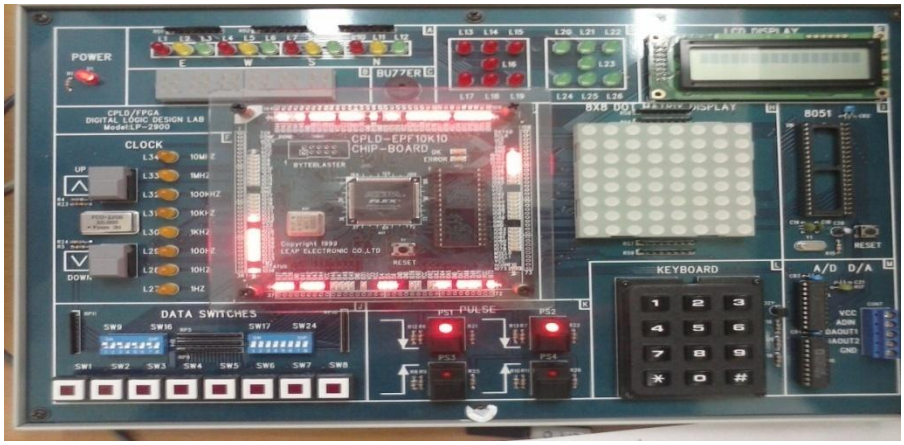
ΤΕΙ Κεντρικής Μακεδονίας, Σέρρες

Τμήμα Μηχανικών Πληροφορικής Τ.Ε.

Πτυχιακή Εργασία

Τίτλος:

Σχεδίαση ελεγκτή ασύγχρονης σειριακής επικοινωνίας σε γλώσσα VHDL και υλοποίηση σε διάταξη FPGA.



Ονοματεπώνυμο: Στεργίου Παναγιώτα, ΑΕΜ: 2297
Επιβλέπων καθηγητής: Καλόμοιρος Ιωάννης

ΣΕΡΡΕΣ 2013

Περίληψη

Η παρούσα πτυχιακή εργασία, αναφέρεται στην υλοποίηση ενός ελεγκτή ασύγχρονης σειριακής επικοινωνίας που δίνει τη δυνατότητα σε έναν υπολογιστή να ανταλλάσει δεδομένα με μια διάταξη FPGA. Η εφαρμογή σχεδιάστηκε σε γλώσσα περιγραφής υλικού VHDL και υλοποιήθηκε σε διάταξη FLEX10K.

Αρχικά σχεδιάστηκε και υλοποιήθηκε ο πομπός της ασύγχρονης σειριακής επικοινωνίας (transmitter) και δοκιμάστηκε με τον Ηλεκτρονικό Υπολογιστή να λειτουργεί ως δέκτης. Στη συνέχεια γίνεται το αντίθετο, ο Η/Υ είναι ο πομπός και η διάταξη FPGA λειτουργεί ως δέκτης (receiver).

Η ορθή λειτουργία των κυκλωμάτων επαληθεύεται με τις προσομοιώσεις που έγιναν με την βοήθεια του προγράμματος ψηφιακής σχεδίασης και προσομοίωσης Quartus II της εταιρίας Altera. Η εφαρμογή ξενιστή που εκτελείται σε Η/Υ υλοποιήθηκε με χρήση του λογισμικού μετρήσεων LabView της εταιρίας National Instruments.

Περιεχόμενα

Κεφάλαιο 1
«Προγραμματιζόμενες λογικές διατάξεις»

Τι τελευταίες δεκαετίες εμφανίστηκαν τα ολοκληρωμένα κυκλώματα προγραμματιζόμενης λογικής, που δίνουν τη δυνατότητα στο σχεδιαστή του υλικού να δημιουργεί τις δικές του υλοποιήσεις με τη μορφή ολοκληρωμένου κυκλώματος, χρησιμοποιώντας τα κατάλληλα εργαλεία σχεδίασης. Με τον τρόπο αυτό μπορούν να πρωτοτυποιηθούν και να δοκιμαστούν πολύπλοκες ψηφιακές σχεδιάσεις, που σε ορισμένες περιπτώσεις μπορεί να χρησιμοποιηθούν και σε καταναλωτικά προϊόντα. Τα σύγχρονα κυκλώματα προγραμματιζόμενης λογικής περιέχουν πολλά λογικά στοιχεία, ώστε έχει πλέον ξεπεραστεί ο περιορισμός των πόρων υλικού που ίσχυε παλιότερα σε τέτοια κυκλώματα. Τα προγραμματιζόμενα λογικά κυκλώματα υπακούουν σε μια δομή που δεν είναι συγκεκριμένη, αλλά διαμορφώνεται κατάλληλα από τον χρήστη.

Οι ***διατάξεις προγραμματιζόμενης λογικής (programmable logic devices-PLD)*** πρωτοεμφανίστηκαν στην δεκαετία του 1970. Τα PLDs χωρίζονται σε δυο κατηγορίες, τα απλά PLDs (SPLDs-Simple programmable logic devices) και τα πολύπλοκα PLDs (CPLDs-Complexity programmable logic devices). Η διαφορά τους είναι στην πολυπλοκότητά τους. Τα SPLDs είναι αυτόνομα ολοκληρωμένα κυκλώματα που προγραμματίζονται το καθένα ξεχωριστά, ενώ τα CPLDs αποτελούνται από πολλά (όχι μόνο από ένα) SPLDs που προγραμματίζονται όλα μαζί, γιατί αποτελούν μέρος της αρχιτεκτονικής του.

Για την υλοποίηση μεγαλύτερων κυκλωμάτων χρησιμοποιούμε ένα ολοκληρωμένο κύκλωμα με μεγαλύτερη χωρητικότητα, την πλέον εξελιγμένη εκδοχή των PLDs, τις διατάξεις FPGA.

1.1 Τι είναι το FPGA

Το ***FPGA ή Field Programmable Gate Array ή συστοιχία επιτόπια προγραμματιζομενων πυλών*** είναι ένας τύπος προγραμματιζόμενου ολοκληρωμέ-νου κυκλώματος γενικής χρήσης το οποίο διαθέτει πολύ μεγάλο αριθμό τυποποιημένων πυλών και άλλων ψηφιακών λειτουργιών όπως απαριθμητές, καταχωρητές μνήμης, γεννήτριες PLL κα. Σε ορισμένα από αυτά ενσωματώνονται και αναλογικές λειτουργίες. Κατά τον προγραμματισμό του FPGA, ο οποίος γίνεται πάντοτε ενώ αυτό είναι

τοποθετημένο στο τυπωμένο κύκλωμα, ενεργοποιούνται οι επιθυμητές λειτουργίες και διασυνδέονται μεταξύ τους έτσι ώστε το FPGA να συμπεριφέρεται ως ολοκληρωμένο κύκλωμα με συγκεκριμένη λειτουργία. Ο κώδικας με τον οποίο προγραμματίζεται το FPGA γράφεται σε γλώσσες περιγραφής υλικού (VHDL, AHDL, Verilog).

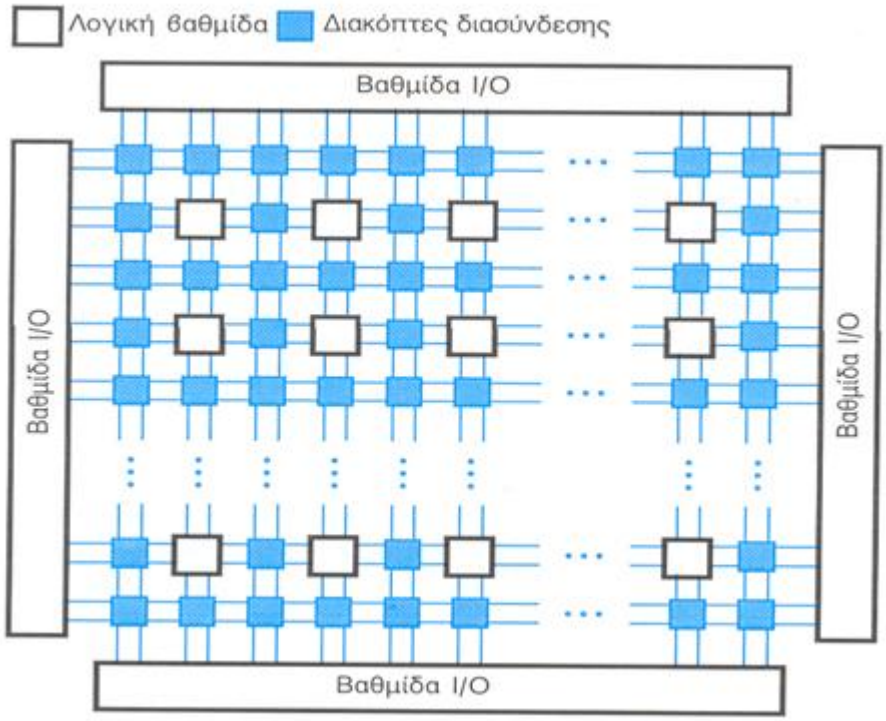
Το FPGA έχει παρόμοιο πεδίο εφαρμογών με άλλα προγραμματιζόμενα ολοκληρωμένα ψηφιακά κυκλώματα όπως τα ***PLD (Programmable Logic Device-ολοκληρωμένο κύκλωμα ημιαγωγών)*** και τα ***ASIC (Application Specific Integrated Circuit-ολοκληρωμένο κύκλωμα για συγκεκριμένες εφαρμογές)*** . Όμως τα ιδιαίτερα χαρακτηριστικά του FPGA είναι τα εξής:

- Το FPGA χάνει τον προγραμματισμό του κάθε φορά που διακόπτεται η τάση τροφοδοσίας του. Επομένως απαιτεί εξωτερικό μικροεπεξεργαστή ή μνήμη με μόνιμη συγκράτηση δεδομένων (non-volatile memory) από τα οποία θα προγραμματίζεται, κάθε φορά που επανέρχεται η τάση τροφοδοσίας.
- Ο προγραμματισμός του FPGA μπορεί να αλλάζει κάθε φορά που τροποποιείται το λογισμικό του μικροεπεξεργαστή ή τα δεδομένα της μνήμης που το ελέγχει.
- Δεν υπάρχει όριο στο πόσες φορές μπορεί να επαναπρογραμματιστεί.
- Η κατανάλωση ισχύος είναι σημαντικά αυξημένη, σε σχέση με τα ASIC.

Δηλαδή το FPGA το χρησιμοποιούμε περισσότερο όταν οι παράμετροι λειτουργίας πρέπει να αλλάζουν συχνά και όταν έχουμε μικρή παραγωγή. Σε αντίθεση το ASIC επειδή χρησιμοποιείται ευρέως έχει χαμηλότερο κόστος όταν απαιτείται μεγάλη ποσότητα και για το λόγω του ότι δεν επαναπρογραμματίζεται όπως ένα FPGA η λειτουργία του είναι αυστηρά προκαθορισμένη.

Βασική δομική μονάδα του FPGA είναι το λογικό μπλοκ, με τη χρήση του οποίου υλοποιούνται οι λογικές συναρτήσεις που εκφράζουν τη λειτουργία ενός ψηφιακού κυκλώματος. Ανάλογα με το μέγεθος του κυκλώματος πολλά λογικά μπλοκ συνδέονται για να υλοποιήσουν το πλήθος των απαραίτητων λογικών συναρτήσεων.

Στο παρακάτω σχήμα (Σχήμα 1.1) βλέπουμε ένα προγραμματιζόμενο κύκλωμα εισόδου(I/O block)στο οποίο τα λευκά τετράγωνα είναι οι



Σχήμα 1.1 Γενική δομή μιας διάταξης FPGA

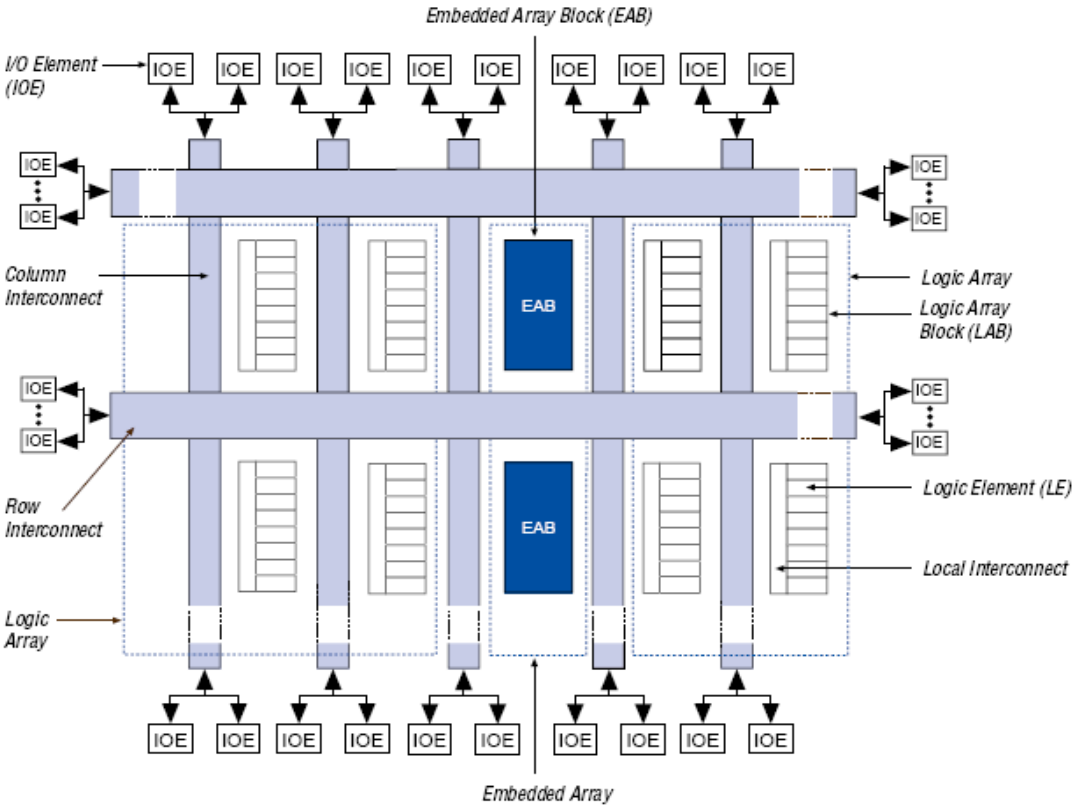
στοιχειώδεις βαθμίδες λογικής(λογικά στοιχεία), ενώ τα μπλε αποτελούν τον προγραμματιζόμενο πίνακα διασυνδέσεων.

1.1.1 Μια διάταξη FPGA τύπου FLEX10K της εταιρείας Altera

Το FLEX10K είναι ένα ολοκληρωμένο κύκλωμα της οικογένειας της Altera το οποίο προσφέρει ευελιξία της παραδοσιακής προγραμματιζόμενης λογικής σε συνδυασμό με την αποτελεσματικότητα και την πυκνότητα των ενσωματωμένων διατάξεων των πυλών. Με τις δύο μοναδικές λογικές δομές της, την ενσωματωμένη και την λογική διάταξη, η οικογένεια FLEX10K επαναστατεί στις προγραμματιζόμενες αρχιτεκτονικές και φέρνει στην αγορά την προγραμματιζόμενη λογική.

Η οικογένεια FLEX10K κυμαίνεται από 10.000 έως 250.000 τυπικές πύλες και χωρίζεται σε τρεις γενιές. Κάθε επόμενη γενιά προσφέρει υψηλότερη απόδοση, χαμηλότερο κόστος και χαμηλότερη κατανάλωση ενέργειας απ' ότι η προηγούμενη. Επίσης παρέχει μια σειρά από συσκευές που ταιριάζουν σε διαφορετικές ανάγκες σχεδιασμού.

Παρακάτω στο Σχήμα 1.2 δίνεται η εικόνα μιας δομής ολοκληρωμένου κυκλώματος FLEX10K. Αυτό περιέχει μια συλλογή βαθμίδων LAB,όπου η



Σχήμα 1.2 Γενική δομή διάταξης FPGA της οικογένειας FLEX10K

κάθε λογική βαθμίδα αποτελείται από οκτώ λογικά στοιχεία, τα οποία στηρίζονται σε πίνακες αναφοράς (lookup tables, LUTs). Εκτός από τις βαθμίδες LAB, το ολοκληρωμένο κύκλωμα περιέχει και βαθμίδες ενσωματωμένων διατάξεων(embedded array blocks, EABs), που είναι βαθμίδες της SRAM μνήμης. Οι βαθμίδες αυτές μπορούν να συνδεθούν με γραμμές και με στήλες γραμμών.

1.2 Εργαλεία σχεδίασης

Το να σχεδιάζεις PLD,CPLD ή FPGA είναι απαραίτητη η χρήση CAD(computer-aided design) εργαλείων σχεδίασης, ειδικά αν πρόκειται για τη σύνθεση του κυκλώματος. Τα πιο γνωστά και διαδεδομένα προγράμματα σχεδίασης κυκλωμάτων όπως και γλώσσας περιγραφής υλικού(HDL) είναι η ABEL, η VHDL και η Verilog.

Πολλά είναι τα εργαλεία σχεδίασης τα οποία εκτός από τη σύνθεση ενός κυκλώματος μπορούν να διαμορφώνουν και να διασυνδέουν λογικά μπλοκ όπως επίσης και να διαιρούν σε επιμέρους κομμάτια, στην ουσία σε διαμορφούμενα και διασυνδεόμενα λογικά μπλοκ ενός FPGA, την περιγραφή σχεδίασης της HDL, έτσι ώστε το FPGA να μπορεί να υλοποιήσει την συνολική σχεδίαση.

Εμείς εδώ θα αναλύσουμε μια από τις γλώσσες περιγραφής υλικού την οποία χρησιμοποιούμε και στη συνέχεια για την σχεδίαση του ελεγκτή μας, την VHDL, του προγράμματος Quartus II της Altera.

1.2.1 Λίγα λόγια για το Quartus II.

Για να χρησιμοποιήσουμε τη γλώσσα περιγραφής υλικού VHDL θα χρειαστούμε το σχεδιαστικό λογισμικό της Altera, Quartus II. Αυτό δίνει τη δυνατότητα για εύκολο σχεδιασμό, προσομοίωση και υλοποίηση σχεδιάσεων λογικής με διαφορετικό βαθμό πολυπλοκότητας.

Αποτελείται από διάφορες εφαρμογές και κάποια από τα βασικά χαρακτηριστικά του περιλαμβάνουν:

- Υλοποίηση της VHDL και Verilog για την περιγραφή υλικού
- Απεικόνιση των λογικών κυκλωμάτων
- και προσομοίωση κυματομορφής (vector waveform).

Τέλος στο Quartus II εκτός από το σχεδιασμό λογικών κυκλωμάτων και το σχεδιασμό με εργαλεία φυσικής σχεδίασης μπορούν να υλοποιηθούν και αριθμητικά κυκλώματα.

1.3 Εισαγωγή στη γλώσσα VHDL

Η γλώσσα **VHDL (Very High Speed Integrated Circuit Hardware Description Language)** είναι μια από τις δύο μορφές γλωσσών HDL των προτύπων του IEEE. Είχε αρχικά διπλό σκοπό: Αρχικά είναι μια γλώσσα κειμένου, δηλαδή, περιγράφει τη δομή περίπλοκων ψηφιακών κυκλωμάτων. Ως επίσημο πρότυπο IEEE, η γλώσσα VHDL παρείχε έναν κοινό τρόπο καταγραφής κυκλωμάτων, σχεδιασμένων από διάφορους σχεδιαστές, σε μορφή κειμένου. Κατά δεύτερο λόγο, η γλώσσα VHDL έδινε τη δυνατότητα για μοντελοποίηση της συμπεριφοράς των ψηφιακών κυκλωμάτων και έτσι μπορούσε να χρησιμοποιηθεί ως είσοδος σε προγράμματα λογισμικού που προσομοίωναν την συμπεριφορά των ψηφιακών κυκλωμάτων.

Τα τελευταία χρόνια, πέρα από την χρήση της για καταγραφή και προσομοίωση, η γλώσσα VHDL χρησιμοποιείται επίσης ευρέως για την εισαγωγή σχεδίων σε συστήματα σχεδίασης CAD. Τα εργαλεία των προγραμμάτων CAD χρησιμοποιούνται για να συνθέσουν με την βοήθεια της γλώσσας VHDL την υλοποίηση των κυκλωμάτων που περιγράφονται από την γλώσσα.

Σε αντίθεση με τα σχηματικά διαγράμματα, η γλώσσα VHDL περιέχει έναν αριθμό πλεονεκτημάτων. Επειδή υποστηρίζεται από τις περισσότερες εταιρίες που προσφέρουν τεχνολογία ψηφιακών συστημάτων, η γλώσσα VHDL παρέχει *φορητότητα(portability),*δηλαδή *συμβατότητα σχεδίασης.* Ένα κύκλωμα που ορίζεται στη γλώσσα VHDL μπορεί να υλοποιηθεί με διάφορα ολοκληρωμένα κυκλώματα και με εργαλεία σχεδίασης CAD διαφόρων εταιριών, χωρίς να πρέπει να αλλάξουμε τις προδιαγραφές της γλώσσας VHDL. Η φορητότητα σχεδίασης είναι σημαντική επειδή η τεχνολογία των ψηφιακών κυκλωμάτων εξελίσσεται ραγδαία. Χρησιμοποιώντας μια πρότυπη γλώσσα, ο σχεδιαστής μπορεί να επικεντρωθεί στη λειτουργικότητα που απαιτείται από το κύκλωμα του και να μην ασχοληθεί με την τεχνολογία που θα χρησιμοποιηθεί για την τελική υλοποίηση αυτού.

Η εισαγωγή σχεδίασης ενός λογικού κυκλώματος πραγματοποιείται γράφοντας κάποιο πρόγραμμα, ή όπως λέγεται κάποιο κώδικα(code), στη γλώσσα VHDL. Τα σήματα που διαβιβάζονται στα κυκλώματα παριστάνονται ως μεταβλητές στον πηγαίο κώδικα και οι λογικές συναρτήσεις εκφράζονται ως δηλώσεις τιμών σε αυτές τις μεταβλητές. Ο πηγαίος κώδικας της γλώσσας VHDL έχει τη μορφή απλού κειμένου και έτσι είναι εύκολο στον σχεδιαστή να τον περιλάβει στα συνοδευτικά κείμενα του κυκλώματος που σχεδίασε, ώστε να μπορεί να γίνεται κατανοητή η λειτουργία του κυκλώματος. Η δυνατότητα αυτή σε συνδυασμό με το γεγονός της ευρείας χρήσης της γλώσσας VHDL, ενθαρρύνει τη χρήση και επαναχρησιμοποίηση κυκλωμάτων που έχουν σχεδιαστεί σε αυτήν τη γλώσσα. Έτσι επιτρέπεται η ταχύτερη ανάπτυξη νέων προϊόντων σε περίπτωση που ο υπάρχων κώδικας VHDL μπορεί να τροποποιηθεί, ώστε να περιγράψει νέα κυκλώματα.

Ομοίως με αυτόν τον τρόπο μπορούμε να μεταχειριστούμε τα μεγάλα κυκλώματα με την μέθοδο των σχηματικών διαγραμμάτων, έτσι και στη γλώσσα VHDL μπορούμε να γράψουμε τον κώδικα με τέτοιο τρόπο ώστε η σχεδίαση να είναι ιεραρχική. Έτσι μπορούμε να παραστήσουμε αποδοτικά με κώδικα της γλώσσα VHDL μεγάλα και μικρά κυκλώματα. Η γλώσσα VHDL έχει χρησιμοποιηθεί για την περιγραφή μεγάλων κυκλωμάτων όπως είναι οι μικροεπεξεργαστές, που διαθέτουν εκατομμύρια τρανζίστορ.

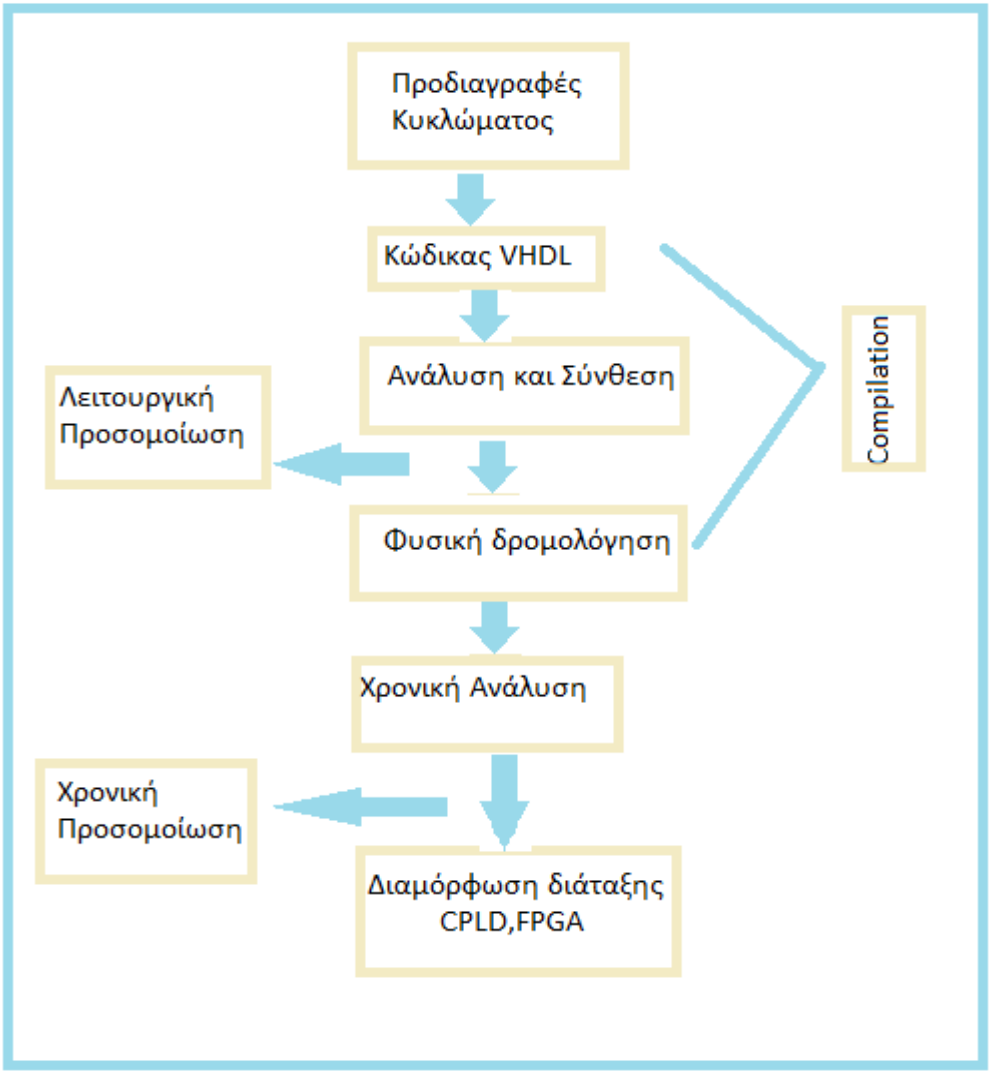
Η εισαγωγή σχεδίασης με την βοήθεια της γλώσσας VHDL μπορεί να συνδυαστεί με άλλες μεθόδους. Για παράδειγμα, μπορεί να χρησιμοποιηθεί το εργαλείο των σχηματικών διαγραμμάτων, στο οποίο για την περιγραφή ενός υποκυκλώματος χρησιμοποιείται η γλώσσα VHDL.

1.3.1 Ροή σχεδίασης της VHDL

Η σχεδίαση της γλώσσας VHDL γίνεται σε επιμέρους κομμάτια με κανόνες ιεραρχικής σχεδίασης. Αρχικά λαμβάνονται υπόψη οι προδιαγραφές του κυκλώματος. Στην συνέχεια γίνεται η συγγραφή του κώδικα VHDL με τα κατάλληλα εργαλεία σχεδίασης ο οποίος περιγράφει ακριβώς τα διάφορα δομικά στοιχεία. Αμέσως μετά ακολουθεί το compilation, η 'συμβολομετάφραση' δηλαδή του κώδικα VHDL. Ο compiler εδώ, βλέπει για τυχόν συντακτικά λάθη και ελέγχει αν ο κώδικας μας είναι συμβατός με τα τμήματα του κώδικα από τον οποίο εξαρτάται.

Η προσομοίωση είναι το αμέσως επόμενο σήμα. Δίνοντας τις κατάλληλες τιμές στις εισόδους, παίρνουμε τις επιθυμητές τιμές στις εξόδους.

Το Verification είναι ο έλεγχος της σωστής λειτουργίας του κυκλώματος μας, χωρίζεται σε Functional και σε Timing. Στο Functional Verification



Σχήμα 1.3 Βασική ροή εργασιών κατά τη σχεδίαση με τη γλώσσα VHDL

η προσομοίωση μας λειτουργεί ανεξάρτητα από τον χρόνο, ενώ στο Timing Verification χρόνος και οι χρονικές καθυστερήσεις λαμβάνονται υπόψη και έτσι γίνεται και ο έλεγχος.

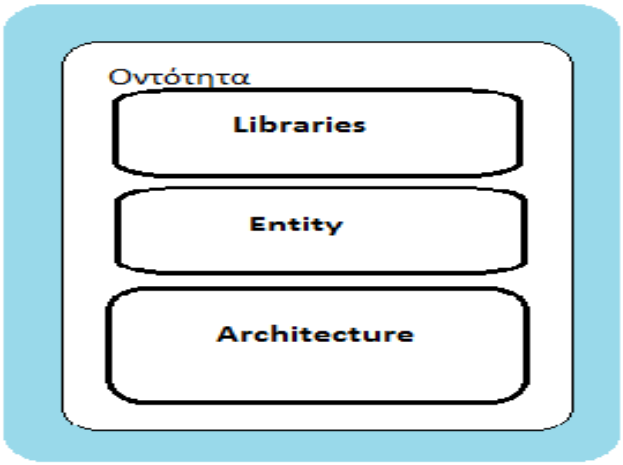
Τέλος γίνεται η σύνθεση του κώδικα. Δηλαδή από περιγραφή VHDL γίνεται μετάφραση σε δομικά στοιχεία που φτιάχνουν την τεχνολογία του κυκλώματος.

Η βασική ροή των εργασιών κατά τη σχεδίαση με τη γλώσσα VHDL φαίνεται στο παρακάτω διάγραμμα.

1.3.2 Ο κώδικας VHDL

Για να σχεδιάσει κάποιος ένα ηλεκτρονικό σχέδιο σε κώδικα γλώσσας VHDL και ταυτόχρονα αυτός ο κώδικας να είναι πιο εύκολα κατανοητός πρέπει αρχικά να χωρίσει το σχέδιο του σε τμήματα ή αλλιώς οντότητες.

Μία οντότητα χωρίζεται σε τρία κυρίως μέρη: την δήλωση βιβλιοθηκών στο αρχικό κομμάτι της(libraries), την δήλωση της οντότητας μετά (entity) και



Σχήμα 1.4 Βασικά τμήματα ενός αρχείου VHDL

τέλος την αρχιτεκτονική στην οποία υπάρχουν οι περαιτέρω λεπτομέρειες για την λειτουργία της οντότητάς μας.

Στο πρώτο μέρος του κώδικα γίνεται η δήλωση της βιβλιοθήκες η οποία περιλαμβάνει κάποια προτυποποιημένων πακέτων από το ινστιτούτο IEEE. Η πλέον γνώστη βιβλιοθήκη είναι η ieee και η δήλωσή της γίνεται με την λέξι-κλειδί LIBRARY.

***LIBRARY** όνομα_βιβλιοθήκης;*

πχ. LIBRARY ieee;

Ο κάθε χρήστης μπορεί βέβαια να δημιουργήσει και την δική του βιβλιοθήκη.

Τα πακέτα ,τώρα, εισάγονται με την λέξη-κλειδί USE και λειτουργούν ως «αποθήκες» μέσα στις οποίες συγκρατιούνται προγράμματα της γλώσσας VHDL είναι γενικής χρήσης, όπως τα προγράμματα που ορίζουν έναν τύπο.

USE** όνομα_βιβλιοθήκης.όνομα_πακέτου**.all;

πχ. USE ieee.std_logic_1164.all;

Μέσα σε μια οντότητα περιγράφεται το κύκλωμα ως μια βαθμίδα εισόδων και εξόδων. Δηλαδή γίνεται η δήλωση των σημάτων εισόδου-εξόδου τα οποία χρησιμοποιούμε ως σήματα στην μονάδα μας.

ENTITY** όνομα_οντότητας **IS
***PORT(**όνομα_σήματος1: τρόπος_λειτουργίας τύπος_σήματος1;*
όνομα_σήματος2: τρόπος_λειτουργίας τύπος_σήματος2;
.
.
.
*όνομα_σήματοςN: τρόπος_λειτουργίας τύπος_σήματοςN**);***
END** όνομα_οντότητας**;

Ο τρόπος λειτουργίας αφορά τον τρόπο με τον οποίο λειτουργεί το σήμα μας, δηλαδή αν είναι IN, OUT, BUFFER.

Ο τύπος του σήματος μπορεί να είναι bit, real, integer κτλ.

Τέλος, η αρχιτεκτονική μιας οντότητας περιλαμβάνει όλες τις λειτουργίες του κυκλώματος μας, δηλαδή με ποιες λογικές συναρτήσεις θα λαμβάνουν τιμές τα σήματά μας σε κάθε χρονική στιγμή.

ARCHITECTURE** όνομα_αρχιτεκτονικής **OF** όνομα_οντότητας **IS
[Δηλώσεις επιπλέον σημάτων]
BEGIN
[Δηλώσεις επιπλέον σημάτων]
Εντολές που περιγράφουν λογικές λειτουργίες και αναθέτουν τιμές σε σήματα.
END** όνομα_αρχιτεκτονικής**;

Το όνομα_αρχιτεκτονικής μπορεί να είναι είτε ένα όνομα που έχουμε επιλέξει εμείς είτε ένα όνομα που θα χαρακτηρίζει την λειτουργία του

κυκλώματός μας, ***behavior*** για απλή περιγραφή της σχέσης της εισόδου με την έξοδο ή ***Structure*** για περιγραφή του κυκλώματος με περισσότερες ενότητες ή πύλες τα οποία συνδέονται όλα μαζί για να δώσουν την επιθυμητή λειτουργία.

Το όνομα_οντότητας προέρχεται από το ENTITY για να μπορέσουμε να τα συνδέσουμε μεταξύ τους.

Όσο αφορά τις δηλώσεις των επιπλέον σημάτων, εδώ αυτά τα σήματα μπορεί να είναι SIGNAL, CONSTANT,TYPE όπως επίσης μπορεί να είναι και ένα ή παραπάνω COMPONENT.

1.3.3 Τα αντικείμενα δεδομένων και τα υποκυκλώματα

Τα αντικείμενα είναι αυτά που μεταφέρουν την πληροφορία και που αποδίδουν τιμές σε διάφορα σημεία μέσα στον κώδικά μας.

Πιο συγκεκριμένα τα σήματα(**SIGNALS**) είναι «καλώδια» που χρησιμοποιούνται για την μεταφορά των δεδομένων μέσα στον κώδικά μας. Οι σταθερές(**CONSTANTS**) περιέχουν μια σταθερή τιμή η οποία δεν αλλάζει καθ' όλη την διάρκεια του simulation και μπορεί να χρησιμοποιηθεί σε όλο το architecture αρκεί να έχει δηλωθεί στην αρχή του. Τέλος οι μεταβλητές(**VARIABLES**) χρησιμοποιούνται για την μεταφορά δεδομένων μεταξύ διαφόρων σειριακών διεργασιών, είναι κάτι σαν προσωρινές θέσεις μνήμης και μπορούν να δηλωθούν μόνο μέσα σε μια εντολή διαδικασίας(PROCESS).

Ένα υποκύκλωμα ή αλλιώς ένα ***COMPONENT*** είναι ένα μικρότερο από το αρχικό κύκλωμα που έχει ήδη σχεδιαστεί από τον χρήστη ή υπάρχει ήδη μέσα στο λογισμικό σχεδίασης για να μπορεί να χρησιμοποιείται αρκετές φορές.

COMPONENT όνομα_συνιστώσας
[GENERIC(όνομα_παραμέτρου:integer:=τιμή;]
PORT(όνομα_ακροδέκτη1: **mode** τύπος;
όνομα_ακροδέκτη2: **mode** τύπος;
.
.
όνομα_ακροδέκτη: **mode** τύπος);
END COMPONENT;

Στο όνομα_συνιστώσας έχουμε το όνομα του component.

Μέσα στο κύριο πρόγραμμά μας πρέπει να δηλώσουμε αυτό το υποκύκλωμα σαν ένα στιγμιότυπο και αυτό γίνεται με τον εξής τρόπο:

Όνομα_στιγμιοτύπου: όνομα_συνιστώσας **PORT MAP (**όνομα_σημάτων).

1.3.4 Η εντολή PROCESS

Τέλος πρέπει να αναφερθούμε στις εντολές διαδικασίας. Μια εντολή διαδικασίας ή αλλιώς μια διεργασία είναι ένα τμήμα του ακολουθιακού κώδικα και είναι ο κυριότερος τρόπος για την περιγραφή των σειριακών λειτουργιών. Αυτό συμβαίνει γιατί με την αλλαγή της κατάστασης των σημάτων που βρίσκονται μέσα σε μία διεργασία γίνεται η εκτέλεση αυτής. Η δήλωσή της γίνεται όπως παρακάτω:

PROCESS
BEGIN
Ακολουθιακές προτάσεις
END PROCESS;

Κεφάλαιο 2
«Τα είδη της σειριακής επικοινωνίας και άλλες βασικές έννοιες»

2.1 Σειριακή επικοινωνία

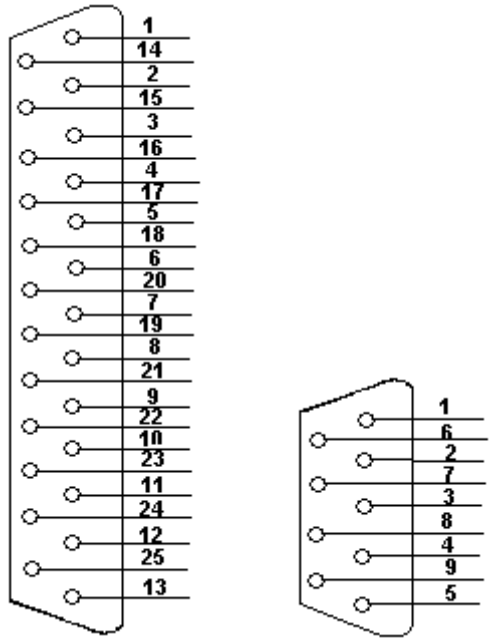
Ένας πολύ διαδεδομένος τρόπος μετάδοσης της πληροφορίας, ειδικά σε σημαντικές αποστάσεις, είναι η σειριακή επικοινωνία. Με τον τρόπο αυτό τα bits της πληροφορίας μεταδίδονται ένα κάθε φορά, δηλαδή στη σειρά, μέσα από έναν αγωγό μεταφοράς των δεδομένων. Για μια τέτοια επικοινωνία, χρειαζόμαστε τρεις συνολικά αγωγούς, έναν για την αποστολή δεδομένων, έναν για τη λήψη και έναν που θα βρίσκεται στο δυναμικό αναφοράς των μεταδιδόμενων σημάτων.

Είναι προφανές ότι για να αποσταλούν με σειριακό τρόπο κάποια δεδομένα μέσω μίας θύρας επικοινωνίας ενός ηλεκτρονικού υπολογιστή, πρέπει πρώτα να μετατραπούν από τη παράλληλη μορφή, με την οποία εμφανίζονται στο διάδρομο δεδομένων, σε σειριακή μορφή. Τη λειτουργία αυτή αναλαμβάνει ένα κύκλωμα που ονομάζεται UART (*Universal Asynchronous Receiver/Transmitter*), το οποίο υπάρχει σε ολοκληρωμένη μορφή επάνω στη μητρική πλακέτα ή στις μονάδες ελέγχου των περιφερειακών συσκευών ενός υπολογιστή. Η λειτουργία του κυκλώματος αυτού στηρίζεται στη λειτουργία του καταχωρητή ολίσθησης, ο οποίος αφού λάβει κάποια δεδομένα και τα καταχωρήσει στα flip-flops που διαθέτει, ολισθαίνει τα bits της ψηφιολέξης που έχει καταχωρήσει ένα-ένα προς τα δεξιά ή προς τα αριστερά.

Σχήμα 2.1 Σχεδιάγραμμα ολίσθησης UART και προσπέλαση δεδομένων

Τα κυριότερο πλεονέκτημα της σειριακής επικοινωνίας είναι ο μικρότερος αριθμός καλωδίων διασύνδεσης που απαιτείται, σε σχέση με την παράλληλη επικοινωνία. Αυτό κάνει την εγκατάσταση φθηνότερη όταν οι αποστάσεις είναι μεγάλες. Επιπλέον, τα πρωτόκολλα επικοινωνίας που χρησιμοποιούνται στη σειριακή επικοινωνία επιτρέπουν μεγάλες στάθμες σημάτων σε σχέση με τα πρωτόκολλα της παράλληλης επικοινωνίας, οπότε οι απώλειες του σήματος δημιουργούν μικρότερο πρόβλημα και η μετάδοση σε μεγάλη απόσταση είναι εφικτή. Εξάλλου, με την σειριακή επικοινωνία είναι πολύ ευκολότερη η *ασύρματη* μετάδοση, ειδικά μέσω διατάξεων υπέρυθρης ακτινοβολίας, που είναι πολύ διαδεδομένες. Τέλος, η σειριακή μετάδοση είναι πιο κατάλληλη για χρήση σε μικροελεγκτές, που επίσης έχουν διαδοθεί τα τελευταία χρόνια. Ο λόγος είναι ότι οι διάφορες διατάξεις, όπως μετατροπείς A/D , μνήμες, υπολογιστές κ.λπ. καταλαμβάνουν πολύ λιγότερους ακροδέκτες του μικροελεγκτή όταν επικοινωνούν σειριακά με αυτόν, παρά όταν επικοινωνούν παράλληλα. Εξάλλου, μερικά συστήματα μικροελεγκτών έχουν ενσωματωμένες θύρες σειριακής διασύνδεσης με το εξωτερικό περιβάλλον, κάτι που καθιστά απλή τη σειριακή διασύνδεση τους με περιφερειακές συσκευές.

Οι σειριακές θύρες επικοινωνούν μέσω αρσενικών συνδέσμων D-25 ή D-9, δηλαδή συνδέσμων των 25 ή των 9 ακροδεκτών. Οι σειριακές θύρες του υπολογιστή χρησιμοποιούν το ασύγχρονο σειριακό πρωτόκολλο RS-232.



Σχήμα 2.2 Σύνδεσμοι D-25 και D-9 για την σειριακή επικοινωνία.

2.1.1 Σύγχρονη σειριακή επικοινωνία

Στη σύγχρονη σειριακή μετάδοση έχουμε καλύτερη εκμετάλλευση του εύρους του καναλιού επικοινωνίας, αφού τα δεδομένα ομαδοποιούνται σε μεγάλα πακέτα δεδομένων, χωρίς START και STOP bits στην αρχή και στο τέλος κάθε χαρακτήρα. Με τον τρόπο αυτό αυξάνεται σημαντικά ο ρυθμός μετάδοσης και γίνεται αποδοτικότερη η χρήση κυκλωμάτων μετάδοσης. Η σύγχρονη σειριακή μετάδοση ενδείκνυται κυρίως για την επικοινωνία ανάμεσα σε υπολογιστές που διακινούν μεγάλους όγκους δεδομένων.

Για να μπορεί ο δέκτης να διακρίνει χωρίς απώλειες τα bits των δεδομένων, ειδικά όταν πρόκειται για πακέτα δεδομένων μεγάλου μήκους(εκατοντάδων ή χιλιάδων χαρακτήρων), είναι αναγκαίος ο συγχρονισμός ανάμεσα στον πομπό και στον δέκτη. Για τον σκοπό αυτό στην αρχή και στο τέλος κάθε ομάδας δεδομένων τοποθετούνται κάποια bytes συγχρονισμού, τα οποία περιέχουν ακολουθίες bits συγχρονισμού που δημιουργεί ο πομπός με χρήση ειδικού ωρολογιακού σήματος. Το τέλος της ομάδας δεδομένων σηματοδοτείται από έναν «κωδικό τέλους» και από έναν ή περισσότερους χαρακτήρες σφαλμάτων.

Ένας άλλος τρόπος συγχρονισμού είναι η ταυτόχρονη αποστολή σήματος συγχρονισμού, το οποίο παράγεται από ρολόι (CLK) του πομπού και το οποίο οδηγεί το ρολόι του δέκτη.

Τέλος, πλεονέκτημα της σύγχρονης μετάδοσης είναι ο μεγάλος ρυθμός αποστολής και η ανοχή στο «θόρυβο». Μειονέκτημα είναι ότι είναι πιο περίπλοκη σε σχέση με την ασύγχρονη και ότι έχει μεγαλύτερο κόστος υλοποίησης.

2.1.2 Ασύγχρονη σειριακή επικοινωνία

Η ασύγχρονη σειριακή επικοινωνία εξυπηρετεί τη μετάδοση χαρακτήρων που εκπέμπονται από κάποιο πομπό χωρίς κανένα συγχρονισμό, όπως συμβαίνει με τους αλφαριθμητικούς χαρακτήρες που δημιουργούνται όταν πιέζουμε τα πλήκτρα ενός πληκτρολογίου.

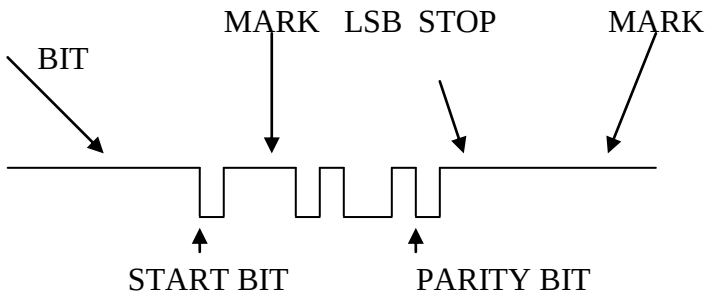
Κάθε τέτοιος χαρακτήρας μετατρέπεται σε ψηφιολέξη μέσω του κώδικα ASCII και η ακολουθία bits που αντιστοιχεί σε αυτόν εμφανίζεται στην γραμμή μετάδοσης. Ο δέκτης πρέπει να είναι σε θέση να αναγνωρίζει ότι έφτασε ένας χαρακτήρας και να δέχεται τα bits του χαρακτήρα με τη σωστή σειρά και χωρίς απώλειες.

Για τον παραπάνω σκοπό, τα bits της ασύγχρονης σειριακής μετάδοσης οργανώνονται σε ομάδες των εννέα έως δώδεκα bits συνολικά, οι οποίες περιέχουν κάποιους καταχωρητές έναρξης και λήξης. Το πρώτο bit κάθε πλαισίου είναι το λεγόμενο START BIT, το οποίο αντιστοιχεί σε λογικό

μηδέν. Ακολουθεί η σειρά των ψηφίων του χαρακτήρα που αποστέλλεται. Για παράδειγμα εάν αποστέλλεται το κεφαλαίο γράμμα Α, τότε η ακολουθία των ψηφίων θα είναι 01001011. Μετά από τον χαρακτήρα ακολουθεί ένα bit άρτιας ή περιττής ισοτιμίας(parity) το οποίο ενεργοποιεί μία διαδικασία ελέγχου σφαλμάτων, για να ανιχνεύει τυχόν λάθη που συνέβησαν κατά την μετάδοση. Το πλαίσιο κλείνει με ένα ή δύο STOP BITS, που υποδηλώνουν το τέλος του χαρακτήρα και την κατάσταση αναμονής για το επόμενο. Η λογική κατάσταση των ψηφίων λήξης(STOP BIT) είναι το λογικό ένα.

Όταν δεν μεταδίδεται κάποιος χαρακτήρας τότε λέμε ότι η σύνδεση είναι ανενεργή, οπότε η γραμμή ευρίσκεται σε λογικό ένα. Η κατάσταση αυτή ονομάζεται «συνθήκη MARK». Όταν μεταδοθεί το bit έναρξης(START BIT) και φτάσει στο δέκτη, τότε ο δέκτης καταλαβαίνει ότι ακολουθούν τα bits του χαρακτήρα που αποστέλλεται, οπότε ενεργοποιεί το σύστημα χρονισμού του και διαβάζει με τη σειρά τα επόμενα bits μέχρι τα bits λήξης(STOP BITS).

Στη συνέχεια τίθεται και πάλι στην κατάσταση αναμονής.



Σχήμα 2.3 Τ*α bits της* ασύγχρονης σειριακής μετάδοσης

Είναι φανερό ότι η διάρκεια του κάθε εκπεμπόμενου bit στην ασύγχρονη σειριακή μετάδοση πρέπει να είναι αυστηρά η ίδια, ώστε να μπορεί ο δέκτης με βάση κάποιο σύστημα χρονισμού, να διακρίνει τα bits μεταξύ τους. Συνεπώς ο πομπός και ο δέκτης πρέπει να συμφωνούν ως προς την ταχύτητα της σειριακής μετάδοσης των bits. Η ταχύτητα αυτή ορίζει τον λεγόμενο «ρυθμό μετάδοσης» (baud rate), που μετριέται σε bits ανά δευτερόλεπτο(bits/sec ή bps). Συνήθεις ρυθμοί στις ασύγχρονες σειριακές επικοινωνίες είναι 2400,4800, 9600,14400, 19200 ,28800 και 33600 bits/sec. Η μέγιστη ταχύτητα που υποστηρίζει μια θύρα UART κατά την αποστολή χαρακτήρων από έναν υπολογιστή προς μια συσκευή επικοινωνίας είναι 115,2kbps.

Τυπική ασύγχρονη σειριακή συσκευή είναι το modem,που διασυνδέει τον υπολογιστή ή κάποιο τερματικό με την τηλεφωνική γραμμή. Σαν παράδειγμα αναφέρουμε ότι για κάθε ρυθμό μετάδοσης 9600 bps η διάρκεια του κάθε bit πρέπει να είναι 104μs. Κάθε χαρακτήρας θα διαρκεί στη γραμμή 1,14ms. Ας σημειωθεί ότι η ακρίβεια στην διάρκεια του κάθε bit είναι σημαντική, ώστε να υπάρχει συγχρονισμός πομπού και δέκτη.

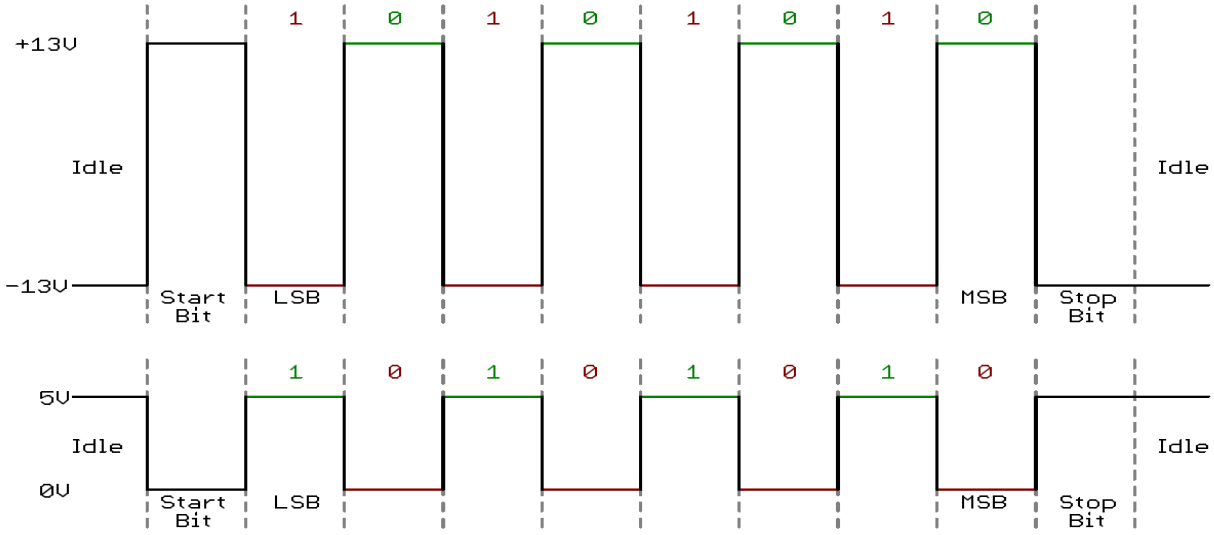
Το βασικό μειονέκτημα της ασύγχρονης σειριακής μετάδοσης είναι η ανάγκη που προκύπτει για START και STOP bits στην αρχή και στο τέλος κάθε χαρακτήρα. Με τον τρόπο αυτό επιβαρύνεται σημαντικά η διαδικασία μετάδοσης με επιπλέον bits που δεν αντιπροσωπεύουν χρήσιμη πληροφορία.

2.2 Το πρωτόκολλο RS-232C

Το πρωτόκολλο RS-232C επιτρέπει την ασύγχρονη σειριακή επικοινωνία ανάμεσα σε δύο συσκευές. Εάν επιθυμούμε να συνδέσουμε περισσότερες από δύο συσκευές σε έναν υπολογιστή, χρειαζόμαστε περισσότερες από μια σειριακές θύρες.

Το πρωτόκολλο RS-232C χρησιμοποιεί αρνητική ψηφιακή λογική και μεγάλες στάθμες, ώστε να επιτρέπει τη διάδοση του σήματος σε μεγάλες αποστάσεις χωρίς απώλειες. Αυτά έχουν σαν αποτέλεσμα οι τάσεις του πρωτοκόλλου RS-232C να μην είναι συμβατές με τις στάθμες TTL.

Το παρακάτω διάγραμμα χρονισμού παρουσιάζει τόσο ένα TTL (κάτω μέρος) όσο και ένα RS-232(πάνω μέρος) για αποστολή του σήματος:0b01010101.



Σχήμα 2.4 Διάγραμμα προσπέλασης δεδομένων σε TTL και RS232

Τα επίπεδα τάσεων του πρωτοκόλλου RS-232C, σύμφωνα με τις προδιαγραφές που θέσπισε η Ένωση EIA είναι τα εξής:

1. Το λογικό 0, που λέγεται και **SPACE**, βρίσκεται μεταξύ +3 και +25 V (στην πράξη λαμβάνονται και εκπέμπονται από +5 έως +15 V).

2. Το λογικό 1, που λέγεται και **MARK**, βρίσκεται μεταξύ –3 και –25V (πρακτικά από –5 έως –15V).

3. Η περιοχή από –3V έως +3V δεν αντιπροσωπεύει καθορισμένη λογική στάθμη.

4. Κανένας από τους ακροδέκτες της σειριακής θύρας δεν μπορεί να δεχτεί δυναμικό μεγαλύτερο από 25V σε σχέση με τη γη.

5. Το μέγιστο ρεύμα δεν μπορεί να ξεπερνά τα 500mA.

Αν και σύμφωνα με το πρωτόκολλο ο μέγιστος ρυθμός μετάδοσης (baud rate) δεν ξεπερνά τα 19,2 kbps, οι σημερινές ταχύτητες μπορεί να είναι σαφώς μεγαλύτερες.

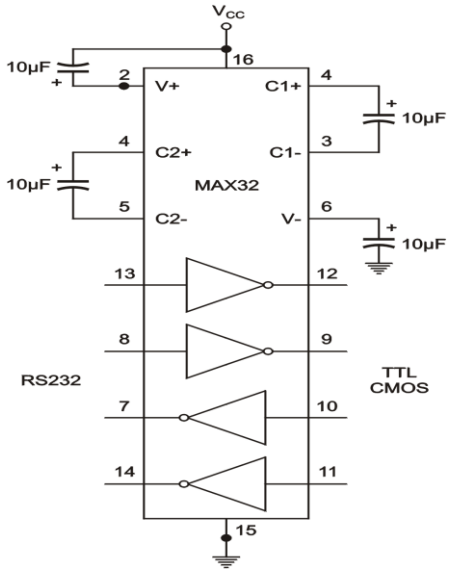
Με χρήση του πρωτοκόλλου RS-232C, ένα τερματικό χαρακτήρων(ASCII terminal) μπορεί να αποστείλει μέσω μιας γραμμής επικοινωνίας δεδομένα, σύμφωνα με τους κανόνες της ασύγχρονης σειριακής μετάδοσης.

Όταν η γραμμή είναι ανενεργή, βρίσκεται σε συνθήκη MARK, δηλαδή -12V περίπου, που αντιστοιχούν σε λογικό ένα. Η γραμμή ενεργοποιείται με την συνθήκη SPACE (λογικό μηδέν ή +12V). Ακολουθεί η μετάδοση επτά ή οκτώ bits για τον αποστελλόμενο χαρακτήρα, ένα προαιρετικό bit, άρτιας ή περιττής ισοτιμίας(parity) και ένα ή δύο STOP bits(συνθήκη MARK), που σηματοδοτούν το τέλος του χαρακτήρα.

2.3 Ο μετατροπέας στάθμης MAX232.

Για τα κυκλώματα μας χρησιμοποιούμε λογικές στάθμες TTL και CMOS για τα οποία το λογικό μηδέν είναι 0Volt και το λογικό ένα είναι ίσο με +5Volt. Για να συνδέσουμε κάποια εφαρμογή με μια σειριακή θύρα πρέπει να χρησιμοποιήσουμε ένα κύκλωμα που θα αλλάζει τις στάθμες του RS232 σε στάθμες TTL των +5Volt για λογικό μηδέν και +10 ή -10Volt για λογικό ένα.

Ένα τέτοιο κύκλωμα δημιουργήθηκε από την εταιρία MAXIM, έχει δηλαδή σκοπό να επικοινωνεί με συσκευές που υποστηρίζουν το ασύγχρονο πρωτόκολλο σειριακής μετάδοσης. Αυτό ονομάστηκε MAX232 και αποτελείται από δύο γραμμές εκπομπής και από δύο γραμμές λήψης.



Σχήμα 2.5 Τυπική εφαρμογή του ολοκληρωμένου MAX232

Δηλαδή μόνο δύο από τα σήματα του RS232 μπορούν να μετατραπούν σε κάθε κατεύθυνση.

2.4 Λίγα λόγια για το κύκλωμα UART

Ο πιο συνηθισμένος τύπος κυκλώματος UART που απαντάται στη σειριακή κάρτα των Η/Υ είναι το κύκλωμα 16550. Αυτό είναι ένα ολοκληρωμένο κύκλωμα 40 ακροδεκτών, το οποίο από τη μια πλευρά βλέπει 8 bits του διαδρόμου δεδομένων και από την άλλη οδηγεί τους ακροδέκτες ενός σειριακού συνδέσμου D-9 ή D-25.

Οι ακροδέκτες 1 έως 8 βλέπουν τον δίαυλο, ενώ οι ακροδέκτες RD και TD (11 και 12 αντίστοιχα) οδηγούν τις γραμμές εκπομπής και λήψης σειριακών δεδομένων. Οι ακροδέκτες DSR, CTS, DTR και RTS (37, 36, 33 και 32) οδηγούν τους ακροδέκτες ελέγχου της σειριακής θύρας.

Το κύκλωμα UART είναι συμβατό με το επίπεδο TTL. Άρα, ανάμεσα στο UART και στον D-τύπου συνδετήρα της σειριακής θύρας παρεμβάλλονται μετατροπείς στάθμης, ώστε τα σήματα του UART να γίνουν συμβατά με τα επίπεδα λογικής του πρωτοκόλλου RS-232.

Το κύκλωμα χρονισμού του κυκλώματος UART ονομάζεται γεννήτρια ρυθμού (baud rate generator). Αυτό αποτελείται από έναν εξωτερικό κρύσταλλο συχνότητας 1.8432 MHz, ο οποίος συνδέεται στους ακροδέκτες XIN-XOUT.

Ο τρόπος λειτουργίας του κυκλώματος UART προγραμματίζεται με την βοήθεια ορισμένων καταχωρητών, τους οποίους ο υπολογιστής βλέπει σαν θέσεις μνήμης. Με την βοήθεια των καταχωρητών αυτών μπορεί ο

χρήστης να έχει πλήρη έλεγχο της διαδικασίας εισόδου/εξόδου μέσω της σειριακής θύρας. Εάν για παράδειγμα κάποιος θέλει να έχει πρόσβαση στους ακροδέκτες ελέγχου (CTS, RTS, DTR ή DSR) τότε πρέπει να χρησιμοποιήσει τους καταχωρητές του κυκλώματος UART και να προσπελάσει τις αντίστοιχες θέσεις μνήμης.

Ακροδέκτες σύνδεσμο D-25	Ακροδέκτε σύνδεσμο D-9	Κωδική Ονομασία	Όνομα	Περιγραφή λειτουργίας	
2	3	TXD	Transmit Data	Από : Προς:DCE	Εκπομπή σειρια δεδομένων από την συσ DTE, και η λήψη τους απ συσκευή DCE
3	2	RXD	Receive Data	Από : Προς:DTE	Λήψη σειριακών δεδομέ από την συσκευή DTE, κ αποστολή τους από τη συσ DCE.
7	5	SGND	Signal Ground		Αγωγός αναφοράς σημά (γείωση)
4	7	RTS	Request To Ser	Από : Προς:DCE	Η συσκευή DTE είναι έτ να αποστείλει δεδομένα DCE.Ως απάντηση αναμένε αποστολή σήματος CTS την DCE.
5	8	CTS	Clear To Send	Από : Προς:DTE	Δηλώνει ότι η συσκευή είναι έτοιμη να λάβει δεδομέ
6	6	DSR	Data Set Ready	Από : Προς:DTE	Η συσκευή DCE πληροφ τη θύρα UART ότι είναι έτ να αποστείλει δεδομένα.
20	4	DTR	Data Term Ready	Από : Προς:DCE	Το αντίθετο του σήμ DSR.Η θύρα UART πληροφ τη συσκευή DCE ότι είναι έτ για να λάβει δεδομένα.
8	1	CD	Carrier Detect	Από : Προς:DTE	Η συσκευή DCE δείχνε συσκευή DTE κατά πό υπάρχει σήμα λήψης γραμμή, και μάλιστα μέσα κατάλληλα αποδεκτά όρια.
22	9	RI	Ring Indicator		

Πίνακας 2.1 Αντιστοίχιση, ονομασία και λειτουργία των ακροδεκτών της σειριακής θύρας .

2.5 Οι ακροδέκτες της σειριακής θύρας και οι λειτουργίες τους.

Για την σύνδεση δύο τερματικών συσκευών σε μεγάλες αποστάσεις είναι απαραίτητο να παρεμβάλλονται συσκευές επικοινωνίας. Τέτοιες συσκευές

είναι το modem, το video, τα πολύμετρα. Οι ταχύτητες ανάμεσα σε μία τερματική συσκευή και σε μια συσκευή επικοινωνίας είναι πολύ μεγαλύτερες από τις ταχύτητες ανάμεσα σε συσκευές επικοινωνίας .

Οι ακροδέκτες της σειριακής θύρας διακρίνονται στις γραμμές δεδομένων και στις γραμμές ελέγχου. Οι σημαντικότεροι ακροδέκτες είναι αυτοί που μεταφέρουν τα δεδομένα εκπομπής και λήψης. Όλοι οι υπόλοιποι ακροδέκτες είναι βοηθητικοί αλλά απαραίτητοι για την επικοινωνία. Στον παρακάτω πίνακα αναφέρονται τα ονόματα, οι κωδικές ονομασίες και οι λειτουργίες του κάθε ακροδέκτη των συνδέσμων D-25 και D-9.

Όπως βλέπουμε από τον παραπάνω πίνακα οι ακροδέκτες DTR και RTS είναι ακροδέκτες εξόδου, ενώ οι ακροδέκτες CTS και DSR είναι ακροδέκτες εισόδου.

Κεφάλαιο 3

«Εφαρμογή ελεγκτή σειριακής επικοινωνίας»

Όπως αναφερθήκαμε στο προηγούμενο κεφάλαιο, η σειριακή θύρα συνδέεται με εξωτερικές συσκευές μέσω ενός συνδετήρα D-25 ή D-9. Οι τάσεις και εν γένει τα ηλεκτρικά χαρακτηριστικά της σειριακής θύρας περιέχονται στο σειριακό πρωτόκολλο επικοινωνίας RS-232C της Ένωσης EIA(Electronics Industry Association). Το πρωτόκολλο αυτό ονομάζεται επίσης V.24/V.28, σύμφωνα με τη διεθνή ένωση προτύπων CCITT. Οι διάφοροι κατασκευαστές υπολογιστών, τερματικών και συσκευών επικοινωνίας, όπως modem, φροντίζουν να συμμορφώνονται με το πρότυπο αυτό, ώστε να υπάρχει απόλυτη συμβατότητα ανάμεσα στις συσκευές.

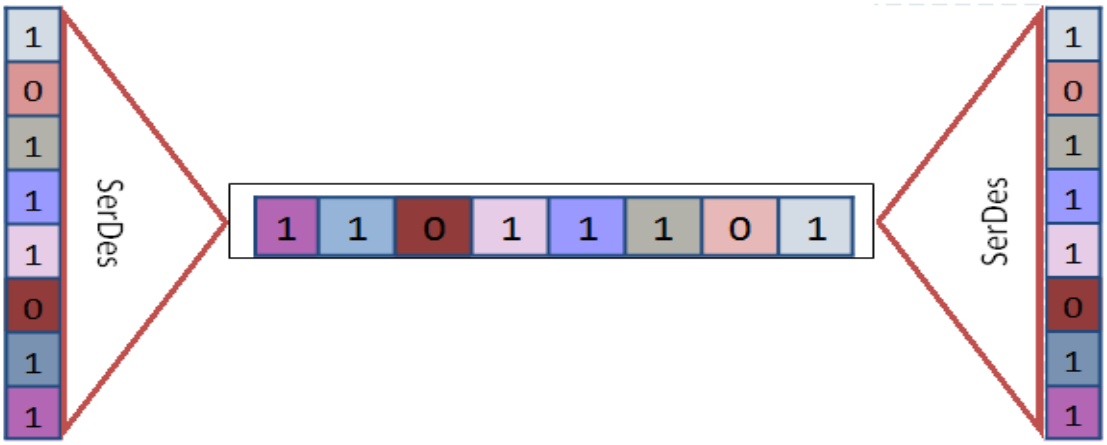
Οι τύποι των συσκευών που καθορίζουν το πρωτόκολλο RS232 είναι δύο. Ο πρώτος τύπος συσκευής ονομάζεται Τερματική Συσκευή Δεδομένων **DTE** (Data Terminal Equipment). Ο δεύτερος τύπος συσκευής ονομάζεται Συσκευή Επικοινωνίας Δεδομένων **DCE** (Data Communications Equipment).

Αυτά τα βλέπουμε και στον πίνακα 2.1 στο προηγούμενο κεφάλαιο στην ενότητα για τους ακροδέκτες της σειριακής επικοινωνίας.

Το UART **(Universal Asynchronous Receiver-Transmitter)** στην ουσία είναι το υποσύστημα ενός υπολογιστή, το οποίο διαχειρίζεται την ασύγχρονη σειριακή επικοινωνία. Κάθε υπολογιστής ενσωματώνει ένα UART για τη διαχείριση των σειριακών θυρών του. Επίσης, όλες οι εσωτερικές συσκευές modem ***(το modem προέρχεται από τη σύνδεση των λέξεων MOdulator-DEModulator δηλαδή κωδικοποιητής-αποκωδικοποιητής. Είναι η συσκευή που κωδικοποιεί και αποκωδικοποιεί)*** έχουν το δικό τους UART, ενώ οι εξωτερικές κάνουν χρήση του UART του υπολογιστή. Με την αύξηση των modems το UART έχει αυξημένα καθήκοντα επιτήρησης και διευθέτησης της επικοινωνίας.

Ο **Transmitter,** δηλαδή ο πομπός έχει την ικανότητα να επεξεργάζεται ένα σήμα μηνύματος με σκοπό να παράγει ένα σήμα που μπορεί να περάσει αξιόπιστα και αποδοτικά μέσα από το κανάλι.

Ο **Receiver,** δηλαδή ο δέκτης αντιστρέφει τη διαδικασία διαμόρφωσης που εκτέλεσε ο πομπός και έτσι μπορεί να ανακτά το αρχικό σήμα-μήνυμα, καθώς



Σχήμα 3.1 Η λογική του Serializer-Deserializer

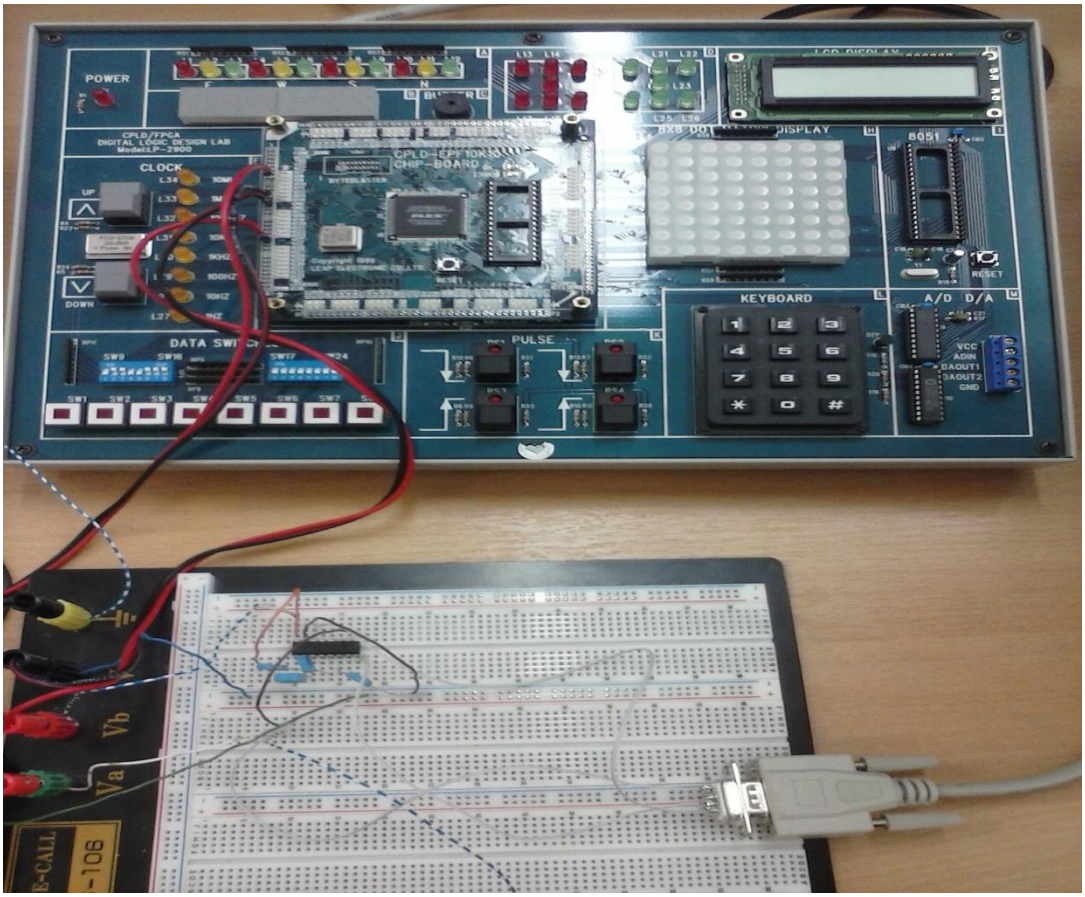
και αντισταθμίζει την υποβάθμιση του σήματος που τυχόν έχει προκαλέσει το κανάλι μετάδοσης.

Για την μετάδοση πομπού-δέκτη πρέπει να φτιάξουμε ένα σύστημα ***serializer-deserializer*** το οποίο θα μας δίνει τα δεδομένα αρχικά από παράλληλη σε σειριακή επικοινωνία και μετά από σειριακή σε παράλληλη με είσοδο στην δεύτερη τα αποτελέσματα της πρώτης.

3.1 Ο στόχος της εργασίας

Σαν στόχο στην πτυχιακή εργασία θέσαμε να φτιάξουμε έναν ελεγκτή ο οποίος θα στέλνει και θα λαμβάνει με ασύγχρονη σειριακή μετάδοση τα δεδομένα-χαρακτήρες που δίνουμε. Στην ουσία θέλουμε να διαμορφώσουμε το δικό μας UART που με την κατάλληλη χρήση των pins του FPGA να μπορεί να μας δείχνει ότι αυτό που στείλαμε μπορεί να το λάβει και σε δεύτερη φάση αυτό που στέλνουμε από αυτό να μπορούμε να το λάβουμε στον υπολογιστή μας.

Προγραμματίζοντας λοιπόν το FPGA μας με τον κώδικα VHDL, φτιάχνουμε δύο projects, ένα για τον έλεγχο του Transmitter και ένα για τον έλεγχο του Receiver. Επίσης χρησιμοποιούμε ένα raster πάνω στο οποίο τοποθετούμε τον μετατροπέα στάθμης MAX232, κάτι πυκνωτές και τα καλώδια με τα οποία συνδέουμε το FPGA και την σειριακή θύρα που ενώνεται με τον υπολογιστή.



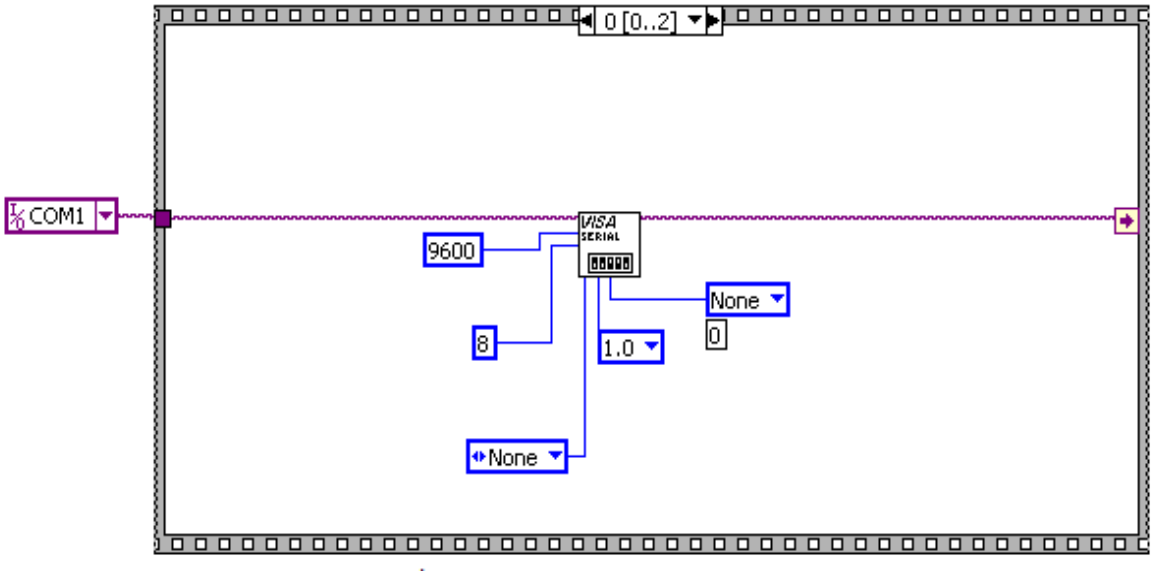
Σχήμα 3.2 Η φυσική διάταξη της υλοποίησης

Τέλος με τα καλώδια συνδέουμε τον παλμογράφο στο FPGA και στο raster για να πάρουμε αποτελέσματα. Το κύκλωμα που φτιάξαμε στο raster δίνεται στο παραπάνω σχήμα (σχήμα 3.2) .

3.2 Ο πομπός

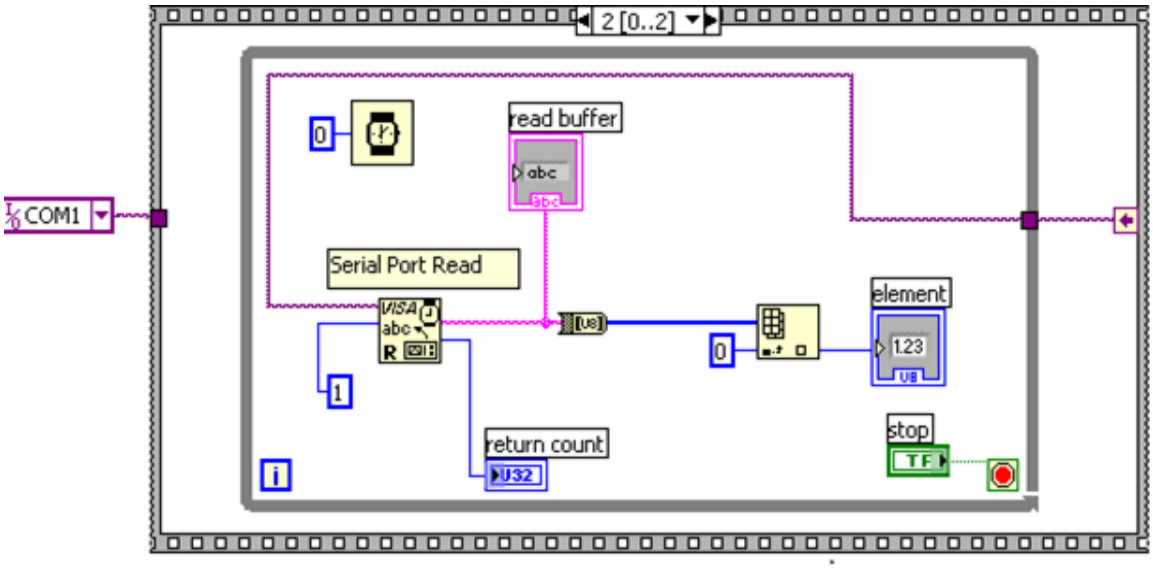
Για να δημιουργήσουμε τον πομπό μας φτιάξαμε στο ερευνητικό εργαστήριο του ΤΕΙ και στο πρόγραμμα Labview 7.1 του National Instruments την αρχικοποίηση για το UART στο οποίο θέσαμε Baud rate=9600bps, 8 bits χαρακτήρα και 1 STOP BIT. Αυτά ισχύουν και για τον δέκτη. Η αρχικοποίηση στο πρόγραμμα φαίνεται παρακάτω στο σχήμα 3.3 . Επίσης το focs=10MHz.

Σε ένα άλλο panel του Labview το οποίο το συνδέσαμε με αυτό της αρχικοποίησης φτιάχνουμε τον δέκτη μας και γι αυτό χρησιμοποιούμε μια σειριακή θύρα που θα διαβάζει αυτά που δώσαμε σχήμα 3.4 .



Σχήμα 3.3 Panel για την αρχικοποίηση του προγράμματος

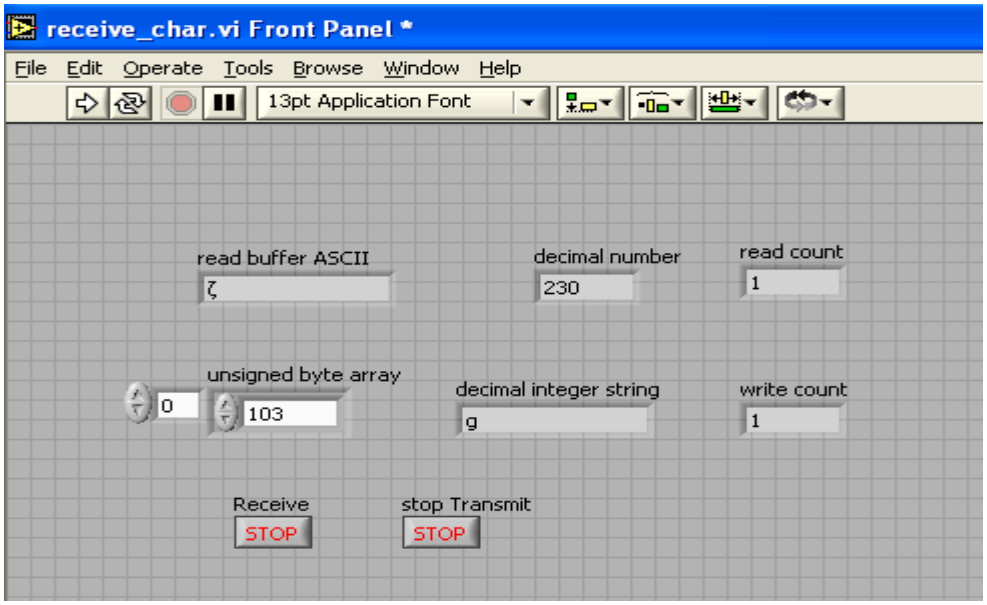
Τα δυο αυτά panels τα συνδέουμε μεταξύ τους.



Σχήμα 3.4 Panel που αντιπροσωπεύει τον δέκτη

Το panel που χρησιμοποιείται κατά το simulation είναι το παρακάτω (σχήμα 3.5) . Για την λειτουργία του transmitter παρατηρούμε την πάνω σειρά για να δούμε αποτελέσματα.

Τα σύμβολα που δίνονται είναι αυτά που έγιναν οι δοκιμές.



Σχήμα 3.5 Panel που δείχνει τα αποτελέσματα

Για τον σχεδιασμό του κώδικα με τον οποίο προγραμματίσαμε το FPGA μας χρησιμοποιήσαμε το πρόγραμμα Quartus II της εταιρείας Altera. Ο κώδικας αυτός περιλαμβάνει τρία επιμέρους κομμάτια, το ένα είναι το κύριο και τα άλλα δύο περιλαμβάνονται μέσα σε αυτό.

Το ένα από τα δύο «υποκυκλώματα» είναι το κύκλωμα Baud_rate_gen το οποίο μετατρέπει το ρολόι που είχαμε σαν είσοδο σε ένα ρολόι-έξοδο που θα είναι έχει περίοδο ίση με 104μs. Αρχικά ορίστηκαν οι βιβλιοθήκες:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
```

Στη συνέχεια ακολουθεί το ENTITY το οποίο περιλαμβάνει τον ορισμό των εισόδων και των εξόδων του Baud_rate_gen.

```
ENTITY baud_rate_gen IS
        PORT(clk,reset:in std_logic;
        baud_clk: out std_logic);
END baud_rate_gen;
```

Σειρά έχει η αρχιτεκτονική(architecture) μέσα στην οποία δηλώνεται ένα σήμα και μετά υπάρχει ένα process που εκτός από την μεταβλητή που

δηλώνεται για να χρησιμοποιηθεί (*VARIABLE sample:std_logic_vector(9 DOWNTO 0);*) ξεκινάει και ο σχεδιασμός και η υλοποίηση των εξισώσεων με βάσει τις οποίες θα πάρουμε αποτελέσματα στις εξόδους του Baud_rate_gen.

```
ARCHITECTURE baud OF baud_rate_gen IS
SIGNAL s_clk: std_logic :='0';
BEGIN
sampling:
PROCESS(clk,reset)
VARIABLE sample:std_logic_vector(9 DOWNTO 0);
        BEGIN
                IF reset='1' THEN
                        sample:=(OTHERS=>'0');
                ELSIF clk 'EVENT AND clk='1' THEN
                        sample:=sample+1;
                        IF sample=520 THEN --  fosc/(2xBaude_Rate)
                                s_clk<=NOT s_clk;
                                sample:=(OTHERS=>'0');
                        END IF;
                END IF;
```

Τέλος και αφού κλείσουμε την process παίρνουμε τα αποτελέσματά μας στην έξοδο baud_clk. Και έτσι κλείνουμε το κύκλωμα Baud_rate_gen.vhd.

```
END PROCESS;
baud_clk<=s_clk;
END baud;
```

Το δεύτερο από τα «υποκυκλώματα» είναι το transmit στο οποίο γίνεται όλη η διαδικασία του serializer-deserializer με σκοπό τα δεδομένα που δίνουμε να μπορούν να βγουν στην οθόνη μας χωρίς απώλειες και με τη σωστή σειρά. Χρησιμοποιούμε πάλι κάποιες βιβλιοθήκες και ορίζουμε τις εισόδους και τις εξόδους της οντότητας(entity) του πομπού.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_signed.all;

ENTITY transmit IS
```

```
        PORT(baud_clk, reset:IN std_logic;
                tx_data:IN std_logic_vector(7 downto 0);
                tx_en:IN std_logic;
                tx: OUT std_logic;
                tx_done:OUT std_logic);
    END transmit;
```

Όπως πρέπει να γίνεται δίπλα από κάθε είσοδο ή έξοδο δίνεται ο τύπος και ο τρόπος λειτουργίας του κάθε σήματος.

Ξεκινώντας την αρχιτεκτονική(architecture) δηλώνεται ένας τύπος με όνομα *state,* όπου δίπλα δηλώνονται οι καταστάσεις του και στην συνέχεια υπάρχουν πάλι κάποια σήματα τα οποία βοηθάνε στην διεξαγωγή του κώδικά μας.

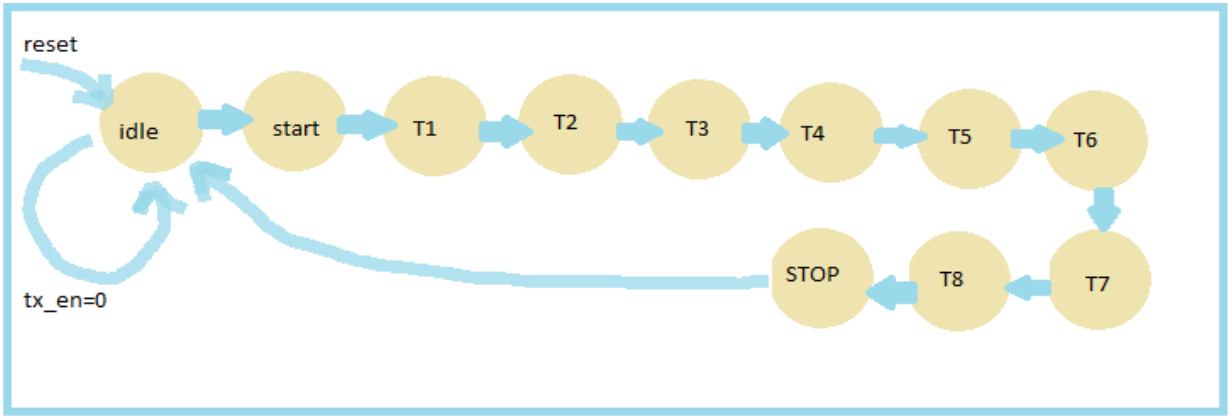
```
    ARCHITECTURE bhv OF transmit IS
    TYPE state IS (idle,start,T1,T2,T3,T4,T5,T6,T7,T8,stop);
        SIGNAL present_state,next_state:state;
        SIGNAL tx_do: std_logic;

    BEGIN
```

Στη συνέχεια ξεκινάει το process, comp1 το οποίο έχει σκοπό να κάνει το serializing μεταβαίνοντας από την μία κατάσταση του τύπου state που έχει σαν σήματα τα *present_state* και *next_state* που κάθε φορά παίρνουν την επόμενη τιμή του χαρακτήρα , ξεκινώντας πάντα από την κατάσταση idle,μεταβαίνοντας στο START BIT, από εκεί στα επόμενα διαδοχικά Bits και τέλος στο STOP BIT. Αυτό παραθέτεται και στο σχεδιάγραμμα παρακάτω.

```
comp1:PROCESS(present_state,baud_clk,tx_en)
    VARIABLE q:std_logic_vector(7 downto 0);
    BEGIN
        CASE present_state IS
            WHEN idle=>
                tx_do<='0';
                tx<='1';
                IF tx_en='1' THEN
                    q:=tx_data;
                    next_state<=start;
                ELSE
                    next_state<=idle;
                END IF;
```

```
WHEN start=>
IF baud_clk='1'THEN
                tx<='0'; --start bit
                next_state<=T1;
END IF;

WHEN T1=>
IF baud_clk='1' THEN
                tx<=q(0);
                next_state<=T2;
END IF;

WHEN T2=>
IF baud_clk='1' THEN
                tx<=q(1);
                next_state<=T3;
END IF;

WHEN T3=>
IF baud_clk='1' THEN
                tx<=q(2);
                next_state<=T4;
END IF;

WHEN T4=>
IF baud_clk='1' THEN
                tx<=q(3);
                next_state<=T5;
END IF;

WHEN T5=>
IF baud_clk='1' THEN
                tx<=q(4);
                next_state<=T6;
END IF;

WHEN T6=>
IF baud_clk='1' THEN
                tx<=q(5);
                next_state<=T7;
END IF;

WHEN T7=>
IF baud_clk='1' THEN
                tx<=q(6);
                next_state<=T8;
END IF;
```

```
                WHEN T8=>
                IF baud_clk='1' THEN
                        tx<=q(7);
                        next_state<=stop;
                END IF;

                WHEN stop=>
                IF baud_clk='1'THEN
                        tx<='1';--stop bit
                        next_state<=idle;
                END IF;
                IF baud_clk='0' THEN
                        tx_do<='1';
                END IF;
END CASE;
    END PROCESS;
```



Σχήμα 3.6 Η διαδικασία μετάβασης από την μια κατάσταση στην άλλη όπως φαίνεται στην εντολή επιλογής case του κώδικα VHDL

Στο process που ακολουθεί(sequential) το πρόγραμμα βλέπει αν υπάρχει και άλλος χαρακτήρας, αν όχι μεταβαίνει στην κατάσταση idle απ' όπου και περιμένει το καινούριο σήμα.

```
sequential:PROCESS(baud_clk,reset)
BEGIN
        IF reset='1' THEN
                present_state<=idle;
        ELSIF baud_clk 'EVENT AND baud_clk='1' THEN
                present_state<=next_state;
        END IF;
END PROCESS;
```

Στο τελευταίο process του υποκυκλώματος εξάγουμε την τιμή για το tx_done, αν έχει γίνει δηλαδή όλη η μεταφορά του χαρακτήρα.

```
comp2:PROCESS(present_state)
        BEGIN
                tx_done<=tx_do;
        END PROCESS;

END bhv;
```

Για να μπορέσουν αν λειτουργήσουν όμως τα δύο αυτά «υποκυκλώματα» χρειάζεται ένα τρίτο το οποίο θα περιέχει τις βασικές τους ιδιότητες και θα περιέχει και αυτά σαν δομικά στοιχεία. Αυτό είναι το top_level_trans.vhd (έτσι έχουμε ονομάσει και το project μας) . Πιο αναλυτικά, δίνονται πάλι οι βιβλιοθήκες και στην συνέχεια το ENTITY το οποίο περιέχει τα σήματα και από τα δύο προηγούμενα κυκλώματα.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY top_level_trans IS
        PORT(clk,reset:in std_logic;
                tx_data: in std_logic_vector(7 downto 0);--transmit data
                tx_en:IN std_logic; --when 1 it transmits
                tx_out: OUT std_logic; --transmit pin
                tx_done_out:OUT std_logic; -- is 1 when transmit is done
                baud_clk: out std_logic);
END top_level_trans;
```

Αφού ξεκινήσει η αρχιτεκτονική δίνονται κάποια βοηθητικά σήματα(SIGNALS) και μετά δηλώνεται η πρώτη συνιστώσα η οποία αντιπροσωπεύει το Baud_rate_gen.vhd . Στην συνέχεια παραθέτεται το δεύτερο component(συνιστώσα) που αντιπροσωπεύει το transmit.vhd.

```
ARCHITECTURE top OF top_level_trans IS
SIGNAL my_baud_clk: std_logic;
SIGNAL my_tx:std_logic;

COMPONENT baud_rate_gen IS --produces baud rate for the transmitter
        PORT(clk,reset:in std_logic;
        baud_clk: out std_logic);
```

```
END COMPONENT;

COMPONENT transmit IS
        PORT(baud_clk, reset:IN std_logic;
                tx_data:IN std_logic_vector(7 downto 0);
                tx_en:IN std_logic;
                tx: OUT std_logic;
                tx_done:OUT std_logic);
END COMPONENT;
```

Αμέσως μετά ξεκινάει η αρχιτεκτονική με το begin, δίνονται τα στιγμιότυπα των δύο components και παίρνουν τιμές τα τελευταία σήματα εξόδου. Με αυτόν τον τρόπο τελειώνει και ο κώδικας για το transmitter και μπορούμε να ξεκινήσουμε την προσομοίωση.

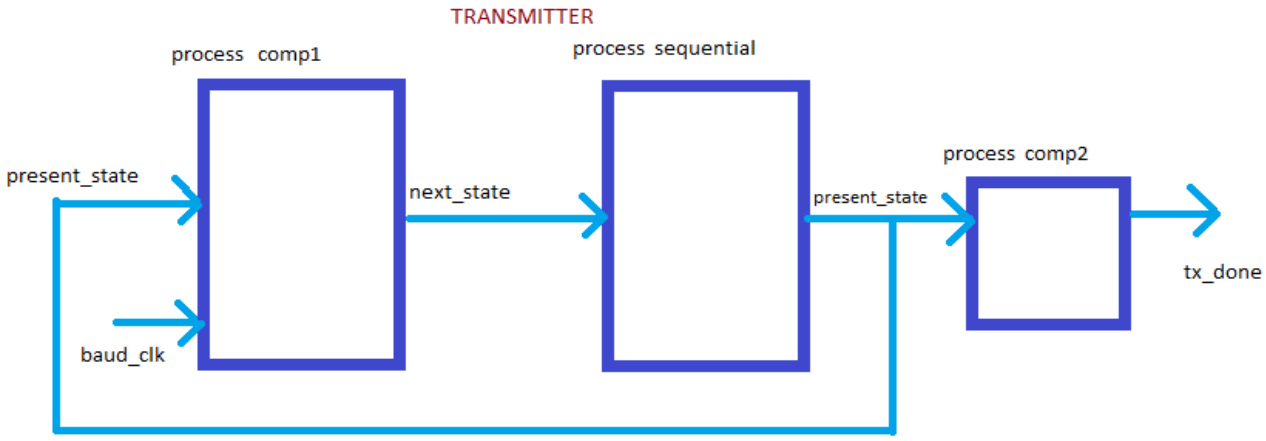
```
BEGIN
baud: baud_rate_gen PORT MAP(clk, reset, my_baud_clk);
baud_clk<=my_baud_clk;
tx_out<=my_tx;
Tx: transmit PORT MAP(my_baud_clk, reset, tx_data, tx_en, my_tx,
                        tx_done_out);
END top;
```

Τα pins που χρησιμοποιήσαμε για τον προγραμματισμό του Transmitter είναι:

Node Name	Direction	Location
clk	Unknown	PIN_55
baud_clk	Unknown	PIN_9
tx_done_out	Unknown	PIN_12
tx_out	Unknown	PIN_7
tx_en	Unknown	PIN_63
reset	Unknown	PIN_47
tx_data[0]	Unknown	PIN_64
tx_data[1]	Unknown	PIN_65
tx_data[2]	Unknown	PIN_67
tx_data[3]	Unknown	PIN_68
tx_data[4]	Unknown	PIN_69
tx_data[5]	Unknown	PIN_70
tx_data[6]	Unknown	PIN_72
tx_data[7]	Unknown	PIN_73

Πίνακας 3.1 Pins για τον Transmitter

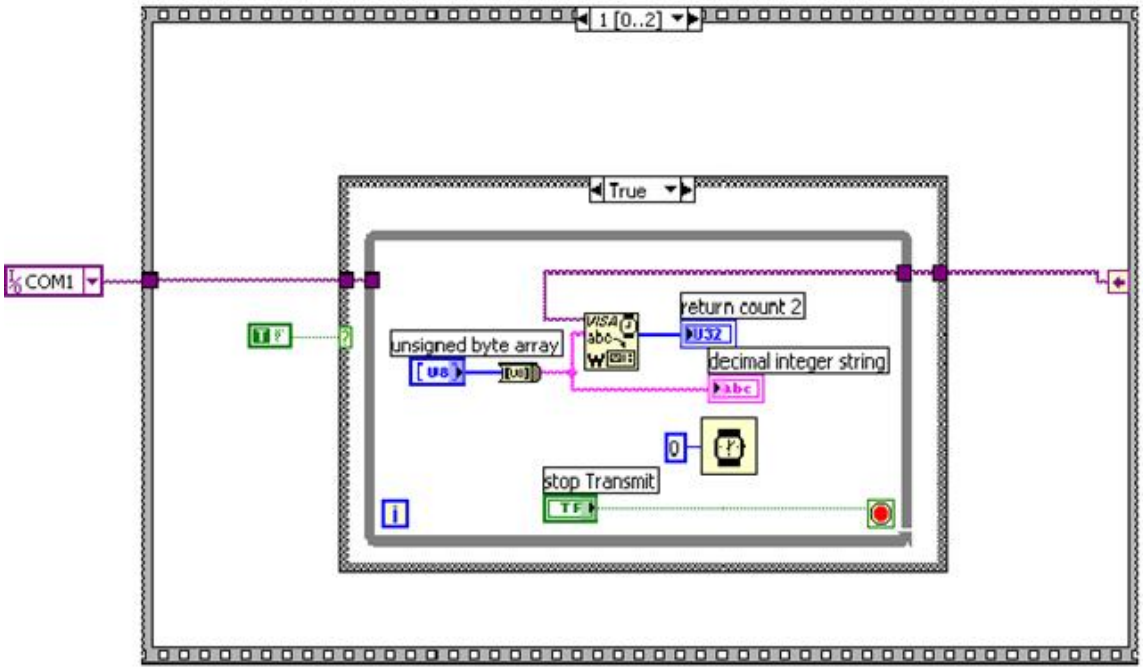
Η μηχανή καταστάσεων που αναπαριστά τον Transmitter του κυκλώματος μας είναι αυτή που φαίνεται στην παρακάτω εικόνα.



Σχήμα 3.7 Μηχανή καταστάσεων για τον Transmitter

3.3 Ο δέκτης

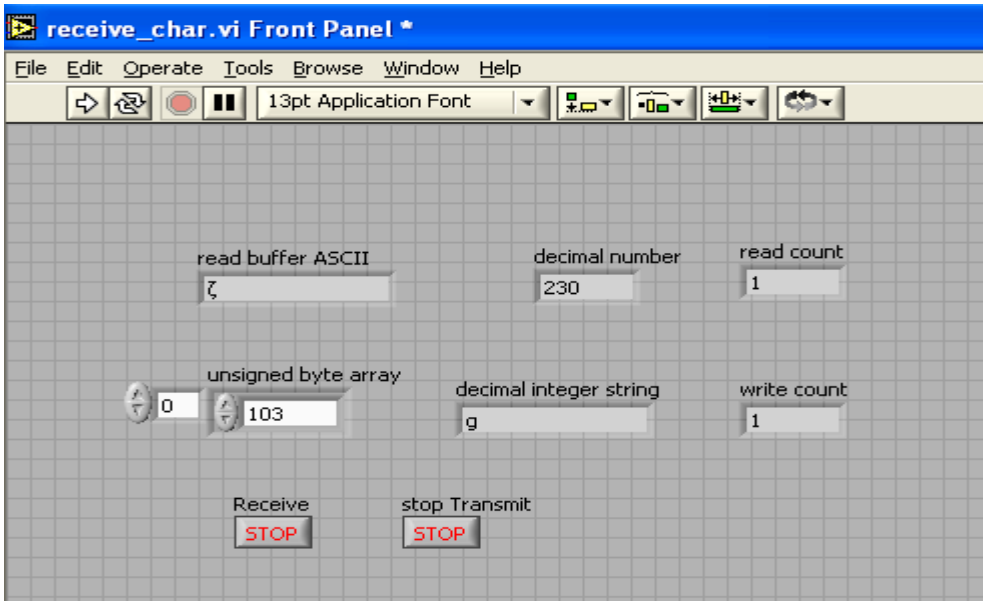
Για να σχεδιάσουμε τον Receiver πρέπει να φτιάξουμε έναν πομπό στο Labview του εργαστηρίου. Οπότε, γνωρίζοντας ότι η αρχικοποίηση είναι η ίδια (εικόνα 1-Labview_ssInit) και έχοντας τις ίδιες προδιαγραφές, Baud rate=9600bps, 8 bits χαρακτήρα και 1 STOP BIT, φτιάχνουμε σε ένα άλλο panel τον πομπό μας(αφού απενεργοποιήσουμε το προηγούμενο). Σε αυτό το panel δηλαδή θα υπάρχει μια σειριακή θύρα που θα γράφει.



Σχήμα 3.8 Panel που αντιπροσωπεύει τον πομπό

Το panel που χρησιμοποιείται κατά το simulation δίνεται στο σχήμα 3.9 . Για την λειτουργία του receiver παρατηρούμε την μεσαία σειρά για να δούμε αποτελέσματα.

Τα σύμβολα που δίνονται είναι αυτά που έγιναν οι δοκιμές.



Σχήμα 3.10 Panel που δείχνει τα αποτελέσματα

Για τον σχεδιασμό του κώδικα VHDL χρησιμοποιείτε και πάλι το Quartus II της εταιρείας Altera και έτσι για να υπάρξει το επιθυμητό αποτέλεσμα για τον Receiver, διαμορφώνονται δύο κυκλώματα. Το ένα αποτελεί το πακέτο (package), που ορίζεται για τη δήλωση των τύπων που χρειάζονται για τη σχεδίαση του Receiver.

Όπως και στα προηγούμενα κυκλώματα έτσι και εδώ δηλώνεται η βιβλιοθήκη και στη συνέχεια γίνεται η δήλωση του πακέτου.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE my_package IS
TYPE state IS (idle,A0,A1,A2,A3,A4,A5,A6,A7,A8,stop);
END PACKAGE;
```

Για το δεύτερο κύκλωμα του Receiver τα δεδομένα που λαμβάνονται πρέπει να είναι αυτά που έχουν δοθεί μέσα από την σειριακή μας θύρα. Οπότε δίνονται πάλι οι βιβλιοθήκες, εδώ τώρα περιλαμβάνεται και το πακέτο που δημιουργήσαμε πριν, που τώρα εισάγεται σαν μια βιβλιοθήκη μαζί με τις άλλες.

Στην συνέχεια ορίζεται η οντότητα (ENTITY) μέσα στην οποία δίνονται τα διάφορα σήματα εισόδου και εξόδου. Σαν όνομα στην οντότητα δίνεται το όνομα του project που είναι *rx_UART2*.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_signed.all;
USE work.my_package.all;--states are defined here

ENTITY rx_UART2 IS
        PORT(clk, reset:IN std_logic;
                rx:IN std_logic;
                rx_data:OUT std_logic_vector(7 downto 0);
                rx_done:OUT std_logic);
END rx_UART2;
```

Αμέσως μετά υπάρχει η αρχιτεκτονική στην οποία αρχικά δηλώνονται κάποια σήματα. Το σήμα ***SIGNAL my_counter:INTEGER RANGE 0 TO 1023;*** έχει τόσο μεγάλο range επειδή το ***state*** που δηλώσαμε στο πακέτο μας περιέχει 10 καταστάσεις και επειδή τα 10bits πληροφορίας είναι ο αριθμός 1024, εδώ υπάρχουν 1024 θέσεις μνήμης από 0-1023.

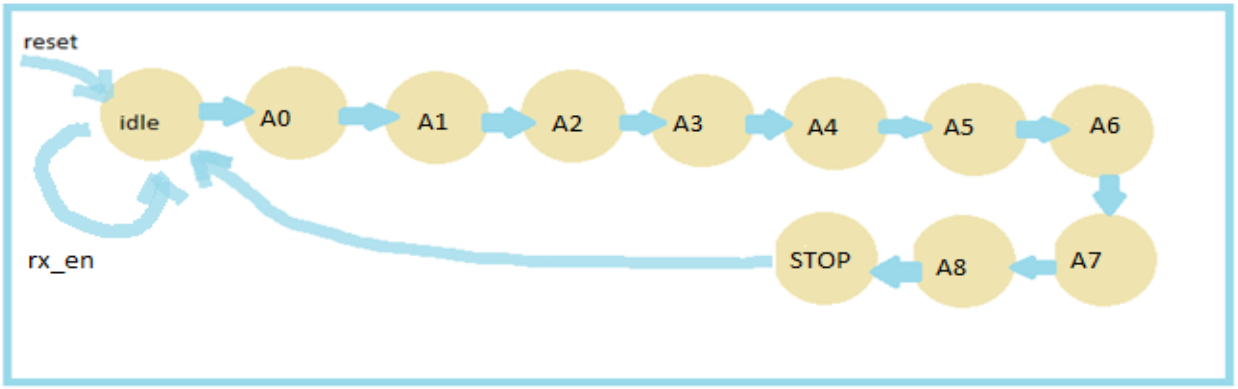
Μετά ξεκινάει η αρχιτεκτονική και το process μέσα στο οποίο δηλώνεται μια μεταβλητή 8bits.

```
ARCHITECTURE bhv OF rx_UART2 IS
        SIGNAL present_state,next_state:state;
        SIGNAL rx_en:std_logic;
        SIGNAL rx_8bits:std_logic_vector(7 downto 0);
        SIGNAL my_counter:INTEGER RANGE 0 TO 1023;

  BEGIN

comp1:PROCESS(present_state, clk) --
        VARIABLE q:std_logic_vector(7 downto 0);
```

Μέσα στο process δίνονται πάλι οι καταστάσεις(όπως και στο Transmitter)για να γίνει το serializing. Κάθε φορά που το my_counter=520(κάτι παραπάνω από το μισό του 1024) βλέπει το πρόγραμμά μας αν έχουμε bit πληροφορίας και μεταβαίνει στην επόμενη κατάσταση. Ξεκινώντας από την idle,μεταβαίνοντας στην A0,διαδοχικά στις επόμενες και τέλος στο STOP BIT. Τέλος επανέρχεται στην κατάσταση idle και τελειώνει το process. Αυτό παραθέτεται και στο γράφημα παρακάτω.

```
BEGIN
        IF clk'EVENT AND clk='1' THEN
        CASE present_state IS
            WHEN idle=>
                q:=(OTHERS=>'0');
                rx_en<='0';
                rx_8bits<=(OTHERS=>'0');
                IF rx='0' THEN
                    next_state<=A0;
                ELSE
                    next_state<=idle;
                END IF;
            WHEN A0=>
                                next_state<=A1;
            WHEN A1=>
                                IF (my_counter=520) THEN
                                q:=rx&q(7 downto 1);

                                END IF;
                                next_state<=A2;
            WHEN A2=>
                                IF (my_counter=520) THEN
                                q:=rx&q(7 downto 1);

                                END IF;
                                next_state<=A3;
            WHEN A3=>
                                IF (my_counter=520) THEN
                                q:=rx&q(7 downto 1);

                                END IF;
                                next_state<=A4;
            WHEN A4=>
                                IF (my_counter=520) THEN
                                q:=rx&q(7 downto 1);

                                END IF;
```

```
                    next_state<=A5;
        WHEN A5=>
                    IF (my_counter=520) THEN
                    q:=rx&q(7 downto 1);

                    END IF;
                    next_state<=A6;
        WHEN A6=>
                    IF (my_counter=520) THEN
                    q:=rx&q(7 downto 1);

                    END IF;
                    next_state<=A7;
        WHEN A7=>
                    IF (my_counter=520) THEN
                    q:=rx&q(7 downto 1);

                    END IF;
                    next_state<=A8;
        WHEN A8=>
                    IF (my_counter=520) THEN
                    q:=rx&q(7 downto 1);

                    END IF;
                    next_state<=stop;
        WHEN stop=>
                    IF (my_counter=520) THEN
                    rx_en<=rx;
                    END IF;
                    next_state<=idle;
    END CASE;
    rx_8bits<=q;
    END IF;
END PROCESS;
```



Σχήμα 3.11 Η διαδικασία μετάβασης από την μια κατάσταση στην άλλη όπως φαίνεται στην εντολή επιλογής case του κώδικα VHDL

Στο επόμενο βήμα δίνεται άλλη μια process στην οποία υπάρχει μια μεταβλητή που αυτή τη φορά είναι για 11καταστάσεις άρα 11bits πληροφορίας, γι αυτό το range της μεταβλητής είναι μέχρι το 2047 (προβλέπονται 11bits άρα 2048 θέσεις μνήμης) και στη συνέχεια ελέγχει αν έχουν τελειώσει όλες οι καταστάσεις για να εξάγει τον χαρακτήρα στην έξοδο rx_data.

Μετά τελειώνει η process και έτσι τελειώνει και ο κώδικας για τον Receiver.

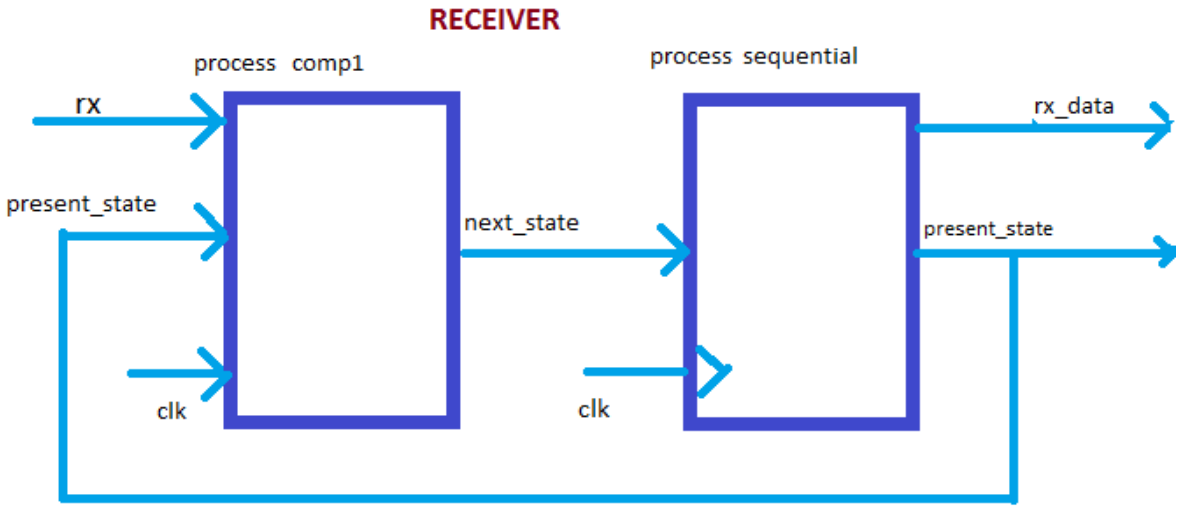
```
sequential:PROCESS(clk,reset)
VARIABLE rx_counter: INTEGER RANGE 0 TO 2047;
BEGIN
        IF reset='1' THEN
                present_state<=idle;
                rx_counter:=0;
        ELSIF clk'EVENT AND clk='1' THEN
                IF(next_state = A0) THEN--
                rx_counter:=0;--
                present_state<=next_state;--
                ELSE
                rx_counter:=rx_counter+1;
                IF rx_counter>1041 THEN
                rx_counter:=0;
                END IF;
                        IF(rx_counter=1041) THEN
                        present_state<=next_state;
                        END IF;
                END IF;
                rx_done<=rx_en;
                IF rx_en='1' THEN
                rx_data<=rx_8bits;
                END IF;
                my_counter<=rx_counter;
        END IF;
END PROCESS;
END bhv;
```

Τα pins που χρησιμοποιήθηκαν για τον προγραμματισμό του FPGA για τον receiver δίνονται στον πίνακα 3.2.

Node Name	Direction	Location
clk	Unknown	PIN_55
reset	Unknown	PIN_47
rx	Unknown	PIN_8
rx_data[7]	Unknown	PIN_26
rx_data[6]	Unknown	PIN_27
rx_data[5]	Unknown	PIN_28
rx_data[4]	Unknown	PIN_29
rx_data[3]	Unknown	PIN_30
rx_data[2]	Unknown	PIN_31
rx_data[1]	Unknown	PIN_32
rx_data[0]	Unknown	PIN_33
rx_done	Unknown	PIN_13

Πίνακας 3.2 Pins για το Receiver

Τέλος η μηχανή καταστάσεων που εφαρμόστηκε για τον receiver δίνεται στο σχήμα 3.12 .



Σχήμα 3.12 Μηχανή καταστάσεων για τον Receiver

3.4 Οι κώδικες σε VHDL

Οι κώδικες που χρησιμοποιήθηκαν σε γλώσσα VHDL δίνονται ολοκληρωμένοι παρακάτω με σχόλια *(τα σχόλια δίνονται με πράσινο χρώμα)* **.**

3.4.1 Κώδικας για τον πομπό (TRANSMITTER)

To Baud_rate_gen.vhd

```
LIBRARY ieee; --δήλωση βιβλιοθήκης
USE ieee.std_logic_1164.all; -- δήλωση πακέτου βιβλιοθήκης
USE ieee.std_logic_unsigned.all; --δήλωση πακέτου βιβλιοθήκης για μη προσημασμένους

ENTITY baud_rate_gen IS
      PORT(clk,reset:in std_logic; -- δήλωση εισόδων και εξόδων
      baud_clk: out std_logic);
END baud_rate_gen;

ARCHITECTURE baud OF baud_rate_gen IS
SIGNAL s_clk: std_logic :='0';  --δήλωση σήματος που λειτουργεί μέσα στην αρχιτεκτονική
BEGIN
sampling:
PROCESS(clk,reset)
VARIABLE sample:std_logic_vector(9 DOWNTO 0);-- δήλωση μεταβλητής
      BEGIN
            IF reset='1' THEN
                  sample:=(OTHERS=>'0');--το sample μηδενίζεται όταν reset=1
            ELSIF clk 'EVENT AND clk='1' THEN
                  sample:=sample+1; -- όταν το resetδεν είναι 1 το sample αυξάνεται
                  IF sample=520 THEN --  fosc/(2xBaude_Rate)
                        s_clk<=NOT s_clk; --όταν φτάσει στο 520 το s_clk αλλάζει
                                          --κατάσταση
                        sample:=(OTHERS=>'0');
                  END IF;
            END IF;
END PROCESS;
baud_clk<=s_clk; --το baud_clk παίρνει στην έξοδο την τιμή του s_clk
END baud;
```

To Transmit.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;--δήλωση πακέτου βιβλιοθήκης μη προσημασμένων
USE ieee.std_logic_signed.all; --δήλωση πακέτου βιβλιοθήκης  προσημασμένων

ENTITY transmit IS
      PORT(baud_clk, reset:IN std_logic;-- δήλωση εισόδων και εξόδων
            tx_data:IN std_logic_vector(7 downto 0);
```

```
            tx_en:IN std_logic;
            tx: OUT std_logic;
            tx_done:OUT std_logic);
END transmit;

ARCHITECTURE bhv OF transmit IS
TYPE state IS (idle,start,T1,T2,T3,T4,T5,T6,T7,T8,stop);--ορισμός τύπου απαρίθμησης
                                                        --με όνομα state
    SIGNAL present_state,next_state:state;-- ορισμός του present_state και
                                          --next_state ως τύπο state
    SIGNAL tx_do: std_logic;

BEGIN
comp1:PROCESS(present_state,baud_clk,tx_en)
    VARIABLE q:std_logic_vector(7 downto 0);--δήλωση μεταβλητής
    BEGIN
        CASE present_state IS--εντολή επιλογής για την τιμή που παίρνει το
                           --present_state
            WHEN idle=>
                tx_do<='0';
                tx<='1';
                IF tx_en='1' THEN
                    q:=tx_data;
                    next_state<=start;
                ELSE
                    next_state<=idle;
                END IF;

            WHEN start=>
            IF baud_clk='1'THEN
                    tx<='0'; --start bit
                    next_state<=T1;
            END IF;

            WHEN T1=>
            IF baud_clk='1' THEN
                    tx<=q(0);
                    next_state<=T2;
            END IF;

            WHEN T2=>
```

Όταν βρίσκεται σε κάθε κατάσταση βάζει διαδοχικά στην μεταβλητή q τον χαρακτήρα που δίνουμε. Κάθε φορά που τελειώνει η μια κατάσταση μεταβαίνουμε στην επόμενη με το next_state.

```
IF baud_clk='1' THEN
                tx<=q(1);
                next_state<=T3;
END IF;

WHEN T3=>
IF baud_clk='1' THEN
                tx<=q(2);
                next_state<=T4;
END IF;

WHEN T4=>
IF baud_clk='1' THEN
                tx<=q(3);
                next_state<=T5;
END IF;

WHEN T5=>
IF baud_clk='1' THEN
                tx<=q(4);
                next_state<=T6;
END IF;

WHEN T6=>
IF baud_clk='1' THEN
                tx<=q(5);
                next_state<=T7;
END IF;

WHEN T7=>
IF baud_clk='1' THEN
                tx<=q(6);
                next_state<=T8;
END IF;

WHEN T8=>
IF baud_clk='1' THEN
                tx<=q(7);
                next_state<=stop;
END IF;
```

```
                WHEN stop=>
                IF baud_clk='1'THEN
                                tx<='1';--stop bit
                                next_state<=idle;
                END IF;
                IF baud_clk='0' THEN--αφού έχει δοθεί ολόκληρος ο χαρακτήρας
                        x_do<='1';      -- και το baud_clk=0 το tx_do=1

                END IF;

        END CASE;
    END PROCESS;

sequential:PROCESS(baud_clk,reset)
BEGIN
    IF reset='1' THEN
        present_state<=idle;
    ELSIF baud_clk 'EVENT AND baud_clk='1' THEN
        present_state<=next_state;
    END IF;
END PROCESS;

comp2:PROCESS(present_state)
    BEGIN
        tx_done<=tx_do;-- δίνουμε στο tx_done την τιμή του tx_do,1 όταν
                        --τελειώνει ο χαρακτήρας
    END PROCESS;
END bhv;
```

To top_level_trans.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY top_level_trans IS
    PORT(clk,reset:in std_logic;-- δήλωση εισόδων και εξόδων
         tx_data: in std_logic_vector(7 downto 0);--transmit data
         tx_en:IN std_logic; --when 1 it transmits
         tx_out: OUT std_logic; --transmit pin
```

```
            tx_done_out:OUT std_logic; -- is 1 when transmit is done
            baud_clk: out std_logic);
END top_level_trans;

ARCHITECTURE top OF top_level_trans IS
SIGNAL my_baud_clk: std_logic;
SIGNAL my_tx:std_logic;

COMPONENT baud_rate_gen IS –βάζουμε σαν δομικό στοιχείο το baud_rate_gen για
                           --να μπορέσουμε να το συμπεριλάβουμε στο κυρίως
                           --πρόγραμμα
    PORT(clk,reset:in std_logic;
    baud_clk: out std_logic);
END COMPONENT;

COMPONENT transmit IS      --βάζουμε σαν δομικό στοιχείο το transmit για να
                           --μπορέσουμε να το συμπεριλάβουμε στο κυρίως
                           --πρόγραμμα
    PORT(baud_clk, reset:IN std_logic;
            tx_data:IN std_logic_vector(7 downto 0);
            tx_en:IN std_logic;
            tx: OUT std_logic;
            tx_done:OUT std_logic);
END COMPONENT;

BEGIN
baud: baud_rate_gen PORT MAP(clk, reset, my_baud_clk);--στιγμιότυπο του
                                                     --υποκυκλώματος
                                                     --baud_rate_gen
baud_clk<=my_baud_clk;--το baud_clk παίρνει τιμή από το my_baud_clk
tx_out<=my_tx;--το tx_out παίρνει τιμή από το my_tx
Tx: transmit PORT MAP(my_baud_clk, reset, tx_data, tx_en, my_tx, tx_done_out); --
                           -- στιγμιότυπο του υποκυκλώματος baud_rate_gen
END top;
```

3.4.2 Κώδικας για τον δέκτη (Receiver)

To my_package.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE my_package IS--δημιουργία πακέτου για χρήση ως βιβλιοθήκη
TYPE state IS (idle,A0,A1,A2,A3,A4,A5,A6,A7,A8,stop);
END PACKAGE;
```

To rx_UART.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_signed.all;
USE work.my_package.all;--states are defined here—δήλωση του πακέτου

ENTITY rx_UART2 IS
      PORT(clk, reset:IN std_logic;--δήλωση εισόδων και εξόδων
                  rx:IN std_logic;
                  rx_data:OUT std_logic_vector(7 downto 0);
                  rx_done:OUT std_logic);
END rx_UART2;

ARCHITECTURE bhv OF rx_UART2 IS
      SIGNAL present_state,next_state:state;--δήλωση σημάτων
      SIGNAL rx_en:std_logic;
      SIGNAL rx_8bits:std_logic_vector(7 downto 0);
      SIGNAL my_counter:INTEGER RANGE 0 TO 1023;

  BEGIN

comp1:PROCESS(present_state, clk) --
      VARIABLE q:std_logic_vector(7 downto 0);--δήλωση μεταβλητής
      BEGIN
            IF clk'EVENT AND clk='1' THEN
            CASE present_state IS--εντολή επιλογής για το present_state
                  WHEN idle=>
```

```
        q:=(OTHERS=>'0');
        rx_en<='0';
        rx_8bits<=(OTHERS=>'0');
        IF rx='0' THEN
                next_state<=A0;
        ELSE
                next_state<=idle;
        END IF;
WHEN A0=>
                        next_state<=A1;
WHEN A1=>
                        IF (my_counter=520) THEN
                        q:=rx&q(7 downto 1);

                        END IF;
                        next_state<=A2;
WHEN A2=>
                        IF (my_counter=520) THEN
                        q:=rx&q(7 downto 1);

                        END IF;
                        next_state<=A3;
WHEN A3=>
                        IF (my_counter=520) THEN
                        q:=rx&q(7 downto 1);

                        END IF;
                        next_state<=A4;
WHEN A4=>
                        IF (my_counter=520) THEN
                        q:=rx&q(7 downto 1);

                        END IF;
                        next_state<=A5;
WHEN A5=>
                        IF (my_counter=520) THEN
                        q:=rx&q(7 downto 1);

                        END IF;
                        next_state<=A6;
WHEN A6=>
```

Ξεκινά η επιλογή κατάστασης και μεταβαίνει στην επόμενη κατάσταση αφού πρώτα πάρει τιμή το q.αυτό γίνεται για όλες τις καταστάσεις από την idle μέχρι και την stop όπως δίνεται η σειρά στο TYPE state.

```
                                    IF (my_counter=520) THEN
                                    q:=rx&q(7 downto 1);

                                    END IF;
                                    next_state<=A7;
                WHEN A7=>
                                    IF (my_counter=520) THEN
                                    q:=rx&q(7 downto 1);

                                    END IF;
                                    next_state<=A8;
                WHEN A8=>
                                    IF (my_counter=520) THEN
                                    q:=rx&q(7 downto 1);

                                    END IF;
                                    next_state<=stop;
                WHEN stop=>
                                    IF (my_counter=520) THEN
                                    rx_en<=rx;
                                    END IF;
                                    next_state<=idle;
        END CASE;
        rx_8bits<=q;--ο rx_8bits παίρνει την τιμή του q που αντιπροσωπεύει τον
                    --χαρακτήρα
        END IF;
    END PROCESS;

sequential:PROCESS(clk,reset)
VARIABLE rx_counter: INTEGER RANGE 0 TO 2047;--μεταβλητή εύρους 2048 θέσεων
                                --μνήμης γιατί έχουμε 11bits πληροφορίας, 2^11=2048
BEGIN
    IF reset='1' THEN
        present_state<=idle;
        rx_counter:=0;
    ELSIF clk'EVENT AND clk='1' THEN
        IF(next_state = A0) THEN--αν ισχύει η συνθήκη μηδενίζει ο rx_counter και
                            --ξεκινάει από την αρχή η διαδικασία
            rx_counter:=0;
            present_state<=next_state;
        ELSE                --αλλιώς μετράει μέχρι το τέλος του μετώπου του
```

```
                                        --παλμού του clk για να μηδενίσει ο rx_counter
                        rx_counter:=rx_counter+1;
                        IF rx_counter>1041 THEN
                                rx_counter:=0;
                        END IF;
                        IF(rx_counter=1041) THEN
                                present_state<=next_state;
                        END IF;
                END IF;
                rx_done<=rx_en;
                IF rx_en='1' THEN
                        rx_data<=rx_8bits;
                END IF;
                my_counter<=rx_counter;
        END IF;
    END PROCESS;
    END bhv;
```

3.5 Η προσομοίωση του ελεγκτή στο πρόγραμμα Quartus II

Φτιάχνοντας την προσομοίωση λάβαμε υπόψη ότι δεν μπορούμε να έχουμε όλες τις θέσεις μνήμης που χρειαζόμαστε, λόγο μεγάλου όγκου. Διαμορφώσαμε λοιπόν το project μας έτσι ώστε να λειτουργεί αλλά να φαίνονται μόνο τρείς καταστάσεις από τις οκτώ (αφού ο χαρακτήρας που χρησιμοποιούμε στο UART είναι των 8 bits) .

Τα VHDL Files που χρησιμοποιήσαμε για την υλοποίηση του πομπού και του δέκτη τα εντάξαμε μέσα σε ένα άλλο από το οποίο παίρνουμε αποτελέσματα. Αυτό το VHDL αρχείο δίνεται στο παράρτημα.

Τα αποτελέσματα της προσομοίωσης φαίνονται στο σχήμα 3.13 .

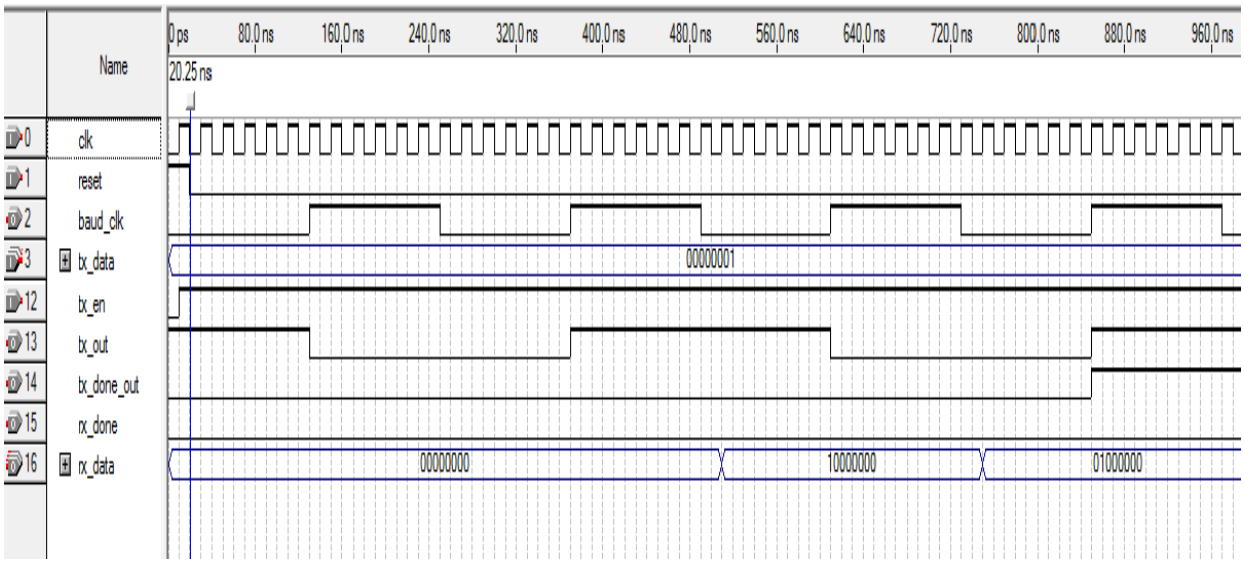
Το σήμα εξόδου baud_clk μας δίνει το διαμορφωμένο ρολόι (clk), το οποίο διαμορφώνεται στο Baud_rate_gen.vhd.

Το σήμα εισόδου tx_data είναι το σήμα που δίνουμε.

Το σήμα εξόδου tx_done_out μας δίνει μονάδα όταν ολοκληρωθεί η αποστολή του χαρακτήρα.

Το σήμα rx_data μας δείχνει πως αποστέλλεται ο χαρακτήρας διαδοχικά, εδώ βέβαια βλέπουμε μόνο τρεις καταστάσεις.

Τέλος, το σήμα rx_done μας δίνει μονάδα όταν έχει ληφθεί ολόκληρος ο χαρακτήρας. Εδώ δεν προλαβαίνει να φανεί γιατί τελειώνει η προσομοίωση.



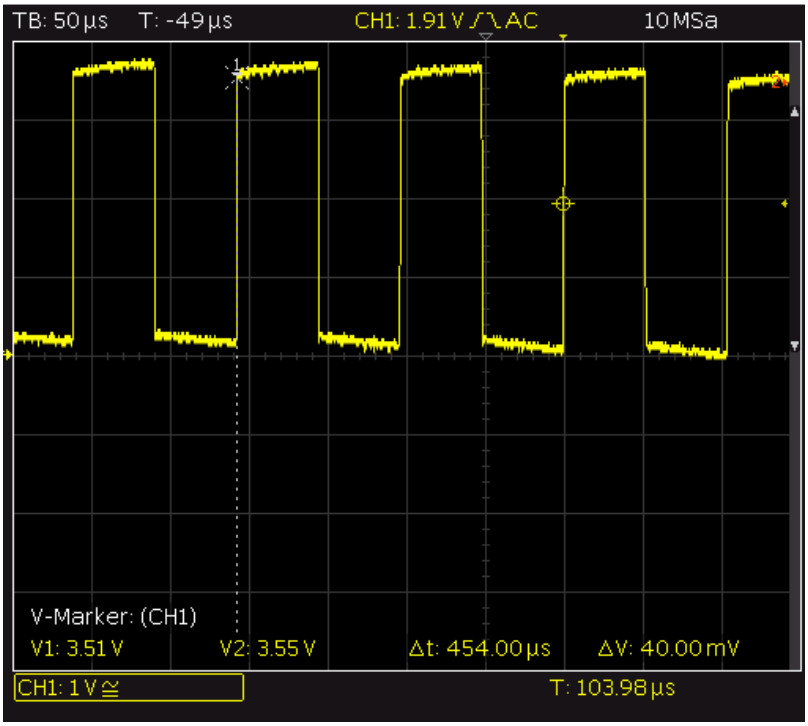
Σχήμα 3.13 Αποτελέσματα προσομοίωσης στο πρόγραμμα Quartus

3.6 Υλοποίηση και Μετρήσεις

Με τις μετρήσεις και τις δοκιμές που έγιναν ο παλμογράφος έβγαλε τα εξής αποτελέσματα:

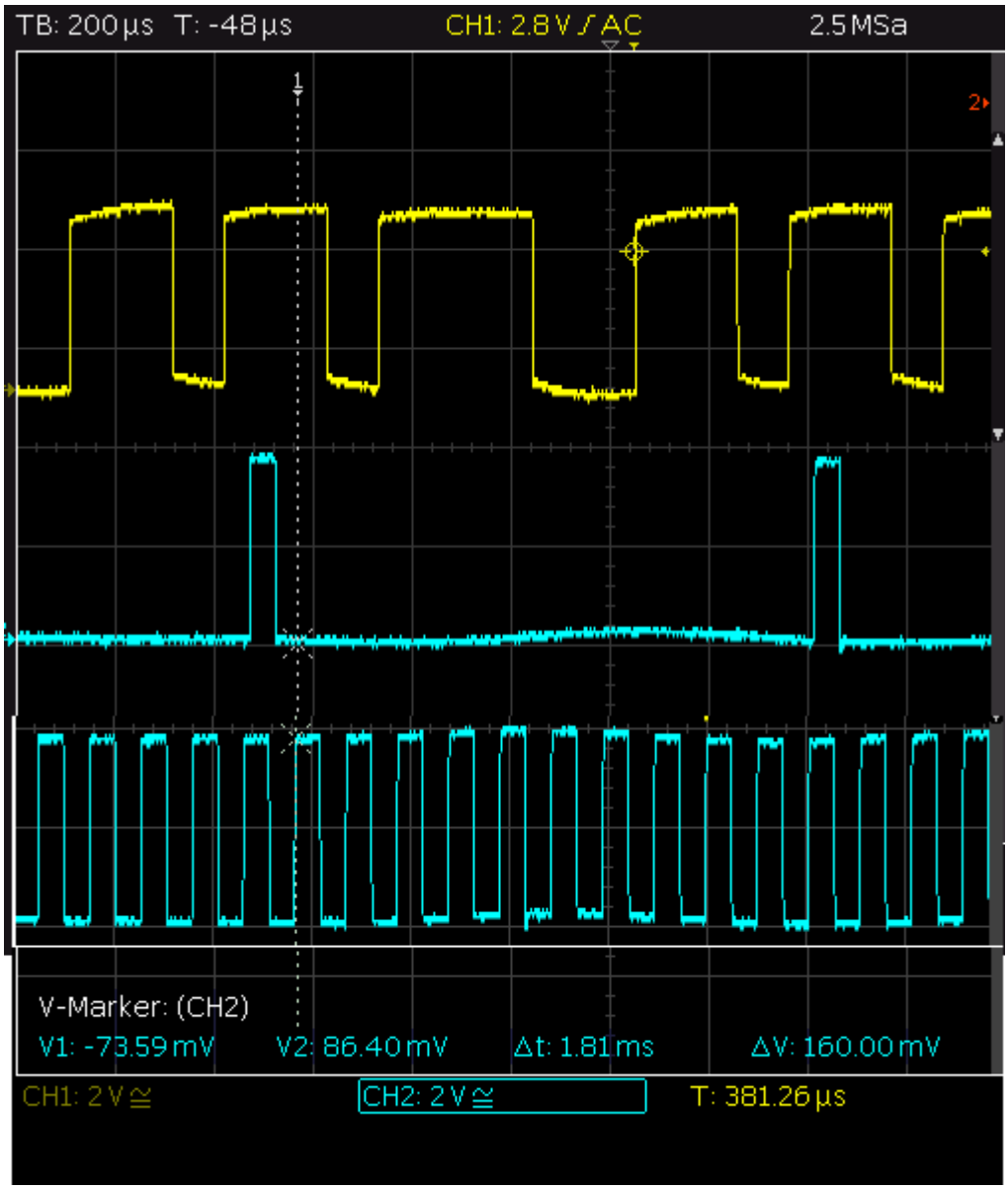
Για την λειτουργία του transmitter

Το Baud_clk έχει πλάτος 3,55Volt και περίοδο 103,98μs.



Σχήμα 3.14 Το διάγραμμα από τον παλμογράφο για το baud_clk

Τα αποτελέσματα για το tx_out,το tx_done_out και το baud_clk δίνονται μαζεμένα στο σχήμα 3.15



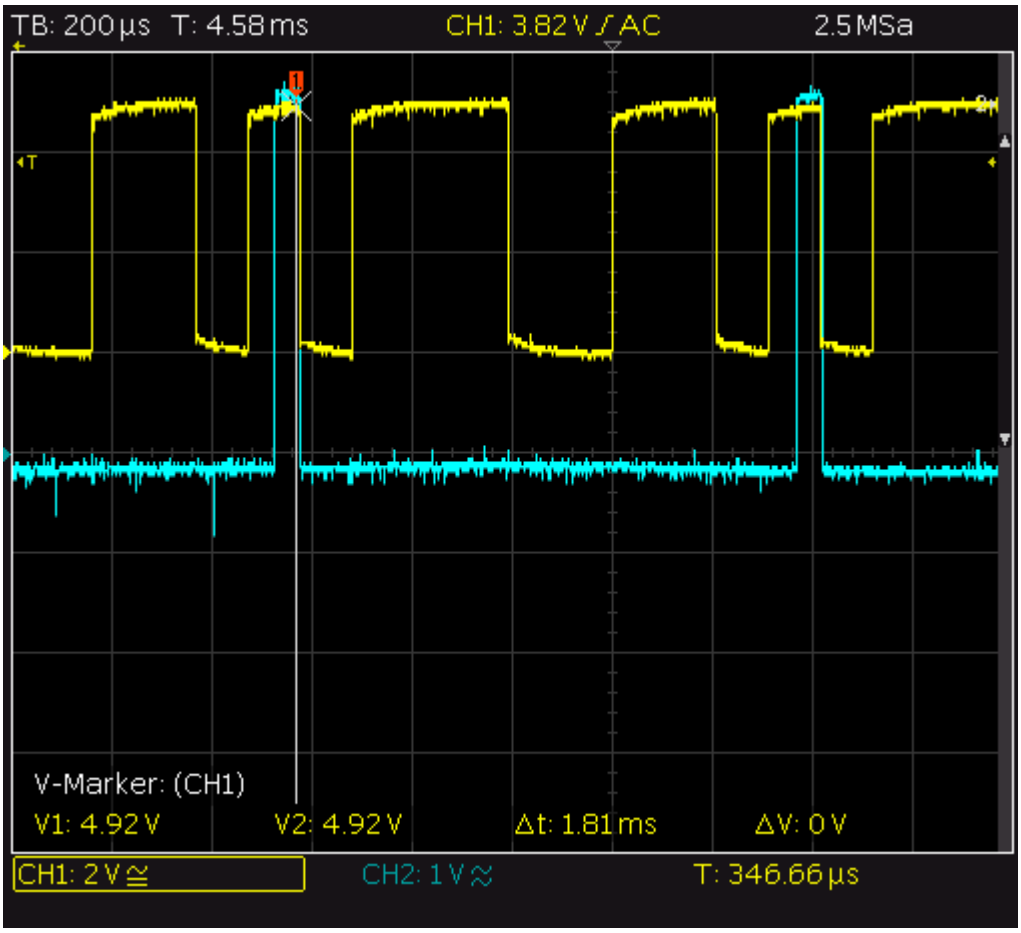
Σχήμα 3.15 Το διάγραμμα από τον παλμογράφο για το Transmitter

Βάζοντας και τα τρία σήματα στο γράφημά μας βλέπουμε ότι το Baud_clk είναι το τελευταίο, στη μέση δίνεται το tx_done_out και πρώτο δίνεται το σήμα για το tx_out. Το tx_out μας δίνει τον χαρακτήρα που δόθηκε ο οποίος είναι το 01100111. Στο αποτέλεσμα μετά το START BIT βλέπουμε το λιγότερο σημαντικό bit του χαρακτήρα που είναι το πρώτο από δεξιά. Είναι κατανοητό και από το γράφημα ότι ο χαρακτήρας τελειώνει όταν το

tx_done_out γίνει 1,μετά υπάρχει η κατάσταση idle και μετά πάλι δίνεται το START BIT.

Για την λειτουργία του receiver

Εδώ δίνοντας πάλι το ίδιο σήμα 01100111 αυτή τη φορά όμως το σήμα μας το δώσαμε στον υπολογιστή παίρνουμε τα παρακάτω αποτελέσματα.

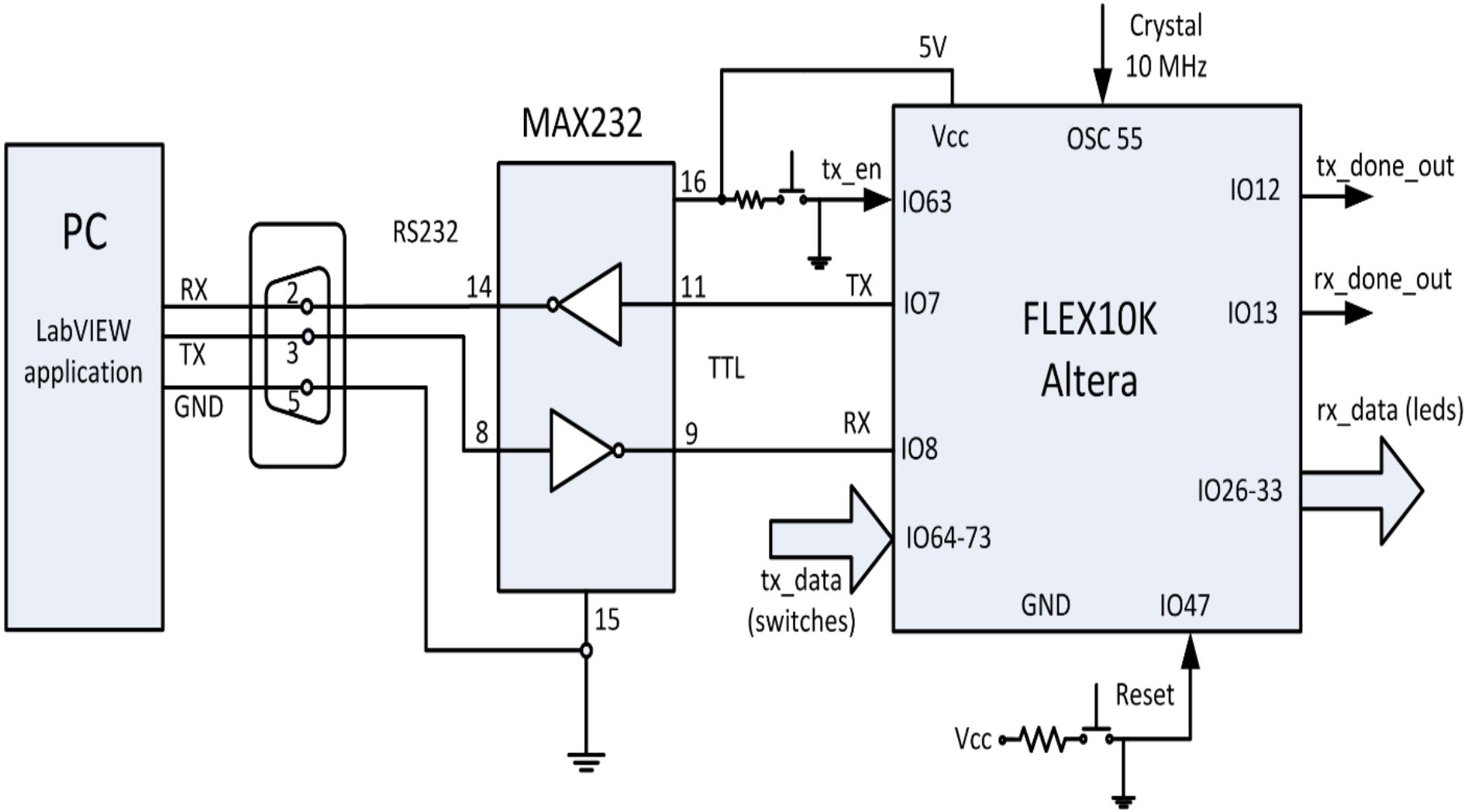


Σχήμα 3.16 Το διάγραμμα από τον παλμογράφο για το Receiver

Με το κίτρινο δίνεται το rx και με το γαλάζιο το rx_done. Στο αποτέλεσμα μετά το START BIT βλέπουμε το λιγότερο σημαντικό bit του χαρακτήρα που είναι το πρώτο από δεξιά. Κάθε φορά που τελειώνει ο χαρακτήρας το σύστημα μας το δείχνει με την rx_done.

Το διάγραμμα που δίνει όλη τη λειτουργία του ελεγκτή που κατασκευάστηκε δίνεται παρακάτω. Περιλαμβάνει το FPGA τον μετατροπέα στάθμης MAX232 την σειριακή θύρα και τον υπολογιστή ο οποίος ενώνεται με την θύρα.

Παρακάτω δίνεται το ολοκληρωμένο διάγραμμα του ελεγκτή ασύρματης σειριακής επικοινωνίας.



ολοκληρωμένο κύκλωμα ελεγκτή ασύρματης σειριακής επικοινωνίας

Παράρτημα

Στο παράρτημα αυτό δίνεται το VHDL αρχείο που χρησιμοποιήσαμε για να ενώσουμε τα VHDL αρχεία του πομπού και του δέκτη και να κάνουμε τον ελεγκτή του UART να λειτουργεί και σε επίπεδο προσομοίωσης με την βοήθεια του προγράμματος Quartus II της εταιρείας Altera.Αυτό είναι το

- Top_level4.vhd

Πιο αναλυτικά τα Files που χρησιμοποιήσαμε είναι:

Baud_rate_gen.vhd: διαμορφώνουμε το αρχικό ρολόι έτσι ώστε να αναπαραχθεί ένα νέο ρολόι με παλμό ίσο με 12 φορές τον αρχικό.

Transmit.vhd: γίνεται η επεξεργασία του σήματος-χαρακτήρα. Βλέπουμε δηλαδή ότι το σήμα μας λόγο ασύγχρονης επικοινωνίας ελέγχει αν έχει δοθεί αρχικό bit=0, δηλαδή START BIT, για να μεταβεί από την κατάσταση idle,δηλαδή λογικό 1, στην οποία βρίσκεται, στη λήψη σήματος το οποίο αποτελείται στην περίπτωση μας από 8 bits. Στη συνέχεια το σήμα μας δίνει το STOP BIT και μετά η κατάσταση γίνεται πάλι idle μέχρι να δώσουμε τον επόμενο χαρακτήρα-σήμα.

My_package.vhd: είναι το κομμάτι του κώδικα που φτιάξαμε για να χρησιμοποιήσουμε σαν βιβλιοθήκη, είναι δηλαδή ένα πακέτο (package) και περιέχει τον τύπο state τον οποίο και χρειαζόμαστε στο rx_UART2.vhd.

Rx_UART2.vhd: αντιπροσωπεύει τον δέκτη μας (receiver), τα δεδομένα που δέχεται σαν είσοδο τα βγάζει στην έξοδο.

Για να μπορούν όμως να λειτουργούν όλα τα παραπάνω VHDL Files πρέπει να είναι όλα συγκεντρωμένα κάτω από μια οντότητα η οποία θα έχει και το όνομα ολόκληρου του project μας. Εδώ το project το έχουμε ονομάσει ***top_level4.vhd*** και οι υπόλοιπες οντότητες εμφανίζονται σε αυτό σαν υποκυκλώματα(COMPONENTS).

Top_level4.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE work.my_package.all;

ENTITY top_level4 IS
      PORT(clk,reset:in std_logic;
            tx_data: in std_logic_vector(7 downto 0);--transmit data
            tx_en:IN std_logic; --when 1 it transmits
            tx_out: OUT std_logic; --transmit pin
            tx_done_out,rx_done:OUT std_logic; -- is 1 when transmit is done
            baud_clk: out std_logic;--outputs baud clock and sample clock for display
            rx_data:OUT std_logic_vector(7 downto 0));
END top_level4;

ARCHITECTURE top OF top_level4 IS
SIGNAL my_baud_clk: std_logic;
SIGNAL my_tx:std_logic;

COMPONENT baud_rate_gen IS --produces baud rate for the transmitter from
sampling rate
          PORT(clk,reset:in std_logic;
                baud_clk: out std_logic);
END COMPONENT;

COMPONENT transmit IS
          PORT(baud_clk, reset:IN std_logic;
                tx_data:IN std_logic_vector(7 downto 0);
                 tx_en:IN std_logic;
                 tx: OUT std_logic;
                 tx_done:OUT std_logic);
END COMPONENT;

COMPONENT rx_UART2 IS
          PORT(clk, reset:IN std_logic;
                rx:IN std_logic;
                rx_data: OUT std_logic_vector(7 downto 0);
                 rx_done:OUT std_logic);
END COMPONENT;

BEGIN
baud: baud_rate_gen PORT MAP(clk, reset, my_baud_clk);
baud_clk<=my_baud_clk;
tx_out<=my_tx;
Tx: transmit PORT MAP(my_baud_clk, reset, tx_data, tx_en, my_tx, tx_done_out);
Rx: rx_UART2 PORT MAP(clk, reset, my_tx, rx_data, rx_done);
END top;
```

Βιβλιογραφία & Ιστότοποι

- «Έλεγχος κυκλωμάτων και μετρήσεων με Η/Υ» Ι. Καλόμοιρος, Σ. Μπουλταδάκης, Ι. Πεταλάς, Εκδόσεις ΤΖΙΟΛΑ.
- «Σχεδίαση Ψηφιακών Κυκλωμάτων με την γλώσσα VHDL» Stephen Brown, Zvonko Vranesic, Εκδόσεις ΤΖΙΟΛΑ
- «Ψηφιακή σχεδίαση», τέταρτη έκδοση M. Morris Mano, Michael D. Ciletti, Εκδόσεις Παπασωτηρίου.
- «Circuit Design and Simulation with VHDL» Volnei A. Pedroni, second edition, The MIT Press, 2010
- «Εισαγωγή στην γλώσσα VHDL» Ι. Καλόμοιρος, ΤΕΙ ΣΕΡΡΩΝ

- http://digilib.lib.unipi.gr/dspace/bitstream/unipi/4989/1/Venetikidis.pdf
- http://www.altera.com/products/devices/flex10k/f10-index.html
- http://en.wikipedia.org/wiki/Main_Page