Σταθουδάκης Σταύρος, AEM:03512
15/12/2023 Lab2 UART

Περίληψη

7 αρχεία συνθέσιμης Verilog, 4 αρχεία δοκμής των συνθέσιμων. Η υλοποίηση του UART πρωτόκολου εποικοινωνίας και η λεπτομερής περιγραφή του κώδικα υλοποίησησης του.

Εισαγωγή

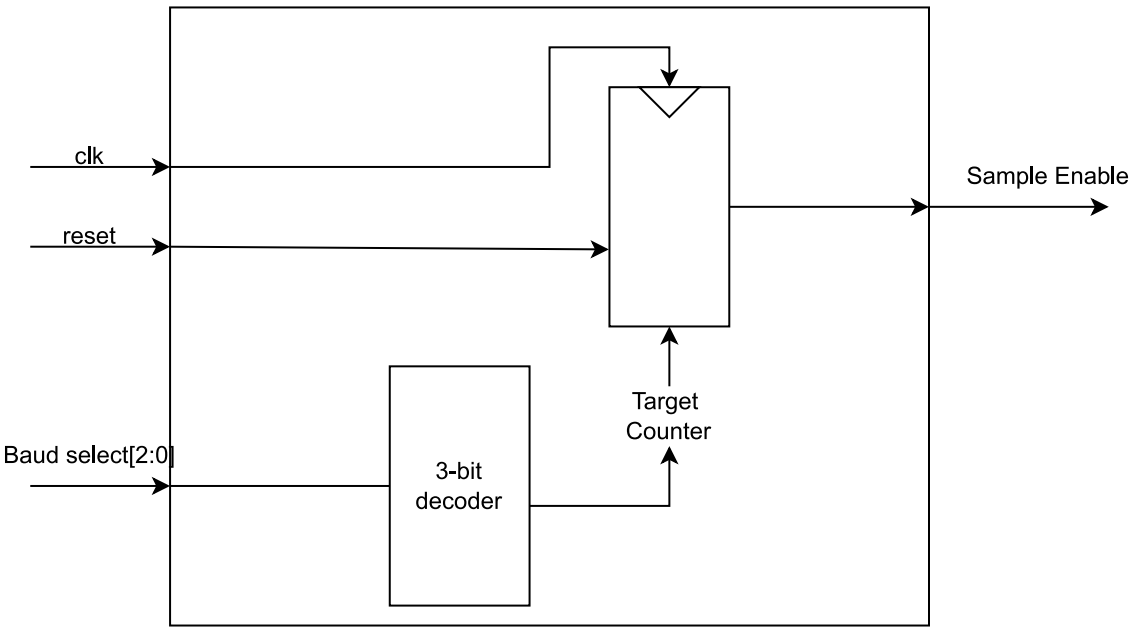
Ο Υποχρεωτικός στόχος της Εργασίας ήταν η υλοπήοιηση του πρωτόκολλου επικοινωνίας UART. Αυτό περιλαμβάνει την υλοποίηση του Baud controller όπου ρυθμίζει το baud rate (bits/sec) έτσι ώστε να μπορεί να συγχρονιστεί η επικοινωνία μεταξύ του transceiver και του receiver, του Transceiver όπου είναι υπεύθυνος για την αποστολή των bits σειριακά και του Receiver για την παραλαβή των bits σειριακά και την αποστολή αυτών παράλληλα στο σύστημα. Αυτός ο στόχος επιτεύψθηκε με επιτυχία.

Όσον αναφορά το προαιρετικό κομμάτι της εργασίας, στόχος ήταν το σήμα του ενός byte όπου στάλθηκε απο τον receiver στο σύστημα να ανδειχθεί σε 2 από τα 4 7-segment-displays. Αυτός ο στόχος δεν ήρθε στο τέλος του λόγω ρονικού περιορισμού.

Μέρος A - Ελεγκτής Baud Rate

Υλοποίηση

- Το κύκλωμα που δημιουρείτε σε αυτό το μέρος είναι ένας 3-bit αποκωδικοποιτής για να οριστεί ένας counter. Έπειτα ακολουθεί ένα Flip-Flop όπου λειτουργεί με το εισαγόμενο ρολόι απο το testbench (χρησιμοποιώ ρολόι των 100 MHz της πλακέτας όπως συστείνετε απο την εκφώνηση) όπου έχει ώς έξοδο ένα σήμα κάθε x όπου x=1/(16*Baud Rate).



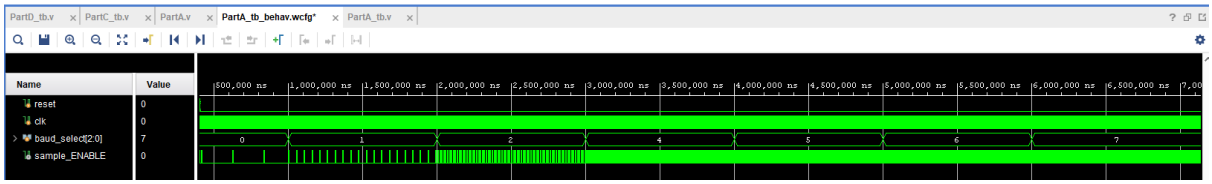
- Ο κώδικας αποτελειτε από 2 always block. Το πρώτο είναι ένας 3-bit decoder όπου ανάλογα με το baud select επιλέγεται και ο κατάλληλος counter όπου αντιστοιχεί στο baud rate αλλα

προσαμοσμένο στο ρολόι της FPGA. Έπειτα το δεύτερο always block είναι ακολουθιακό, με ασύγχρονο always και βγάζει το sample enable=1 κάθε 1/(16*BaudRate).

- Ο counter της πρώτης always βγαίνει από την πράξη: (1/(16*baudRate)) / 0.00000001

Επαλήθευση

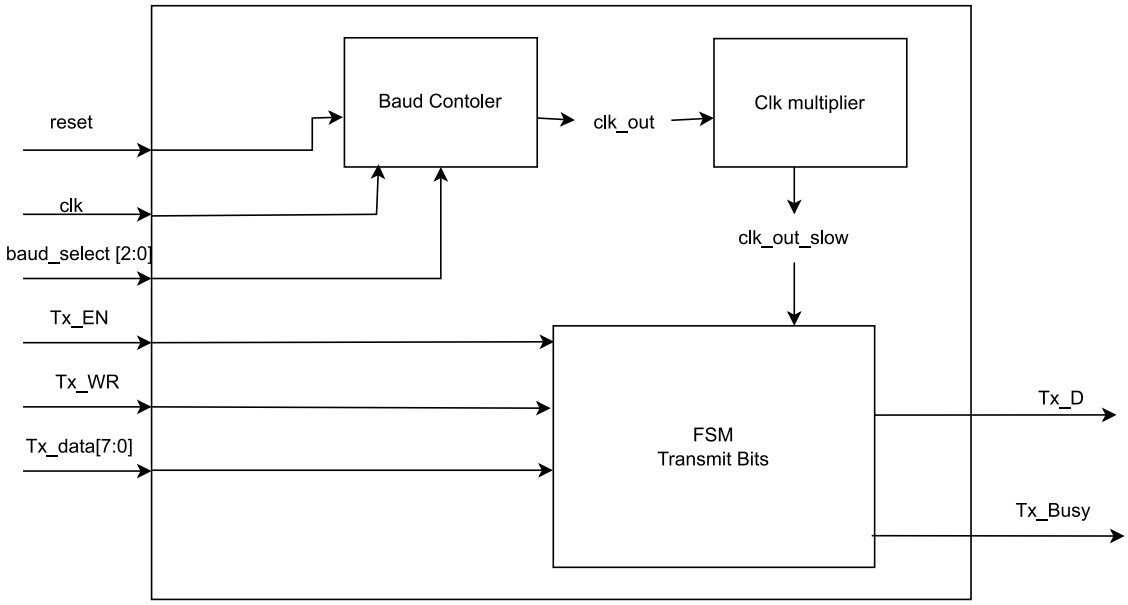
- Το πλαίσιο ελέγχου στην ουσία δοκιμάζει κάθε τιμή του baud_select και τεστάρει αν το sample enable βγαίνει στα σωστά χρονικά διαστήματα
- Η μεταβλητή ελέγχου είναι μία και είναι ο counter για να μπορέσω να παρατηρήσω πότε μηδενίζεται και να επιβεβαιώσω με ακρίβεια την σωστή λειτουργία του. Προφανως αυτός ο έλεγχος συνοδεύεται απο την παρατήρηση του sample_enable και η ορθή ενεργοποίηση και απενεργοποίηση του. Το ποσοστό κάλυψης της λειτουργία του κώδικα είναι 100% επειδή έξετάζετε ή λειτουργία για όλες τις πιθανές εισόδους.
- Το μόνο πρόβλημα που αντιμετώπισα είναι ότι στην εναλλαγή τιμών του baud_select δεν μηδενιζόταν ο counter, κάτι που επιλύθηκε αλλάζωντας τον if περιορισμό και βάζοντας <= αντί για σκέτο =, στην γραμμή 27.
- Παρακάτω παραδίδω μία φωτογραφία από τις κυματομορφές του πλαισίου ελέγχου όπου και φαίνονται όσα δείχνω παραπάνω. Το sample_enable δεν είναι μόνιμα ανοιχτό για baud_select μεγαλύτερο του 4, απλά ενεργοποιείτε πολύ γρήγορα και είναι αρκετό για την κατανόηση λειτουργίας του κυκλώματος.



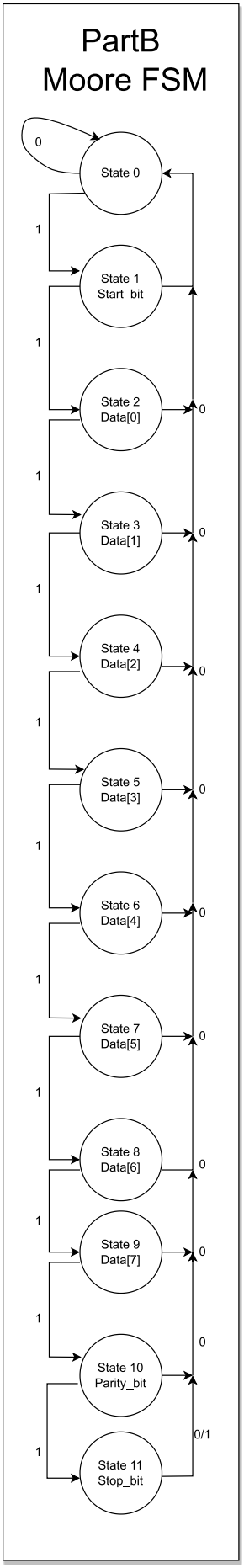
Μέρος Β - UART Αποστολέας (Transmitter)

Υλοποίηση

- Στον κώδικα εμπεριέχονται:
  - Το κάλεσμα του baud controller(Part A) για τον transmitter
  - Clk_multiplier: ενεργοποιεί ένα σήμα κάθε 16 κύκλους του sample enable
  - Send_bits module: Το module το οποίο εμπεριέχει την FSM η οποία είναι κατάλληλη για την μετάδοση των δεδομένων.



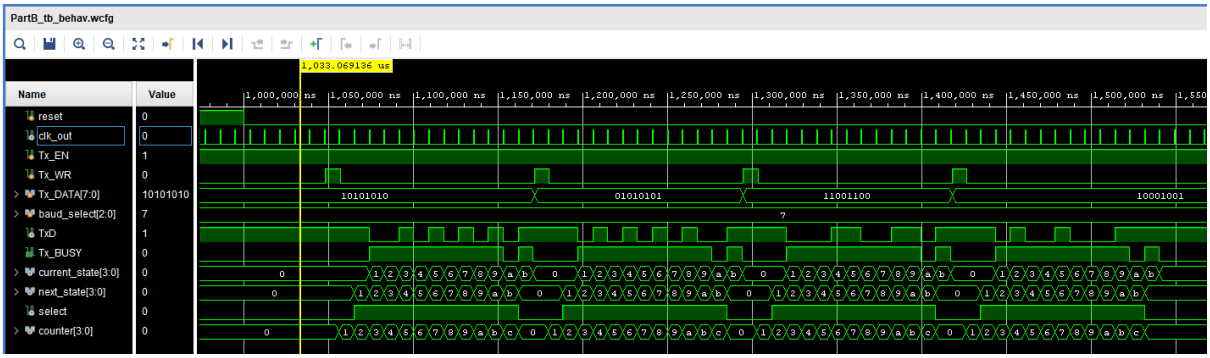
- Περιγραφή κύριων τμημάτων δομής.
    - Στο module clk_multiplier είναι μία ακολουθιακή always (ένα flip-flop δηλαδή) όπου στην ουσία είναι ένας counter που μετατρέπει ένα output σε 1 (για έναν κύκλο του sample enable) κάθε 16 ενεργοποιήσεις του sample_enable.
    - Στο module send_bits εμπεριέχεται και μία FSM όπως ανέφερα και παραπάνω. Αυτή η FSM είναι υπεύθυνη για την σειριακή μετάδοση των δεδομένων. Επέλεξα να την κάνω moore FSM γιατί μου φάνηκε πιο εύκολη η υλοποίηση της.
        - Στην συνδυαστική always: Το πρώτο state ελέγχει αν κάποιο καινούργιο μύνημα έχει φτάσει απο το σύστημα και είναι έτοιμο να μεαταδοθεί. Αν αυτό είναι αληθές, τότε η FSM μπαίνει στο επόμενο state όπου και αρχίζει να στέλενει ένα-ένα τα bits. Εδώ η FSM πάει στην επόμενη κατάσταση μόνο όταν το output από το Clk_multiplier είναι 1. Έπειτα η FSM στέλνει ένα-ένα τα bit εισόδου αλλάζωντας την μεταβλητή Tx_D με την αλλαγή κάθε state. Αυτό γίνεται μέχρι και το state 10 όπου εκεί η FSM θα υπολογίσει (βάζωντας σε xor όλα τα εισαγόμενα bits) και θα στείλει το parity bit. Έπειτα στην επόμενη κατάσταση θα στείλει το stop_bit(=1) και θα επιστρέψει την FSM στην κατάσταση 0 θέτοντας την κατάλληλη για την παραλαβή και την αποστολή ενός καινούργιου μυνήματος.
        - Στην ακολυθιακή always ελέγχεται αν η FSM θα μετακινηθεί στο επόμενο state ή θα επιστρέψει/παραμείνει στο state 0.
        - Παρακάτω σας παραδίδω το σχετικό διάγραμμα που περιγράφει την FSM


PartB
Moore FSM
0
State 0
1
State 1
Start_bit
1
State 2
Data[0]
0
1
State 3
Data[1]
0
1
State 4
Data[2]
0
1
State 5
Data[3]
0
1
State 6
Data[4]
0
1
State 7
Data[5]
0
1
State 8
Data[6]
0
1
State 9
Data[7]
0
1
State 10
Parity_bit
0
0/1
1
State 11
Stop_bit

- Γενικά ο κώδικας αποτελείται από το main module (uart_transmitter) το οποίο περιέχει μόνο τα instantiations όλων των υπόλοιπων modules που ανέφερα παραπάνω. Αυτό γίνεται για να υπάρχει μία τάξη στον κώδικα και να είναι πιο εύκολη η αντιμετώπιση πιθανών προβλημάτων. Σε κάθε module υπάρχουν και τα ατάλληλα σχόλια για την καλύτερη και ευκολότερη κατανόηση του κειμένου.

Επαλήθευση

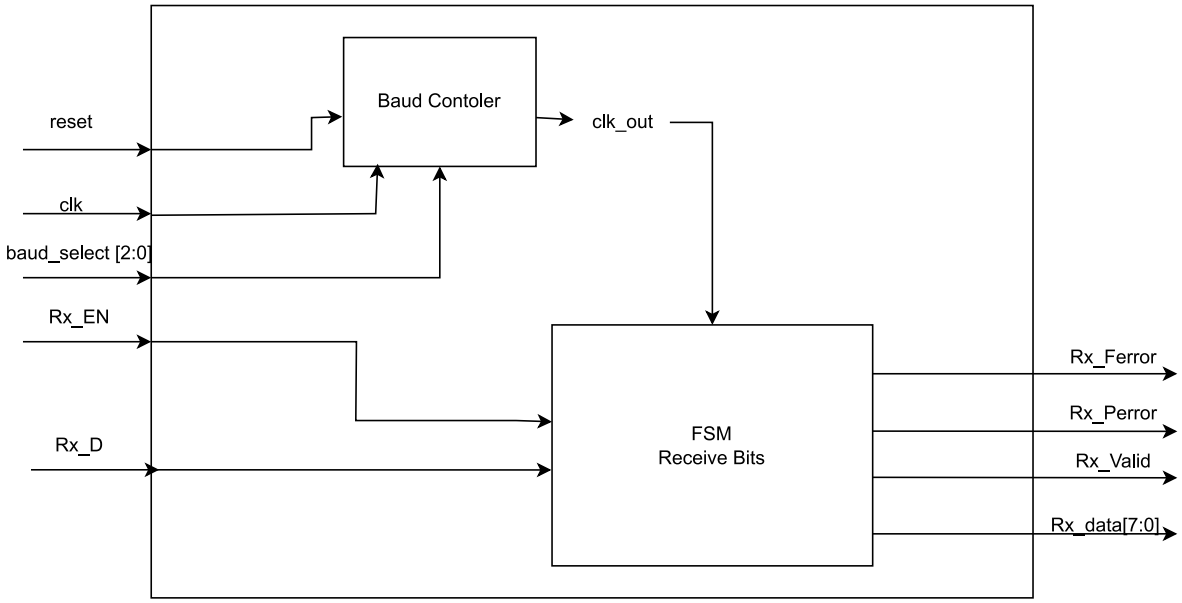
- Το testbench τεστάρει τον κώδικα στέλοντας 4 μυνήματα στην σειρά αφού ένα μύνημα έχει σταλθεί, όπως προτείνετε και στην εκφώνηση. Αυτό γίνεται αφού έχει οριστεί ένα ρολόι με περίοδο 100 MHz, το Tx_En=1 και το baud select έχει πάρει την επιθυμητή τιμή.
- Για να βεβαιωθώ ότι ο κώδικας έχει την επιθυμητή λειτουργία οι μεταβλητές όπου τσεκάρω είναι τα states της FSM, τον counter όπου χρησιμοποιεί η clk_multiplier. Γενικά ο κώδικας τεστάρετε όπως προτείνετε απο την εκφώνηση και θεωρώ πως καλύπτει ένα μεγάλο ποσοστό της υλοποίησης (>90%).
- ‘Οσο δοκίμαζα τον κώδικα προέκυψαν κάποια προβλήματα. Ένα πολύ εύκολο για να λυθεί ήταν ότι δεν έβαζα την sensitivity list της συνδυαστικής always όλες τις μεταβλητές όπου λειτουργούσαν σαν είσοδοι μέσα σε αυτή. Αυτό είχε ως αποτέλεσμα να μην προχωράει η FSM στο επόμενο state. ‘Επειτα είχα ένα προβλημα με το reset καθώς δεν έλεγχα για reset με το ρολόι της FSM αλλά με το output του clk_multiplier. Αυτό ειχες ως αποτέλεσμα όταν είχα πολύ μικρό Baud_select να μην προλαβαίνει να κάνει reset η FSM και να μην λειτουργέι σωστά το πρόγραμμα. Αυτό λύθηκε με την εξέταση για reset με το ρολόι της FSM.
- Παρακάτω σας παραδίδω ένα στιγμιότυπο απο τις κυματομορφές που είχε το testbench όπου φαίνονται και όσα ανέφερα απο πάνω:



Μέρος Γ - UART Δέκτης (Receiver)

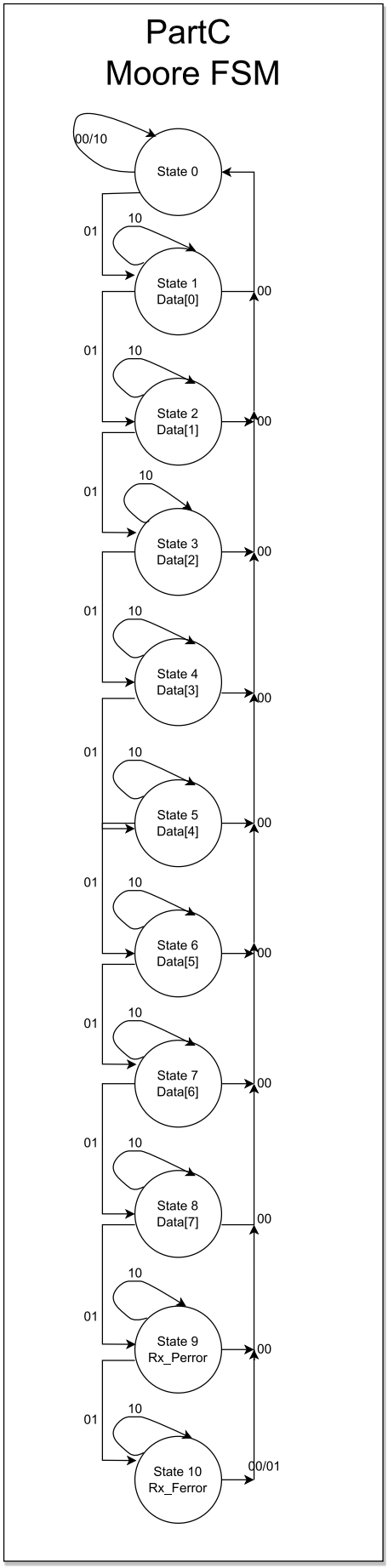
Υλοποίηση

- Στον κώδικα εμπεριέχονται:
    - Το κάλεσμα του baud controller(Part A) για τον transmitter
    - Receive_bits module: Είναι υπεύθνο για την παραλαβή των δεδομένω τα οποία προέρχονται από τον transmitter



- Στο module receive_bits εμπεριέχεται η FSM όπου είναι υπεύθυνη για την σειριακή παραλαβή των bits. Επέλεξα να κάνω την FSM moore γιατί μου φάνηκε πιο εύκολη η υλοποίηση της.
    - Σε μία εξωτερική βοηθητική για την FSM always, υλοποιώ τους κατάλληλους ελέγχους για την σωστή λειτουργία της FSM. Εκεί είναι που συγχρονίζω την δειγματοληψία του κάθε bit στην μέση της αποστολής και αλλάζω το select signal το οποίο είναι υπεύθυνο για την μετατόπιση της FSM στο επόμενο state. Το select αυτό έχει 3 καταστάσεις επειδή ο transmitter λειτουργεί με ρολόι 16 φορές πιο αργό απο αυτό του receiver. Οπότε το select γίνεται 2’b01 κάθε 16 κύκλους, έπειτα γίνεται 2’b10 όταν περιμένουμε να περάσουν αυτοί οι 16 κύκλοι που οφείλοταν στην διαφορετική λειτουργία μεταξυ αποστολέα και δέκτη. Στην περίπτωση που θέλουμε να επιστρέψουμε στο αρχικο state τότε ορίζω το select σε 2’b00. Επίσης σε αυτή την always αποθκεύω και εισερχόμενα bit σε μία προσωρινή μεταβλητή για να μπορέσω έπειτα να τα περάσω στο σύστημα παράλληλα εφόσον τηρούνται όλες οι προυποθέσεις.
    - Στην συνδυαστική always, αρχικοποιώ στην αρχή εκτός του case statement όσες μεταβλητές θέλω να αλλάζω για έναν μόνο κύκλο, για να μην χρειαστεί να τις ορίζω μέσα σε κάθε state. Ο σκοπός του κάθε state φαίνεται και στο σχετικό σχεδιάγραμμα που παραδίδω παρακάτω. Στα states που αποθηκεύω τα εισερχόμενα bit ορίζω μία μεβλητή ώς το bit όπου θέλω να αποθηκεύσω την κάθε μεταβλητή και το αποθηκεύω σε μία προσωρινή μεταβλητή στην εξωτερική always όπως ανέφερα και παραπάνω. Στο state 9 ελέγχω αν το parity bit που στάθηκε είναι ίδιο με το parity bit που υπολογίζω για τα εισερχόμενα bit. Αν διαφέρουν τότε θέτω το Rx_Perror σε 1 και επαναφέρω την FSM στο αρχικό της state. Διαφορετικά η FSM προχωράει στο επόμενο state(10) όπου ελέγχεται η ορθότητα του stop_bit. Άν το τελευταίο bit που στάλθηκε δεν είναι 1, σημαίνει ότι υπήρξε κάποιο πρόβλημα συγχρονισμού και θέτω το Rx_Ferror σε 1. Σε κάθε περίπτωση το επόμενο state της FSM είναι το αρχικό state 0.
    - Στην ακολυθιακή always, επιτρέπω την FSM να προχωρήσει στο επόμενο state εάν το Rx_EN είναι 1, αλλιώς την διατηρώ/επαναφέρω στο state 0.
    - Παρακάτω σας παραδίδω το dataflow της FSM.

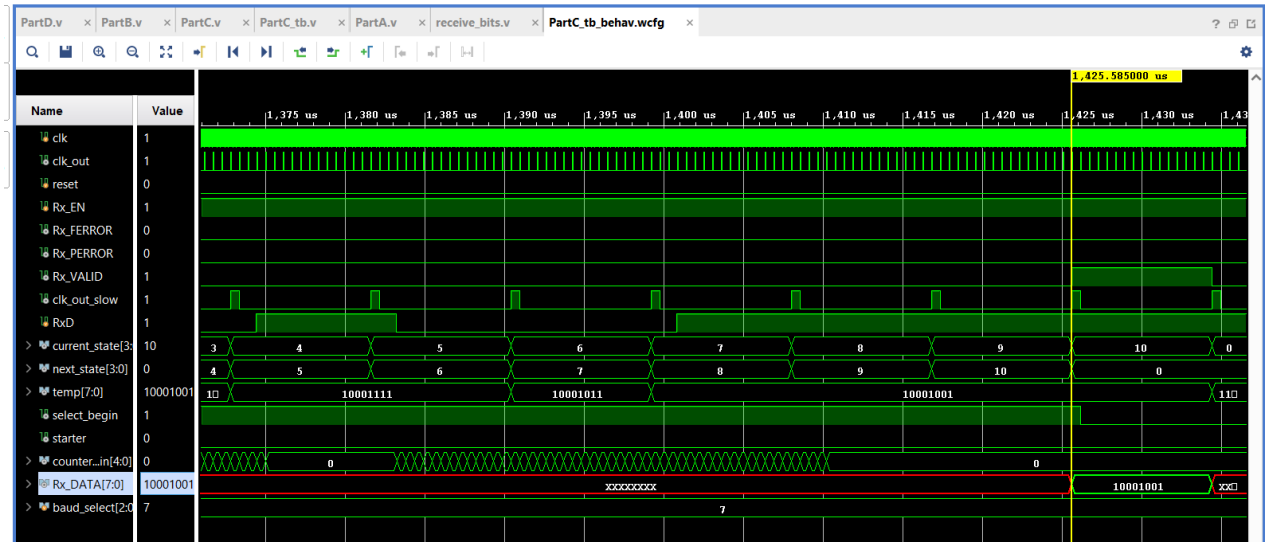
PartC
Moore FSM

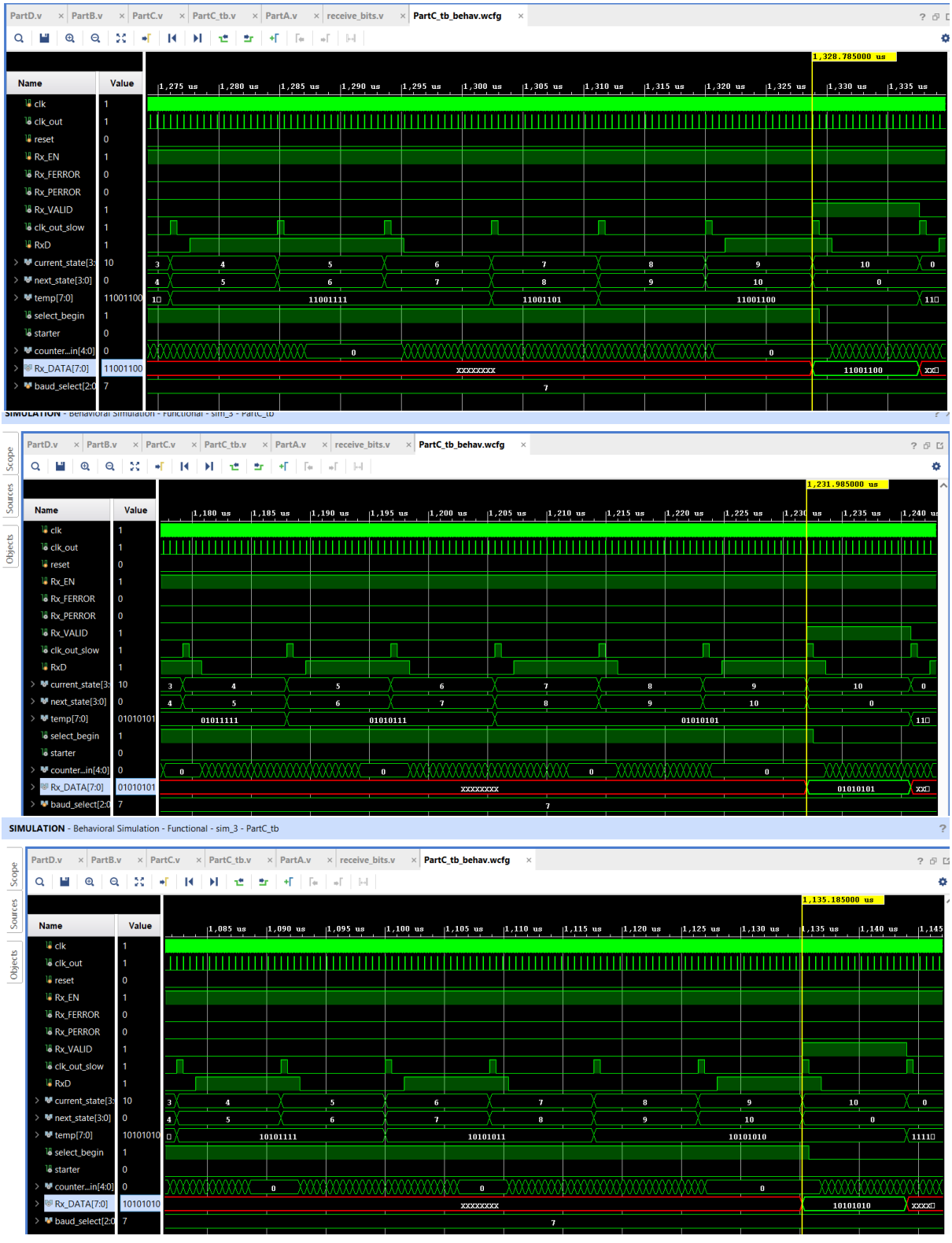


- Γενικά ο κώδικας αποτελείται από το main module (uart_receiver) το οποίο περιέχει μόνο τα instantiations όλων των υπόλοιπων modules που ανέφερα παραπάνω. Αυτό γίνεται για να υπάρχει μία τάξη στον κώδικα και να είναι πιο εύκολη η αντιμετώπιση πιθανών προβλημάτων. Σε κάθε module υπάρχουν και τα ατάλληλα σχόλια για την καλύτερη και ευκολότερη κατανόηση του κειμένου.

Επαλήθευση

- Το testbench έχει ως input τέσσερα συννεχόμενα μυνήματα (όπως προτείνετε και από την εκφώνση) και εξετάζει εάν το Rx_Data παίρνει την κατάλληλη τιμή στο τέλος της κάθε αποστολής.
- Για την ορθή εξέταση του κώδικα χρησιμοποιώ πάλι τις μεταβλητές της FSM για να δω ότι λειουργεί κανονικά, καθώς και τις μεταβλητές counter, temp, select και starter όπου χρησιμοποιούνται στην βοηθητική always της FSM. Για να ελέγξω και την σωστή λειτουργία των ελέγχων για τα Rx_Ferror και Rx_Perror, χρησιμοποίησα και κάποια επίτηδες λάθος μυνήματα εισόδου. Μετά από δοκιμές λειτουργούσαν και τα δύο ορθά. Το ποσοστό της εξέτασης του κώδικα είναι το ίδιο με τον transmitter δηλαδή τουλάχιστον 90%.
- Αντιμετώπισα ένα πρόβλημα συγχρονισμού των εισερχόμενων bit με την FSM. Οφειλότανε στην λάθος ανάθεση τιμών στην βοηθητική always. Το πρόβλημα επιλυθηκε με επιτυχία. Επίσης μου είχαν δημιουργηθεί Latch για την μεταβλητή temp, επειδή δεν της έθετα τιμή σε κάθε state της FSM. Αυτό το έλεισα θέτωντας της τιμή στην βοθητική always.
- Παρακάτω σας παραδίδω μία φωτογραφία απο τις κυματομορφές του testbench. Οι φωτογραφίες είναι 4 όσα και τα μυνήματα που στέλνω στον Δέκτη, για να φαίνεται με λεπτομέρεια η λειτουργία του σε κάθε περίπ τωση. ΤΟ RX_DATA ΕΊΝΑΙ ΕΠΙΤΗΔΕΣ Χ ΟΤΑΝ ΔΕΝ ΤΟΥ ΑΝΑΘΕΤΩ ΤΙΜΗ ΓΙΑ ΝΑ ΕΙΝΑΙ ΠΙΟ ΕΥΚΟΛΟ ΣΤΟ ΜΑΤΙ ΝΑ ΕΝΤΟΠΙΣΕΙ ΠΟΤΕ ΤΟ ΜΥΝΗΜΑ ΑΠΟΣΤΕΛΛΕΤΑΙ

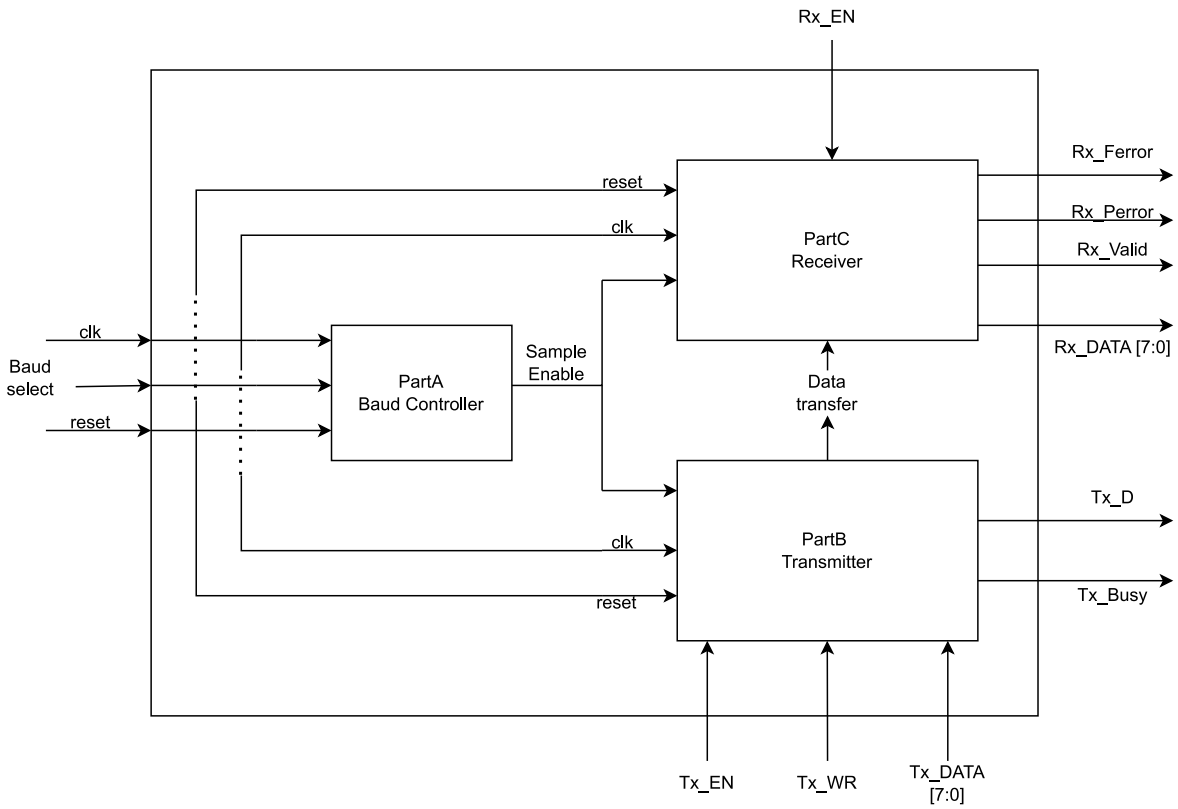




Μέρος Δ - UART (Receiever-Transmitter serial data transfer)

Υλοποίηση

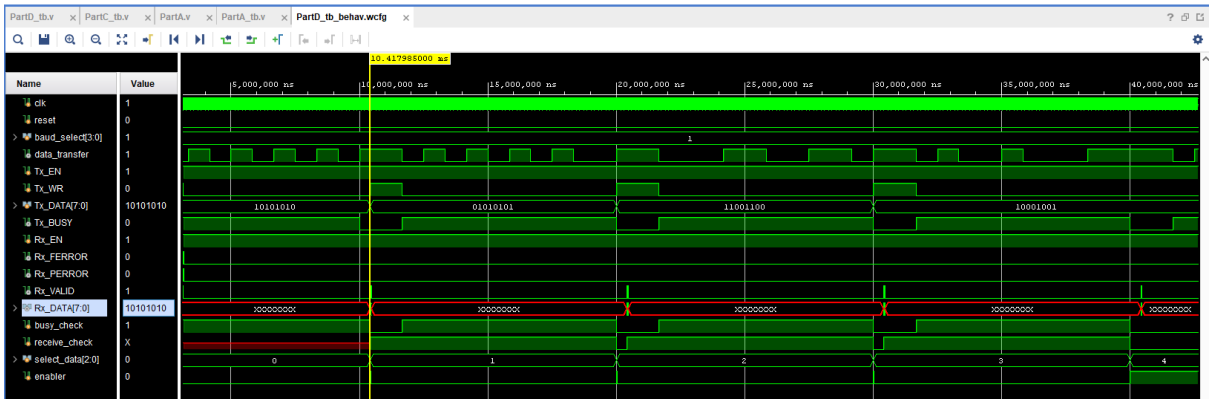
- Στον κώδικα εμπεριέχονται:
  - Uart Transmitter: Το Partb της εργασίας
  - Uart Receiver: Το PartC της εργασίας



- Ο κώδικας περιγράφεται με λεπτομέρεια στα 2 προηγούμενα Μέρη της εργασίας.
- Γενικά ο κώδικας αποτελείται από το main module (UART) το οποίο περιέχει μόνο τα instantiations όλων modules που ανέφερα παραπάνω. Σε κάθε module υπάρχουν και τα ατάλληλα σχόλια για την καλύτερη και ευκολότερη κατανόηση του κειμένου.

Επαλήθευση

- Πάλι η δαιδικασία εξέτασης είναι η εισαγωγή 4 μυνημάτων. Αυτή την φορά όμως δεν τα στέλνω ένα-ένα όπως στο μέρος Γ, αλλά παράλληλα όπως στο Μέρος Β, και εξετάζω αν το Rx_Data είναι ίδιο με το Tx_Data μετά την εκτέλεση των FSM. Αυτή την φορά επίσης εκτελώ το testbench με always και όχι με initial γιατί η εκφώνηση λέει ότι το επόμενο μύνημα πρέπει να στέλνετε με τα μυνήματα που επικοινωνουνε τα δύο modules και όχι με προκαθορισμένο χρόνο. Έτσι λοιπόν το επόμενο μύνημα στέλνετε μόνο όταν τα Tx_busy και Rx_Valid/Rx_Ferror/Rx_Perror ενεργοποιηθούν. Αυτό σημαίνει ότι τα μυνήματα έχουν σταλθεί από τον αποστολέα και έχουν ληφθεί από τον δέκτη.
- Το δυάνυσμα δειγματοληψίας είναι το data_transfer το οποίο είναι το καλώδιο που ενώνει το Tx_D και το Rx_D. Το ποσοστό κάλυψης ελέγχου του κώδικα είναι το ίδιο με τα προηγούμεναμ, δηλαδή τουλάχιστον 90%.
- Είχα πρόβλημα με τον συγχρονισμό των 2 modules επειδή δεν είχα 16 φορές πιο αργή την αποστολή μυνημάτων απο τον αποστολέα.
- Παρακάτω παραδίδω μία εικόνα απο τις κυματομορφές της προσομοίωσης. ΤΟ RX_DATA ΕΊΝΑΙ ΕΠΙΤΗΔΕΣ Χ ΟΤΑΝ ΔΕΝ ΤΟΥ ΑΝΑΘΕΤΩ ΤΙΜΗ ΓΙΑ ΝΑ ΕΙΝΑΙ ΠΙΟ ΕΥΚΟΛΟ ΣΤΟ ΜΑΤΙ ΝΑ ΕΝΤΟΠΙΣΕΙ ΠΟΤΕ ΤΟ ΜΥΝΗΜΑ ΑΠΟΣΤΕΛΛΕΤΑΙ



Συμπέρασμα

Γενικά ήταν πολύ ενδιαφέρον εργασία και με παροτρυνε να την τελειώσω. Γενικά πήγε αρκετά καλά το γράψιμο του κώδικα. Αυτό που με βοήθησε ότι είχα καταλάβει εξαρχής ποιο ήταν το ζητούμενο καθώς σαν concept θεωρώ δεν είναι κάτι τρομερό. Επίσης με βοήθησε είναι ότι στα εργαστήρια πήγαινα έτοιμος με ερωτήσεις και δεν τα έκανα τελευταία στιγμή. Εκεί ήταν και που κατάλαβα το λάθος μου στο Μέρος Δ όπου έπρεπε να αλλάξω ένα μεγάλο μέρος του Γ, παρόλο που απο μόνο του λειτουργούσε σωστά. Ήθελα να προλάβω και το προεραιτικό αλλά δεν τα κατάφερα.