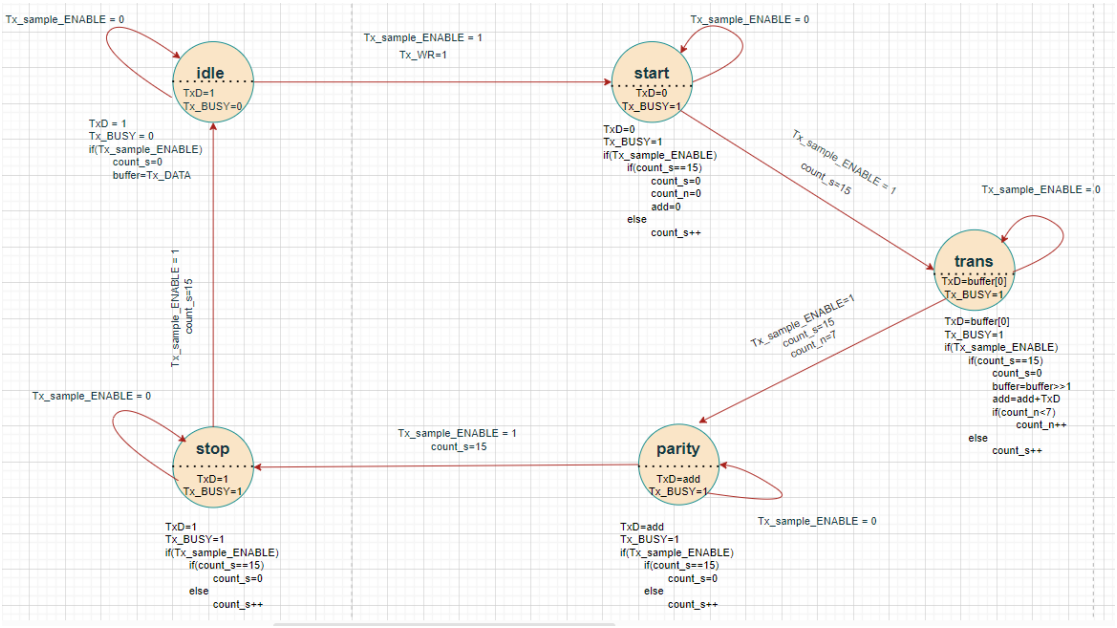
- **Tx_FSM**: Υλοποιούμε μία FSM, στην οποία κάνουμε τις κατάλληλες ενέργειες, για τη μετάδοση του σήματος. Συνδέονται όλες οι είσοδοι του κυκλώματός μας σε αυτήν την ΜΠΣ, εκτός από το baud_rate και παράγει και τα 2 σήματα εξόδου μας. Οι καταστάσεις από τις οποίες αποτελείται είναι:
    1) **idle(000):** Σε αυτή την κατάσταση ο transmitter βρίσκεται σε κατάσταση αδράνειας, οπότε κάνει το σήμα TxD = 1 και το σήμα Tx_BUSY = 0. Περιμένει να ενεργοποιηθεί το σήμα Tx_ENABLE και το σήμα Tx_WR, για να αλλάξει κατάσταση, αλλιώς παραμένει ανενεργό.
    2) **start(001):** Σε αυτήν την κατάσταση κάθε φορά που το Tx_sample_ENABLE, ενεργοποιείται, θα αυξάνεται ο μετρητής μας και όταν μετρήσει 16 κύκλους θα περάσουμε στην επόμενη κατάσταση, αρχικοποιώντας μεταβλητές που θα χρησιμοποιήσουμε στο επόμενο βήμα. Επίσης, ενεργοποιεί το start bit που σημαίνει την έναρξη μετάδοσης στο κύκλωμά μας και θέτουμε τις εξόδους σε TxD=0 και Tx_BUSY=1.
    3) **trans(010):** Σε αυτή την κατάσταση το σήμα εξόδου των bits μας θα λαμβάνει την τιμή του κατάλληλου bit, από το σήμα που έχουμε χρησιμοποιήσει σαν είσοδο. Αυτό θα υλοποιηθεί κάνοντας δεξί shift κατά 1 τα bits που Tx_DATA, τα οποία έχουμε αποθηκεύσει σε ένα buffer. Συνάμα, αυξάνουμε την τιμή του add, που ελέγχει το parity bit, στην παρακάτω κατάσταση. Αυτά γίνονται κάθε 16 κύκλους ρολογιού, όταν το Tx_sample_ENABLE, είναι ενεργό. Τέλος με την μέτρηση των 8 bits, που υπάρχουν προς μετάδοση, μεταβαίνουμε στην επόμενη κατάσταση.
    4) **parity(011):** Σε αυτήν την κατάσταση θέτουμε την τιμή του TxD ίση με το άθροισμα όλων των ψηφίων προς μετάδοση του σήματος στην προηγούμενη κατάσταση, το ονομάζουμε parity bit και ελέγχει την ισοτιμία των bits.
    5) **stop(100):** Σε αυτήν την κατάσταση φτάνουμε, για να δηλώσουμε την λήξη της μετάδοσης των bits μας, στέλνοντας το stop bit, που σηματοδοτεί την λήξη μετάδοσης του συμβόλου (TxD = 1).

Για την υλοποίηση της FSM χρησιμοποιούμε δύο always blocks:

Το πρώτο είναι sequential always block και είναι υπεύθυνο για να δίνει τιμές στην στην τωρινή κατάσταση εξισώνοντάς τη με την επόμενη κατάσταση. Κάθε φορά που αλλάζει μία μεταβλητή αυτή ανατίθεται στην τωρινή κατάσταση. Αλλιώς εάν πατηθεί το reset, έχουμε αρχικοποίηση όλων των τιμών σε 0.

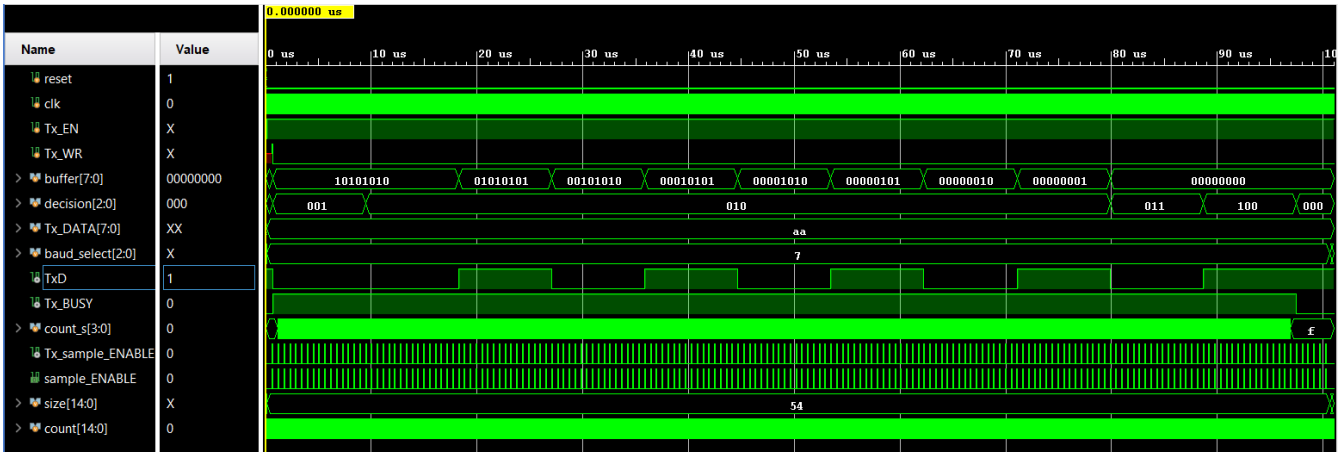
Το δεύτερο είναι combinational always block και τροποποιεί τις επόμενες καταστάσεις με την ενεργοποίηση του Tx_sample_ENABLE και όταν ο μετρητής μας φτάσει την τιμή 15, τότε επιτρέπουμε τις αλλαγές στις μεταβλητές και την μετάβαση από τη μία στην άλλη κατάσταση.

Παρακάτω παραθέτουμε το FSM μοντέλο σχηματικά, ως μοντέλο Moore:



Επαλήθευση:

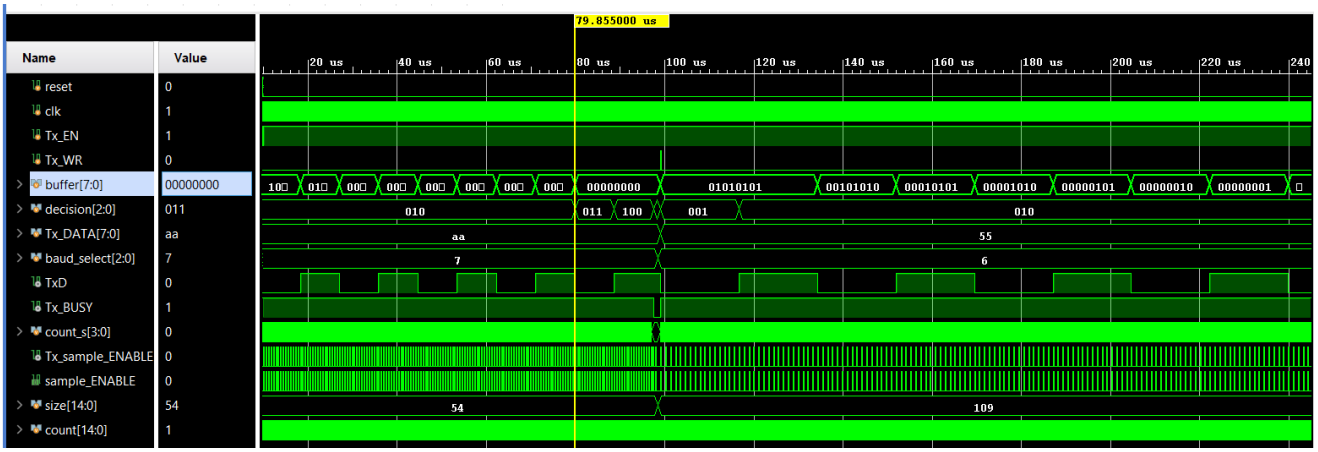
Στο testbench, δίνουμε baud_select = 111, που είναι η γρηγορότερη από τις δεδομένες ταχύτητες, και Tx_DATA = 10101010. Ενεργοποιούμε το σήμα Tx_WR, για έναν κύκλο ρολογιού όταν είναι ενεργοποιημένο και το σήμα Tx_sample_ENABLE, για την μετάβαση από την idle κατάσταση στην start. Η κυματομορφή είναι η ακόλουθη:



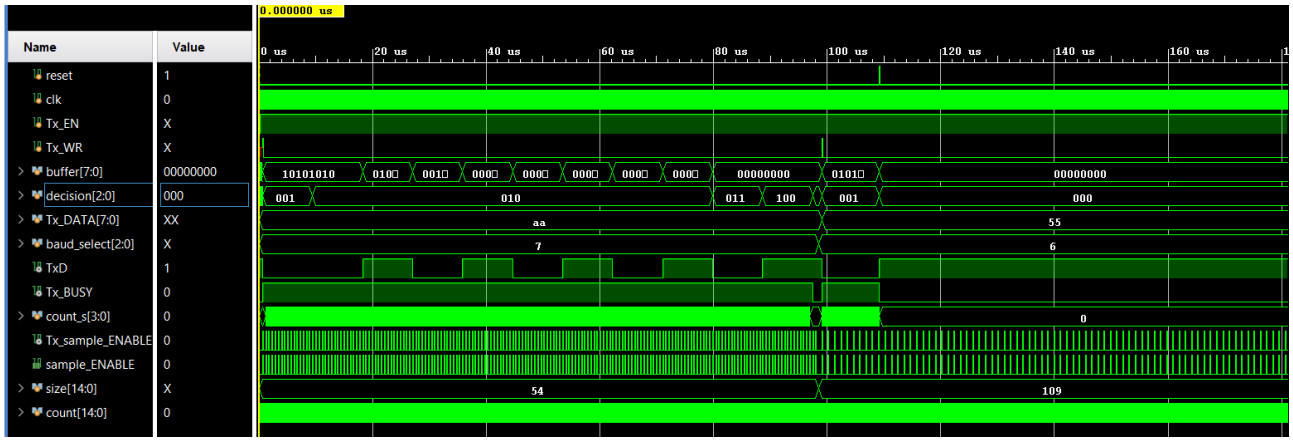
Στην παραπάνω κυματομορφή παρατηρούμε πως οι καταστάσεις αλλάζουν από την idle έως την stop,η μία μετά την άλλη. Από 000->100, (5 καταστάσεις). Το σήμα TxD, είναι αρχικά 1 στην ανενεργή κατάσταση (idle), μετά όμως τη μετάβασή στην κατάσταση έναρξης(start) γίνεται μηδέν και ύστερα στην κατάσταση μετάδοσης(trans), λαμβάνει την τιμή που έχει το lsb(least significant bit), του buffer, που έχουμε δημιουργήσει, στον οποίο εφαρμόζουμε right shift, οπότε βλέπουμε τα most significant bits του να πάνε μία θέση προς τα δεξιά κάθε φορά. Στην κατάσταση ελέγχου του ψηφίου (parity), παρατηρούμε ότι παίρνει την τιμή της πρόσθεσης όλων

των ψηφίων προς αποστολή, στην προκειμένη περίπτωση είναι 0, δηλαδή άρτιο, οπότε τα ψηφία του μεταδόθηκαν σωστά επειδή το σήμα μας έχει άρτιο αριθμό άσων και μηδενικών. Τέλος, φτάνουμε στην κατάσταση που σταματάμε την μετάδοση (stop), εκεί το TxD, γίνεται ξανά 1, η κατάσταση αυτή σημαίνει την λήξη μετάδοσης και επιστρέφει ξανά στην αρχική ανενεργή κατάσταση, μέχρι να ενεργοποιηθεί ξανά το σήμα Tx_WR, για την αποστολή νέου μηνύματος.

Καθ' όλη την διάρκεια της μετάδοσης το σήμα Tx_BUSY είναι 1, με τη λήξη της μετάδοσης γίνεται 0, όταν δηλαδή φτάσει στην κατάσταση idle, όπως φαίνεται στις κυματομορφές.



Εδώ βλέπουμε πως όταν ενεργοποιηθεί εκ νέου το σήμα Tx_WR, το σύστημά μας είναι έτοιμο να υποδεχθεί το καινούργιο σήμα προς αποστολή, το οποίο αποστέλλει με διαφορετικό baud rate, εκτελώντας τις ίδιες λειτουργίες.

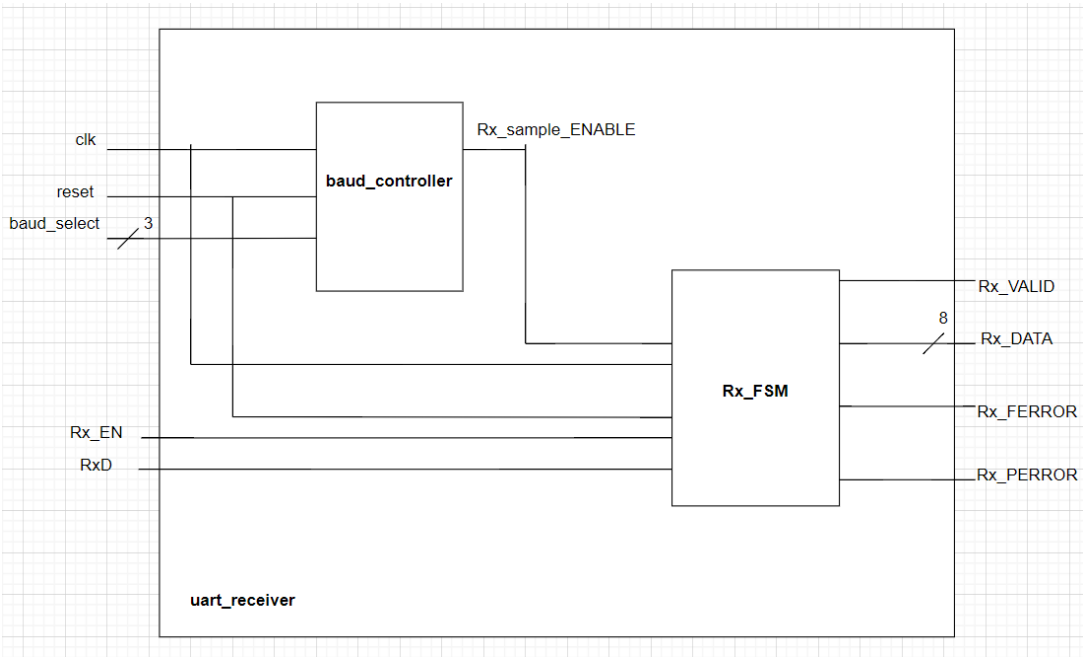


Τέλος σε αυτές τις κυματομορφές βλέπουμε πως μετά την ενεργοποίηση του reset, το κύκλωμα επανέρχεται στην ανενεργή (idle), κατάστασή του.

ΜΕΡΟΣ Γ- Υλοποίηση UART Δέκτη (Receiver)*:*

Υλοποίηση:

Η υλοποίηση του UART Receiver, φαίνεται στο παρακάτω σχήμα:



Σε αυτό το μέρος υλοποιούμε τον Δέκτη UART, ως το top level module, uart_receiver.

Το οποίο έχει ως εισόδους:

- **reset**: Ασύγχρονο σήμα αναίρεσης του κυκλώματος.
- **clk**: Ρολόι του κυκλώματος μας.
- **baud_select**: Σήμα επιλογής της ταχύτητας δειγματοληψίας.
- **Rx_EN**: Σήμα που ενεργοποιεί τον receiver, για να αρχίσει την λειτουργία του, και παραμένει ανοιχτό μέχρι την ολοκλήρωση της μεταφοράς δεδομένων.
- **RxD**: Σήμα, το οποίο λαμβάνει τα δεδομένα από το TxD και τα κάνει δειγματοληψία bit προς bit.

Έχει ως εξόδους:

- **Rx_DATA:** Τα δεδομένα τα οποία λάβαμε από τον αποστολέα, εφόσον τα έχει αποστείλει χωρίς λάθη.
- **Rx_FERROR**: Σήμα, το οποίο δηλώνει πως ο δέκτης έχει κάνει λανθασμένη δειγματοληψία είτε στην αρχή του σήματος (start bit) είτε στο τέλος (stop bit).
- **Rx_PERROR**: Σήμα, το οποίο δηλώνει πως ο δέκτης έχει λάβει διαφορετικό parity bit από αυτό το οποίο υπολογίζουμε, σύμφωνα με το σύμβολο που έχουμε δειγματοληπτήσει.

- **Rx_VALID**: Σήμα, το οποίο όταν γίνεται ένα δηλώνει πως τα δεδομένα που έχουμε λάβει έφτασαν επιτυχώς, χωρίς λάθη. Γίνεται ένα, για ένα κύκλο ρολογιού.

Στο παραπάνω data flow φαίνεται ότι γίνονται instantiate τα παρακάτω modules:

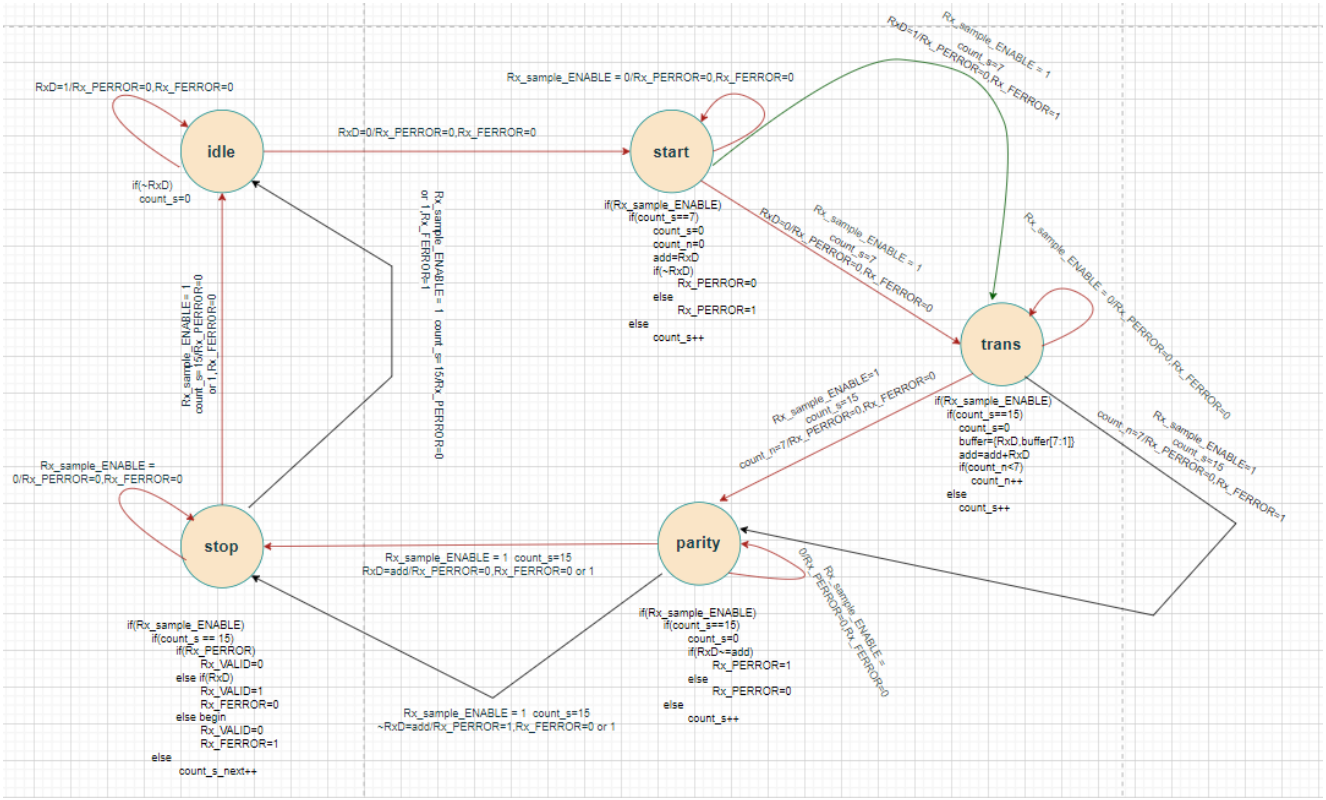
- **baud_controller**: Αποτελεί τον ελεγκτή, τον οποίο έχουμε περιγράψει στο Α Μέρος της εργασίας μας. Παράγει ως έξοδό του το σήμα Tx_sample_ENABLE. Όταν το σήμα αυτό, είναι ανοιχτό επιτρέπει στο FSM, που έχουμε δημιουργήσει να προβεί σε αλλαγές των μεταβλητών του κυκλώματος.

Όπως φαίνεται, στο top module γίνονται instantiate το module baud_controller, που λειτουργούν με το ίδιο τρόπο όπως περιεγράφηκε στον Αποστολέα. Η λογική και οι έξοδοι της μονάδας προκύπτουν από το τρίτο module (Rx_FSM):

- Το module **Rx_FSM**, υλοποιεί μία FSM, της οποίας οι καταστάσεις είναι οι ακόλουθες:
    1) **idle (000):** Η κατάσταση αυτή είναι η κατάσταση αναμονής. Περιμένουμε να λάβουμε αρχικά το σήμα Rx_EN, το οποίο δηλώνει την λειτουργία του δέκτη και ύστερα το start bit, από τον transmitter, το οποίο σημαίνει την έναρξη δειγματοληψίας του σήματός μας.
    2) **start (001):** Η κατάσταση αυτή αρχίζει την δειγματοληψία του μηνύματός μας. Ο μετρητής μας μετράει μέχρι τα μισά δηλαδή 8 κύκλους και τότε ενεργοποιούμε την επόμενη κατάσταση, για να μπορέσουμε να δειγματολειπτήσουμε στο κέντρο του μηνύματος μετράμε 8 κύκλους, σε αυτήν την κατάσταση.
    3) **trans (010):** Η κατάσταση αυτή σηματοδοτεί την έναρξη δειγματοληψίας του συμβόλου, το οποίο έχουμε στείλει μέσω του transmitter. Η δειγματοληψία γίνεται κάθε 16 κύκλους, στο κέντρο πάλι του παλμού. Κάθε φορά που ολοκληρώνονται 16 κύκλοι που το Rx_sample_ENABLE είναι ενεργό, προσθέτουμε στο add τα bits μεταξύ τους για τον ύστερο έλεγχο εάν το parity bit είναι λάθος. Επιπλέον, κάνουμε δεξί shift στο buffer ανά 16 κύκλους, το μήνυμά μας σε έναν buffer, τον οποίο τελικά θα αναθέσουμε στο Rx_DATA.
    4) **parity (011):** Η κατάσταση αυτή περιέχει τον έλεγχο εάν το σήμα το οποίο έχει φτάσει στον δέκτη και έχουμε δειγματοληπτίσει έχει φτάσει όντως σωστά. Εάν έχει φτάσει σωστά μετά από 16 κύκλους το Rx_PERROR, γίνεται 0, αλλιώς 1 και προβαίνουμε στην τελική αλλαγή της κατάστασης.
    5) **stop (100):** Η κατάσταση αυτή μετά από τους 16 κύκλους που το Rx_sample_ENABLE, είναι ενεργό ελέγχει εάν είχε ενεργοποιηθεί

προηγουμένως το Rx_PERROR και θέτει το Rx_VALID 0. Από την άλλη εάν όταν φτάσω στο stop bit και έχει την τιμή 1, τότε σημαίνει ότι έχω δειγματοληπτήσει σωστά. Θέτω Rx_VALID=1 και το Rx_DATA, παίρνει τα δεδομένα του buffer, εάν η δειγματοληψία γίνει λανθασμένα, τότε ενεργοποιείται το Rx_FERROR = 1 και το Rx_VALID=0, επιστρέφοντας στην αρχική ανενεργή κατάσταση.

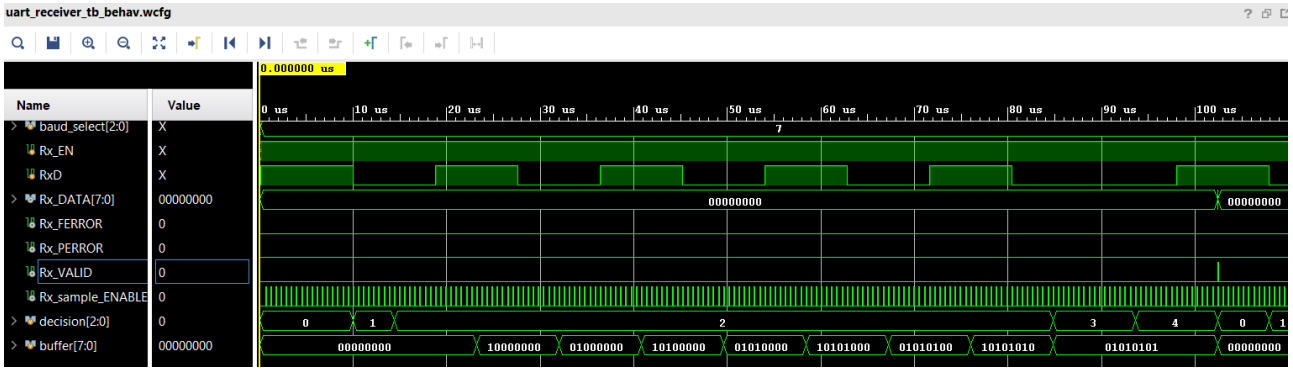
Το σχήμα υλοποίησης της FSM είναι το παρακάτω:

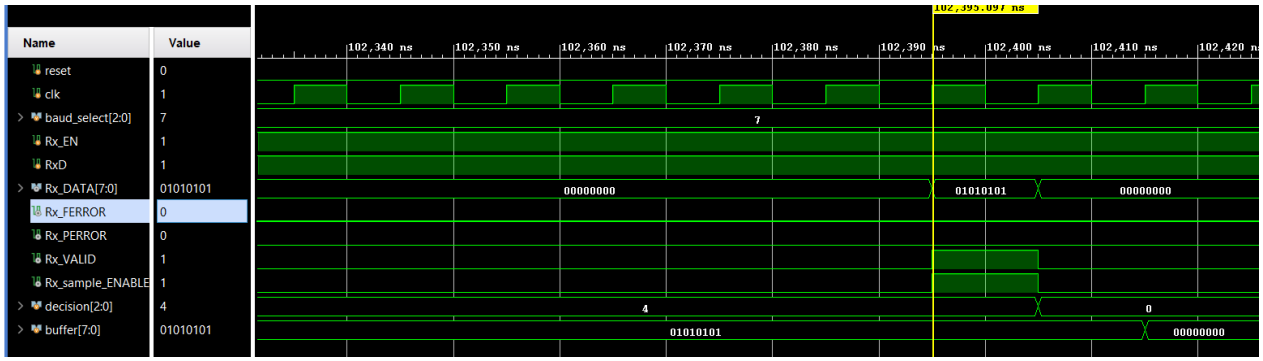


Επιπρόσθετα στον κώδικα χρησιμοποιούμε άλλα 2 combinational always blocks, ένα για να δώσω τιμή στο Rx_VALID, ανάλογα με τις τιμές που έχουν πάρει τα σήματα Rx_PERROR και Rx_FERROR. Δηλαδή εφόσον δεν αντιμετωπίζουμε λάθη και δεν είναι κανένα από τα δύο αυτά σήματα 1, τότε το Rx_VALID, γίνεται 1,αλλιώς 0. Το άλλο always block, για να δώσω την τιμή του buffer, που έχουμε υπολογίσει στην FSM στο Rx_DATA, εάν δεν υπάρχουν λάθη στην λήψη του συμβόλου, τότε το Rx_VALID, θα είναι ενεργό, αλλιώς δεν δίνουμε τιμή στο Rx_DATA, επειδή τα δεδομένα προς αποθήκευση είναι λανθασμένα.

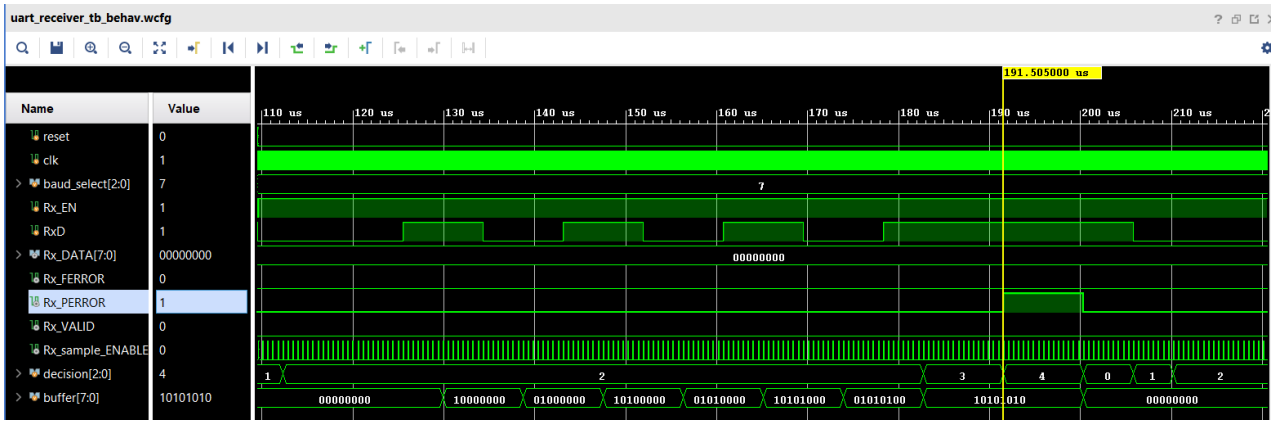
Επαλήθευση:

Σε αυτό το πλαίσιο δοκιμής θέλω να ελέγξω αν γίνεται σωστά η δειγματοληψία στη μέση της περιόδου των μεταδιδόμενων ψηφίων, αν τα σήματα λάθους ενεργοποιούνται ή όχι και τελικά αν οι τιμές τον εξόδων είναι οι αναμενόμενες, οι οποίες πρέπει να διατηρούνται και μετά το τέλος της επικοινωνίας.

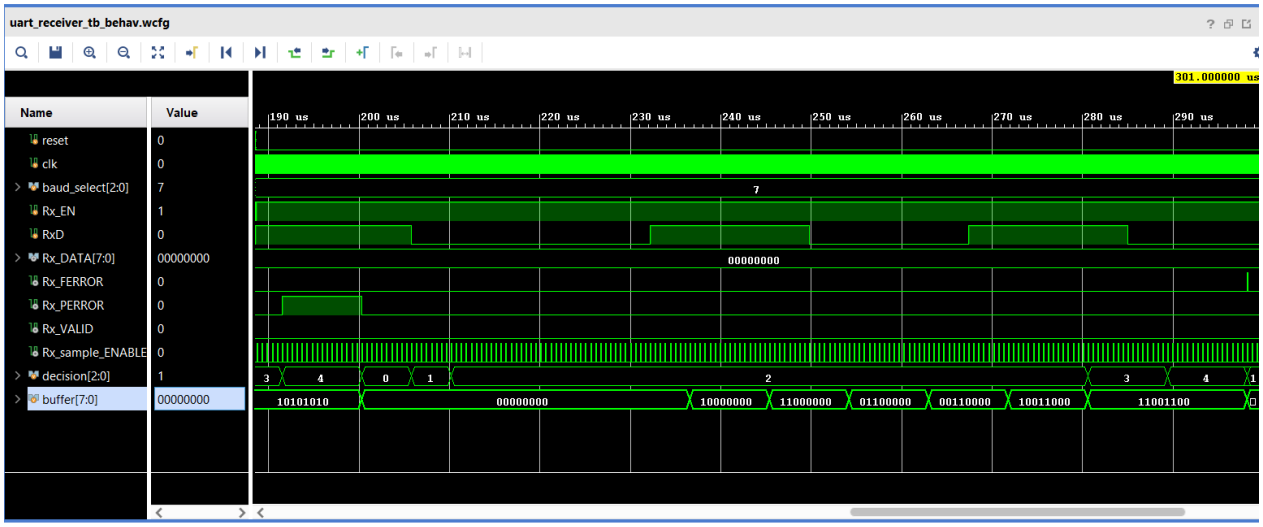




Από το άνω σχήμα παρατηρούμε πως η έξοδος Rx_DATA, εφόσον δεν έχει τελειώσει η FSM, τους ελέγχους δεν παίρνει ακόμα τιμή και θα πάρει μόνο τιμή εφόσον όλα λειτουργήσουν σωστά και δεν ενεργοποιηθούν τα μηνύματα λάθους στο κύκλωμα, με την λήξη του stop bit. Η τιμή αυτή αποθηκεύεται στο Rx_DATA, για ένα κύκλο ρολογιού όσο, δηλαδή είναι ανοιχτό το Rx_VALID. Στο παραπάνω κύκλωμα δεν υπάρχουν λάθη οπότε δεν ενεργοποιούνται και τα αντίστοιχα μηνύματα λάθους.



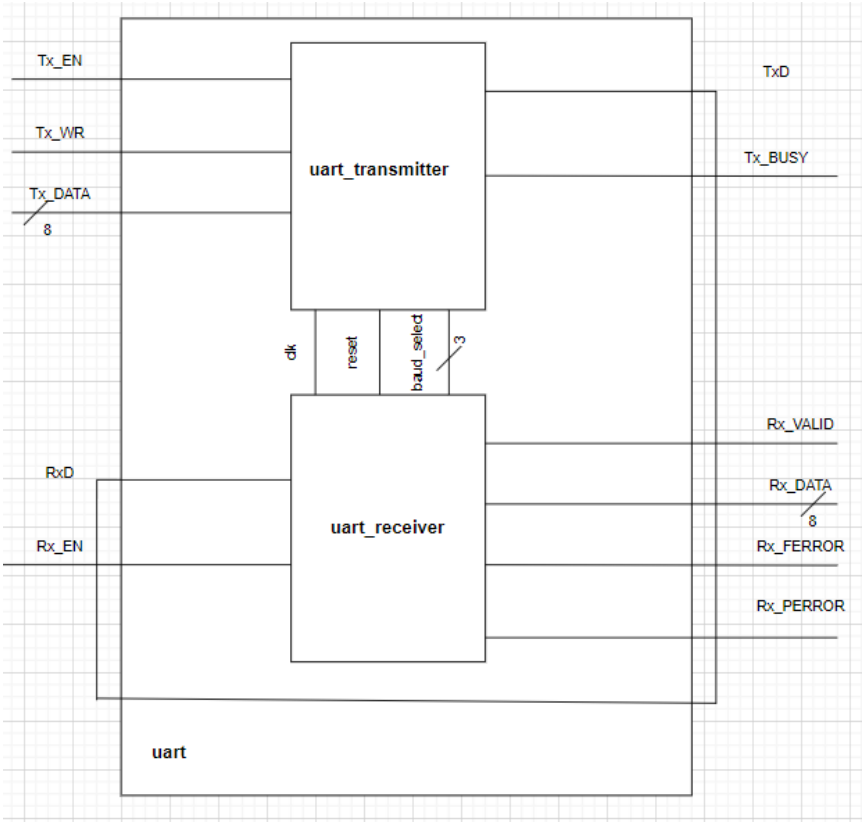
Στις παραπάνω κυματομορφές βλέπουμε την περίπτωση ύπαρξης λάθους στο κύκλωμά μας. Έχουμε PERROR, άρα ενεργοποιείται το σήμα του λάθους, ενώ το σήμα valid παραμένει 0, με αποτέλεσμα το Rx_DATA, δεν λαμβάνει δεδομένα, από το σύστημα, και παραμένει 0. Βλέπουμε πως το σήμα Rx_PERROR, ενεργοποιείται μετά το στάδιο parity.



Στις παραπάνω κυματομορφές βλέπουμε την περίπτωση ύπαρξης λάθους στο κύκλωμά μας. Έχουμε FERROR, άρα ενεργοποιείται το σήμα του λάθους, ενώ το σήμα valid παραμένει 0, με αποτέλεσμα το Rx_DATA, δεν λαμβάνει δεδομένα, από το σύστημα, και παραμένει 0. Το σήμα FERROR, ενεργοποιείται στιγμιαία όπως φαίνεται πάνω.

ΜΕΡΟΣ Δ- Υλοποίηση UART Αποστολέα-Δέκτη για Σειριακή Μεταφορά Δεδομένων:

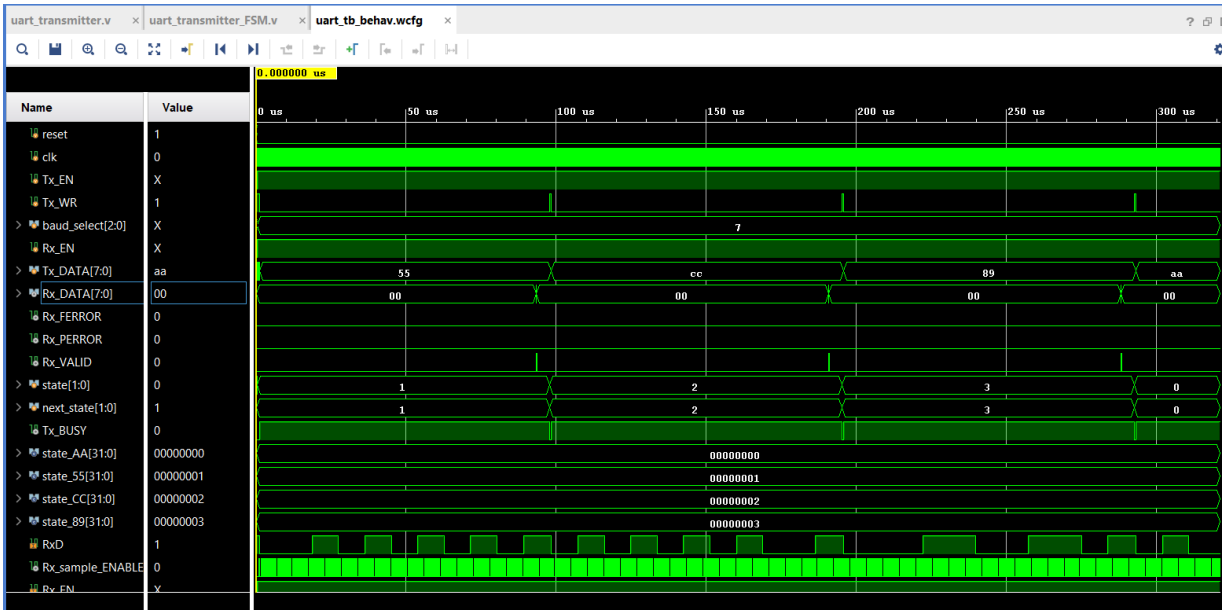
Υλοποίηση:

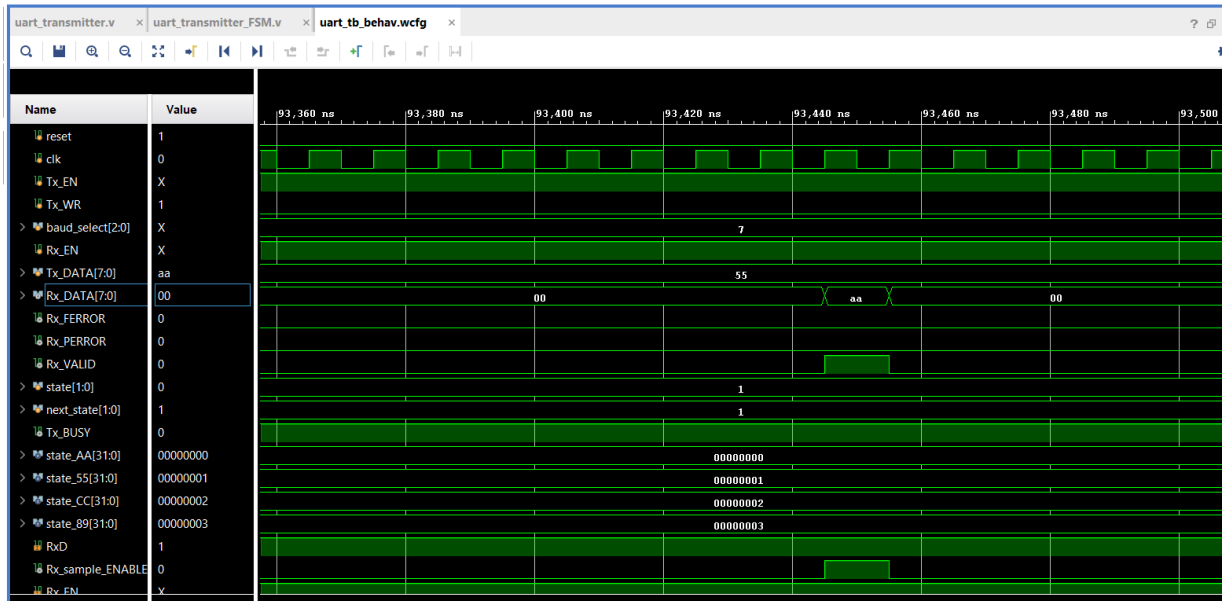


Επαλήθευση:

Έχοντας δημιουργήσει ένα πλήρες κανάλι UART, συνενώνουμε τις μονάδες του Αποστολέα και του Δέκτη, τις οποίες έχουμε περιγράψει στα μέρη Β και Γ. Δημιουργώ ένα testbench, το οποίο θα ελέγξει τις παρακάτω τέσσερις διαδοχικές λέξεις:

1) 10101010(AA)
2) 01010101(55)
3) 11001100(CC)
4) 10001001(89)





Όπως φαίνεται στις παραπάνω κυματομορφές, το Rx_DATA, παίρνει την τιμή του Tx_DATA, μόλις τελειώσει τον εσωτερικό έλεγχο του στο τέλος, για ένα κύκλο ρολογιού, οπότε η τιμή του Rx_DATA δεν είναι διακριτή Επίσης, φαίνεται καθυστερημένα το προς μετάδοση σύμβολο στην Rx_DATA, που είναι η έξοδος των συμβόλων μας. Η επικοινωνία είναι επιτυχής, δεν φαίνεται να δημιουργούνται λάθη στην επικοινωνία μεταξύ του πομπού – δέκτη.

Υλοποιήσαμε το testbench, με ένα FSM πολύ απλό που απλά αν το Tx_BUSY είναι 0, ενεργοποιεί το Tx_WR και σε πάει στην επόμενη κατάσταση, για κάθε κατάσταση συμβαίνει το ίδιο, μέχρι τη λήξη της προσομοίωσής μου.