ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ (ΗΥ430)

ΤΕΧΝΙΚΗ ΑΝΑΦΟΡΑ 2ΗΣ ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

ΠΑΝΑΓΙΩΤΗΣ ΑΝΑΣΤΑΣΙΑΔΗΣ 2134

ΤΙΤΛΟΣ: «ΥΛΟΠΟΙΗΣΗ U.A.R.T.»

Περίληψη τεχνικής αναφοράς

Η δεύτερη εργασία θέτει ως ζητούμενο την υλοποίηση ενός UART (Universal Asynchronous Receiver Transmitter). Η δομή της ξεκινά από την υλοποίηση του ελεγκτή του Baud Rate, προχωρά με την υλοποίηση της μονάδας του Transmitter και της αντίστοιχης του Receiver και καταλήγει με τη σύνδεση των δυο σε ένα ενιαίο σύστημα, στο οποίο οι δυο μονάδες επικοινωνούν μεταξύ τους. Η επικοινωνία αυτή βασίζεται στη σειριακή μεταφορά δεδομένων από τον αποστολέα στον δέκτη.

- Εισαγωγή στην 2η εργαστηριακή εργασία, αναλύοντας τα ζητούμενα και τους στόχους που απαιτεί να επιτευχθούν κατά τη δομή της τελικής υλοποίησης.
- *Μέρος Α*: Σχεδιασμός του ελεγκτή Baud Rate. Ο Baud rate Controller είναι η μονάδα της οποίας βασική λειτουργία αποτελεί ο ορισμός του κατάλληλου σήματος δειγματοληψίας ανάλογα με το επιλεγμένο Baud Rate.
- *Μέρος Β*: Σχεδιασμός του UART αποστολέα. Η υλοποίηση του αποστολέα δέχεται μια ακολουθία δεδομένων 8 bit και τα στέλνει μαζί με το start bit, το stop bit και το parity bit σειριακά στην έξοδο.
- *Μέρος Γ:* Υλοποίηση του UART δέκτη. Ο δέκτης σχεδιάζεται ώστε να δέχεται σειριακά bit προς bit δεδομένα, τα οποία κατόπιν τα δειγματοληπτεί με συχνότητα 16 φορές το συμφωνημένο baud rate (μεταξύ αυτού και του δέκτη). Μόλις λάβει ολόκληρη την ακολουθία δεδομένων τα αποθηκεύει σε καταχωρητές , ώστε αυτά να μπορούν να χρησιμοποιηθούν σε υψηλότερο επίπεδο.
- *Μέρος Δ:* Η κατασκευή συστήματος που συνδέει τις υλοποιημένες μονάδες του αποστολέα και του δέκτη μεταξύ τους. Το ζεύγος αυτό υλοποιείται κατά τρόπο ώστε ο αποστολέας δεχόμενος ακολουθία δεδομένων τη στέλνει στην έξοδο του. Η έξοδος αυτή (TxD) συνδέεται με την είσοδο του δέκτη (RxD) και κατά συνέπεια ο δέκτης δειγματοληπτεί την ακολουθία των δεδομένων εξάγοντας τα δεδομένα.
- *Συμπέρασμα:* Εξαγωγή συμπεράσματος για το σύνολο των τμημάτων της εργασίας, τις δυσκολίες που προέκυψαν κατά τον σχεδιασμό αλλά και το τελικό αποτέλεσμα.

1. Στόχοι της 2ης εργασίας

Ζητούμενο της 2ης εργαστηριακής εργασίας είναι η υλοποίηση ενός συστήματος σειριακής επικοινωνίας, το οποίο θα λειτουργεί με βάση το πρωτόκολλο UART.

Αντικείμενο της δεύτερης σε αριθμό εργασίας είναι η σταδιακή υλοποίηση του πρωτοκόλλου UART, όπου στα 4 μέρη που αυτή διακρίνεται κατασκευάζονται κατά σειρά ο Baud Rate ελεγκτής, ο UART Αποστολέας, ο UART δέκτης και τέλος το συνθέσιμο σύστημα ένωσης Αποστολέα-Δέκτη.

Για το πρώτο μέρος εργασίας ο Baud Rate ελεγκτής σχεδιάστηκε επιτυχώς με αποτέλεσμα να παράγει το κατάλληλο σήμα δειγματοληψίας με τη ζητούμενη συχνότητα 16 x Baud Rate.

Στο δεύτερο μέρος, κατασκευάστηκε ο αποστολέας UART, ο οποίος διαθέτει τα σήματα και τις προδιαγραφές λειτουργίας που ζητά η εκφώνηση της άσκησης και έχει τη δυνατότητα, επικοινωνώντας με το σύστημα και λαμβάνοντας ένα σύμβολο ως δεδομένο, να το στέλνει σειριακά στην έξοδο του.

Ο δέκτης UART αφορά την υλοποίηση που ζητά η εργασία για το τρίτο μέρος. Υλοποιήθηκε έτσι ώστε να ναι συμβατός με τον δέκτη και το πλήρες κανάλι UART που υλοποιείται στο μέρος Δ και ελέγχεται ως ξεχωριστή μονάδα για τις ανάγκες του μέρους Γ. Συγκεκριμένα οι βασικές λειτουργίες του αποστολέα υλοποιούνται στο πλαίσιο δοκιμής και με τον τρόπο αυτό πραγματοποιείται ο έλεγχος σωστής παραλαβής των δεδομένων κατά τη σειριακή μετάδοση.

Στο τέταρτο και τελευταίο μέρος της εργαστηριακής άσκησης κατασκευάζεται το πλήρες κανάλι UART μέσα από το οποίο ενώνονται οι μονάδες του αποστολέα και του δέκτη σε ένα συνθέσιμο σύστημα.

2. Μέρος Α – Υλοποίηση Ελεγκτή Baud Rate

> *Ο ελεγκτής εξάγει το κατάλληλο σήμα δειγματοληψίας ανάλογα τις σχετικές ταχύτητες επικοινωνίας του UART. Οι ταχύτητες αυτές ή αλλιώς Baud Rates είναι τα **300 bits/sec**, **1200 bits/sec**, **4800 bits/sec**, **9600 bits/sec**, **19200 bits/sec**, **38400 bits/sec**, **57600 bits/sec** και τέλος τα **115200 bits/sec**.*

Η υλοποίηση του ελεγκτή Baud Rate σχεδιάζεται με τέτοιο τρόπο ώστε να δουλεύει κάτω από το ρολόι της FPGA SPARTAN 3 πλακέτας, δηλαδή τα 50 MHz. Να σημειωθεί ότι το σήμα που παράγεται από τη μονάδα ισοδυναμεί με τη συχνότητα την οποία θα πρέπει να δουλεύει ο δέκτης και **όχι** ο αποστολέας, συγκεκριμένα τη ταχύτητα 16 x Baud Rate. Η συχνότητα αυτή αποτελεί τη ταχύτητα με την οποία δειγματοληπτεί ο δέκτης τα δεδομένα που δέχεται, ενώ ο αποστολέας ο οποίος πρέπει να λειτουργεί με συχνότητα Baud Rate και όχι 16 x Baud Rate, διαχειρίζεται τη μονάδα του ελεγκτή μέσα από τη δική του υλοποίηση αναλόγως.

Υλοποίηση:

Για την υλοποίηση της μονάδας του ελεγκτή χρησιμοποιείται ένας μετρητής, ο οποίος μετρά τους κύκλους ρολογιού των 50 MHz που απαιτούνται, ώστε το σήμα δειγματοληψίας να οδηγείται στο 1 με συχνότητα 16 x Baud Rate.

Οι απαιτούμενοι αυτοί κύκλοι ρολογιού υπολογίζονται διαιρώντας τα 50 MHz με το πόσο 16 x Baud Rate που κατά συνέπεια έχει ως αποτέλεσμα έναν αριθμό με δεκαδικά ψηφία. Επειδή ο μετρητής είναι συγχρονισμένος με το ρολόι και δεν είναι εφικτό μέσω της Verilog να φτάνει σε μια μέγιστη τιμή, ακριβώς ίση με το πηλίκο της παραπάνω διαίρεσης, η τιμή αυτή στρογγυλοποιείται στη κοντινότερη ακέραια τιμή.

Η ενέργεια αυτή επιφέρει ένα ελάχιστο σφάλμα στη διαδικασία, επομένως πιο κάτω παρατίθενται τα ποσοστά σφαλμάτων εξαιτίας της διαίρεσης για κάθε επιλεγμένο Baud Rate.

```
always @(posedge clk or posedge reset) begin
    if (reset) begin
        counter = 0;
        sample_ENABLE = 0;
        stored_baud = 3'b000;
    end
    else begin

        counter = counter + 1;

        //if during process baud select is changed
        //make counter zero in order to start from scratch again.


        if ( stored_baud != baud_select ) begin
            stored_baud = baud_select;
            counter = 0;
        end
        //case for each of the eight (16 x)baud rates.
        //For each baud rate if counter reaches max value drive sample signal to one

        case( stored_baud )

            3'b000:
            begin
            if ( counter == 14'd10417 )
                begin
                    counter = 0;
                    sample_ENABLE = 1'b1;
                end
                else if ( sample_ENABLE == 1'b1 )
                begin
                    sample_ENABLE = 1'b0;
                end
            end
            3'b001:
            begin

                if ( counter == 14'd2604 )
                begin
                    counter = 0;
                    sample_ENABLE = 1'b1;
                end
                else if ( sample_ENABLE == 1'b1 )
                begin
                    sample_ENABLE = 1'b0;
                end
            end
            3'b010:
            begin

                if ( counter == 14'd651 )
                begin
                    counter = 0;
                    sample_ENABLE = 1'b1;
                end
                else if ( sample_ENABLE == 1'b1 )
                begin
                    sample_ENABLE = 1'b0;
                end
            end
```

```
//if during process baud select is changed
//make counter zero in order to start from scratch again.


if ( stored_baud != baud_select ) begin
    stored_baud = baud_select;
    counter = 0;
end
```

Στα τμήματα κώδικα που δίνονται παραπάνω, στα αριστερά φαίνεται η διαχείριση του μετρητή και η ανάθεση μέγιστης τιμής για κάθε μία από τις περιπτώσεις 000, 001 και 010. Για τις 5 υπόλοιπες κωδικοποιήσεις ακολουθείται η ίδια διαδικασία. Στα δεξιά φαίνεται η if συνθήκη που τίθεται να μηδενίζει το counter στον επόμενο κύκλο ρολογιού κάθε φορά που η είσοδος του baud select αλλάζει εν μέσω λειτουργίας του ελεγκτή.

-Να σημειωθεί πως η συγκεκριμένη υλοποίηση μηχανής πεπερασμένων καταστάσεων της μονάδας προηγήθηκε της διάλεξης του μαθήματος για τις FSM με συνέπεια να μην ακολουθεί τον τρόπο λειτουργίας που συνίσταται για τις μηχανές αυτές.-

Σφάλματα:

Όπως εξηγήθηκε παραπάνω, λόγω της διαίρεσης 50 MHz / (16 x Baud Rate), ο αριθμός των κύκλων ρολογιού που προκύπτει ως πηλίκο δεν είναι ακέραιος, με συνέπεια να πρέπει να στρογγυλοποιηθεί στην πλησιέστερη τιμή.

Έτσι για κάθε Baud Rate, μεταξύ των ιδανικών και πραγματικών τιμών -οι πραγματικές τιμές προκύπτουν βάσει των κυματομορφών κατά την επαλήθευση και μετατρέποντας τα nanoseconds της περιόδου στα αντίστοιχα Hz- υπολογίζουμε τα παρακάτω ποσοστά σφαλμάτων.

Calculate Error	***300 bits/sec***	***1200 bits/sec***	***4800 bits/sec***	***9600 bits/sec***	***19200 bits/sec***	***38400 bits/sec***	***57600 bits/sec***	***115200 bit/sec***
16 x Baud Rate (Ideal Value) (Hz)	4800	19200	76800	153600	307200	614400	921600	1843200
16 x Baud Rate (Real Value) (Hz)	4799.8464	19201.2289	76804.9155	153374.233	306748.466	617283. 951	925925.926	1851851.85
Error (%)	***0,0032***	***0.0064***	***0.0063***	***0.1469***	***0.1469***	***0.4672***	***0.4672***	***0.4671***

Επαλήθευση:

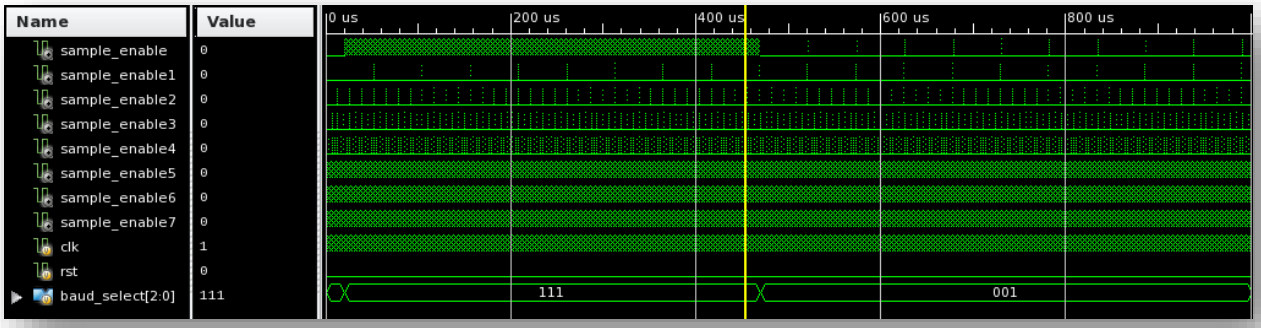
Για την επαλήθευση του Baud Rate ελεγκτή, σχεδιάστηκε ένα testbench, το οποίο παρέχει στη μονάδα κάθε πιθανό διάνυσμα εισόδου, συγκεκριμένα κάθε πιθανή τιμή Baud Rate από το πίνακα των σχετικών κωδικοποιήσεων. Στο κώδικα της Verilog πραγματοποιείται instantiate της μονάδας του ελεγκτή 8 φορές, μία για κάθε διαφορετικό Baud Rate. Αυτό γίνεται για το λόγο ότι είναι εφικτό να φανούν όλες οι διαφορετικές περίοδοι ταυτόχρονα στις κυματομορφές και κατά συνέπεια να μπορεί να γίνει η κατάλληλη σύγκριση μεταξύ τους.

Παρόλα αυτά, για να ελεγχτεί η συμπεριφορά της μονάδας και στη περίπτωση που αλλάξουμε τη σχετική κωδικοποίηση κατά τη διάρκεια λειτουργίας της, διαφοροποιούμε 2 φορές τη κωδικοποίηση για το πρώτο Instantiate του ελεγκτή μετά από το πέρας αρκετών nanoseconds.

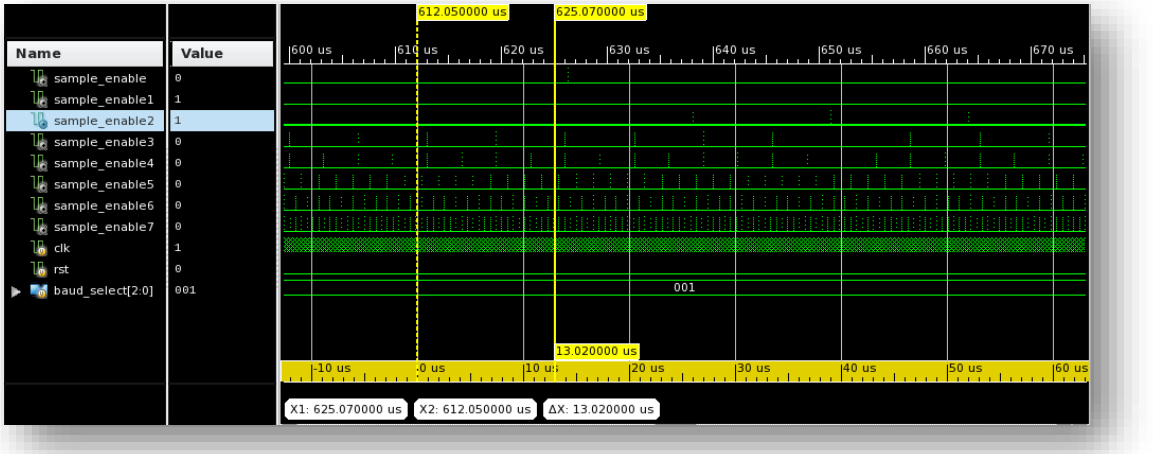
Τα παραπάνω μπορούν να γίνουν αντιληπτά από τον κώδικα της Verilog για το testbench του ελεγκτή.

```
module baud_controller_tb ();

reg clk = 0;
reg rst = 1;
reg [2:0] baud_select = 3'b000;

//Initialize module
//Set baud select to new value for baud_controller_0 after 450000ns,
//in order to test the functionality of the ciruit after the change.

initial begin
    #74 rst=0;
    #20000 baud_select = 3'b111;
    #450000 baud_select = 3'b001;
end

//Fpga spartan 3 clock frequency ==> 50 Mhz

always #10 clk <= ~clk;

wire sample_enable;

//Instantiate baud controller with all the possible values for baud select

baud_controller baud_controller_0(rst, clk , baud_select, sample_enable );
baud_controller baud_controller_1(rst, clk , 3'b001, sample_enable1);
baud_controller baud_controller_2(rst, clk , 3'b010, sample_enable2);
baud_controller baud_controller_3(rst, clk , 3'b011, sample_enable3);
baud_controller baud_controller_4(rst, clk , 3'b100, sample_enable4);
baud_controller baud_controller_5(rst, clk , 3'b101, sample_enable5);
baud_controller baud_controller_6(rst, clk , 3'b110, sample_enable6);
baud_controller baud_controller_7(rst, clk , 3'b111, sample_enable7);

endmodule
```

Οι κυματομορφές που προκύπτουν από την προσομοίωση του κυκλώματος μετά και το **Place and Route,** επαληθεύουν την ορθή λειτουργία του ελεγκτή για όλες τις τιμές:



Διαισθητικά μπορεί να φανεί πως για κάθε instantiation του ελεγκτή με διαφορετική κωδικοποίηση, το σήμα δειγματοληψίας οδηγείται σε κάθε περίπτωση ορθώς. Επιπροσθέτως, στο sample_enable σήμα στη πρώτη σειρά, διακρίνεται η περίπτωση που η κωδικοποίηση αλλάζει 2 φορές τυχαία εν μέσω της διαδικασίας, από την 000 στην 111 και τέλος στην 001 κωδικοποίηση.



Στη δεύτερη κυματομορφή που η μεγέθυνση είναι μεγάλη, τονίζεται η περίοδος 16 x Baud Rate για τη κωδικοποίηση 010, όπου το Baud Rate είναι ίσο με 4800 bits/sec. Η περίοδος είναι ίση με 13,02 us που αντιστοιχούν σε 76804,9155 Hz, όπως επισημάνθηκε και στο πίνακα των σφαλμάτων παραπάνω.

Κάλυψη λειτουργίας: Το πλαίσιο ελέγχου ελέγχει όλες τις δυνατές κωδικοποιήσεις – δηλαδή όλα τα αντίστοιχα Baud Rates που παραμετροποιούν τον Baud Controller. Ελέγχει επίσης, την ειδική περίπτωση που η κωδικοποίηση αλλάξει κατά τη διάρκεια της λειτουργίας, δηλαδή μια στιγμή που ο μετρητής αυξάνεται για τη προηγούμενη επιλογή στη κωδικοποίηση. Συνεπώς, το πλαίσιο ελέγχου εισάγει κάθε πιθανό διανύσματα ελέγχου και κάθε δυνατή περίπτωση στο κύκλωμα και έτσι η κάλυψη φτάνει στο μέγιστο.

3. Μέρος Β – Υλοποίηση UART Αποστολέα

> *Ο αποστολέας είναι η μονάδα που λαμβάνει το σύμβολο προς μεταφορά και το στέλνει σειριακά στην έξοδο 1-bit με συχνότητα **Baud Rate**.*

Ο αποστολέας διαθέτει λειτουργία ενεργοποίησης/απενεργοποίησης που σημαίνει ότι ένα σήμα στην είσοδο του (**Tx_EN**) τον ενεργοποιεί ή τον απενεργοποιεί. Εφόσον βρίσκεται σε ενεργή κατάσταση, είτε είναι στάσιμος σε αναμονή του επόμενου συμβόλου προς μεταφορά είτε είναι απασχολημένος μεταφέροντας τα δοσμένα δεδομένα στην έξοδο με συχνότητα Baud Rate. Το σήμα που επιδεικνύει στον αποστολέα να μεταδώσει το εισερχόμενο σε αυτόν σύμβολο, είναι το **Tx_WR** και μπορεί να σταλθεί μόνο σε περίπτωση που είναι μη απασχολημένος. Τέλος ένα σήμα **Tx_BUSY** στην έξοδο της μονάδας, καταδεικνύει στο υψηλότερο επίπεδο που τη διαχειρίζεται, αν υφίσταται μεταφορά δεδομένων τη δεδομένη στιγμή.

Υλοποίηση:

Το σύστημα του UART αποστολέα αποτελείται από ένα σύνολο τεσσάρων μονάδων.

- Η μονάδα του Baud Ελεγκτή έτσι όπως παρουσιάστηκε παραπάνω.
- Ο μετρητής Baud Counter που ουσιαστικά μετατρέπει τη συχνότητα 16 x Baud Rate σε συχνότητα Baud Rate, βάσει της οποίας οφείλει να στέλνει ο αποστολέας σειριακά τα δεδομένα.
- Μια απλή μονάδα συγχρονιστή του ασύγχρονου σήματος reset, της οποίας η δομή αποτελείται από 2 Flip-Flops.
- Η βασική μονάδα του αποστολέα που αναλαμβάνει τη μετάδοση των δεδομένων, με τον τρόπο που επισημάνθηκε.

UART Transmitter Module:

```
//// States of transmitter ////

always @(posedge clk or posedge reset)
  if ( reset )
    state <= IDLE;
  else if (Tx_EN == 1'b0)
    state <= IDLE;
  else
      state <= next_state;
```

```
always @(*) begin

  //Default values
  //Stay in the same state by default

  next_state = state;
  load_shifter = 0;
  counter_ENABLE = 0;

  case (state)

    //Idle state
    //Remain in this state until Tx_WR is 1

    IDLE: begin
      Tx_BUSY = 0;
      if (Tx_WR == 1'b1 && Tx_EN == 1'b1)
        next_state = START;
    end

    //1 cycle long
    //turn on the baud counter and load the shift register

    START: begin
      load_shifter = 1;
      counter_ENABLE = 1;
      Tx_BUSY = 1;
      next_state = TRANSMISSION;
    end

    //Stay here until all the bits have been sent

    TRANSMISSION: begin
      counter_ENABLE = 1;
      Tx_BUSY = 1;
      if (bit_counter == 11)
        next_state = IDLE;
    end

    default:
      Tx_BUSY = 1;

  endcase

end
```

Όπως φαίνεται παραπάνω από τις εικόνες, στη βασική μονάδα του UART transmitter εμπεριέχεται μια μηχανή πεπερασμένων καταστάσεων που ορίζει τις καταστάσεις λειτουργίας του αποστολέα.

- **IDLE State:** Στην κατάσταση αυτή βρίσκεται συνεχώς η μονάδα του αποστολέα εφόσον αρχικοποιείται η μονάδα ή μένει απενεργοποιημένη. Εφόσον ο αποστολέας είναι ενεργός και του σταλθεί σήμα Tx_WR == 1, τότε μεταβαίνει στην επόμενη κατάσταση, την κατάσταση φόρτωσης δεδομένων προς αποστολή.
- **START State:** Στην κατάσταση START ξεκινά η φόρτωση του συμβόλου που θα μεταδοθεί. Συγκεκριμένα στέλνεται το σήμα εκκίνησης του μετρητή που ορίζει τη συχνότητα Baud Rate και αποθηκεύονται τα δεδομένα σε ένα shift register (θα εξηγηθεί παρακάτω) μαζί με το parity bit που υπολογίζεται μέσω bitwise XOR στα δεδομένα και το start bit που είναι 0. Τέλος το Tx_BUSY σήμα γίνεται 1 δηλώνοντας πως ο αποστολέας είναι πια απασχολημένος με τη διαδικασία μεταφοράς.
- **TRANSMISSION State:** Στην κατάσταση TRANSMISSION μένει ο αποστολέας μέχρι να σταλθούν όλα τα δεδομένα σειριακά, δηλαδή και τα 11 bits (start bit + 8 data bits + parity bit + stop bit). Μόλις αυτό πραγματοποιηθεί, δίνεται εντολή και ο αποστολέας μεταβαίνει στην κατάσταση IDLE.

Η μεταφορά των δεδομένων υλοποιείται με τα τμήματα Verilog που φαίνονται παρακάτω και τα αντίστοιχα always blocks.

```
//If Tx_WR signal is enabled store value of Tx_Data to register.
//Otherwise set register to value 8'b1111111.

always @(posedge clk or posedge reset) begin
    if (reset) begin
        stored_data = 8'b11111111;
    end
    else if ( Tx_WR == 1'b1 && state == IDLE && Tx_EN == 1'b1 ) begin
        stored_data = Tx_DATA;
    end
  else if ( Tx_EN == 0 ) begin
    stored_data = 8'b11111111;
  end
end
```

```
//1.If transmitter is initialized in START STATE (load_shifter == 1) store to shifter 8-bit data + start bit + stop bit + parity bit.
//2.Every baud_tx_clk a new bit must be send. Increment the bit counter and set shifter[0] with the next value.

always @(posedge clk or posedge reset) begin
    if ( reset ) begin
        shifter = 10'b1_1111_11111;
        bit_counter = 0;
    end
  else if (Tx_EN == 1'b0) begin

    shifter = 10'b1_1111_11111;
    bit_counter = 0;

  end
    else if ( load_shifter == 1 ) begin
        shifter = { (^stored_data), stored_data, 1'b0 };
        bit_counter = 0;
    end
    else if ( load_shifter == 1'b0 && baud_tx_clk == 1'b1) begin
        shifter = {1'b1, shifter[9:1]};
        bit_counter = bit_counter + 1;
    end
end
```

```
//Every posedge clock send to output TxD the value of shifter[0]

always @(posedge clk) begin
    TxD = shifter[0];
end
```

1. Στην πρώτη σε σειρά εικόνα επισημαίνεται το always block που είναι υπεύθυνο για την αποθήκευση των δεδομένων προς αποστολή, από την είσοδο σε ένα register αντίστοιχων bits. Αυτό συμβαίνει μόνο στον κύκλο που ο αποστολέας είναι ενεργοποιημένος, βρίσκεται στη κατάσταση αναμονής IDLE και το σήμα εγγραφής Tx_WR οδηγείται στο 1.
2. Σο δεύτερο τμήμα κώδικα που παρουσιάζεται, τονίζεται η διαδικασία φόρτωσης και διαχείρισης του shift register. Στη διαδικασία φόρτωσης δεδομένων, αποθηκεύεται στον shifter η ακολουθία 8 bits μαζί με το start bit και το parity bit. Στη φάση που αποστολέας στέλνει δεδομένα, αυξάνεται ένας bit counter δηλώνοντας τον αριθμό των bits που έχουν σταλθεί, ενώ στον shifter καταχωρείται στο LSB το επόμενο bit προς μεταφορά και στο MSB η τιμή 1. Η τελευταία ενέργεια γίνεται, ώστε το τελευταίο bit προς μεταφορά να είναι το stop bit το οποίο πρέπει να βρίσκεται στο 1.
3. Στη τελευταία εικόνα και όπως εξηγήθηκε παραπάνω, μεταφέρεται στην έξοδο του αποστολέα σε κάθε ακμή του ρολογιού το LSB του shifter.

Baud Counter Module:

```
//baud counter for transmitter

always @( posedge reset or posedge clk ) begin
    if (reset) begin
        baud_counter = 4'b0000;
        baud_tx_clk = 1'b0;
    end

    //after 16 baud signals send signal for transmitter

    else if (counter_ENABLE == 1 && baud_counter == 4'b1111 && Tx_sample_ENABLE == 1'b1) begin
        baud_counter = 4'b0000;
        baud_tx_clk = 1'b1;
    end
    else if ( counter_ENABLE == 1 && Tx_sample_ENABLE == 1'b1) begin
        baud_counter = baud_counter + 1;
        baud_tx_clk = 1'b0;
    end
    else begin
        baud_tx_clk = 1'b0;
    end
end
```

Ο Baud Counter εφόσον ενεργοποιηθεί, μετρά 16 σήματα δειγματοληψίας του Baud Controller. Μόλις αυτό επιτευχθεί στέλνει στην έξοδο ένα σήμα οδηγημένο στο 1. Το σήμα αυτό επιδεικνύει στη μονάδα του αποστολέα πότε να στείλει το επόμενο bit. Με αυτόν το τρόπο επιτυγχάνεται η μετατροπή της συχνότητας 16 x Baud Rate σε συχνότητα τιμής Baud Rate.

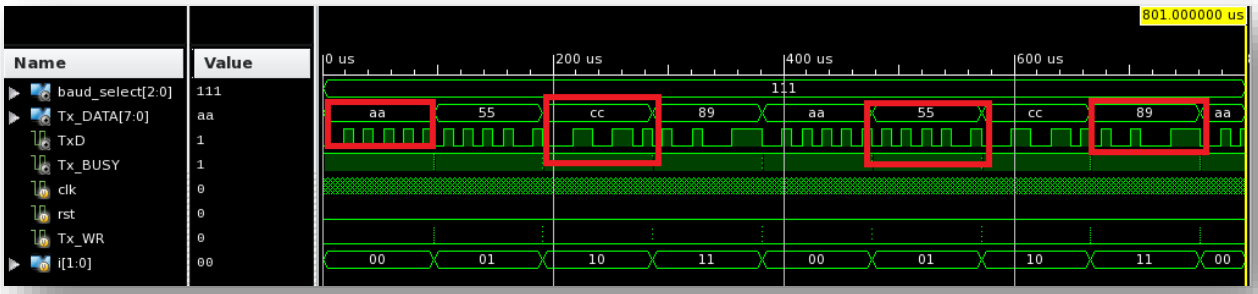
Επαλήθευση:

Στο πλαίσιο ελέγχου του transmitter, δοκιμάζεται η μεταφορά τεσσάρων ακολουθιών στη μονάδα οι οποίες στέλνονται η μια μετά την άλλη συνεχόμενα. Οι ακολουθίες που χρησιμοποιούνται είναι αυτές που προτείνονται από την εκφώνηση, δηλαδή σε δυαδικό σύστημα οι 10101010, 01010101, 11001100 και 10001001 ή αλλιώς σε δεκαεξαδικό σύστημα οι aa , 55, cc, 89 αντιστοίχως.

Ως στρατηγική επαλήθευσης του κυκλώματος, το πλαίσιο ελέγχου επιλέγει μια συχνότητα λειτουργίας baud rate και συγκεκριμένα τα 115200 bits/sec (κωδικοποίηση 111). Στη συνέχεια επιλέγεται για ρολόι το ρολόι της πλακέτας (50 MHz) και αρχικοποιείται το κύκλωμα με το σήμα reset σε τυχαία επιλεγμένη στιγμή, στα 74 ns. Το πλαίσιο ελέγχου κρατά το κύκλωμα του αποστολέα διαρκώς ενεργοποιημένο (Tx_EN = 1), ενώ το πρώτο σήμα εγγραφής Tx_WR στέλνεται εξίσου τυχαία, στα 1500 ns. Μετά από αυτό, κάθε επόμενο σήμα εγγραφής στέλνεται έναν κύκλο μετά που το σήμα Tx_BUSY του αποστολέα οδηγηθεί στο 0. Το βασικό τμήμα κώδικα Verilog του testbench παρουσιάζεται παρακάτω.

```
//Set value of baud select to 115200 bits/sec

assign baud_select = 3'b111;

//Initialize transmission by setting Tx_WR to value 1 after 1500 ns

initial begin
    #74 rst=0;
    #1500 Tx_WR = 1;
end

//Fpga spartan 3 clock frequency ==> 50 Mhz

always #10 clk <= ~clk;

always @(posedge clk or posedge rst) begin
    if (rst) begin

        // reset

        Tx_WR = 0;
        i = 0;
        sequence[0] = 8'hAA;
        sequence[1] = 8'h55;
        sequence[2] = 8'hCC;
        sequence[3] = 8'h89;
        first = 2'b10;
    end
    else if (Tx_WR) begin
        Tx_WR = 0;
    end
    else if (Tx_BUSY == 0 && i == 2'b11) begin
        i = 0;
        Tx_WR = 1;
    end
    else if (Tx_BUSY == 1) begin
        first = 2'b01;
    end
    else if (Tx_BUSY == 0 && first == 2'b00) begin
        Tx_WR = 1;
        i = i + 1;
    end
    else if (Tx_BUSY == 0 && first == 2'b01) begin
        Tx_WR = 1;
        i = i + 1;
        first = 2'b00;
    end
end

//assign sequence value to Tx_Data
//At every Tx_WR == 1 assign continuously 4 values of sequence register to Tx_Data

assign Tx_DATA = sequence[i];


 uart_transmitter uart_transmitter_0(.rst (rst), .clk(clk), .Tx_DATA (Tx_DATA), .baud_select(baud_select), .Tx_WR(Tx_WR), .Tx_EN(1'b1), .TxD(TxD), .Tx_BUSY(Tx_BUSY));
```

Η εκτέλεση της προσομοίωσης συμπεριφοράς μετά και το **Place and Route** του μέρους Β της εργασίας, εξάγει τις παρακάτω κυματομρφές, κυματομορφές οι οποίες αποδεικνύουν πως τα σύμβολα που δίνονται στον αποστολέα για μεταφορά στέλνονται σωστά στην έξοδο του. Στα κόκκινα πλαίσια φαίνεται το σήμα εξόδου Tx_D να στέλνει σωστά την ακολουθία μαζί με το start bit, το parity bit και το stop bit.



Κάλυψη λειτουργίας: Η λειτουργία του κυκλώματος πραγματοποιείται με διανύσματα εισόδου όλες τις ακολουθίες-σύμβολα που προτείνει η εκφώνηση της εργαστηριακής άσκησης. Η κάλυψη λειτουργίας πλησιάζει το μέγιστο καθώς πέρα των τεσσάρων συμβόλων, διαχειρίζεται τόσο το σήμα Tx_WR για την εγγραφή των δεδομένων όσο και το σήμα Tx_BUSY που υποδηλώνει στο πλαίσιο ελέγχου πότε να στείλει τα επόμενα bits. Ιδανικά και για κάλυψη στο 100%, θα μπορούσε να οδηγηθεί το σήμα ενεργοποίησης Tx_EN κάποια στιγμή στο 0, κατά τη διάρκεια λειτουργίας του αποστολέα, εφόσον όμως το Tx_BUSY βρίσκεται στο 0, όπως απαιτεί η εκφώνηση.

4. Μέρος Γ – Υλοποίηση UART Δέκτη

> *Ο δέκτης είναι το σύστημα το οποίο λαμβάνει σειριακά τα δεδομένα που στέλνει ο δέκτης του UART και τα δειγματοληπτεί με συχνότητα 16 φορές το συμφωνημένο Baud Rate.*

Ο δέκτης έχει παρόμοια υλοποίηση με τον αποστολέα που εξηγήθηκε παραπάνω. Ομοίως με τον αποστολέα διαθέτει λειτουργία απενεργοποίησης/ενεργοποίησης (είσοδος **Rx_EN**). Εφόσον λάβει στην είσοδο το start bit ξεκινά για αυτόν η διαδικασία δειγματοληψίας των εισερχόμενων bits με τη συχνότητα που παράγει ο Baud Controller, δηλαδή 16 x Baud Rate. Όταν δειγματοληπτήσει και τα 11 bits, πραγματοποιεί ελέγχους εγκυρότητας που αν και εφόσον περαστούν επιτυχώς, στέλνει στην έξοδο του το σύμβολο που παρέλαβε (**Rx_DATA**) και οδηγεί για ένα κύκλο ρολογιού ένα σήμα **Rx_VALID**, υποδηλώνοντας ότι τη χρονική στιγμή αυτή τα δεδομένα είναι έγκυρα. Οι έλεγχοι που πραγματοποιούνται στα δεδομένα είναι 2 ειδών. Το πρώτο είδος αφορά τον έλεγχο στο start bit και stop bit, που σε περίπτωση λάθους σε αυτά, στέλνεται για ένα κύκλο σήμα σφάλματος frame error (**Rx_FERROR).** Ο δεύτερος και τελευταίος έλεγχος αφορά το parity bit. Ο δέκτης από την ακολουθία των 8-bit δεδομένων που έλαβε, υπολογίζει το parity bit και εφόσον υπάρχει ασυμφωνία μεταξύ του parity bit που υπολόγισε και του parity bit που έλαβε από τον αποστολέα, στέλνει σήμα σφάλματος parity error (**Rx_PERROR**).

Υλοποίηση:

Το σύστημα του UART αποστολέα αποτελείται από ένα σύνολο τεσσάρων μονάδων.

- Η μονάδα του Baud Ελεγκτή έτσι όπως παρουσιάστηκε στο μέρος Α.
- Ο μετρητής Baud Counter Rx που αναλαμβάνει να στείλει σήμα αποθήκευσης του εισερχόμενου bit ακριβώς στο κέντρο του μεταδιδόμενου ψηφίου. Αυτό επιτυγχάνεται μετρώντας 8 κύκλους στο start bit και ακριβώς 16 για κάθε επόμενο bit.
- Η μονάδα συγχρονιστή του ασύγχρονου σήματος reset, της οποίας η δομή αποτελείται από 2 Flip-Flops.
- Η βασική μονάδα του δέκτη που αναλαμβάνει τη λήψη των δεδομένων από τον αποστολέα, με τον τρόπο που επισημάνθηκε.

UART Receiver Module:

```
//Transition between states

always @(posedge clk or posedge reset)
  if ( reset )
    state <= IDLE;
  else if ( Rx_EN == 1'b0 )
    state <= IDLE;
  else
    state <= next_state;

//if we are at bit zero we are going to sample startbit.
//Make STARTBIT signal 1 for baud_counter_rx

assign STARTBIT = (bit_counter == 0) ? 1:0 ;
```

```
case(state)

  //Idle state
  //Remain in this state until a start bit is received in Tx_WR

  IDLE: begin
    clear_counter = 1;
    Rx_VALID = 0;
    if ( stored_bit == 0 && Rx_EN == 1'b1 ) begin
      next_state = RECV;
      counter_ENABLE = 1;
    end

  end

  //Receiving state
  //Enable counter and wait for the serial package to be received

  RECV: begin
    counter_ENABLE = 1;
    Rx_VALID = 0;
    if (bit_counter == 4'd11) begin
      next_state = LOAD;
    end
  end

  //Change value of load to 1. Check if there are errors otherwise store the data to Rx_DATA.

  LOAD: begin
    load = 1;
    Rx_VALID = 0;
    next_state = DAV;
  end

  //No error -> Data Available (1 cycle).
  //Error    -> None of the data are stored, go to state idle.

  DAV: begin
      if ( error == 0) begin
          Rx_VALID = 1;
          next_state = IDLE;
      end
    else begin
          next_state = IDLE;
        Rx_VALID = 1'b0;
    end
  end

  default:
    Rx_VALID = 0;
endcase
```

Όπως φαίνεται και στα παραπάνω τμήματα κώδικα, η μονάδα του UART Receiver έχει ομοίως με τον αποστολέα, δική της μηχανή πεπερασμένων καταστάσεων λειτουργίας.

- **IDLE State:** Στην κατάσταση αυτή βρίσκεται συνεχώς η μονάδα του δέκτη εφόσον αρχικοποιείται η μονάδα ή μένει απενεργοποιημένη. Δεδομένου ότι ο δέκτης είναι ενεργός, εφόσον στην σειριακή 1-bit είσοδο του σταλθεί το Start Bit, ενεργοποιεί την διαδικασία δειγματοληψίας και μεταβαίνει στην επόμενη κατάσταση.
- **RECV State:** Στην κατάσταση RECV παραμένει ο αποστολέας μέχρι να δειγματοληπτήσει 11 bits.
- **LOAD State:** Η κατάσταση LOAD έπεται της κατάστασης RECV. Αφού έχουν ληφθεί τα 11 bits, η κατάσταση LOAD σηματοδοτεί την εκκίνηση των ελέγχων εγκυρότητας των δεδομένων και την αποθήκευση της ακολουθίας των 8-bit δεδομένων σε register, αν και μόνο αν οι έλεγχοι ήταν επιτυχείς. Σε άλλη περίπτωση οδηγείται ένα από τα σήματα λάθους parity error η frame error.
- **DAV State:** Στη κατάσταση DAV (μετά το πέρας της LOAD κατάστασης) και στη περίπτωση που τα δεδομένα είναι έγκυρα και αποθηκεύτηκαν επιτυχώς, οδηγείται για ένα κύκλο ρολογιού το Rx_VALID στην τιμή 1, δείχνοντας με αυτόν το τρόπο την εγκυρότητα των δεδομένων. Μετά την DAV το κύκλωμα ξανά-επιστρέφει στη κατάσταση IDLE.

Τα τμήματα κώδικα που παρουσιάζονται παρακάτω επιδεικνύουν το τρόπο με τον οποίο ο δέκτης δειγματοληπτεί τα δεδομένα που λαμβάνει με τρόπο σειριακό και τον αντίστοιχο τρόπο με τον οποίο τα εξάγει στην έξοδο του, εφόσον αυτά είναι έγκυρα.

```
//After every sampling of incoming bit, store bit to [10:0] register raw_data along with parity bit.
//Store every new bit at MSB position of raw_data register

always @( posedge clk or posedge reset ) begin
    if ( reset ) begin
        raw_data = 11'd0;
        parity_check = 0;
    end
    else if (Rx_EN == 1'b0) begin
        raw_data = 11'd0;
        parity_check = 0;
    end
    else if ( sample_now == 1  ) begin
        raw_data = {stored_bit, raw_data[10:1]};
        parity_check = ^(raw_data[8:1]);
    end
end
```

```
//Data register. Store the character received
//1.When all bits are sent and load signal is 1, store data to Rx_DATA
//2.If stop bit and start bit are wrong send frame error and do not store anything
//3.Calcuate parity bit of receiving data.
//  if calculated parity bit and received parity bit do not match send parity error.

always @(posedge clk or posedge reset)
  if ( reset ) begin
    Rx_DATA = 0;
    error = 0;
    Rx_PERROR = 1'b0;
    Rx_FERROR = 1'b0;
  end
  else if (Rx_EN == 0) begin
    Rx_DATA = 0;
    error = 0;
    Rx_PERROR = 1'b0;
    Rx_FERROR = 1'b0;
  end
  else if (load && parity_check != raw_data[9]) begin
    Rx_PERROR = 1'b1;
    error = 1;
  end
  else if (load && ((raw_data[10] == 0) || (raw_data[0] == 1))) begin
    Rx_FERROR = 1'b1;
    error = 1;
  end
  else if ( Rx_FERROR == 1'b1 || Rx_PERROR == 1'b1 ) begin
    Rx_PERROR = 1'b0;
    Rx_FERROR = 1'b0;
  end
  else if ( load ) begin
    error = 0;
    Rx_DATA = raw_data[8:1];
  end
```

1. Στην πρώτο τμήμα κώδικα φαίνεται η διαδικασία με την οποία σταδιακά εξάγει ο δέκτης τα δεδομένα που δέχεται σειριακά. Το sample_now είναι το σήμα που καταδεικνύει τη δειγματοληψία στο κέντρο του ψηφίου είτε αυτό είναι το start bit είτε κάποιο άλλο. Το sample_now εξάγεται από τη μονάδα του baud_counter_rx που θα εξηγηθεί παρακάτω. Στη συνέχεια, κάθε νέο bit που δειγματοληπτείται, αποθηκεύεται στο MSB του register raw_data με όλα τα προηγούμενα να κάνουν ολίσθηση προς τα δεξιά. Τέλος, σε ένα νέο καταχωρητή parity_check υπολογίζεται το parity bit των δεδομένων που λαμβάνει ο δέκτης.
2. Σο δεύτερο κατά σειρά τμήμα κώδικα, τονίζεται η διαδικασία με την οποία γίνονται έλεγχοι εγκυρότητας στα δεδομένα, εφόσον έχει ολοκληρωθεί η εξαγωγή τους στον καταχωρητή raw_data. Σε πρώτο στάδιο ελέγχεται η ύπαρξη ασυμφωνίας στο parity bit που έχει λάβει ο δέκτης από τον αποστολέα και στο parity bit που έχει υπολογίσει ο ίδιος. Σε περίπτωση διαφοράς στα 2 bits στέλνεται σήμα Rx_PERROR = 1 και τα δεδομένα δεν αποθηκεύονται στην έξοδο του δέκτη. Αν ο έλεγχος είναι επιτυχής, ελέγχονται εν συνέχεια λάθη στο start bit και στο stop bit. Εφόσον υπάρχουν, οδηγείται στο ένα το σήμα Rx_FERROR χωρίς και πάλι να πραγματοποιηθεί αποθήκευση δεδομένων στην έξοδο. Τελικά, αν και οι 2 έλεγχοι δεν αναδείξουν σφάλματα, η ακολουθία δεδομένων (χωρίς το start bit, το stop bit και το parity bit) στέλνεται στην έξοδο του δέκτη.

Baud Counter Module (Δέκτη):

```
//baud counter for receiver

always @( posedge reset or posedge clk ) begin
    if (reset) begin
        baud_counter = 4'b0000;
        sample_now = 1'b0;
    end

    //If startbit is 1 calculate 8 baud cycles

    else if (counter_ENABLE == 1 && STARTBIT == 1 && baud_counter == 4'b1000 && Rx_sample_ENABLE == 1'b1) begin
        baud_counter = 4'b0000;
        sample_now = 1'b1;
    end

    //After starbit for every other bit (data bit, parity bit or stop bit) calculate 16 cycles

    else if (counter_ENABLE == 1 && baud_counter == 4'b1111 && Rx_sample_ENABLE == 1'b1) begin
        baud_counter = 4'b0000;
        sample_now = 1'b1;
    end
    else if ( counter_ENABLE == 1 && Rx_sample_ENABLE == 1'b1) begin
        baud_counter = baud_counter + 1;
        sample_now = 1'b0;
    end
    else begin
        sample_now = 1'b0;
    end
end
```

Η δομή της μονάδας του baud_counter_rx είναι απλή. Εφόσον ενεργοποιηθεί, αν πρόκειται για το Start Bit (είσοδος STARTBIT = 1), μετρά 8 σήματα δειγματοληψίας του Baud Controller, ενώ σε κάθε άλλη περίπτωση μετρά 16 αντίστοιχα σήματα. Μόλις ολοκληρωθεί η διαδικασία μέτρησης, στέλνεται σήμα sample_now = 1 στην έξοδο. Αυτό καταδεικνύει στην μονάδα του δέκτη πως πρέπει να αποθηκεύσει τη στιγμή που το sample_now είναι 1, το bit που δέχεται στην είσοδο του, ως το νέο ληφθέν bit.

Επαλήθευση:

Το πλαίσιο ελέγχου του δέκτη χωρίζεται σε 2 σκέλη:

- Στο πρώτο σκέλος προσομοιώνεται η λειτουργία του αποστολέα η οποία στέλνει διαρκώς την ακολουθία 10101010 (16αδικό- aa) στην μονάδα του δέκτη. Αυτό έχει ως σκοπό να ελέγξει κατά πόσο ο δέκτης εξάγει σωστά στην έξοδο του, τα δεδομένα που έλαβε και οδηγεί ορθώς στο 1 το σήμα valid για ένα κύκλο.
- Στο δεύτερο σκέλος, εισάγεται **σκοπίμως** μαζί με την ακολουθία 10101010, **λανθασμένο** parity bit, έτσι ώστε να ελεγχθεί αν η μονάδα του δέκτη στείλε σήμα σφάλματος parity error και δεν αποθηκεύσει τα δεδομένα στην έξοδο της.

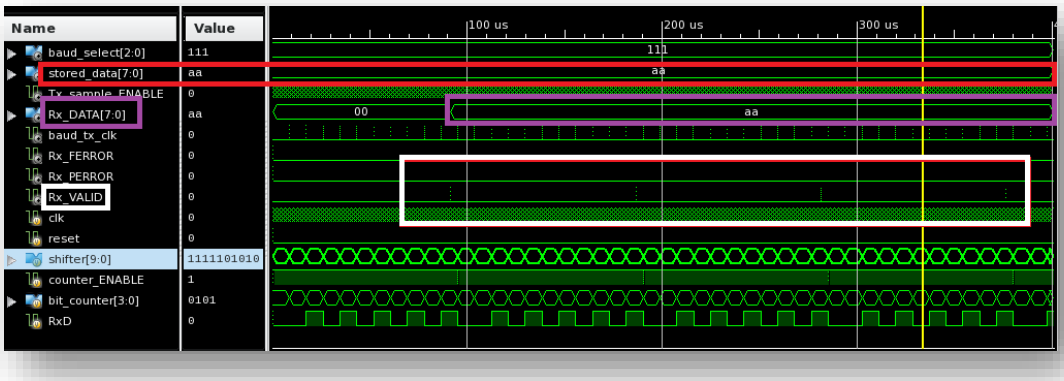
Όπως και στο πλαίσιο ελέγχου του αποστολέα στο μέρος Β, έτσι και εδώ γίνονται οι εξής ενέργειες:

- Περίοδος ρολογιού στα 50 MHz.
- Αρχικοποίηση κυκλώματος με τυχαίο σήμα reset στα 74 ns.
- Συμφωνημένο baud rate στα 115200 bits/sec (κωδικοποίηση 111).

Πρώτο σκέλος

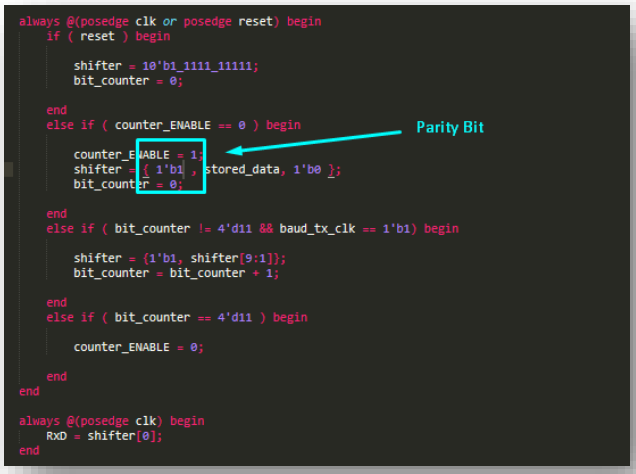
```
initial begin
    #74 reset=0;
end

//Fpga spartan 3 clock frequency ==> 50 Mhz

always #10 clk <= ~clk;

//Set value of baud select to 115200 bits/sec

assign baud_select = 3'b111;

//Set 8-bit sequence to send to receiver

assign stored_data = 10'b10101010;

//////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////---------Simulate transmitter behaviour, in order to send data to receiver----------//////////

baud_controller baud_controller_for_tb(reset, clk, baud_select, Tx_sample_ENABLE);


baud_counter_tx baud_counter_for_tb ( reset, clk, counter_ENABLE, Tx_sample_ENABLE, baud_tx_clk );

always @(posedge clk or posedge reset) begin
    if ( reset ) begin

        shifter = 10'b1_1111_11111;
        bit_counter = 0;

    end
    else if ( counter_ENABLE == 0 ) begin

        counter_ENABLE = 1;
        shifter = { 1'b0 , stored_data, 1'b0 };
        bit_counter = 0;

    end
    else if ( bit_counter != 4'd11 && baud_tx_clk == 1'b1) begin

        shifter = {1'b1, shifter[9:1]};
        bit_counter = bit_counter + 1;

    end
    else if ( bit_counter == 4'd11 ) begin

        counter_ENABLE = 0;

    end
end

always @(posedge clk) begin
    RxD = shifter[0];
end

//////////////////////////////////////////////////////////////////////////////////////////////////////////

//Instantiate Receiver Module
//Set Rx_Enable always to 1

uart_receiver uart_receiver_for_tb(reset, clk, Rx_DATA, baud_select, 1'b1, RxD,
Rx_FERROR, Rx_PERROR, Rx_VALID);
```

Οι κυματομορφές αποδεικνύουν την ορθή λειτουργία του κυκλώματος, όταν δέχεται έγκυρα δεδομένα από τον αποστολέα (Σημείωση: Η ακολουθία aa στέλνεται στον δέκτη συνεχώς). Η προσομοίωση έγινε πάνω στη μονάδα μετά και το **Place & Route.**



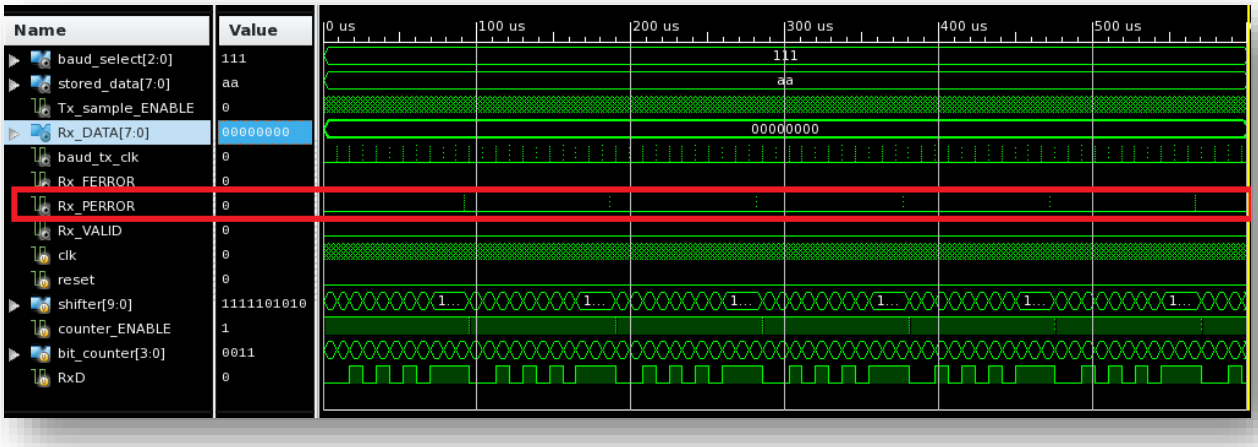
Δεύτερο σκέλος

Για το δεύτερο σκέλος της προσομοίωσης, ο κώδικας παραμένει ο ίδιος με τη μόνη διαφορά να υφίσταται στην μετάδοση του parity bit. Για την ακολουθία 10101010, το parity bit είναι 0, καθώς υπάρχει ζυγός αριθμός άσσων. Συνεπώς, για τις ανάγκες της προσομοίωσης στο δεύτερο σκέλος, αλλάζουμε το parity bit που στέλνει ο αποστολέας για τη δεδομένη ακολουθία σε 1.




```
always @(posedge clk or posedge reset) begin
    if ( reset ) begin

        shifter = 10'b1_1111_11111;
        bit_counter = 0;

    end
    else if ( counter_ENABLE == 0 ) begin

        counter_ENABLE = 1;
        shifter = { 1'b1 , stored_data, 1'b0 };
        bit_counter = 0;

    end
    else if ( bit_counter != 4'd11 && baud_tx_clk == 1'b1) begin

        shifter = {1'b1, shifter[9:1]};
        bit_counter = bit_counter + 1;

    end
    else if ( bit_counter == 4'd11 ) begin

        counter_ENABLE = 0;

    end
end

always @(posedge clk) begin
    RxD = shifter[0];
end
```

Η νέα προσομοίωση συμπεριφοράς με το διαφοροποιημένο πλαίσιο ελέγχου παράγει τις παρακάτω κυματομορφές στις οποίες φαίνεται πως το σήμα Rx_PERROR ορθώς οδηγείται στο 1. (Σημείωση: Η ακολουθία aa στέλνεται στον δέκτη συνεχώς).



Κάλυψη λειτουργίας: Η κάλυψη λειτουργίας του κυκλώματος του receiver, μέσα από το πλαίσιο ελέγχου που επισημάνθηκε παραπάνω, πλησιάζει το μέγιστο αλλά η λειτουργία δε καλύπτεται ολοκληρωτικά. Αυτό συμβαίνει διότι λαμβάνοντας την οντότητα του receiver ξεχωριστά, όπως και συνέβη στο μέρος Β, δεν είναι αρκετά εύκολο να διαμορφωθεί το πλαίσιο ελέγχου με τέτοιο τρόπο, έτσι ώστε να επιβάλλει τον δέκτη να μη δειγματοληπτήσει σωστά το start bit ή το stop bit και κατά επέκταση να ελέγξει αν παράγει το σήμα σφάλματος Rx_FERROR.

5. Μέρος Δ: Σύστημα UART Αποστολέα-Δέκτη για Σειριακή Μεταφορά Δεδομένων

> *Το σύστημα UART Αποστολέα – Δέκτη συνενώνει τις δυο μονάδες του μέρους Β και Γ και με αυτό τον τρόπο σχηματίζεται ένα πλήρες κανάλι UART.*

Για τις ανάγκες του μέρους Δ υλοποιείται ένα **συνθέσιμο** σύστημα που εμπεριέχει τις μονάδες του αποστολέα και δέκτη και τις συνδέει μεταξύ τους, ενώνοντας την σειριακή έξοδο TxD του πρώτου με την σειριακή είσοδο RxD του δεύτερου.

Υλοποίηση:

Το κανάλι UART είναι ένα σύνολο που εμπεριέχει τις εξής μονάδες:

- Baud Controller
- Baud Counter Tx
- Baud Counter Rx
- Reset Synchronizer
- Transmitter
- Receiver
- UART

Με εξαίρεση τη μονάδα UART που αποτελεί το συνθέσιμο σύστημα ένωσης Αποστολέα – Δέκτη, οι υπόλοιπες μονάδες είναι αυτούσιες από τα προηγούμενα μέρη της εργαστηριακής άσκησης.

Σημείωση: Μικρή διαφοροποίηση υφίστανται **μόνο** οι μονάδες του Transmitter και του Receiver, από τις οποίες αφαιρέθηκε το instantiation του reset synchronizer και αυτό διότι προστέθηκε στο “topmodule” UART για το μέρος Δ.

UART Module

```
//////////////////////////////////////////////////////////////////

module uart (rst, clk, Tx_DATA, Tx_BUSY, baud_select, Tx_WR, Tx_EN, Rx_EN, Rx_DATA, Rx_VALID, Rx_FERROR, Rx_PERROR);

input rst, clk;
input [7:0] Tx_DATA;
input [2:0] baud_select;
input Tx_WR,Tx_EN, Rx_EN;

output [7:0] Rx_DATA;
output Rx_PERROR;
output Rx_FERROR;
output Rx_VALID;
output Tx_BUSY;

wire TxD;
wire Tx_BUSY;
wire [7:0] Rx_DATA;
wire [7:0] Tx_DATA;
wire [2:0] baud_select;
wire Tx_EN, Rx_EN;
wire Rx_FERROR;
wire Rx_PERROR;
wire Rx_VALID;

//Synchronize reset using two flip flops to avoid setup and hold violations.

reset_synchronizer reset_synchronizer_0(rst, clk, reset);

//Instantiate Transmitter

uart_transmitter uart_transmitter_0(reset, clk, Tx_DATA, baud_select, Tx_WR,
Tx_EN, TxD, Tx_BUSY);

//Instantiate Receiver. Connect Transmitter to Receiver (TxD <===> RxD)

uart_receiver uart_receiver_0 (reset, clk, Rx_DATA, baud_select, Rx_EN, TxD,
Rx_FERROR, Rx_PERROR, Rx_VALID);

endmodule
```

Η υλοποίηση του UART Module είναι απλή στη δομή της. Αποτελεί το top module του συστήματος και κάνει στις πρώτες γραμμές instantiate τον συγχρονιστή reset, ώστε να μεταδίδει το συγχρονισμένο σήμα reset στις μονάδες του αποστολέα και του δέκτη. Σε δεύτερη φάση περιέχει τα instantiations του transmitter και του receiver, ενώ τη σύνδεση μεταξύ τους πραγματοποιεί η ανάθεση της εξόδου TxD του πρώτου ως είσοδο στην θέση RxD του δεύτερου.

Επαλήθευση:

Για την επαλήθευση του καναλιού UART κατασκευάστηκε ένα πλαίσιο ελέγχου, το οποίο στέλνει ως δεδομένα στο κανάλι τις 4 ακολουθίες-σύμβολα της εκφώνησης (AA, 55, CC, 89 σε 16αδικο). Τα σύμβολα αυτά στέλνονται συνεχόμενα στο κανάλι με τη σειρά που αναγράφτηκαν παραπάνω. Το πλαίσιο ελέγχου διαχειρίζεται το κανάλι αποκλειστικά βάσει των σημάτων Tx_WR,Tx_BUSY του αποστολέα και των σημάτων Rx_VALID, Rx_PERROR και Rx_FERROR του δέκτη.

Η στρατηγική του συγκεκριμένου πλαισίου δοκιμής είναι η εξής:

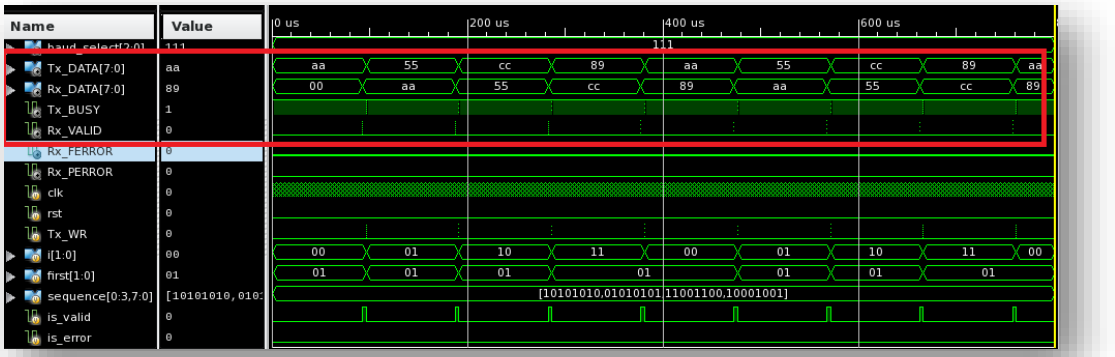
- Οι μονάδες του αποστολέα και δέκτη μένουν πάντα ενεργοποιημένες.
- Μετά το πρώτο τυχαίο Tx_WR σήμα εγγραφής που στέλνεται στη μονάδα και αφορά τη πρώτη ακολουθία, οι υπόλοιπες ακολουθίες στέλνονται ένα κύκλο μετά την οδήγηση του Tx_BUSY στο 0, **αν και εφόσον** το Rx_VALID καταδείξει πως τα προηγούμενα δεδομένα παραλήφθηκαν επιτυχώς από τον δέκτη.
- Σε περίπτωση σφάλματος Rx_FERROR ή Rx_PERROR, το πλαίσιο δοκιμής εκτυπώνει μήνυμα λάθους και σταματά τη συνεχόμενη μετάδοση των ακολουθιών.

Όπως συνέβη και στα πλαίσια δοκιμής των προηγουμένων μερών, το testbench για το μέρος Δ περιέχει τα εξής επιπλέον στοιχεία.

- Περίοδος ρολογιού στα 50 MHz.
- Αρχικοποίηση κυκλώματος με τυχαίο σήμα reset στα 74 ns.
- Πρώτο και μοναδικό από οπτική τυχαιότητας σήμα Tx_WR στέλνεται με καθυστέρηση 1500 ns από την εκκίνηση της προσομοίωσης.
- Συμφωνημένο baud rate στα 115200 bits/sec (κωδικοποίηση 111).

Το βασικό τμήμα του κώδικα Verilog του πλαισίου δοκιμής φαίνεται αναλυτικά στη παρακάτω εικόνα.

Το σύστημα UART δέκτη – αποστολέα μετά και το πέρας του σταδίου **Place & Route**, προσομοιώνεται με το πλαίσιο ελέγχου που επισημαίνεται παραπάνω. Στις κυματομορφές που παράγονται, διακρίνεται η ορθή λειτουργία του συνολικού κυκλώματος.



Κάλυψη λειτουργίας: Τα βασικά διανύσματα εισόδου που ορίζουν ή ωθούν διαφορετικές καταστάσεις στις μονάδες από τις οποίες αποτελείται το κανάλι UART, καλύφθηκαν στα αντίστοιχα μέρη Α,Β,Γ της αναφοράς. Στο πλαίσιο δοκιμής που ελέγχεται το σύστημα Αποστολέα-Δέκτη, απαιτείται ο έλεγχος γύρω από την ορθή μετάδοση των δεδομένων μεταξύ των ενωμένων μονάδων. Συνεπώς, η σωστή λειτουργία του κυκλώματος που αποδεικνύεται μέσα από τις κυματομορφές, δείχνει πως η κάλυψη λειτουργίας του κυκλώματος βρίσκεται στο μέγιστο επίπεδο για το μέρος Δ.

6. Συμπέρασμα

Με το τέλος της προθεσμίας που επιβλήθηκε για τη 2η εργαστηριακή άσκηση, παραδίδεται ολοκληρωμένη η υλοποίηση των τεσσάρων μερών της.

Η πορεία της 2ης εργαστηριακής εργασίας ολοκληρώνεται και το σύστημα UART μαζί με τα υπόλοιπα μέρη είναι πλέον ολοκληρωμένο.

Η δεύτερη εργασία αποτελεί τη συνέχεια στη πορεία του μαθήματος των ψηφιακών συστημάτων.

Μοναδική εξαίρεση που αξίζει να παρατηρηθεί πάνω στη συγκεκριμένη εργασία είναι το γεγονός ότι δεν υπήρξε ανάγκη να γίνει πειραματικός έλεγχος πάνω στη πλακέτα.

ΤΕΛΟΣ ΑΝΑΦΟΡΑΣ