Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Πανεπιστήμιο Θεσσαλίας

ECE333 - Εργαστήριο Ψηφιακών Συστημάτων

Εργαστηριακή Εργασία 2η

Περιεχόμενα

1.Τίτλος

Όνομα: Τσελεπή Ελένη

ΑΕΜ:03272

Ημερομηνία:8/11 - 4/12

Τίτλος εργασίας: Εργαστηριακή Εργασία 2η, Υλοποίηση Μονάδας Γενικού Ασύγχρονου Δέκτη Αποστολέα

2.Περίληψη

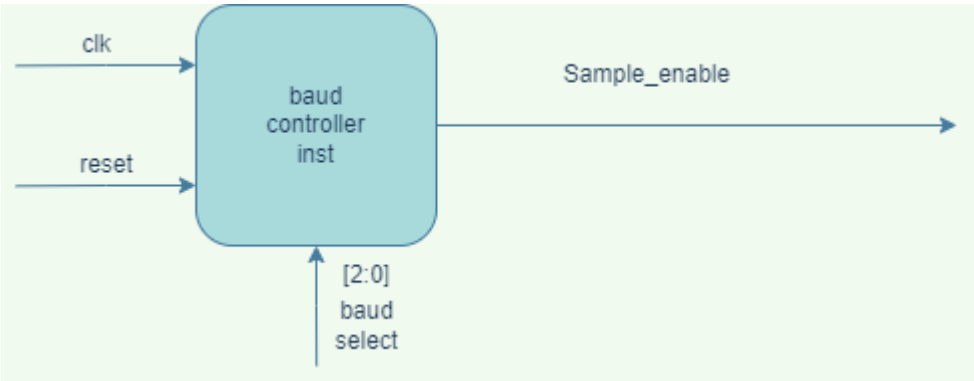
Το παρακάτω κείμενο αποτελεί την εργαστηριακή αναφορά για την 2η εργαστηριακή εργασία όπου στόχος της αναφοράς είναι η σαφής περιγραφή της διαδικασίας σχεδίασης, επαλήθευσης, δοκιμής και τελικής υλοποίησης του κυκλώματος. Παρακάτω αναφέρονται αναλυτικά όλα τα μέρη υλοποίησης της εργασίας καθώς και στιγμιότυπα από τις κυματομορφές στο vivado.

3.Εισαγωγή

Ο στόχος της 2ης εργαστηριακής εργασίας ήταν η υλοποίηση ενός συστήματος σειριακής επικοινωνίας, το οποίο θα χρησιμοποιεί το πρωτόκολλο UART (Universal Asynchronous Receiver Transmitter — Γενικού Ασύγχρονου Δέκτη Αποστολέα). Το σύστημα αποτελείται από έναν UART αποστολέα και έναν UART Δέκτη, τους οποίους και υλοποιήσαμε για να μεταφέρουν δεδομένα στη μια κατεύθυνση, από τον Αποστολέα στον Δέκτη, μέσω μιας σειριακής σύνδεσης ενός σήματος. Αναλυτικά τα βήματα που ακολουθήθηκαν για την υλοποίηση του αποστολέα και του δεκτή τα παραθέτω παρακάτω .

4. Μέρος Α - Ελεγκτής Baud Rate

Για τον ορθό ρυθμό δειγματοληψίας έτσι ώστε να μην χάνονται ή να πολλαπλασιάζονται δεδομένα ο Αποστολέας και ο Δέκτης προσυμφωνούν την ταχύτητα της μεταξύ τους επικοινωνίας σε μονάδες Baud. Για αυτό τον λόγο χρειαζόμαστε ένα ελεγκτή ο οποίος ανάλογα με το επιλεγμένο Baud Rate θα παρέχει το κατάλληλο σήμα δειγματοληψίας .



Συγκεκριμένα χρειαζόμαστε το σήμα δειγματοληψίας να έχει συχνότητα
16 φορές x Baud Rate. Ισοδύναμα δηλαδή το σήμα δειγματοληψίας έχει περίοδο 1/(16 x Baud Rate) και για να πέτυχουμε αυτή τη περίοδο χρησιμοποιούμε έναν μετρητή, ο οποίος όταν ολοκληρώσει την μέτρηση κάνει το σήμα δειγματοληψίας (sample enable) ένα, για έναν κύκλο ρολογιού. Συγκεκριμένα , ο ελεγκτής παίρνει ως είσοδο ένα 3bit baud select το οποίο ορίζει την ταχύτητα επικοινωνίας , το clk στα 100MHZ και το reset. Ο ελεγκτής με βάση το baud select καθορίζει την περίοδο δειγματοληψίας σε κύκλους ρολογιού και έπειτα ένας μετρητής μετράει αυτούς τους κύκλους ρολογιού . Επειδή ο μετρητής ξεκινάει από το 0 όταν φτάσει στην τιμή cycles-1 τότε το σήμα δειγματοληψίας sample_enable γίνεται ένα για ένα κύκλο ρολογιού και ο μετρητής επιστρέφει στο 0 .

Για τα 8 διαφορετικά baud Rates υπολογίζουμε τους κύκλους ρολογιού με βάση την περίοδο του ρολογιού (10ns,100MHZ) και παρακάτω σας παραθέτω τα αποτελέσματα .

Baud Rate, περίοδος δειγματοληψίας Tsc=1/(16 x BaudRate) , κύκλοι ρολογιού cycles=Tsc/10ns

Baud Rate (bits/sec)	**Tsc=1/(16xBaud Rate)**	**Cycles (Tsc/10ns)**
300	208*10^-6	20833
1200	5.2*10^-5	5208
4800	1.3*10^-5	1302
9600	6.5*10-6	651
19200	3.25*10^-6	325
38400	1.62*10^-6	162
57600	1.08*10^-6	108
115200	5.42*10^-7	54

Υπολογισμός σχετικού σφάλματος :

Το σφάλμα δίνεται από τον τύπο $\frac{|y-x|}{x}$, οπού y η προσέγγιση των κύκλων και x ο πραγματικός αριθμός των κύκλων

Για baud rate 300 το σχετικό σφάλμα είναι : $\frac{|20833-20833.333|}{20833.333}$ *100%= 0.0015999%

Για baud rate 1200 το σχετικό σφάλμα είναι : $\frac{|5208-5208.333|}{5208.333} * 100\%$= 0.00639%

Για baud rate 4800 το σχετικό σφάλμα είναι : $\frac{|1302-1302.08333|}{1302.08333} * 100\%$= 0.00639%

Για baud rate 9600 το σχετικό σφάλμα είναι : $\frac{|651-651.0416|}{651.0416} * 100\%$= 0.00639%

Για baud rate 19200 το σχετικό σφάλμα είναι : $\frac{|325-325.520833|}{325.520833} * 100\%$= 0.159%

Για baud rate 38400 το σχετικό σφάλμα είναι : $\frac{|162-162.7604|}{162.7604} * 100\%$= 0.46%

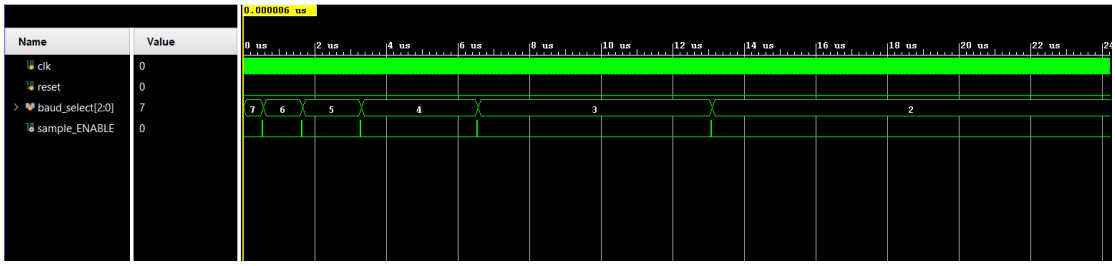
Για baud rate 57600 το σχετικό σφάλμα είναι : $\frac{|108-108.50694|}{108.50694} * 100\%$ = 0.46%

Για baud rate 115200 το σχετικό σφάλμα είναι : $\frac{|54-54.25347|}{54.25347} * 100\%$ = 0.46%

Παρατηρούμε λοιπόν πως για τιμές του Baud Rate μεγαλύτερες από 19200 bits/sec το ποσοστιαίο σχετικό σφάλμα μεγαλώνει σε σχέση με μικρότερες τιμές του baud rate.Για αυτό τον λόγο πρέπει να είμαστε ακριβείς με τον υπολογισμό των κύκλων ρολογιού που αντιστοιχούν στην περίοδο δειγματοληψίας .

Τα αποτελέσματα από το testbench για το Μέρος Α

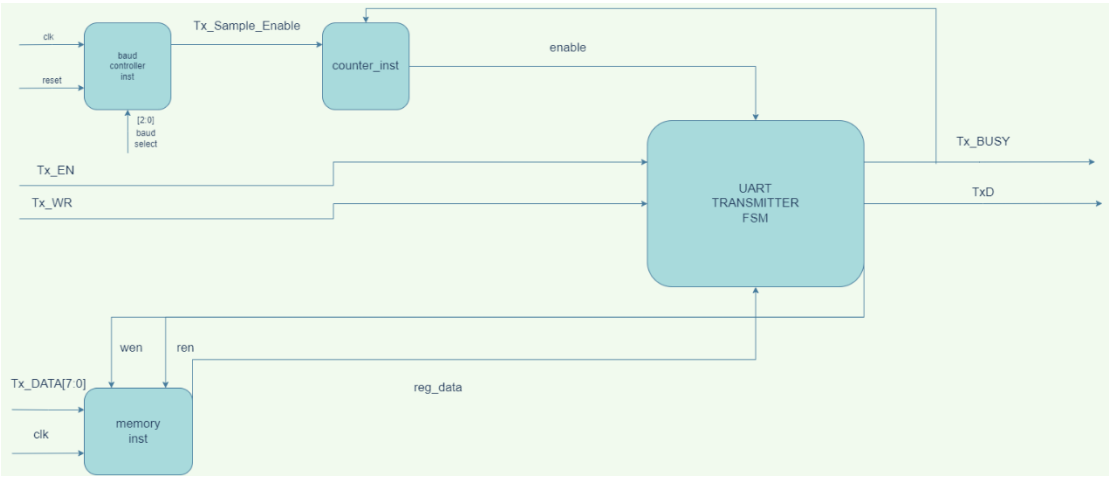
Για τον έλεγχο του Α μέρους έφτιαξα ένα αυτόματο testbench το οποίο αλλάζει το baud select και ελέγχει αν σε (cycles-1)*period το σήμα δειγματοληψίας έχει γίνει 1 (baud select==1).



Στιγμιότυπο από το testbench του Α Μέρους, στο οποίο φαίνονται πως για διαφορετικό baud rate αλλάζει η περίοδος οπού το sample enable γίνεται ένα .

5.Μέρος Β - Υλοποίηση UART Αποστολέα (Transmitter)

Dataflow Μέρους Β



Στο μέρος Β υλοποιήσαμε τον Αποστολέα (transmitter) της σειριακής επικοινωνίας ο οποίος ουσιαστικά αντιμετωπίζεται ως μια **FSM** . Συγκεκριμένα ο Αποστολέας ενεργοποιείται με το σήμα Tx_EN και θεωρούμε ότι μένει ενεργό έως ότου ολοκληρωθεί η τρέχουσα μεταβίβαση . Όταν έρθει από το σύστημα το σήμα Tx_WR (γίνεται ένα για έναν κύκλο ρολογιού) τότε ξεκινάει η διαδικασία μετάδοσης με τα

δεδομένα προς μετάδοση να βρίσκονται στο Tx_DATA[7:0] . Τότε ο αποστολέας κάνει το σήμα Tx_BUSY ένα και για όσο διαρκεί η μετάδοση των δεδομένων αυτό παραμένει ένα ώστε το σύστημα να μην μπορεί να κάνει ξανά το Tx_WR ένα και να στείλει δεδομένα. Για το συγκεκριμένο η υλοποίηση μου δεν το κάνει κάποιο παραπάνω έλεγχο. Δηλαδή αν είναι το Tx_BUSY=1 τότε ο χρήστης πρέπει να ξανά στείλει δεδομένο μόνο εφόσον το Tx_BUSY=0. Όσο ο αποστολέας μας δεν στέλνει δεδομένα (Tx_WR=0 && Tx_En=1) η 1bit έξοδος του αποστολέα είναι στο 1 .

Όταν έρθει το Tx_WR και είναι ενεργοποιημένος επειδή έχω αποφασίσει η υλοποίηση της FSM μου να είναι MOORE και δεν επιτρέπεται η είσοδος να καθορίζει την έξοδο ο αποστολέας μου πηγαίνει στην κατάσταση State_write_mem οπού το σήμα write enable(we) γίνεται ένα και στην μνήμη γράφονται τα δεδομένα από το Tx_DATA. Στον επόμενο κύκλο ρολογιού(CurrentState<=NextState) ο αποστολέας πηγαίνει στην κατάσταση state_start και στέλνει το start bit=0 και το Tx_BUSY γίνεται 1 έως ότου ολοκληρωθεί η διαδικασία αποστολής του συμβόλου . Οι κύκλοι που το σήμα Tx_sample_ENABLE είναι 1, σηματοδοτούν ενεργούς κύκλους για τον Αποστολέα.Ετσι, η μετάδοση δεδομένων , συμπεριλαμβανομένου και του Start bit, γίνεται κάθε 16 κύκλους του Tx_sample_ENABLE. Τους συγκεκριμένους κύκλους τους μετράω με ένα counter ο οποίος όταν φτάσει στην τιμή 15 (16-1) στέλνει ένα σήμα enable και μπορεί ο αποστολέας να στείλει το πρώτο bit από τα δεδομένα που βρίσκονται στη μνήμη . Επειδή μας ζητήθηκε ο αποστολεας να στέλνει τα δεδομένα από το least significant bit προς το most significant bit στέλνουμε πρώτα το reg_data**[0].** Όσο το σήμα enable δεν έρχεται τότε ο αποστολεας παραμένει στην ίδια κατάσταση και στέλνει το ίδιο δεδομένο. Όταν στείλει και τα 8 bit τότε στέλνει και το parity που έχει υπολογίσει κάνοντας xor και 8bit(θα στείλει 0 για ζυγό αριθμό άσσων και 1 για μονό αριθμό άσσων). Τέλος μετά θα στείλει το stop bit =1 και από 16 Tx_sample_ENABLE θα ξανά γυρίσει στην κατάσταση State_before_start όπου θα περιμένει το Tx_WR ώστε να στείλει ένα δεδομένο .

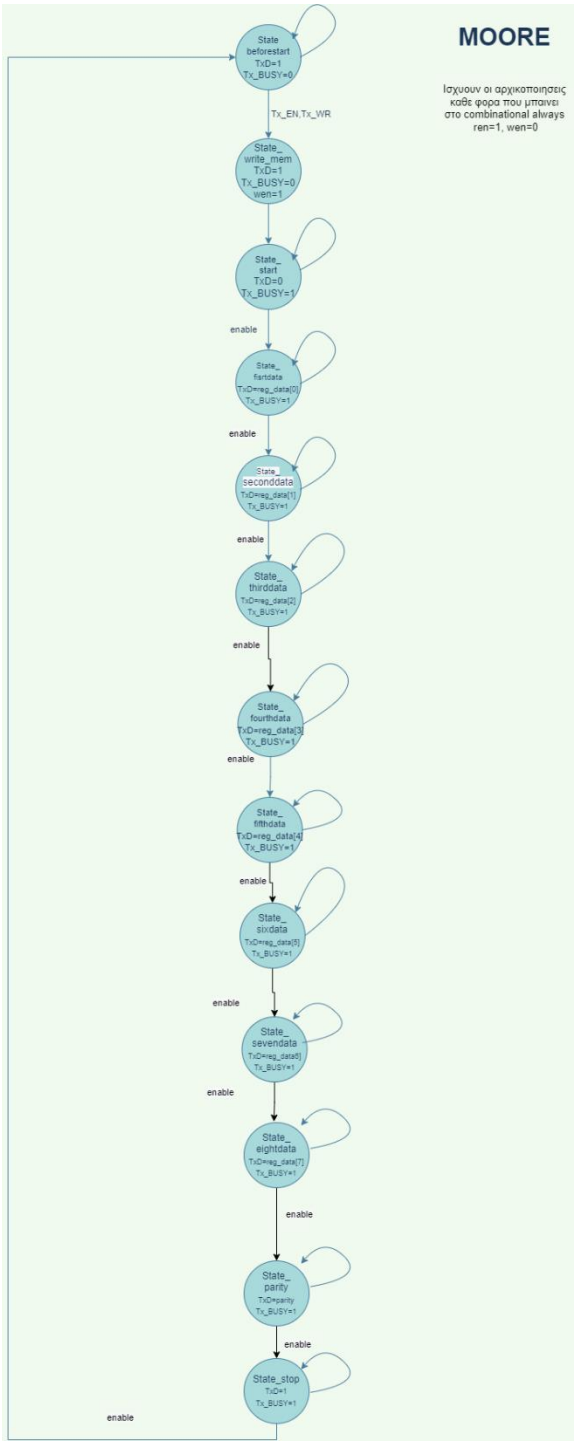
Παρατηρήσεις :

- Ο counter που μετράει τις 16 εμφανίσεις του Tx_sample_ENABLE ξεκινάει να μετράει όταν αρχίσει η διαδικασία αποστολής με το Tx_BUSY να γίνεται 1 και συνεχίζει να μετράει όσο το Tx_Busy είναι 1 .
- Η εντολή reg_enable!=enable && enable==1'b1, όπου το reg_enable έρχεται από ένα flip flop για να κρατάμε την παλιά τιμή του enable , βρίσκεται για να ελέγχω την πρώτη φορά όπου το enable σήμα μας γίνεται 1 δηλαδή θέλω να πέτυχω κάτι σαν posedge enable . Αυτή η εντολή υπάρχει διότι με την εντολή if(enable) ο αποστολεας όταν άλλαζε μια κατάσταση άλλαζε απευθείας και στην επόμενη κατάσταση διότι στον επόμενο κύκλο ρολογιού όπου το enable έπεφτε στο 0 επειδή πρώτα αποτιμούνταν η έκφραση if(enable)και μετά γινόταν το enable 0 γινόταν το NextSate η επόμενη κατάσταση και στον επόμενο κύκλο ρολογιού θα άλλαξε κατάσταση χωρίς να περιμένει τους 16 κύκλους ρολογιού . Δηλαδή άλλαζε 2 καταστάσεις αντί για μια .
- Το κάθε ξεχωριστό bit προς μετάδοση είναι μια ξεχωριστή κατάσταση διότι στέλνω σε κάθε κατάσταση το συγκεκριμένο bit (reg_data[0], reg_data[1]…)διότι σε μια προηγουμένη υλοποίηση μου με counter που έδειχνε ποιο bit πρέπει να στείλουμε δημιουργούνταν latch γιατί ο counter

έπρεπε να κρατάει την προηγουμένη τιμή του κάθε φορά που έμενε στην ίδια κατάσταση .

Όλα αυτά που προαναφέρθηκαν ήδη παρουσιάζονται και στο παρακάτω σχήμα της FSM του αποστολέα. Στο παρακάτω σχήμα παρουσιάζονται στους κύκλους μόνο τα σήματα εξόδου που αλλάζουν τιμές , ισχύει η αρχικοποίηση για κάθε που μπαίνουμε στο combinational always ότι η έξοδος read enable (re) είναι πάντα 1 για να διαβάζουμε από την μνήμη και we=0 γιατί γράφουμε στη μνήμη μόνο όταν έρθει το Tx_WR.

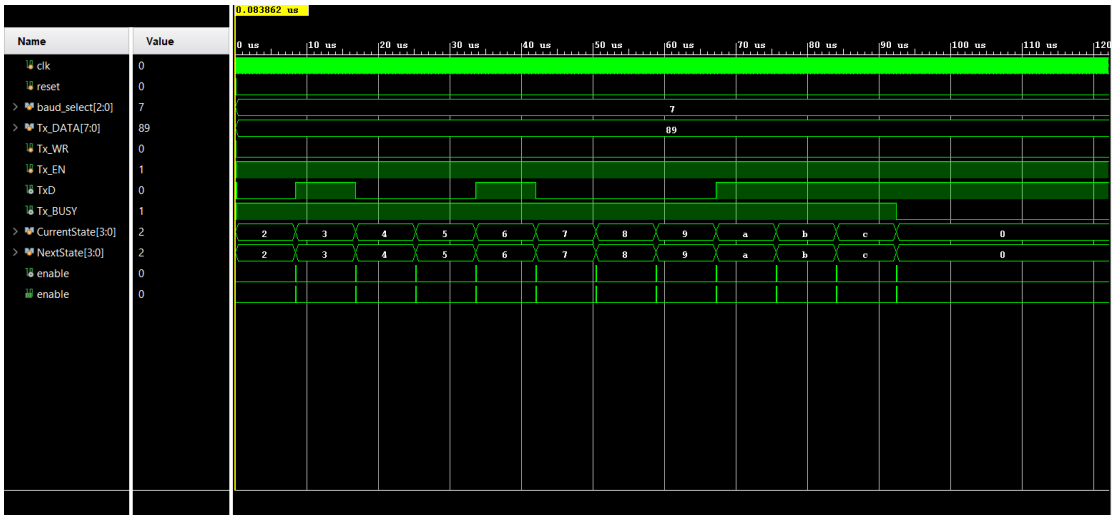
FSM Μέρους Β



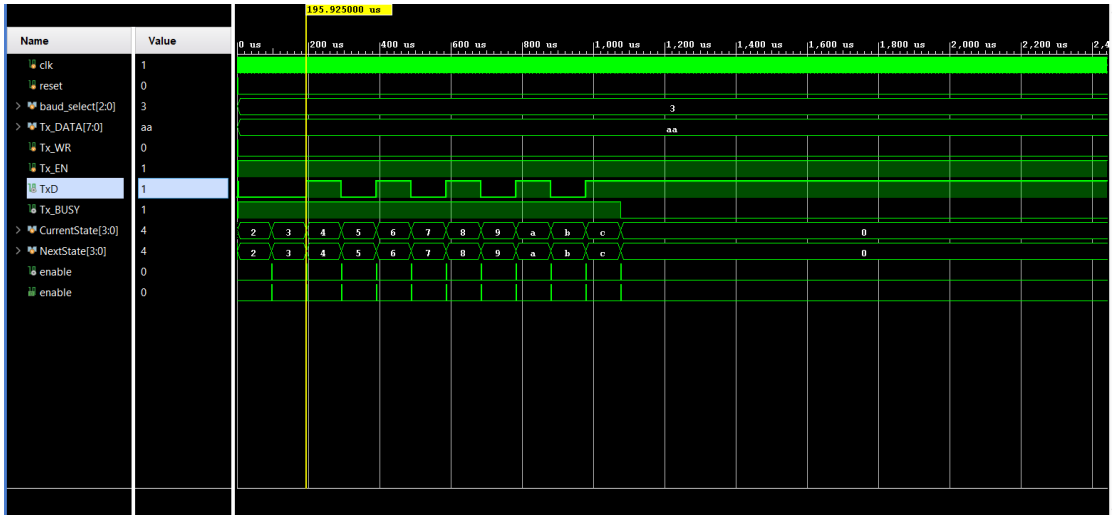
Την υλοποίηση της FSM του αποστολέα την έχω υλοποιήσει με 2 always blocks . 1 sequential για την τρέχουσα κατάσταση και ένα combinational always block για την έξοδο και την επόμενη κατάσταση του αποστολέα . Όπως φαίνεται και στο dataflow η FSM με το σήμα TX_BUSY κάνει τον counter να λειτουργεί αλλά και ο counter με το σήμα enable μετά από 16 Tx_sample_ENABLE αλλάζει την επομένη κατάσταση του αποστολέα μας . Τέλος το module μνήμη βρίσκεται για να κρατάμε τα δεδομένα από το TX_DATA[7:0].

<u>Τα αποτελέσματα από το testbench για το Μέρος Β</u>

Για την επαλήθευση του μέρους Β δοκίμασα να δίνω ένα σύμβολο και να δω την έξοδο του αποστολέα μου σε διαφορετικά Baud Rates .
<u>Συγκεκριμένα:</u>

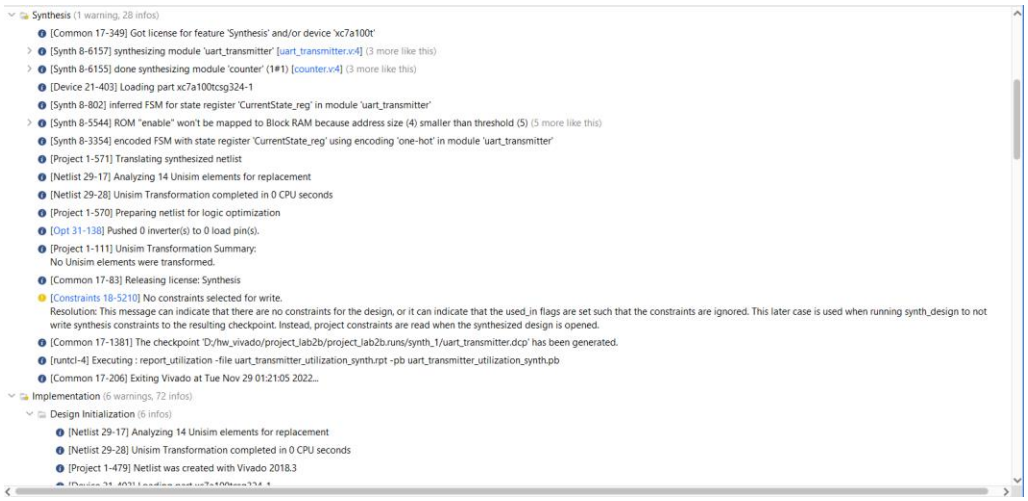


Στιγμιότυπο από την αποστολή του συμβόλου 10001001(89) με baud rate 115200bits/sec. Στο συγκεκριμένο στιγμιότυπο φαίνεται πως σε κάθε σήμα enable η κατάσταση του αποστολέα μας αλλάζει . Στην κατάσταση 2 (state start bit) ο αποστολεας έχει στην έξοδο του TxD το start bit =0 και στην επόμενη ακριβώς κατάσταση στέλνει το τελευταίο bit του 89 (δηλαδή το 1) και συνεχίζει να στέλνει και τα υπόλοιπα 7 bits . Στο τέλος στέλνει το parity =1 (μονό αριθμό άσσων) και stop bit =1 στην κατάσταση c. Μετά από 16 Tx_sample_ENABLE επιστρέφει ξανά στο state start before start.



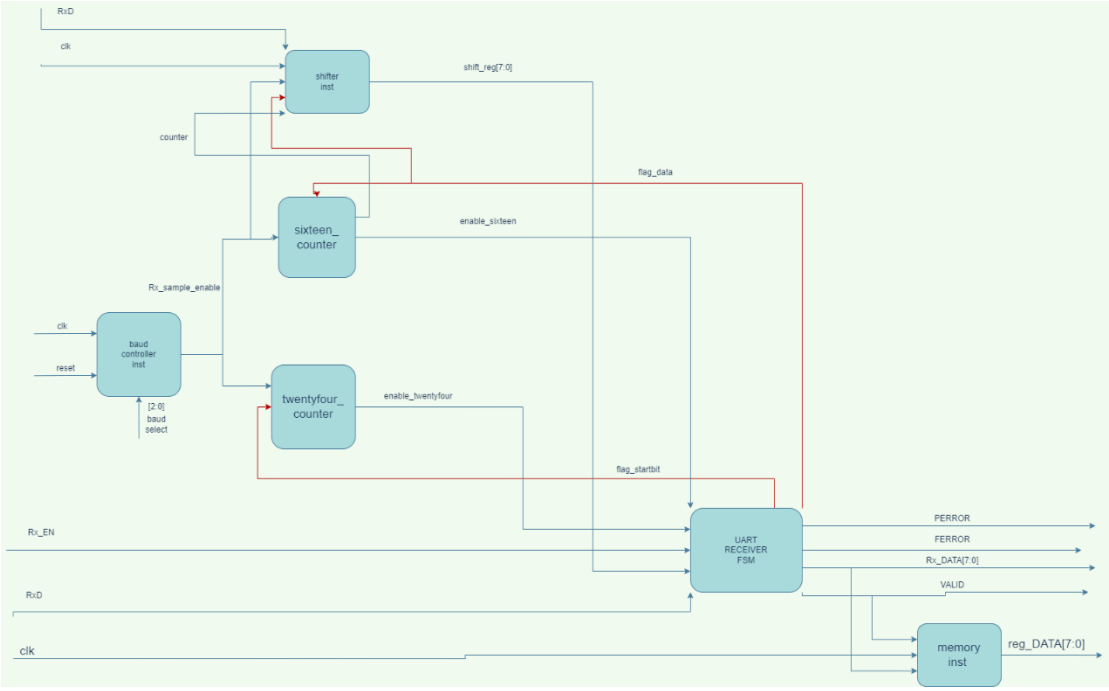
Στο συγκεκριμένο στιγμιότυπο παρουσιάζω την αποστολή του συμβόλου 101010101(aa) με baud rate 4800bits/sec. Για το διαφορετικό αυτό baud Rate παρατηρούμε πως το σειριακό τρένο των δεδομένων πλαταίνει κάτι το οποίο είναι φυσιολογικό αφού έχουμε πιο μικρή ταχύτητα επικοινωνίας. Και για αυτές όμως τις ταχύτητες επικοινωνίας παρατηρούμε ότι η έξοδος του transmitter μας είναι οι αναμενόμενες .

<u>Τα warning του implementation :</u>



Όπως θα δείτε δεν υπάρχουν σημαντικά warnings και το συγκεκριμένο δεν με ανησύχησε .

6.Μέρος Γ - Υλοποίηση UART Δέκτη (Receiver)



Στο Μέρος Γ υλοποιήσαμε τον Δεκτή (receiver) της σειριακής επικοινωνίας ο οποίος ουσιαστικά αντιμετωπίζεται και αυτός ως μια **FSM** . Ο Δέκτης ενεργοποιείται με το σήμα Rx_EN και θεωρούμε ότι μένει ενεργό έως ότου ολοκληρωθεί η διαδικασία αποστολής δεδομένων . Για να ξεκινήσει η διαδικασία λήψης των δεδομένων ο δέκτης πρέπει να λάβει το start bit.Η δειγματοληψία του Δέκτη πρέπει να πραγματοποιείται στην προβλεπόμενη μέση του επομένου bit. Η συγκεκριμένη πρόβλεψη, και η κατάλληλη ευθυγράμμιση της δειγματοληψίας, γίνεται σε σχέση με το Start bit, έχοντας έναν counter ο οποίος μετράει τον κατάλληλο αριθμό ενεργών κύκλων . Στο Δέκτη, οι ενεργοί κύκλοι σηματοδοτούνται από το σχετικό σήμα Rx_sample_ENABLE. Έτσι μόλις στον Δεκτή φτάσει το πρώτο Start bit έχω υλοποιήσει ένα 5bit-counter(twentyfour_counter) ο οποίος μετράει 24 ενεργούς κύκλους(24 Rx_sample_ENABLE) ώστε να ευθυγραμμιστούμε στην μέση του 1^{ου} δεδομένου (16 ενεργοί κύκλοι startbit + 16/2 first data = 16+8=24). Ο συγκεκριμένος counter συγχρονίζεται με το startbit με την χρήση ενός flag(flag_startbit) πετυχαίνοντας να μετράει μονό όσο αυτό το flag είναι 1 . Ένα σήμα (enable_tewntyfour) έρχεται εκείνη την στιγμή για να σηματοδοτήσει το πέρας των 24 ενεργών κύκλων και να πάμε στην επόμενη κατάσταση . Στη συνέχεια για να ευθυγραμμιστούμε στη μέση του 2^{ου} δεδομένου χρησιμοποιούμε έναν 4bit counter(sixteen_counter) ο οποίος μετράει 16 ενεργούς κύκλους ρολογιού(16 Rx_sample_ENABLE) (8 Rx_sample_ENABLE 1^{ου} δεδομένου + 16/2 Rx_sample_ENABLE 2^{ου} δεδομένου =16 Rx_sample_ENABLE). Ο συγκεκριμένος counter ξεκινάει να μετράει με την χρήση ενός flag(flag_data) ο οποίος σηματοδοτεί ότι ξεκίνησε η διαδικασία λήψης των δεδομένων .

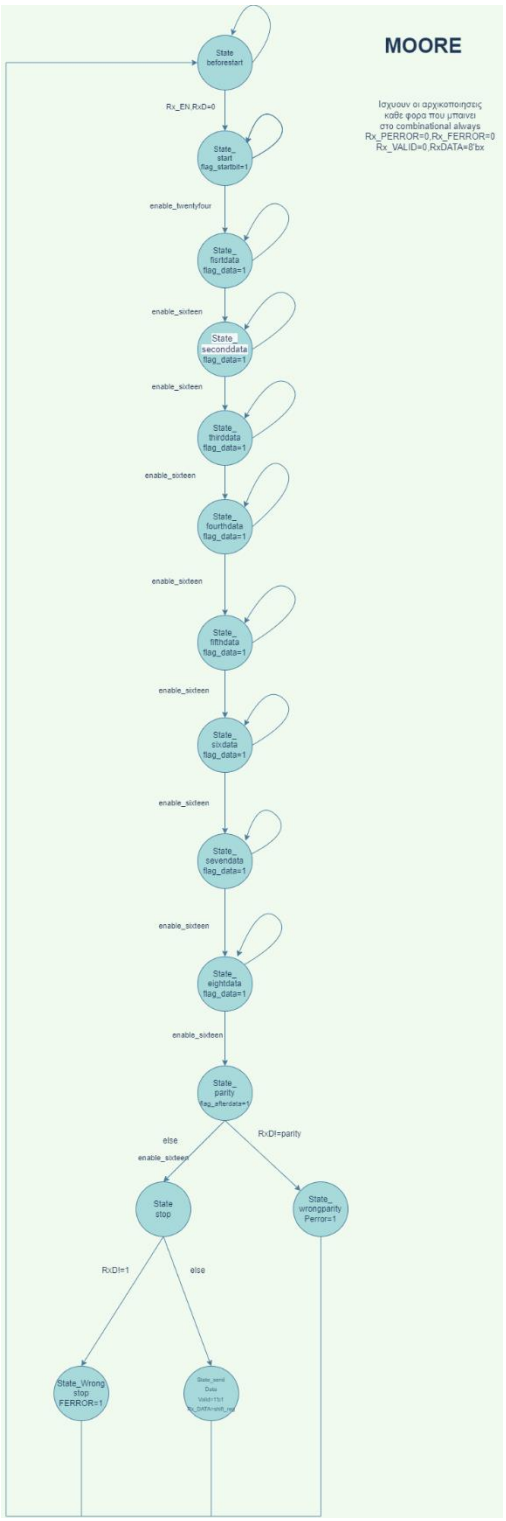
Ο δέκτης ολισθαίνει τα bit δεδομένων ένα προς ένα σε έναν καταχώρηση έτσι ώστε να παραδοθούν μετά παράλληλα στο σύστημα . Επειδή ο αποστολέας στέλνει τα δεδομένα από το least significant bit χρησιμοποιούμε έναν right sifter (sifter_inst). Με τον τρόπο που έχω υλοποιήσει τον receiver στην κατάσταση fisrtdata δειγματοληπτούμε την μέση του 1^{ου} δεδομένου . Επειδή η FSM όσο δεν έρχεται το σήμα από τον counter (enable_sixteen) που σηματοδοτεί ότι πέρασαν οι 16 ενεργοί

κύκλοι παραμένει στην ίδια κατάσταση χρειάζεται να ξέρω την πρώτη φορά που μπήκε σε αυτή την κατάσταση ώστε να περάσω μια φορά το δεδομένο στον καταχώρηση . Αυτό το επιτυγχάνω με την εντολή
if(sample_enable && counter=='d0 && flag) . Την πρώτη φορά που μπει στην κατάστασης ο counter έχει την τιμή 0 αλλά και πρέπει να είναι και στην κατάσταση αποθήκευσης δεδομένων για αυτό υπάρχει και το flag (flag_data).Με το sample_enable ο καταχώρησης συγχρονίζεται με την FSM receiver .
Μετά την αποθήκευση των 8 bit δεδομένων ο δέκτης επαληθεύει ότι η ισοτιμία του συμβόλου είναι σωστή. Σε περίπτωση που διαπιστωθεί σφάλμα στην ισοτιμία, ο Δέκτης κάνει το σήμα Rx_PERROR=1 ώστε να πληροφορήσει την πλευρά του συστήματος ότι τα δεδομένα που παρελήφθησαν δεν είναι σωστά, για να αγνοηθούν σε υψηλότερο επίπεδο και επιστρέφει στην αρχική κατάσταση state_beforestart. Αλλιώς μετά από 16 ενεργούς κύκλους πηγαίνει στην κατάσταση state stop. Εκεί στην περίπτωση που ο Δέκτης δε δειγματοληπτίσει το Stop bit στον χρόνο που το περιμένει σημαίνει ότι τα δεδομένα δεν έχουν πλαισιωθεί σωστά και πληροφορεί το σύστημα για το σφάλμα με το σήμα Rx_FERROR=1 και μετά γυρνάει στην κατάσταση state_beforestart. Αν δεν έχει υπάρξει σφάλμα ο Δέκτης κάνει για έναν κύκλο ρολογιού το σήμα Rx_VALID=1 το οποίο σηματοδοτεί ότι τα δεδομένα είναι έγκυρα και μπορεί να διαβαστούν από το σύστημα και στον Rx_DATA υπάρχουν τα δεδομένα από τον καταχωρητή. Επιπλέον εκείνη την στιγμή τα δεδομένα αποθηκεύονται και στην μνήμη ώστε μέχρι να γίνει ξανά το Rx_VALID =1 τα δεδομένα να βρίσκονται στον Rx_DATA.

Παρατηρήσεις:

- Μια και ο Δέκτης είναι ασύγχρονος ως προς τον Αποστολέα, θα πρέπει πριν την οποιαδήποτε δειγματοληψία, να συγχρονιστεί η ασύγχρονη σειριακή είσοδος. Για αυτό τον λόγο υπάρχουν 2 flip flop (για αποφυγή της μεταστάθειας) που συγχρονίζουν το TxD με τον Δεκτή . Το συγχρονισμένο σήμα είναι το final_RxD το οποίο αν δείτε δίνω και όχι το RxD.
- Η μνήμη είναι διαφορετική από αυτή του transmitter γιατί δεν έχει το ren ώστε να γίνεται πάντα ανάθεση στο Rx_DATA από το reg_data χωρίς να χρειάζεται ren=1.
- Η γραμμή 157 *if(final_RxD!=parity && sixteen_counter=='d0 && flag_afterdata)* υπάρχει για να να ελέγχω μονό μια φορά που είμαι σε αυτή την κατάσταση αν υπάρχει ισοτιμία ή όχι με το parity και όχι και για τις 16 φορές που παραμένω στην ίδια κατάσταση .
- Ο sixteen counter θέλω να μετράει και στις καταστάσεις με τα δεδομένα αλλά και να μετρήσει τους 16 ενεργούς κύκλους από το parity στο stop bit γι αυτό υπάρχει η εντολή με τα 2 διαφορετικά flags (Rx_sample_ENABLE && (flag_data||flag_afterdata))
- Όπως προαναφέρθηκε χρειάζομαι ο shifter να αποθηκεύει μόνο όταν είμαι στην κατάσταση λήψης των δεδομένων γι αυτό έχω το flag_data.
- Χρειάζεται να αναφέρω πως στην υλοποίηση μου δεν ελέγχω αν ο δέκτης έχει ευθυγραμμιστεί με την μέση του Start bit για αυτό ο μονός λόγος για Ferror είναι στο Stop bit.

FSM Μέρους Γ



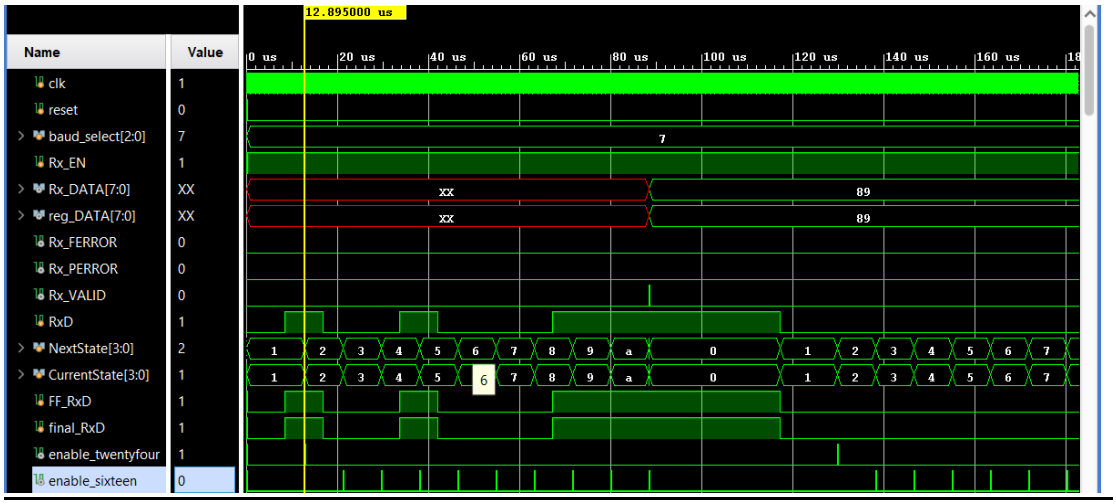
Όλα αυτά που προαναφέρθηκαν υπάρχουν και στο παραπάνω σχήμα της FSM του receiver . Στο παραπάνω σχήμα παρουσιάζονται στους κύκλους μόνο τα σήματα εξόδου που αλλάζουν τιμές , ισχύει η αρχικοποίηση για κάθε που μπαίνουμε στο

combinational always ότι τα σήματα Rx_PERROR, Rx_FERROR,Rx_VALID και RX_DATA είναι 0.

Όπως και στον Αποστολέα υλοποίησα την FSM του δέκτη με 2 always blocks . 1 sequential για την τρέχουσα κατάσταση και ένα combinational always block για την έξοδο και την επόμενη κατάσταση του αποστολέα . Η FSM όπως φαίνεται και στο dataflow αρχικοποιεί τους counter με τα Flag αλλά και οι counters με το σήμα enable που σηματοδοτούν το πέρας των ενεργών κύκλων ορίζουν την επόμενη κατάσταση της FSM .

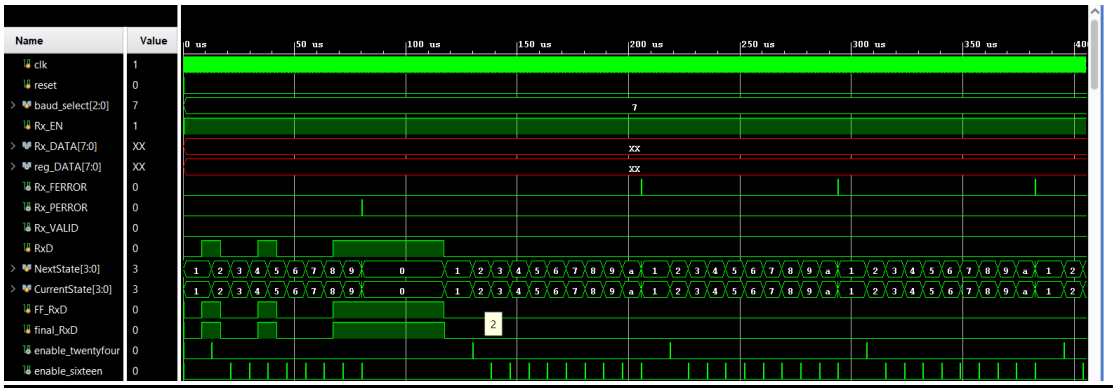
<u>Τα αποτελέσματα από το testbench για το Μέρος Γ</u>

Για να ελέγξω την σωστή λειτουργεία του receiver με καθυστερήσεις στο testbench εισήγαγα 1 bit δεδομένων για να μοντελοποιήσω την λειτουργεία του transmitter. Οι καθυστερήσεις προκύπτουν από την λογική υλοποίησης του transmitter ο οποίος στέλνει 1 bit ανά 16 ενεργούς κύκλους ρολογιού. Ένας ενεργός κύκλος είναι baud_select_cycles * περίοδο ρολογιού μας (10ns) . Επομένως οι 16 ενεργοί κύκλοι είναι **16 * cycles*10ns**. Για κάθε λοιπόν από τα 11 bit δεδομένων έχω καθυστέρηση ίση με 16 ενεργούς κύκλους .

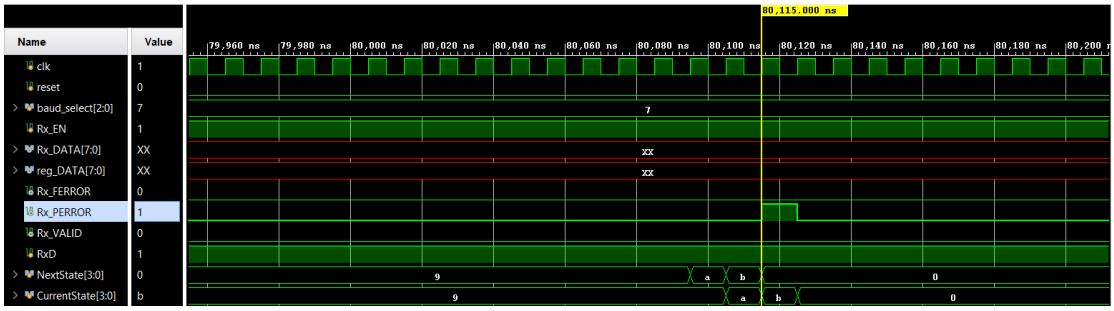


Στιγμιότυπο από την λήψη του συμβόλου 10001001(89) για baud Rate 115200bits/sec. Στο συγκεκριμένο στιγμιότυπο φαίνεται πως σε κάθε σήμα enable η κατάσταση του δέκτη μας αλλάζει. Στην κατάσταση 2 (state first data) όπως φαίνεται και στο στιγμιότυπο ο δέκτης μας δειγματοληπτεί στη μέση του πρώτου bit=1 . Με το πέρας της δειγματοληψίας όπως φαίνεται το Rx_VALID γίνεται ένα για έναν κύκλο ρολογιού και το Rx_DATA έχει όλα τα δεδομένα. Να σημειωθεί πως παρόλο που τα δεδομένα βρίσκονται στο Rx_DATA δεν μπορούν να διαβαστούν από το σύστημα γιατί το Rx_VALID είναι μηδέν.

Στιγμιότυπο με PERROR

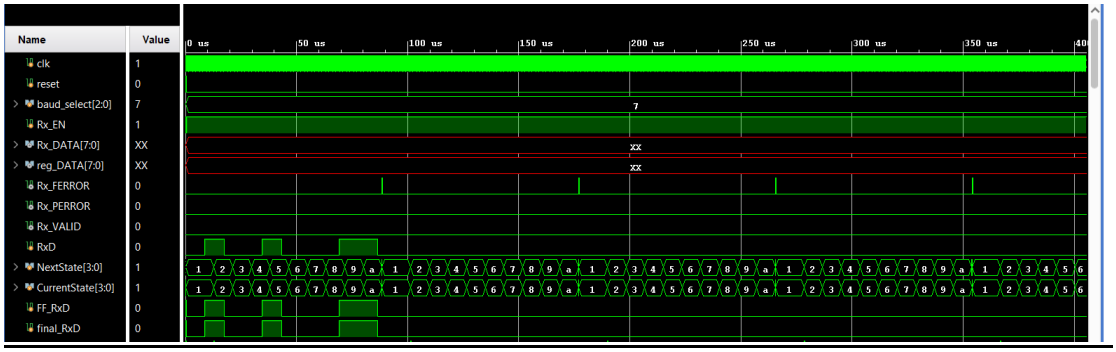


Στο παραπάνω στιγμιότυπο υπάρχει πρόβλημα με το parity. Όπως θα δείτε για έναν κύκλο ρολογιού το PERROR γίνεται ένα, το Rx_DATA δεν έχει δεδομένα και το Rx_VALID παραμένει στο 0 .

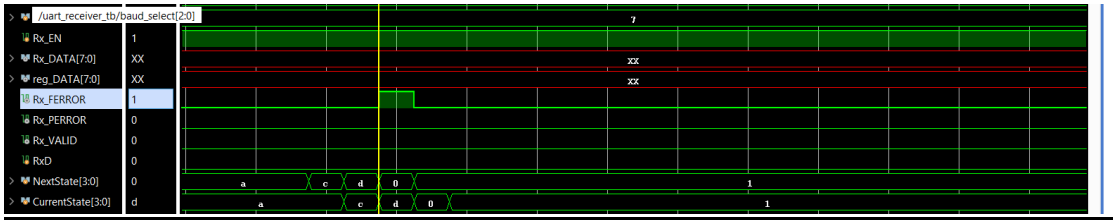


Σε μεγέθυνση φαίνεται πως όταν η FSM πάει στην κατάσταση b currentState είναι το b (wrong parity) το PERROR γίνεται ένα και μετά επιστρέφει στην αρχική κατάσταση state_before_start.

Στιγμιότυπο με FERROR

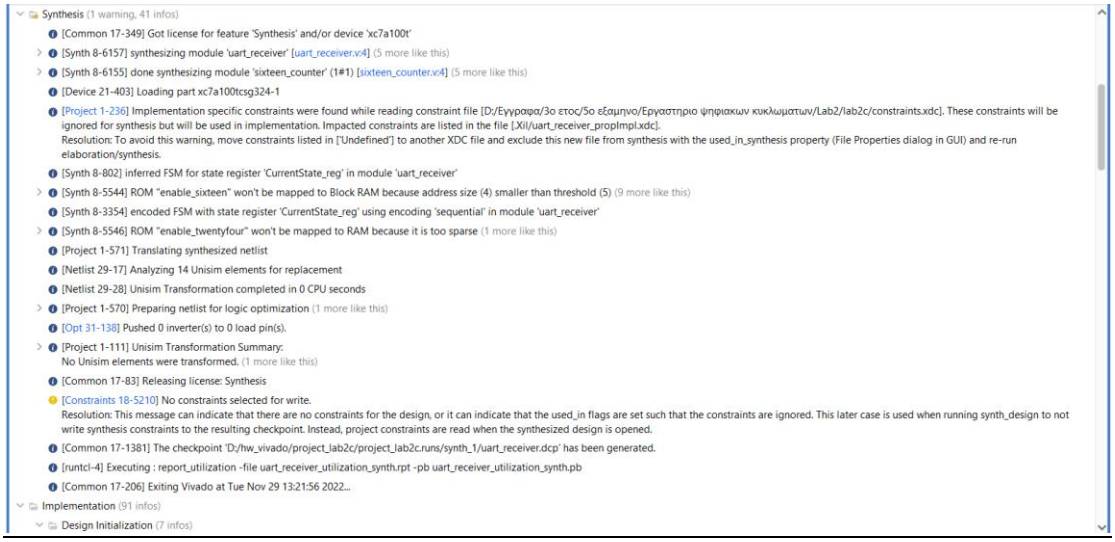


Στο παραπάνω στιγμιότυπο υπάρχει πρόβλημα με το stop bit (είναι μηδέν αντί για 1). Όπως θα δείτε για έναν κύκλο ρολογιού το FERROR γίνεται ένα, το Rx_DATA δεν έχει δεδομένα και το Rx_VALID παραμένει στο 0.



Σε μεγέθυνση φαίνεται πως όταν η FSM πάει στην κατάσταση d (wrong stop) το FERROR γίνεται ένα και μετά επιστρέφει στην αρχική κατάσταση state_before_start.

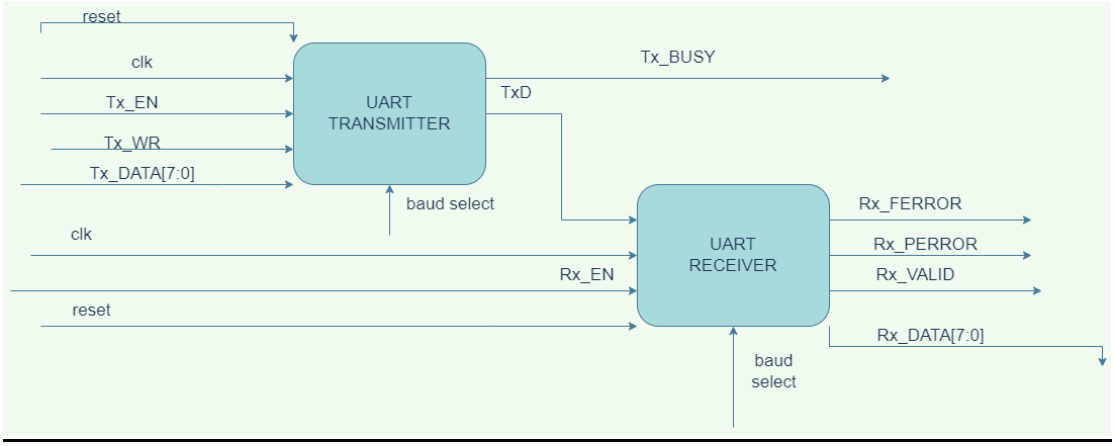
Τα warning του implementation :



Όπως θα δείτε δεν υπάρχουν σημαντικά warnings και το συγκεκριμένο δεν με ανησύχησε .

7.Μέρος Δ - Υλοποίηση UART Αποστολέα-Δέκτη για Σειριακή Μεταφορά Δεδομένων

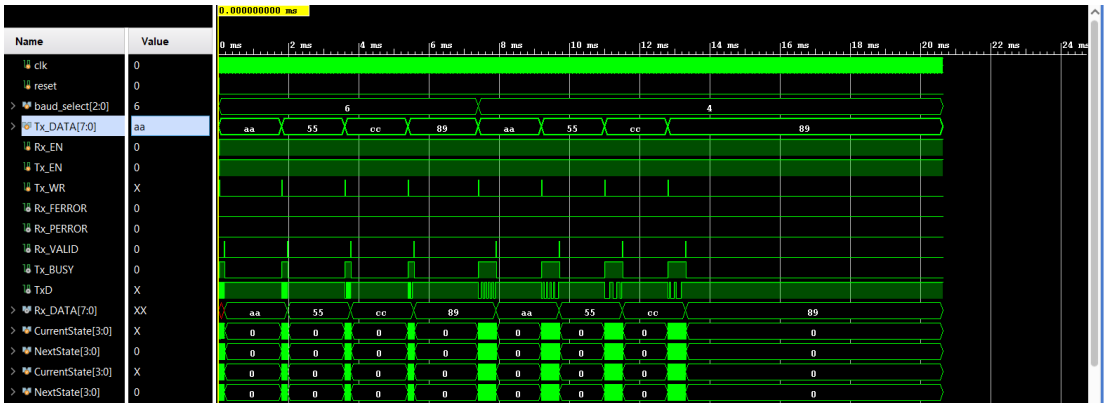
Dataflow Μέρος Δ



Στο Μέρος Δ συνένωσα τον αποστολέα και τον δεκτή σε ένα κανάλι UART όπως φαίνεται και στο παραπάνω dataflow . Συγκεκριμένα, η έξοδος του αποστολέα TxD γίνεται είσοδος στον Δεκτή η RxD αφού πρώτα περάσει από δυο flip flop όπως εξηγήθηκε και στο Μέρος Γ . Η επικοινωνία του πλαισίου δοκιμής με τους Αποστολέα και Δέκτη, βασίζεται πλέον αποκλειστικά στο πρωτόκολλο των σχετικών σημάτων και όχι σε καθυστερήσεις όπως ανέφερα στο μέρος Γ .

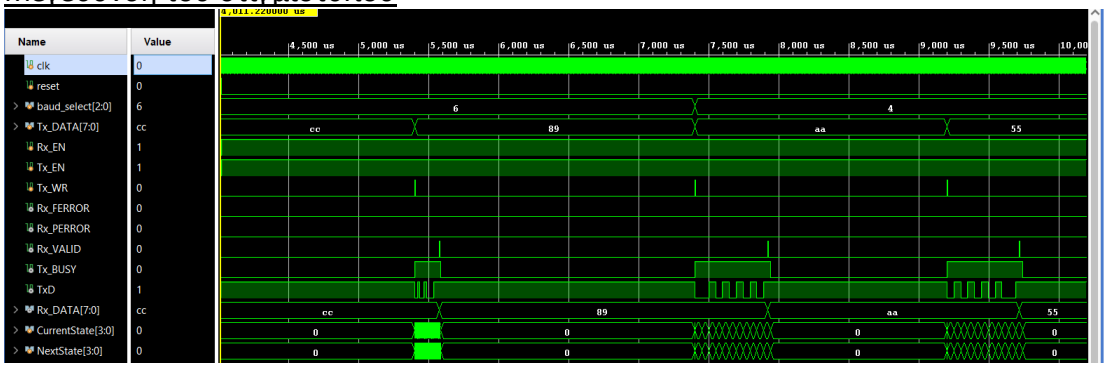
Τα αποτελέσματα από το testbench για το Μέρος Δ

Για τον έλεγχο του Δ μέρους έδωσα στο Tx_DATA τα 4 διαφορετικά σύμβολα με 2 διαφορετικά baud Rates(57600 bits/sec και 19200bits/sec) και περίμενα να δω την έξοδο του Rx_DATA. Οι καθυστερήσεις που θα δείτε στο testbench προέκυψαν μετά από παρατήρηση των κυματομορφών καθώς έπρεπε να έχει ολοκληρωθεί η αποστολή κάθε συμβόλου και μετά να ακολουθεί η επόμενη αποστολή , με το TxWR να γίνεται 1. Έτσι αν παρατηρήσετε υπάρχουν μεγάλες καθυστερήσεις στην αλλαγή του Tx_DATA για να αποφύγω ο αποστολεας να μην έχει προλάβει στείλει το προηγούμενο σύμβολο και γω να αλλάζω το Tx_WR .



Στο στιγμιότυπο φαίνεται η αποστολή των τεσσάρων συμβολών με 2 διαφορετικά Baud Rates . Όπως φαίνεται και στο στιγμιότυπο το Rx_DATA έχει κάθε σύμβολο που έχει στείλει ο transmitter με το Rx_VALID να γίνεται ένα για έναν κύκλο ρολογιού .

Μεγέθυνση του στιγμιότυπου



Σε μεγέθυνση του προηγουμένου στιγμιότυπου φαίνεται πως για μικρότερο baud rate (baud select 4 baud rate 19200 bits/sec) το σειριακό τρένο είναι πλατύτερο από ότι για baud select 6 ,αυτό προέκυψε παρατηρώντας το TxD.

8. Συμπεράσματα

Η 2η εργαστηριακή εργασία ήταν μια καλή πρώτη επαφή με τις μηχανές πεπερασμένων καταστάσεων FSM .Ήταν αρκετά απαιτητική και χρειάστηκαν πολλές ώρες στο testbench για να καταλάβω όλα εκείνα τα λάθη για να λειτουργούν σωστά ο Αποστολέας και ο Δέκτης. Τέλος, ήταν μια καλή εύκαιρα να ασχοληθώ περισσότερο με τα testbench αφού ο μόνος τρόπος να ελέγξουμε αυτή την εργασία ήταν μονό με την επαλήθευση από το testbench.