ΤΕΙ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη ενσωματωμένου συστήματος με δυνατότητα ασύρματης σύνδεσης σε κυψελοειδή δίκτυα, με χρήση HSPA modem, για διαδικτυακή πρόσβαση σε απομακρυσμένα ψηφιακά συστήματα.

Μπακογιάννης Δημήτριος
Α.Μ. 2214

Επιβλέπων Καθηγητής:
Κορνάρος Γεώργιος

Αφιέρωση

Αφιερώνω κάθε βήμα προς την ολοκλήρωση αυτής της εργασίας, αλλά και το τελικό αποτέλεσμα αυτής, στην Φραγκιουδάκη Χρυσούλα η οποία βρισκόταν παρούσα καθ' όλη την διάρκεια εκπόνησής της και με στήριξε με κάθε δυνατό τρόπο αλλά και πίστεψε στις ικανότητες μου.

Ευχαριστίες

Ολόθερμες και ειλικρινείς ευχαριστίες προς τον επιβλέποντα καθηγητή μου κ. Κορνάρο Γεώργιο που χάρη στη στήριξη, καθοδήγηση και εμπιστοσύνη που μου επέδειξε, καθ' όλη τη διάρκεια εκπόνησης της εργασίας, μπόρεσε αυτή να έρθει εις πέρας και να ικανοποιήσει τις αρχικές προσδοκίες.

Abstract

The purpose of this paper is the development of a system, consisting of hardware as well as software, which provide the ability of communication, interaction, and data exchanging between a user and remote devices and modules.

Many times, has been noted the necessity of accessing digital devices that are located remotely and either hold information that we need to know or waiting for data from the user for executing specific commands. Examples are sensors, microcontrollers and switches which find numerous applications from automated houses to robotic systems and weather stations. The system that was developed in this paper is a mediator between such user requirements and digital means such as those mentioned previously.

Part of this paper was carried out by using the development board ML403, which hosts a FPGA for the implementation of the necessary hardware. The design of the hardware as well as the development of the code executed by the embedded system of this paper was accomplished by using the Xilinx Platform Studio suite. More code was developed with Netbeans platform for the creation, in java language, of the necessary applications which give the remote user the opportunity to interact with the embedded system. There were also used sensors, microcontrollers and other electronic means to document the efficiency of the systems.

Σύνοψη

Σκοπός της πτυχιακής αυτής εργασίας είναι η ανάπτυξη ενός συστήματος, αποτελούμενου τόσο από υλικό (hardware) όσο και από λογισμικό(software) , το οποίο παρέχει την δυνατότητα επικοινωνίας, αλληλεπίδρασης και ανταλλαγής δεδομένων μεταξύ ενός χρήστη και απομακρυσμένων συσκευών και εξαρτημάτων.

Πολλές φορές, έχει διαπιστωθεί η ανάγκη πρόσβασης σε ψηφιακές συσκευές οι οποίες είναι τοποθετημένες απομακρυσμένα και είτε κρατούν πληροφορίες, τις οποίες χρειάζεται να γνωρίζουμε, είτε αναμένουν δεδομένα από τον χρήστη για την εκτέλεση εντολών. Παραδείγματα αποτελούν αισθητήρες, μικροελεγκτές και διακόπτες τα οποία βρίσκουν πληθώρα εφαρμογών από αυτοματοποιημένα σπίτια έως ρομποτικά συστήματα και μετεωρολογικούς σταθμούς. Το σύστημα που αναπτύχθηκε εδώ αποτελεί έναν διαμεσολαβητή μεταξύ τέτοιων αναγκών του χρήστη και των ψηφιακών μέσων που αναφέρονται παραπάνω.

Μέρος της εργασίας αυτής πραγματοποιήθηκε με χρήση της αναπτυξιακής πλακέτας ML403, που φιλοξενεί ένα FPGA για την υλοποίηση του απαραίτητου υλικού. Ο σχεδιασμός του υλικού αλλά και η ανάπτυξη του κώδικα που εκτελεί το ενσωματωμένο σύστημα της εργασίας πραγματοποιήθηκε με χρήση της σουίτας Xilinx Platform Studio. Επιπλέον κώδικας αναπτύχθηκε με την πλατφόρμα Netbeans για την δημιουργία, σε γλώσσα java, των αναγκαίων εφαρμογών που δίνουν την ευκαιρία στον απομακρυσμένο χρήστη να αλληλεπιδράσει με το ενσωματωμένο σύστημα. Χρησιμοποιήθηκαν δε αισθητήρες, μικροελεγκτές και άλλα ηλεκτρονικά μέσα για την τεκμηρίωση της αποτελεσματικότητας του συστήματος.

Πίνακας περιεχομένων

1. Εισαγωγή

Η εργασία αυτή υλοποιεί ένα διπύρηνο ενσωματωμένο σύστημα σε αναπτυξιακή πλακέτα FPGA το οποίο ενεργεί ως ένας διαμεσολαβητής μεταξύ ψηφιακών συσκευών (αισθητήρες, μικροελεγκτές κλπ.) και ενός χρήστη. Ο χρήστης ανά πάσα στιγμή μπορεί να αντλήσει πληροφορίες από τις συσκευές αυτές ή και να ασκήσει επιρροή πάνω τους με την χρήση του συστήματος που αναλύεται σε αυτή την εργασία.

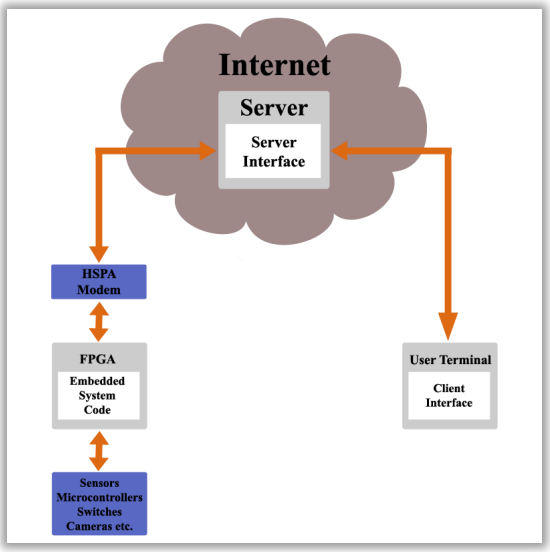
Είναι πολλές οι περιπτώσεις όπου έχει προκύψει η ανάγκη για ασύρματη πρόσβαση σε απομακρυσμένες ψηφιακές συσκευές. Παράδειγμα θα μπορούσε να είναι ένας μετεωρολογικός σταθμός, σε κάποιο δυσπρόσιτο σημείο, όπου είναι αναγκαία η απόκτηση δεδομένων από τους εγκατεστημένους σε αυτόν αισθητήρες προκειμένου να διεξαχθούν τα απαραίτητα συμπεράσματα για τις καιρικές συνθήκες. Ένα άλλο παράδειγμα θα μπορούσε να είναι ένα τηλεκατευθυνόμενο σύστημα το οποίο αναμένει δεδομένα από το χρήστη προκειμένου να ακολουθήσει την απαραίτητη πορεία ή ένα αυτοματοποιημένο σπίτι του οποίου είναι επιθυμητός ο έλεγχος (φώτα, σύστημα ποτίσματος, θερμοσίφωνας, σύστημα ασφαλείας κλπ).

Το ενσωματωμένο σύστημα που αναλύεται εδώ αποτελεί μια συσκευή στην οποία μπορεί ένας διαχειριστής να προσαρτήσει ψηφιακά συστήματα όπως αισθητήρες, κάμερες, διακόπτες και μικροελεγκτές. Η συσκευή υλοποιεί ασύρματη σύνδεση σε κυψελοειδή δίκτυα με την χρήση HSPA modem και προσφέρει την δυνατότητα στον απομακρυσμένο χρήστη να έχει πρόσβαση σε αυτά τα συστήματα μέσω του παγκόσμιου ιστού.

Η εργασία αυτή, όμως, είναι παραπάνω από ένα ενσωματωμένο σύστημα καθώς απαιτείται η ύπαρξη ενός περιβάλλοντος διάδρασης ώστε ο απομακρυσμένος χρήστης να μπορεί να αποκτήσει πρόσβαση στο ενσωματωμένο σύστημα και κατ' επέκταση να ασκήσει επιρροή στα προαναφερθέντα ψηφιακά συστήματα. Ως συνέπεια αναπτύχθηκε κατάλληλο λογισμικό, σε γλώσσα java, για τον παραπάνω χρήστη που ικανοποιεί αυτή την ανάγκη.

Το ενσωματωμένο σύστημα και η εφαρμογή στο τερματικό αποτελούν τα δυο άκρα της διαδικτυακής επικοινωνίας. Η πολυπλοκότητα του διαδικτύου και η χρήση, στις περισσότερες περιπτώσεις, δυναμικών διευθύνσεων IP κάνει σχεδόν κάθε φορά ανέφικτη την άμεση επικοινωνία από το ένα άκρο στο άλλο. Η λύση που δόθηκε, για να γίνει τελικά εφικτή η επικοινωνία μεταξύ των δυο άκρων, ήταν η ανάπτυξη επιπλέον λογισμικού, επίσης σε γλώσσα java, το οποίο εκτελείται σε έναν server με στατική διεύθυνση IP και το οποίο λειτουργεί ως μεσολαβητής που μεταβιβάζει τα πακέτα επικοινωνίας μεταξύ των δυο άκρων.

Από τα παραπάνω προκύπτει η εργασία αυτή, ως σύνολο, να αποτελείται από το ενσωματωμένο σύστημα, την εφαρμογή στο τερματικό του χρήστη και την ενδιάμεση εφαρμογή στον server. Τα δυο άκρα συνδέονται με την εφαρμογή του server και πλέον στέλνουν δεδομένα με παραλήπτη την άλλη πλευρά. Δεδομένα που στέλνονται από το τερματικό του χρήστη προς το ενσωματωμένο σύστημα αρχικά λαμβάνονται από την εφαρμογή του server η οποία στη συνέχεια θα τα προωθήσει προς τον τελικό παραλήπτη. Αντίστοιχα και αντίστροφα, δεδομένα από το ενσωματωμένο σύστημα προς τον χρήστη αποστέλλονται ασύρματα μέσω του HSPA modem προς την εφαρμογή του server η οποία θα τα προωθήσει στο τερματικό του χρήστη.



Σχήμα 1.1 Το σύστημα αυτής της εργασίας ως σύνολο

Το ενσωματωμένο σύστημα αυτής της εργασίας αναπτύχθηκε στην αναπτυξιακή πλακέτα ML403 της εταιρείας Xilinx και ο προγραμματισμός του πραγματοποιήθηκε με χρήση του λογισμικού Xilinx ISE Design Suite 12.2 της ίδιας εταιρείας. Οι εφαρμογές στο τερματικό του χρήστη και στον server δημιουργήθηκαν με χρήση της πλατφόρμας Netbeans 7.1.

1.1 Κίνητρο για την διεξαγωγή της εργασίας

Τα ενσωματωμένα συστήματα έχουν κατακτήσει ένα μεγάλο μερίδιο τεχνολογικών εφαρμογών καθώς δίνουν λύση σε πολλές ανάγκες τις καθημερινότητας όπως κινητή τηλεφωνία, επικοινωνίες, αυτοματισμοί, διασκέδαση, οικιακές συσκευές και μεταφορές. Έχουν στις περισσότερες περιπτώσεις υψηλή, σε σχέση με το χαμηλό τους κόστος, απόδοση και ταυτόχρονα χαμηλή κατανάλωση ενέργειας. Έρευνα και ανάπτυξη βρίσκεται σε συνεχή εξέλιξη, όσον αφορά τα ενσωματωμένα συστήματα, και ως συνέπεια προστίθενται ολοένα και περισσότερες τεχνολογίες και καινοτομίες που βασίζονται σε αυτά.

Μια ανάγκη, που πιθανότατα πάντα ήταν επιθυμητή, είναι η ύπαρξη ενός τρόπου πρόσβασης αλλά και ελέγχου απομακρυσμένων συσκευών. Ο έλεγχος των συστημάτων ασφαλείας ενός σπιτιού ενώ ο ιδιοκτήτης βρίσκεται σε διακοπές ή το πότισμα των λουλουδιών με το πάτημα ενός κουμπιού ενώ ο ιδιοκτήτης συνεχίζει να βρίσκεται σε διακοπές. Η τηλεκατεύθυνση ενός οχήματος το οποίο βρίσκεται στη Γαλλία από έναν χειριστή που βρίσκεται στη Κίνα. Πρόσβαση σε αισθητήρες ενός απομακρυσμένου μετεωρολογικού σταθμού για άμεση απόκτηση δεδομένων σχετικά με τις καιρικές συνθήκες. Ζωντανή παρακολούθηση της πορείας ενός οχήματος από το τερματικό ενός χρήστη.

Οι παραπάνω είναι μερικές από τις πολλές περιπτώσεις που ανήκουν στην κατηγορία πρόσβασης και ελέγχου απομακρυσμένων συσκευών και αποτέλεσαν το κίνητρο για την διεξαγωγή αυτής της εργασίας. Το ενσωματωμένο σύστημα που υλοποιήθηκε έρχεται να ικανοποιήσει ανάγκες, όπως τις προαναφερθείσες, και ενώ έχουν αναπτυχθεί πολλές τεχνολογίες για τέτοιους σκοπούς η συγκεκριμένη υλοποίηση στοχεύει στο να προσφέρει τον ταυτόχρονο έλεγχο ποικίλων τύπων συσκευών με ευελιξία και αξιοπιστία.

1.2 Σκοπός και στόχοι της εργασίας

Οι στόχοι στα πλαίσια αυτής της εργασίας είναι οι εξής:

- Ανάπτυξη ενός συστήματος με δυνατότητα ασύρματης σύνδεσης σε κυψελοειδή δίκτυα, με χρήση HSPA modem, για διαδικτυακή πρόσβαση σε απομακρυσμένα ψηφιακά συστήματα.
- Υλοποίηση κατάλληλου hardware για τις ανάγκες του ενσωματωμένου συστήματος που αποτελεί τμήμα αυτής της εργασίας.
- Ανάπτυξη του απαραίτητου κώδικα που θα εκτελέσει το ενσωματωμένο σύστημα για την επικοινωνία του τόσο με τον απομακρυσμένο χρήστη όσο και με τα ψηφιακά συστήματα.
- Ανάπτυξη κατάλληλου software για τις εφαρμογές που εκτελούνται στο τερματικό του απομακρυσμένου χρήστη αλλά και στον server.
- Έλεγχος ορθής λειτουργίας του συστήματος ως σύνολο και πειραματισμός με ποικίλα ψηφιακά συστήματα, που προσαρτώνται στο ενσωματωμένο σύστημα.

1.3 Δομή της εργασίας

Το υπόλοιπο της εργασίας είναι οργανωμένο ως εξής:

- Στο κεφάλαιο 2 δίνεται μια σύντομη εξήγηση σχετικά με τα ενσωματωμένα συστήματα αλλά και μιας κατηγορίας αυτών, τα FPGA-Based συστήματα. Σκοπός αυτού του κεφαλαίου είναι να προσφέρει στον αναγνώστη μια βασική εξοικείωση σχετικά με τα ενσωματωμένα συστήματα που χρησιμοποιούνται για να καλύψουν το υλικό (hardware) σε αυτή την εργασία.
- Στο κεφάλαιο 3 γίνεται αναφορά στις μεθόδους υλοποίησης του συστήματος αλλά και στα εργαλεία (υλικό και λογισμικό) που χρησιμοποιήθηκαν.
- Στο κεφάλαιο 4 αναλύεται η αρχιτεκτονική του υλικού (hardware) που υλοποιήθηκε ως τμήμα αυτής της εργασίας.
- Στο κεφάλαιο 5 αναλύεται το λογισμικό που αναπτύχθηκε τόσο για το ενσωματωμένο σύστημα όσο και για τις εφαρμογές στο τερματικό του χρήστη και τον server.
- Στο κεφάλαιο 6 γίνεται αναφορά σε επεκτάσεις και βελτιστοποιήσεις, που θα μπορούσαν να εφαρμοστούν μελλοντικά, αλλά και στα συμπεράσματα που προκύπτουν από την εκπόνηση αυτής της εργασίας.

1.4 Συναφείς εργασίες

Όπως αναφέρθηκε και νωρίτερα, υπάρχει μεγάλη έρευνα και ανάπτυξη, όσον αφορά τα ενσωματωμένα συστήματα, και προφανώς καινοτομίες έχουν εμφανιστεί και στα πλαίσια παροχής τεχνολογιών απομακρυσμένης πρόσβασης και ελέγχου. Πιθανότατα, υπάρχουν πολλά παραδείγματα στα οποία θα μπορούσε να γίνει αναφορά. Σε αυτή την ενότητα θα δοθεί σύντομη περιγραφή για μια αξιόλογη καινοτομία που ανήκει στον ίδιο χώρο με το σύστημα της εργασίας αυτής ενώ θα γίνει αναφορά σε σημεία που αυτή μπορεί να υπερτερεί ή να υστερεί.

1.4.1 Το σύστημα “Wireless Sensor Tag”

Πρόκειται για μια από τις δυο συναφείς καινοτομίες στις οποίες γίνεται αναφορά. Το σύστημα αυτό δίνει τη δυνατότητα σε έναν χρήστη να λαμβάνει πληροφορίες από απομακρυσμένους αισθητήρες αλλά και να τους ρυθμίζει κατ’ ανάγκη.

Βρίσκει, κατά κύριο λόγο, εφαρμογή σε οικιακούς χώρους χωρίς αυτό να είναι υποχρεωτικό. Ο χρήστης μπορεί να τοποθετήσει τον επιθυμητό αριθμό αισθητήρων στον χώρο του και να ενημερώνεται από απόσταση και άμεσα για αλλαγές όπως το άνοιγμα ή κλείσιμο θυρών, η αλλαγή της θερμοκρασίας, το επίπεδο υγρασίας ή ο εντοπισμός κίνησης.

Το βασικό εξάρτημα του συστήματος αυτού είναι μια συσκευή που ονομάζεται Ethernet Tag Manager. Πρόκειται, στην ουσία, για ένα ενσωματωμένο σύστημα το οποίο έχει πρόσβαση στο διαδίκτυο μέσω πρωτοκόλλου Ethernet και λειτουργεί ως διαχειριστής αισθητήρων. Μπορούν να συνδεθούν σε αυτό έως και 255 αισθητήρες οι οποίοι μέσω του διαχειριστή Ethernet Tag Manager μπορούν να αλληλεπιδράσουν με τον απομακρυσμένο χρήστη. Ο διαχειριστής αυτός βρίσκεται σε συνεχή επικοινωνία με τους συνδεδεμένους αισθητήρες και προωθεί στον απομακρυσμένο χρήστη δεδομένα από αυτούς. Ταυτόχρονα ο χρήστης είναι σε θέση να αποστείλει δεδομένα μέσω του διαχειριστή προς τους αισθητήρες και να τροποποιήσει τις ρυθμίσεις τους.

Ο απομακρυσμένος χρήστης μπορεί να αποκτήσει πρόσβαση στον διαχειριστή Ethernet Tag Manager και κατ’ επέκταση στους αισθητήρες είτε μέσα από τον υπολογιστή του, με χρήση ενός Web Browser, ή από ειδικά σχεδιασμένη εφαρμογή στο Android ή i-Phone κινητό του. Μπορεί επίσης να λάβει ειδοποιήσεις από αυτούς μέσω e-mail, twitter και μέσω των εφαρμογών στο κινητό του.

Οι αισθητήρες συνδέονται με τον διαχειριστή Ethernet Tag Manager ασύρματα και έχουν εμβέλεια 60 μέτρων. Έχουν πολύ μικρό μέγεθος, ποιότητα και υψηλή ακρίβεια και πλαισιώνονται από χαμηλή κατανάλωση μπαταρίας η οποία, ανάλογα με τις ρυθμίσεις του χρήστη, μπορεί να έχει διάρκεια έως και αρκετών χρόνων. Προς το παρόν για αυτό το σύστημα περιλαμβάνονται αισθητήρες κίνησης, υγρασίας και θερμοκρασίας. Σύντομα θα περιλαμβάνονται αισθητήρες επαφής, για πόρτες και παράθυρα, αισθητήρες ανίχνευσης υπέρυθρης ακτινοβολίας καθώς και θερμοστάτες.

Μερικά από τα πλεονεκτήματα του συστήματος Wireless Sensor Tag σε σχέση με το σύστημα αυτής της εργασίας είναι:

- Δυνατότητα προσθήκης μεγάλου αριθμού αισθητήρων.
- Πρόσβαση στους αισθητήρες τόσο από υπολογιστή όσο και από το κινητό.
- Ποικίλοι τρόποι άμεσης ειδοποίησης όπως e-mail, twitter και εφαρμογές στο κινητό.
- Οι αισθητήρες συνδέονται με το ενσωματωμένο σύστημα ασύρματα.
- Έξυπνες δυνατότητες όπως η αναπαραγωγή ήχου beep για ανεύρεση χαμένου αισθητήρα.

Πλεονεκτήματα του συστήματος αυτής της εργασίας σε σχέση με το σύστημα Wireless Sensor Tag είναι:

- Δυνατότητα πρόσβασης σε ποικίλες συσκευές, πέραν των αισθητήρων, και με ποικίλα πρωτόκολλα επικοινωνίας.
- Μεγάλη ευελιξία και δυνατότητες τροποποίησης.
- Μεγάλη γκάμα εφαρμογών μεταξύ των οποίων ρομποτικά συστήματα, αυτοματισμοί κτιρίων, έλεγχος αισθητήρων, τηλεκατεύθυνση οχημάτων κλπ.
- Δυνατότητα εκτέλεσης ενεργειών σε απομακρυσμένες συσκευές.

Πληροφορίες σχετικά με το σύστημα Wireless Sensor Tag μπορούν να βρεθούν στην επίσημη ιστοσελίδα του: *http://www.mytaglist.com/index.html.*

2. Ενσωματωμένα συστήματα, FPGA-Based συστήματα

Σε αυτό το κεφάλαιο θα γίνει αναφορά στα ενσωματωμένα συστήματα αλλά και στα FPGA-Based συστήματα που ανήκουν στην κατηγορία των ενσωματωμένων συστημάτων.

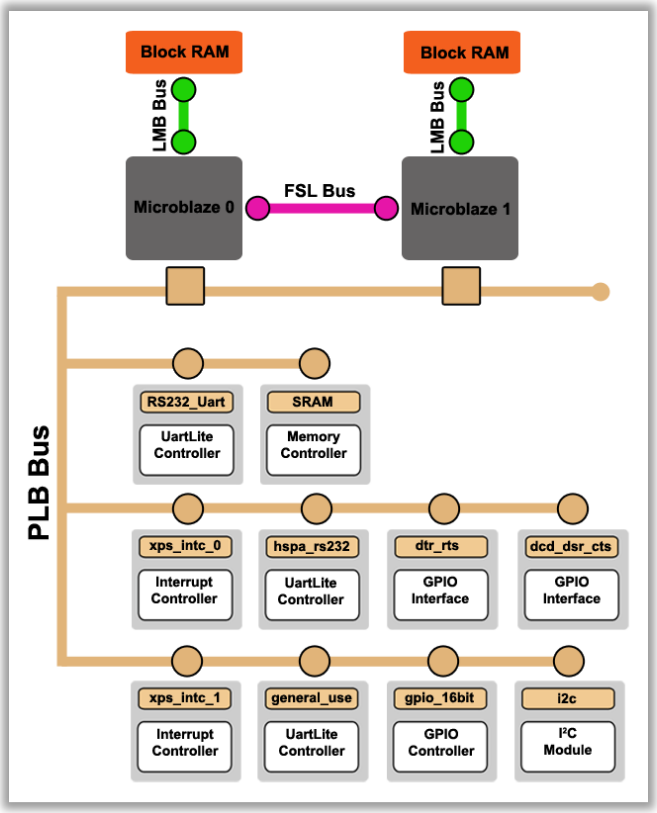
2.1 Ενσωματωμένα συστήματα

Τα ενσωματωμένα συστήματα είναι υπολογιστικά συστήματα με συγκεκριμένες λειτουργίες τα οποία αποτελούν μέρος ενός μεγαλύτερου μηχανικού ή ηλεκτρικού συστήματος συχνά με περιορισμούς λειτουργίας σε πραγματικούς χρόνους. Ενσωματώνονται ως τμήμα μιας ολοκληρωμένης συσκευής η οποία συχνά περιλαμβάνει τόσο hardware όσο και μηχανικά μέρη. Σε αντίθεση, οι υπολογιστές γενικού σκοπού, όπως οι προσωπικοί υπολογιστές, είναι σχεδιασμένοι να έχουν ευελιξία και να καλύπτουν ένα μεγάλο εύρος αναγκών του χρήστη.

Τα σύγχρονα ενσωματωμένα συστήματα, τα οποία ελέγχουν μεγάλο μέρος των συσκευών κοινής χρήσης, βασίζονται σε μικροελεγκτές (επεξεργαστές με ενσωματωμένη μνήμη και περιφερειακά) αλλά και σε μικροεπεξεργαστές, που χρησιμοποιούν εξωτερική μνήμη αλλά και εξωτερικά περιφερειακά.

Σημαντικό χαρακτηριστικό των ενσωματωμένων συστημάτων είναι ότι υλοποιήθηκαν για να χειρίζονται πολύ συγκεκριμένες διεργασίες. Αυτό το χαρακτηριστικό επιτρέπει στους μηχανικούς σχεδίασης να βελτιστοποιήσουν το προϊόν με τρόπους που μειώνουν το μέγεθος και το κόστος ενώ παράλληλα αυξάνουν την αξιοπιστία και την απόδοση του.

Από φυσικής άποψης, τα ενσωματωμένα συστήματα ευρύνονται από φορητές συσκευές όπως ψηφιακά ρολόγια και κινητά έως μεγάλες εγκαταστάσεις όπως φώτα κυκλοφορίας, εργοστασιακούς ελεγκτές και πολύπλοκα υβριδικά οχήματα. Η πολυπλοκότητα τους ποικίλει από χαμηλή όπως απλοί μικροελεγκτές έως πολύ υψηλή όπως πολλαπλές μονάδες με περιφερειακά και δίκτυα.



Σχήμα 2.1 Αρχιτεκτονική ενός ενσωματωμένου συστήματος

2.1.1 Παραδείγματα εφαρμογών των ενσωματωμένων συστημάτων

Τα ενσωματωμένα συστήματα βρίσκουν εφαρμογή σε ένα μεγάλο εύρος καθημερινών αναγκών. Παρακάτω παρατίθενται μερικές από αυτές τι εφαρμογές:

- Τηλεπικοινωνιακά συστήματα όπως κινητά τηλέφωνα, switches, routers και διαδικτυακές γέφυρες.
- Συσκευές καταναλωτών όπως mp3 players, κονσόλες παιχνιδιών, ψηφιακές κάμερες, δέκτες GPS και εκτυπωτές.
- Οικιακές συσκευές όπως πλυντήρια και ψυγεία.
- Συστήματα ασφαλείας και παρακολούθησης.
- Συστήματα οχημάτων όπως Anti-lock Braking System (ABS), Electronic Stability Control (ESC/ESP) και Traction Control (TCS).
- Ιατρικός εξοπλισμός όπως μονάδες ζωτικών ενδείξεων και ηλεκτρονικά στηθοσκόπια.

2.1.2 Παράμετροι για σχεδιασμό ενσωματωμένων συστημάτων

Παρακάτω παρατίθενται μερικοί από τους λόγους που κάνουν επιθυμητό τον σχεδιασμό ενσωματωμένων συστημάτων:

- Κόστος: τα ενσωματωμένα συστήματα σχεδιάζονται να ικανοποιούν συγκεκριμένες απαιτήσεις κάτι, που στο στάδιο της παραγωγής, έχει ως αποτέλεσμα να δημιουργείται υλικό με συγκεκριμένα χαρακτηριστικά. Ως συνέπεια, στις περισσότερες περιπτώσεις, προκύπτει χαμηλό κόστος κατασκευής και, προφανώς, χαμηλό κόστος αγοράς.

- Κατανάλωση ενέργειας: στις περισσότερες περιπτώσεις, τα ενσωματωμένα συστήματα έχουν χαμηλή κατανάλωση ενέργειας πράγμα που τα κάνει ιδιαίτερα δημοφιλή, ιδιαίτερα σε περιπτώσεις φορητών συσκευών για τις οποίες η μέγιστη δυνατή διάρκεια της μπαταρίας είναι ύψιστης σημασίας.

- Αξιοπιστία: τα ενσωματωμένα συστήματα έχουν συνήθως μεγάλη αξιοπιστία κάτι που οφείλεται σε μεγάλο βαθμό στο ότι είναι σχεδιασμένα να εκτελούν συγκεκριμένες διεργασίες. Η αξιοπιστία είναι ένας πολύ σημαντικός παράγοντας, ιδίως σε περιπτώσεις που κρίνεται η απώλεια ανθρώπινης ζωής ή σε διαστημικές εφαρμογές.

- Ισχύς: πολλά ενσωματωμένα συστήματα είναι σχεδιασμένα να συνδυάζουν χαμηλή κατανάλωση με υψηλή απόδοση. Παραδείγματα είναι οι φορητές κονσόλες παιχνιδιών Nintendo και PSP που απαιτούν υψηλή απόδοση για αναπαραγωγή γρήγορων γραφικών.

- Ευελιξία: πολλές φορές είναι επιθυμητή η αναβάθμιση μιας συσκευής. Πολλά ενσωματωμένα συστήματα έχουν την δυνατότητα να αναβαθμιστούν τόσο σε επίπεδο λογισμικού (software) όσο και σε επίπεδο υλικού (hardware). Τα ενσωματωμένα συστήματα που μπορούν να αναβαθμίσουν και το υλικό ανήκουν συνήθων στην κατηγορία των FPGA.

- Λειτουργία πραγματικού χρόνου: Πολλά ενσωματωμένα συστήματα είναι σχεδιασμένα να ικανοποιούν συγκεκριμένες χρονικές απαιτήσεις. Σε εφαρμογές όπως στον τομέα της βιομηχανίας τα συστήματα απαιτείται να ανταποκρίνονται με μεγάλη χρονική ακρίβεια.

2.1.3 Επεξεργαστές στα ενσωματωμένα συστήματα

Οι επεξεργαστές των ενσωματωμένων συστημάτων μπορούν να χωριστούν σε δυο κατηγορίες. Στην πρώτη κατηγορία ανήκουν οι μικροεπεξεργαστές οι οποίοι είναι συνδεδεμένοι με εξωτερικές μνήμες και περιφερειακά μέσω ειδικών ολοκληρωμένων κυκλωμάτων. Στη δεύτερη κατηγορία ανήκουν οι μικροελεγκτές που έχουν τα περισσότερα περιφερειακά τους εσωτερικά μειώνοντας έτσι το μέγεθος, το κόστος και την κατανάλωση ενέργειας. Τα ενσωματωμένα συστήματα μπορούν να χρησιμοποιούν τόσο RISC όσο και non-RISC αρχιτεκτονικές επεξεργαστών ενώ τα μήκη λέξεων τους κυμαίνονται από 4 έως 64 bits.

2.1.4 Περιφερειακά των ενσωματωμένων συστημάτων

Τα ενσωματωμένα συστήματα χρησιμοποιούν μεγάλη ποικιλία περιφερειακών για επικοινωνία με τον έξω κόσμο. Παραδείγματα είναι:

- Serial Communication Interfaces όπως RS-232, RS-422 και RS-485.
- Synchronous Serial Communication Interfaces όπως I^2C και SPI.
- Universal Serial Bus (USB).
- Multi Media Cards όπως SD Cards και Compact Flash.
- Περιφερειακά δικτύου όπως Ethernet.
- Discrete IOs όπως General Purpose Input Output (GPIO).
- Analog to Digital (ADC) και Digital to Analog (DAC) Converters
- Περιφερειακά για Debugging όπως JTAG, ISP και ICSP.

2.2 FPGA-Based συστήματα

Το FPGA (Field Programmable Gate Array ή συστοιχία επιτόπια προγραμματιζόμενων πυλών) είναι ένας τύπος προγραμματιζόμενου ολοκληρωμένου κυκλώματος γενικής χρήσης. Σχεδιάστηκε ώστε να μπορεί να προσδιορίζεται από έναν σχεδιαστή και μετά την κατασκευή του κάτι που μαρτυράει και το όνομα του "Field Programmable". Διαθέτει πολύ μεγάλο αριθμό τυποποιημένων πυλών και άλλων ψηφιακών λειτουργιών όπως απαριθμητές, καταχωρητές, μνήμες και γεννήτριες PLL. Ορισμένα από αυτά ενσωματώνουν και αναλογικές λειτουργίες. Κατά τον προγραμματισμό του FPGA 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 όπως VHDL, AHDL και Verilog.

Το FPGA έχει παρόμοιο πεδίο εφαρμογών με άλλα ολοκληρωμένα ψηφιακά κυκλώματα όπως τα PLD και τα ASIC. Τα ιδιαίτερα, όμως, χαρακτηριστικά του είναι τα εξής:

- Το FPGA χάνει τον προγραμματισμό του κάθε φορά που διακόπτεται η τάση τροφοδοσίας του. Επομένως απαιτεί εξωτερικό επεξεργαστή 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 επεξεργαστή 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.

Βασικά δομικά στοιχεία σε ένα FPGA είναι τα λογικά blocks και μια ιεραρχία από επαναπροσδιορίσιμες διασυνδέσεις που επιτρέπουν στα λογικά blocks να «καλωδιωθούν» μεταξύ τους. Τα λογικά blocks μπορούν να προσδιοριστούν ώστε να πραγματοποιούν περίπλοκες συνδυαστικές λειτουργίες ή να υλοποιούν λογικές πύλες όπως AND και XOR. Στα περισσότερα FPGA, τα λογικά blocks συμπεριλαμβάνουν στοιχεία μνήμης που μπορεί να είναι απλά flip-flops ή ολοκληρωμένα blocks μνήμης.

2.2.1 Εφαρμογές των FPGA-Based συστημάτων

Εφαρμογές των FPGA περιλαμβάνουν, μεταξύ άλλων, ψηφιακή επεξεργασία σήματος, πρωτοτυποποίηση ASIC, αναγνώριση φωνής, computer vision, κρυπτογράφηση, βιοπληροφορική και προσομοίωση του hardware υπολογιστών.

Παραδοσιακά, τα FPGA χρησιμοποιούνται για συγκεκριμένες κάθετες εφαρμογές όπου ο όγκος παραγωγής είναι μικρός. Για τέτοιου είδους εφαρμογές, μικρού όγκου παραγωγής, είναι προνόμιο και οικονομικά πιο ανεκτό οι εταιρείες να χρησιμοποιήσουν FPGA από το να ξοδέψουν μεγάλα ποσά σε πηγές ανάπτυξης για την δημιουργία ενός ASIC.

Μερικές κοινές εφαρμογές των FPGA είναι:

- Αεροδιαστημική και άμυνα:
  - Επικοινωνίες.
  - Πύραυλοι και πυρομαχικά
  - Συστήματα ασφαλείας
  - Διαστημικές εφαρμογές
- Πρωτοτυποποίηση ASIC:
  - Λύσεις συνδεσιμότητας
  - Φορητές ηλεκτρονικές συσκευές
  - Ψηφιακή επεξεργασία σήματος
- Μετάδοση:
  - Real-Time Video Engine
  - Αποκωδικοποιητές
  - Switches και Routers
- Συσκευές καταναλωτών:
  - Ψηφιακές οθόνες
  - Ψηφιακές κάμερες
  - Εκτυπωτές
  - Φορητές συσκευές
- Υπολογισμοί υψηλής απόδοσης:
  - Servers
  - Υπερυπολογιστές
  - Radar
- Βιομηχανικές εφαρμογές:
  - Βιομηχανική απεικόνιση
  - Βιομηχανικά δίκτυα
  - Έλεγχος motor
- Ιατρικές εφαρμογές:
  - Ακτίνες Χ
  - Υπέρηχοι
  - Χειρουργικά συστήματα
- Επεξεργασία βίντεο και εικόνας:
  - Βίντεο υψηλής ανάλυσης
  - Video Over IP Gateway
  - Βιομηχανική απεικόνιση
- Ενσύρματες και ασύρματες επικοινωνίες:
  - Επεξεργασία δικτύου
  - Radio
  - Baseband
  - Connectivity interfaces

2.2.2 Αρχιτεκτονική των FPGA-Based συστημάτων

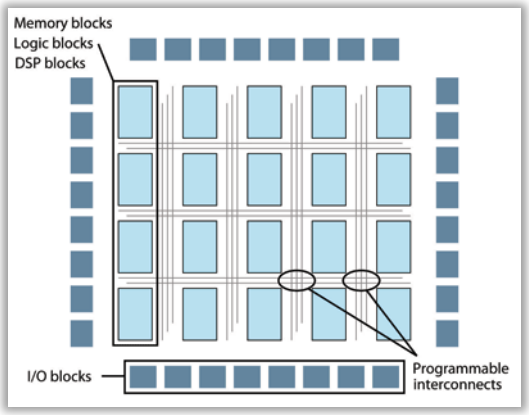
Οι πιο κοινές αρχιτεκτονικές FPGA αποτελούνται από μια συστοιχία λογικών blocks (Combinational Logic Block ή CLB), I/O blocks και κανάλια δρομολόγησης (Routing Channels). Όλα τα κανάλια δρομολόγησης έχουν το ίδιο πλάτος (αριθμός καλωδίων).

Το κύκλωμα μιας εφαρμογής πρέπει να χαρτογραφείται μέσα σε ένα FPGA που διαθέτει επαρκείς πόρους. Παρόλο που ο αριθμός των CLB και των I/O blocks που απαιτούνται μπορεί να καθοριστεί εύκολα κατά τον σχεδιασμό, ο αριθμός των καναλιών που χρειάζονται μπορεί να διαφέρει πολύ ακόμη και ανάμεσα σε σχέδια με το ίδιο ποσό λογικής. Για παράδειγμα, ένας διακόπτης crossbar χρειάζεται πολλά παραπάνω κανάλια δρομολόγησης από έναν συστολικό πίνακα (systolic array) με τον ίδιο αριθμό πυλών.

Ένα λογικό block (CLB) αποτελείται από δυο τεσσάρων εισόδων LUTs (Look-Up Table), έναν πλήρη αθροιστή και ένα ή δυο flip-flop τύπου D. Ένα LUT είναι μια 16x1 ram. Μπορεί να χρησιμοποιηθεί ως γεννήτρια λειτουργιών, ως ram ή ως ένας 16 bit καταχωρητής ολίσθησης.

Τα I/O blocks επιτρέπουν σε έναν ακροδέκτη (pin) να χρησιμοποιηθεί ως είσοδος έξοδος ή συνδυασμός των δυο. Κάθε ακροδέκτης στο FPGA διαθέτει ένα I/O block. Οι είσοδοι μπορούν να χρησιμοποιηθούν άμεσα, μετά από καταχωρητές ή να οδηγήσουν λογική αποκωδικοποιητή (διευθύνσεις).

Η διασύνδεση (routing) των CLBs και των I/O blocks γίνεται με χρήση πλέγματος συρμάτων δυο διαστάσεων. Το πλέγμα αυτό έχει στοιχεία διακοπτών στις διασταυρώσεις των συρμάτων ώστε να γίνεται η διασύνδεση. Οι συνδέσεις των διακοπτών είναι προγραμματιζόμενες. Προκειμένου να βελτιστοποιηθεί η ταχύτητα, πολλά FPGA περιλαμβάνουν σύρματα διαφόρων μήκων. Υπάρχουν σύρματα απλού μήκους (για σύνδεση γειτονικών διακοπτών), διπλού μήκους αλλά και μακριά σύρματα που διατρέχουν κατά μήκος όλο το chip.



Σχήμα 2.2 Αρχιτεκτονική ενός FPGA

3. Μεθοδολογία και εργαλεία ανάπτυξης και σχεδίασης

Σε αυτό το κεφάλαιο, αρχικά, θα γίνει αναφορά στις μεθόδους που χρησιμοποιήθηκαν για να υλοποιηθεί αυτή η εργασία. Στη συνέχεια θα δοθεί περιγραφή των εργαλείων με τα οποία σχεδιάστηκαν και αναπτύχθηκαν τα τμήματα που απαρτίζουν το σύστημα αυτής της εργασίας.

3.1 Μεθοδολογία

Δεδομένων των απαιτήσεων του συστήματος, που υλοποιήθηκε σε αυτή την εργασία, χρησιμοποιήθηκε ένας συνδυασμός υλικού και λογισμικού ώστε να παραδοθεί το βέλτιστο δυνατό αποτέλεσμα παράλληλα με το χαμηλότερο δυνατό κόστος.

Στα πλαίσια ανάπτυξης του απαραίτητου υλικού χρησιμοποιήθηκε η αναπτυξιακή πλακέτα ML403 της Xilinx η οποία , σε αντίθεση με άλλα ενσωματωμένα συστήματα, παρέχει μεγάλη ευελιξία και δυνατότητα τροποποίησης του υλικού. Χάρη σε αυτή την αναπτυξιακή πλακέτα, ύστερα από συνεχείς πειραματισμούς, τροποποιήσεις και βελτιστοποιήσεις, μπόρεσε να υλοποιηθεί ένα επιτυχημένο ενσωματωμένο σύστημα με συγκεκριμένες προδιαγραφές. Παράλληλα, με χρήση της σουίτας Xilinx Platform Studio και των βιβλιοθηκών που παρέχει η Xilinx αναπτύχθηκε ο απαραίτητος κώδικας που εκτελείται από τους επεξεργαστές του ενσωματωμένου συστήματος και δίνει την δυνατότητα επικοινωνίας με τα περιφερειακά και της μνήμες του τελευταίου.

Όσον αφορά την διαδικτυακή επικοινωνία του ενσωματωμένου συστήματος επιλέχτηκε το HSPA modem MTSMC-H4 της εταιρείας Multi-Tech Systems. Η επιλογή αυτού ήταν σκόπιμη καθώς υποστηρίζει μεγάλες ταχύτητες δεδομένων που το έκαναν το καταλληλότερο υποψήφιο για το σύστημα. Στους ως τώρα πειραματισμούς, το modem διακινεί μικρούς όγκους δεδομένων. Παρόλα αυτά, το ενσωματωμένο σύστημα μπορεί να μεταδώσει δεδομένα που πιθανότατα χρειάζονται υψηλές ταχύτητες (παράδειγμα είναι η μετάδοση εικόνας μέσω κάμερας) και, συνεπώς, το συγκεκριμένο modem είναι σε θέση να ικανοποιήσει τέτοιες απαιτήσεις υψηλής ταχύτητας μετάδοσης.

Για τις ανάγκες επικοινωνίας του απομακρυσμένου χρήστη με το ενσωματωμένο σύστημα αναπτύχθηκε εφαρμογή, σε γλώσσα java, η οποία εκτελείται στο τερματικό του χρήστη. Για τον προγραμματισμό της εφαρμογής χρησιμοποιήθηκε η πλατφόρμα Netbeans 7.1. που παρέχει μεγάλη ποικιλία εργαλείων και δυνατοτήτων και διευκολύνει τον προγραμματιστή. Η εφαρμογή αυτή αποτελείται από ένα φιλικό, προς τον χρήστη, γραφικό περιβάλλον σε συνδυασμό με τις απαραίτητες δυνατότητες που διασφαλίζουν την επικοινωνία του χρήστη με το απομακρυσμένο ενσωματωμένο σύστημα και κατ' επέκταση με τις συνδεδεμένες συσκευές σε αυτό.

Η εφαρμογή που εκτελείται στον server αναπτύχθηκε επίσης σε γλώσσα java με χρήση της ίδιας πλατφόρμας. Συνδυάζει, όπως και η προαναφερθείσα, φιλικό γραφικό περιβάλλον και δυνατότητες που επιτρέπουν την διαχείριση του server. Η εφαρμογή αυτή φιλοξενείται σε έναν εικονικό server (VPS) που χρησιμοποιεί λειτουργικό σύστημα Windows Server 2008 R2.

Για τους πειραματισμούς προς έλεγχο της αξιοπιστίας του συστήματος χρησιμοποιήθηκαν αισθητήρες, μικροελεγκτές, leds και διακόπτες ώστε να δοκιμαστούν όλα τα πρωτόκολλα επικοινωνίας που υποστηρίζει το ενσωματωμένο σύστημα. Για δοκιμές, σχετικά με το σειριακό πρωτόκολλο RS-232 χρησιμοποιήθηκε ο μικροελεγκτής ATMega32 της Atmel. Ο μικροελεγκτής αυτός λαμβάνει μέσω της σειριακής θύρας μια συμβολοσειρά που περιλαμβάνει μια απλή μαθηματική πράξη. Υπολογίζει την πράξη και επιστρέφει, πάλι μέσω της σειριακής θύρας το αποτέλεσμα. Για δοκιμή του πρωτοκόλλου I^2C χρησιμοποιήθηκαν δυο αισθητήρες. Πρόκειται για ένα ψηφιακό θερμόμετρο και μια ψηφιακή πυξίδα που επικοινωνούν μέσω διαύλου I^2C. Τέλος για δοκιμή του GPIO interface χρησιμοποιήθηκαν leds, διακόπτες καθώς και ένας αισθητήρας ανίχνευσης κίνησης.

Για την παρακολούθηση των ενεργειών του ενσωματωμένου συστήματος χρησιμοποιήθηκε η εφαρμογή Termite που δίνει την δυνατότητα προβολής δεδομένων που προέρχονται από την σειριακή του θύρα σε ένα τερματικό. Το ενσωματωμένο σύστημα με κάθε ενέργεια που πραγματοποιεί τυπώνει μέσω σειριακής θύρας μηνύματα στην εφαρμογή Termite που εκτελείται σε ένα τερματικό.

3.2 Εργαλεία ανάπτυξης και σχεδίασης

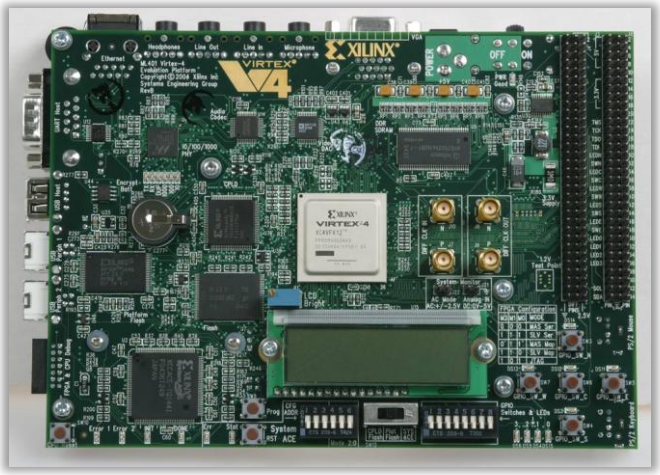
Σε αυτή την ενότητα θα γίνει αναφορά στα εργαλεία ανάπτυξης και σχεδίασης τόσο του υλικού (hardware) όσο και του λογισμικού (software). Ένα από αυτά τα εργαλεία είναι η αναπτυξιακή πλακέτα ML403 της Xilinx που φιλοξενεί το υλικό του ενσωματωμένου συστήματος. Επίσης χρησιμοποιήθηκε η σουίτα Xilinx Platform Studio με την οποία σχεδιάστηκε το υλικό αλλά και αναπτύχθηκε ο κώδικας του ενσωματωμένου συστήματος. Τέλος χρησιμοποιήθηκε η πλατφόρμα Netbeans 7.1 με την οποία αναπτύχθηκε η εφαρμογή του απομακρυσμένου χρήστη αλλά και του server.

3.2.1 Η αναπτυξιακή πλακέτα ML403

Για την ανάπτυξη του υλικού (hardware) χρησιμοποιήθηκε η αναπτυξιακή πλακέτα ML403 της εταιρείας Xilinx. Η πλακέτα αυτή φιλοξενεί ένα FPGA της οικογένειας Virtex 4 που δίνει την δυνατότητα ανάπτυξης υλικού με συγκεκριμένες προδιαγραφές. Χάρη στη φύση των FPGA ο προγραμματιστής είναι σε θέση να τροποποιεί συνεχώς το υλικό, βελτιώνοντας το, έως ότου καταλήξει στο επιθυμητό αποτέλεσμα. Η πλακέτα, επί των πλείστων, περιλαμβάνει μνήμες, κουμπιά, οθόνη led και πολλούς τρόπους διασύνδεσης με εξωτερικές πηγές. Ο σχεδιαστής αναπτύσσει το επιθυμητό υλικό, με τη βοήθεια ειδικών εργαλείων στον υπολογιστή, και στη συνέχεια προγραμματίζει το FPGA το οποίο πλέον συμπεριφέρεται ως ένα ολοκληρωμένο σύστημα. Το σύστημα αυτό, πιθανότατα, περιλαμβάνει επεξεργαστές, μνήμες αλλά και περιφερειακά. Τα περιφερειακά αυτά συνδέονται με τους ποικίλους τρόπους διασύνδεσης που παρέχει η αναπτυξιακή πλακέτα και δίνουν, τελικά, την δυνατότητα επικοινωνίας του συστήματος που αναπτύχθηκε με εξωτερικές πηγές.

Το ενσωματωμένο σύστημα που σχεδιάστηκε στα πλαίσια αυτής της εργασίας είναι αποτέλεσμα συνεχούς πειραματισμού, τροποποιήσεων και βελτιώσεων. Ως συνέπεια, η χρήση της συγκεκριμένης πλακέτας, και κατ' επέκταση του FPGA, αποδείχτηκε καθοριστικής σημασίας. Στα διάφορα στάδια σχεδιασμού προέκυψε πολλές φορές η ανάγκη για αλλαγή χαρακτηριστικών του ενσωματωμένου συστήματος. Παράδειγμα τέτοιων αλλαγών είναι η προσθήκη επιπλέον περιφερειακών που δεν είχαν προβλεφθεί κατά τον αρχικό σχεδιασμό. Άλλο παράδειγμα είναι η αύξηση του μεγέθους της μνήμης BRAM καθώς προέκυψε η ανάγκη για μεγαλύτερο όγκο εκτελέσιμου κώδικα. Ένα τελευταίο παράδειγμα είναι η αύξηση του μεγέθους της ουράς FIFO των ελεγκτών Uart προκειμένου να διορθωθεί η απώλεια εισερχόμενων δεδομένων από αυτούς.

Σε περίπτωση που είχε προτιμηθεί ένα διαφορετικό τυποποιημένο σύστημα, όπως ένα ASIC, θα είχε χαθεί το πλεονέκτημα της ευελιξίας. Ενώ ο σχεδιαστής αναζητά να προσαρμόσει το σύστημα στις δικές του προσδοκίες θα ήταν, πιθανότατα, υποχρεωμένος να μειώσει τις προσδοκίες του ώστε να ταιριάξουν στις περιορισμένες δυνατότητες του συστήματος.



Εικόνα 3.1 Η αναπτυξιακή πλακέτα ML403

3.2.2 Η σουίτα Xilinx Platform Studio

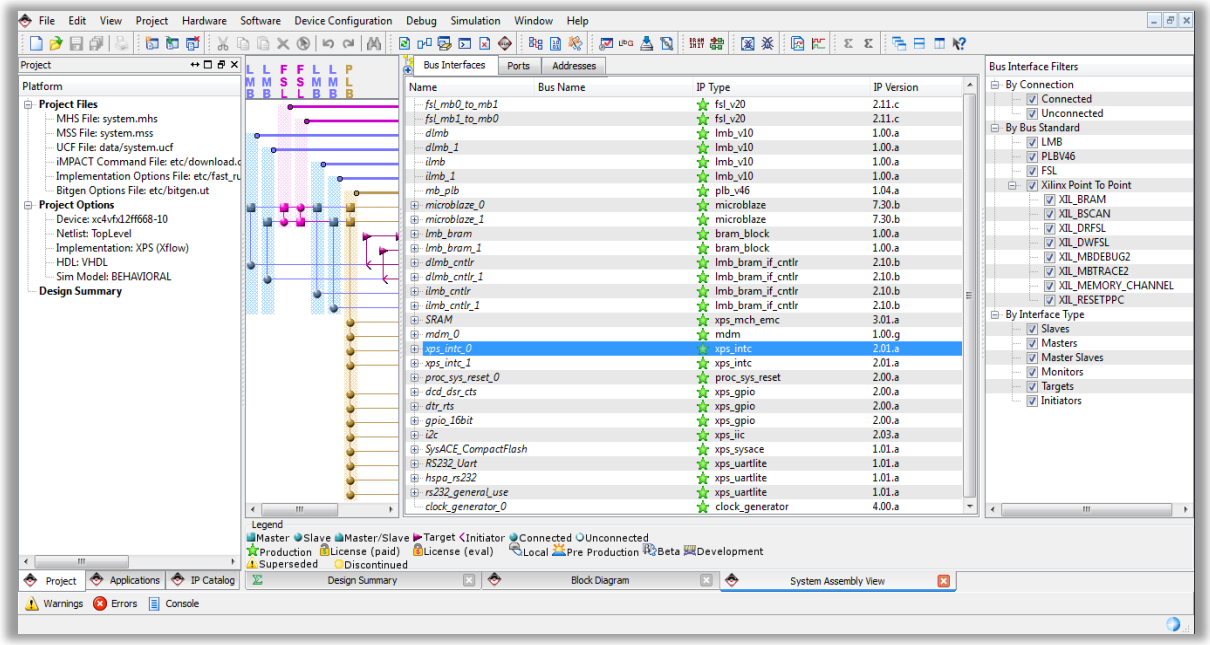
Η σουίτα Xilinx Platform Studio αποτελεί το εργαλείο με το οποίο σχεδιάστηκε το ενσωματωμένο σύστημα αυτής της εργασίας. Αποτελεί επίσης το εργαλείο με το οποίο αναπτύχθηκε ο κώδικας που εκτελεί το ενσωματωμένο σύστημα που υλοποιήθηκε.

Γενικότερα το συγκεκριμένο λογισμικό, που παρέχεται από την Xilinx, αποτελεί μια έκδοση για ανάπτυξη ενσωματωμένων συστημάτων. Όσον αφορά το υλικό, η σουίτα Xilinx Platform Studio παρέχει βιβλιοθήκες, είτε της Xilinx είτε ανεξάρτητων εταιρειών, που δίνουν την δυνατότητα σχεδιασμού συστημάτων που περιλαμβάνουν επεξεργαστές, μνήμες , περιφερειακά και διαύλους. Ο σχεδιαστής είναι σε θέση να δημιουργήσει ενσωματωμένα συστήματα με ευκολία και ευελιξία και σύμφωνα με τις ανάγκες του. Ταυτόχρονα μπορεί να τροποποιήσει τα χαρακτηριστικά των στοιχείων που απαρτίζουν το ενσωματωμένο σύστημα όπως τις μνήμες και τα περιφερειακά.

Όσον αφορά το λογισμικό, η σουίτα επίσης παρέχει βιβλιοθήκες μέσα από τις οποίες ο προγραμματιστής μπορεί να αναπτύξει κώδικα και να διαχειριστεί τα διάφορα στοιχεία που απαρτίζουν το ενσωματωμένο σύστημα που σχεδιάστηκε νωρίτερα.

Στο τελικό στάδιο, όπου έχει ολοκληρωθεί ο σχεδιασμός του επιθυμητού συστήματος αλλά και έχει αναπτυχθεί ο κώδικας που αυτό πρόκειται να εκτελέσει, δημιουργείται το τελικό αρχείο με το οποίο πρόκειται να προγραμματιστεί το FPGA. Στη συνέχεια, το αρχείο αυτό αποθηκεύεται σε εξωτερική κάρτα μνήμης τύπου Compact Flash και η κάρτα μνήμης τοποθετείται στην αναπτυξιακή πλακέτα. Έπειτα αντλείται το αρχείο σύμφωνα με το οποίο θα γίνει ο προγραμματισμός του υλικού στο FPGA και φορτώνεται ο κώδικας που θα εκτελεστεί από το ενσωματωμένο σύστημα που μόλις υλοποιήθηκε.

Στα πλαίσια αυτής της εργασίας, η σουίτα αυτή χρησιμοποιήθηκε τόσο για τον σχεδιασμό του υλικού όσο και για ανάπτυξη εκτελέσιμου κώδικα. Δημιουργήθηκε ένα ενσωματωμένο σύστημα δυο επεξεργαστών και εισήχθησαν τα απαραίτητα περιφερειακά και μνήμες καθώς και οι απαραίτητοι δίαυλοι που τα διασυνδέουν μεταξύ τους. Στα διάφορα στάδια σχεδιασμού προέκυψαν ανάγκες για τροποποίηση και συνεπώς έγιναν αλλαγές όπως προσθήκη περιφερειακών και μετατροπή των χαρακτηριστικών σε περιφερειακά και μνήμες. Στη συνέχεια αναπτύχθηκε, σε συνδυασμό με χρήση των βιβλιοθηκών της σουίτας, κώδικας για κάθε επεξεργαστή που δίνει την δυνατότητα στους επεξεργαστές να αλληλεπιδράσουν με τα περιφερειακά, τις μνήμες αλλά και μεταξύ τους και τελικά να επικοινωνήσουν με εξωτερικές πηγές.



Εικόνα 3.2 Η σουίτα Xilinx Platform Studio

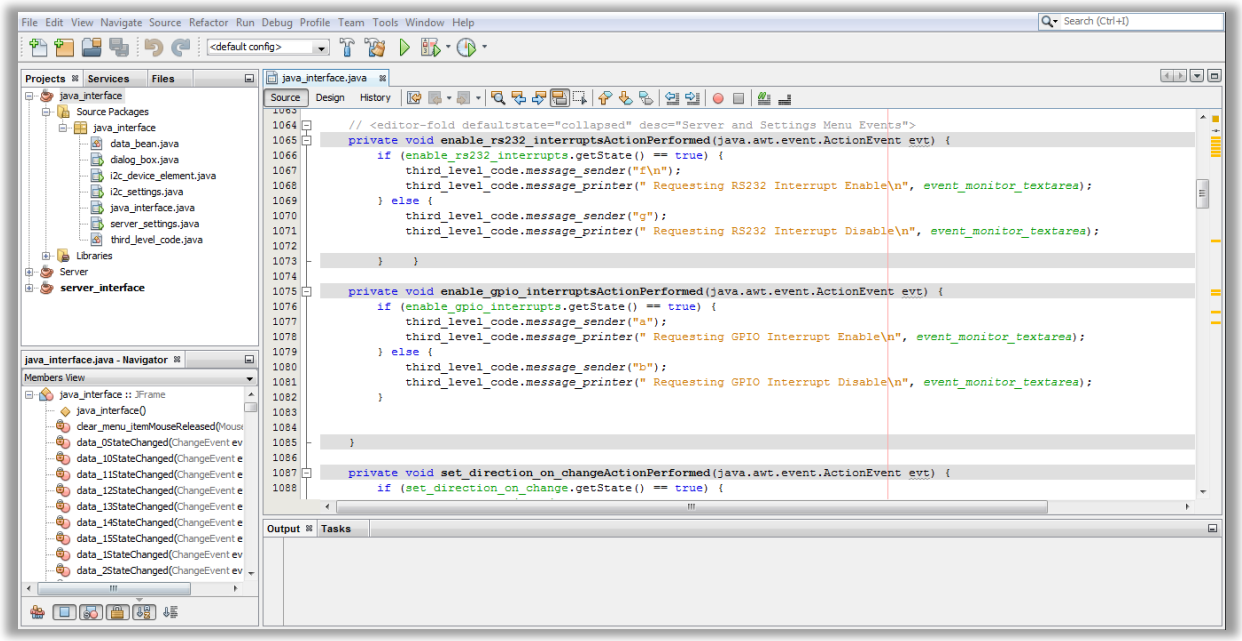
3.2.3 Η πλατφόρμα Netbeans 7.1

Πρόκειται για ένα ολοκληρωμένο περιβάλλον ανάπτυξης, που δημιουργήθηκε αρχικά από την εταιρεία Sun, για ανάπτυξη κατά κύριο λόγο σε java αλλά και σε άλλες γλώσσες όπως PHP, C/C++ και HTML5. Είναι και το ίδιο σχεδιασμένο σε γλώσσα java και μπορεί να εκτελεστεί σε πολλά λειτουργικά συστήματα συμπεριλαμβανομένων των Windows και Linux. Περιλαμβάνει, μεταξύ άλλων, συναρτήσεις διάδρασης χρήστη, source code editor, GUI editor καθώς και υποστήριξη για κατανεμημένες εφαρμογές (CORBA, RMI κλπ) και Web εφαρμογές (JSPs, servlets, struts κλπ).

Για τις ανάγκες αυτής της εργασίας η πλατφόρμα Netbeans χρησιμοποιήθηκε για την ανάπτυξη τόσο της εφαρμογής του απομακρυσμένου χρήστη όσο και της εφαρμογής του server.

Αρχικά, με χρήση του GUI editor σχεδιάστηκε το γραφικό περιβάλλον των εφαρμογών. Ο GUI editor είναι ένα από τα εργαλεία που περιλαμβάνονται στη πλατφόρμα Netbeans. Διευκολύνει στον σχεδιασμό γραφικών χωρίς ο σχεδιαστής να χρειάζεται να επέμβει σε κώδικα καθώς παράγεται αυτόματα. Ο GUI editor παρέχει ένα μεγάλο αριθμό γραφικών στοιχείων όπως κουμπιά, textfields, μενού καθώς και δυνατότητες ρύθμισης αυτών. Ο σχεδιαστής είναι σε θέση να εισάγει (drag and drop) τέτοια γραφικά στοιχεία σε μια περιοχή σχεδιασμού και να δημιουργεί το επιθυμητό περιβάλλον για την εφαρμογή του.

Μόλις ολοκληρωθεί ο σχεδιασμός του επιθυμητού περιβάλλοντος ο προγραμματιστής κάνει χρήση του source code editor με τον οποίο αναπτύσσει τον κώδικα που συνδέει το γραφικό περιβάλλον με τις ενέργειες που εκτελούνται στα παρασκήνια. Στα πλαίσια αυτής της εργασίας ένα μέρος του κώδικα που αναπτύχθηκε, και για τις δυο εφαρμογές, αφορά την διαχείριση των ενεργειών που πρέπει να πραγματοποιηθούν κατά την αλληλεπίδραση με τα στοιχεία του γραφικού περιβάλλοντος (κουμπιά, sliders, combo boxes κλπ). Ένα άλλο μέρος αφορά την ανάπτυξη κώδικα που εκτελείται καθόλη τη διάρκεια λειτουργίας της κάθε εφαρμογής, ανεξάρτητα από τις ενέργειες του χρήστη, και ο οποίος διαχειρίζεται τις διαδικτυακές συνδέσεις αλλά και την μετάδοση των δεδομένων.



Εικόνα 3.3 Η πλατφόρμα Netbeans 7.1

4. Ανάλυση του υλικού (Hardware)-Αρχιτεκτονική

Στο κεφάλαιο αυτό θα εξηγηθεί η αρχιτεκτονική του συστήματος όσον αφορά το υλικό. Αρχικά θα πραγματοποιηθεί μια γενική επισκόπηση και στη συνέχεια θα αναλυθούν σε μεγαλύτερο βάθος τα επιμέρους στοιχεία που αποτελούν τμήματα αυτής. Στα πλαίσια περιγραφής της αρχιτεκτονικής θα αναλυθούν οι επεξεργαστές, οι δίαυλοι, οι μνήμες και τα περιφερειακά του συστήματος. Στη συνέχεια θα δοθεί περιγραφή σχετικά με την συνδεσμολογία και την επικοινωνία των περιφερειακών με εξωτερικές συσκευές. Τέλος θα γίνει αναφορά στο HSPA Modem, που εξασφαλίζει την επικοινωνία με το internet, και στον τρόπο διασύνδεσης του με το ενσωματωμένο σύστημα.

4.1 Γενική επισκόπηση της αρχιτεκτονικής

Το σύστημα που αναφέρεται σε αυτή την εργασία έχει υλοποιηθεί σε FPGA στην αναπτυξιακή πλακέτα ML403 της εταιρείας Xilinx. Πρόκειται για ένα ενσωματωμένο σύστημα του οποίου σημαντικό χαρακτηριστικό αποτελεί η χρήση δυο επεξεργαστών microblaze, οι **microblaze_0** και **microblaze_1**, που λειτουργούν σε συχνότητα 100MHz και οι οποίοι είναι προσαρτημένοι σε ένα κοινό δίαυλο PLB (Processor Local Bus). Ο κοινός αυτός δίαυλος τους παρέχει την δυνατότητα να επικοινωνήσουν με περιφερειακά που είναι και αυτά συνδεδεμένα στον ίδιο δίαυλο. Οι δυο αυτοί επεξεργαστές κάνουν χρήση και ενός δεύτερου διαύλου, τύπου FSL (Fast Simplex Link), που τους επιτρέπει να έχουν άμεση επικοινωνία μεταξύ τους αποφεύγοντας να χρησιμοποιήσουν τον κοινό δίαυλο PLB, τον οποίο διαπερνούν δεδομένα τόσο των επεξεργαστών όσο και των περιφερειακών. Ως συνέπεια, μειώνεται η καθυστέρηση και βελτιώνεται η ολική απόδοση.

Κάθε επεξεργαστής έχει πρόσβαση σε τοπική μνήμη BRAM (Block RAM) μέσω διαύλου LMB (Local Memory Bus). Το μέγεθος της μνήμης είναι 32Kbyte για κάθε επεξεργαστή και αποτελεί τον χώρο στον οποίο φορτώνεται ο κώδικας που θα εκτελέσει ο επεξεργαστής κατά την λειτουργία του. Το ενσωματωμένο σύστημα κάνει επίσης χρήση μνήμης SRAM μεγέθους 1mb η οποία είναι προσαρτημένη στο δίαυλο PLB και χρησιμοποιείται κοινά από τους δυο επεξεργαστές για την ανταλλαγή δεδομένων μεταξύ τους.

Μέρος των καθηκόντων των επεξεργαστών αποτελεί η επικοινωνία με τα περιφερειακά που είναι συνδεδεμένα στο δίαυλο PLB. Κάποια από αυτά είναι κοινά και για τους δυο επεξεργαστές ενώ άλλα επικοινωνούν μόνο με συγκεκριμένο από τους δυο.

Τα κοινά, και για τους δυο επεξεργαστές, περιφερειακά είναι:

- Η μνήμη SRAM η οποία όπως αναφέρθηκε πιο πάνω χρησιμοποιείται για ανταλλαγή δεδομένων.
- Ένας ελεγκτής UART που επιτρέπει στους επεξεργαστές να τυπώσουν μηνύματα μέσω σειριακού πρωτοκόλλου RS-232 σε ένα τερματικό (π.χ. Hyper Terminal).

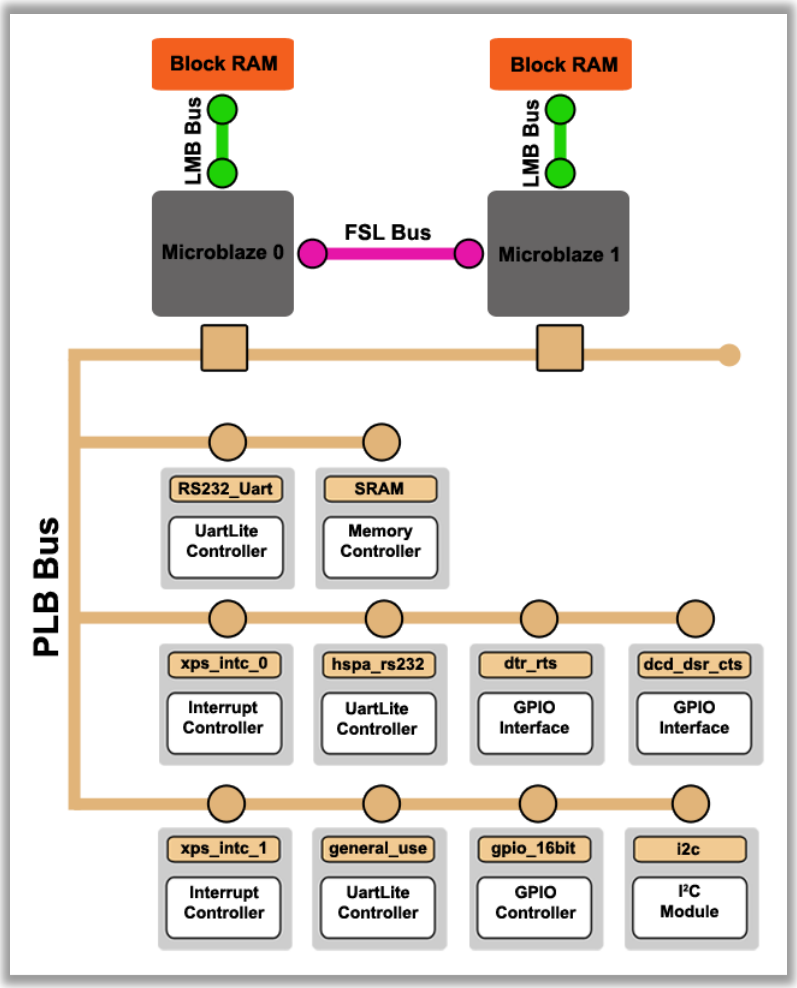
Τα περιφερειακά που χρησιμοποιούνται από τον επεξεργαστή **microblaze_0** είναι:

- Ένας δεύτερος ελεγκτής UART που χρησιμοποιείται για την επικοινωνία του **microblaze_0** με το HSPA Modem.
- Ένα GPIO (General Purpose Input Output) interface πλάτους 2 bit που υλοποιεί δυο σήματα για τον έλεγχο λειτουργιών του HSPA Modem από τον **microblaze_0.**
- Ένα GPIO interface πλάτους 3 bit το οποίο χρησιμοποιείται από τον **microblaze_0** προκειμένου να λάβει πληροφορίες για την κατάσταση του modem, «διαβάζοντας» τρία ακόμα από τα σήματα του τελευταίου.

Τα περιφερειακά που χρησιμοποιούνται από τον επεξεργαστή **microblaze_1** είναι:

- Ένα GPIO Interface πλάτους 16 bit που ο **microblaze_1** χρησιμοποιεί για να επικοινωνήσει με ποικίλα εξωτερικά ψηφιακά συστήματα και ηλεκτρονικά στοιχεία όπως αισθητήρες, μικροελεγκτές, leds και διακόπτες.
- Ένα I^2C module που δίνει την δυνατότητα στον **microblaze_1** να αλληλεπιδράσει μέσω του σειριακού διαύλου I^2C με αντίστοιχα ψηφιακά συστήματα όπως τα προαναφερθέντα, αυτή τη φορά όμως μέσω διαύλου I^2C.
- Ένας ακόμα ελεγκτής UART που αποτελεί για τον **microblaze_1** το μέσο για να μιλήσει με εξαρτήματα όπως κάμερες, GPS και υπολογιστές που και αυτά υλοποιούν σειριακό πρωτόκολλο RS-232 για την ανταλλαγή δεδομένων.

Στο δίαυλο PLB είναι επίσης προσαρτημένοι δυο interrupt controllers που στην ουσία είναι και αυτοί περιφερειακά αλλά αναφέρονται σκόπιμα τελευταίοι. Κάθε ένας από αυτούς αντιστοιχεί σε έναν επεξεργαστή και είναι υπεύθυνος στην περίπτωση που προκληθεί κάποιο γεγονός από κάποια περιφερειακή συσκευή(π.χ. δεδομένα που αποστέλλονται από τον ελεγκτή UART) να ειδοποιήσει τον επεξεργαστή ο οποίος πρέπει διαχειριστεί το γεγονός (π.χ. λήψη και επεξεργασία των δεδομένων).



Σχήμα 4.1 Γενική Επισκόπηση Αρχιτεκτονικής

4.2 Επεξεργαστής Microblaze

Το ενσωματωμένο σύστημα που αναπτύχθηκε κάνει χρήση επεξεργαστών Microblaze. Οι Microblaze ανήκουν στην κατηγορία των FPGA-Based Soft Processors και βασίζονται σε αρχιτεκτονική 32 bit RISC του Harvard. Είναι βελτιστοποιημένοι για χρήση σε εφαρμογές ενσωματωμένων συστημάτων και παρουσιάζουν υψηλό βαθμό ευελιξίας. Η ευελιξία αυτή, που είναι αποτέλεσμα της FPGA φύσης τους, επιτρέπει τροποποιήσεις στα χαρακτηριστικά των επεξεργαστών όπως το μέγεθος της μνήμης cache, το βάθος διασωλήνωσης(3 ή 5 επιπέδων), τα ενσωματωμένα περιφερειακά, τη μονάδα διαχείρισης μνήμης (Memory Management Unit ή MMU) και τους διαύλους. Τροποποιήσεις για βέλτιστη απόδοση επιτρέπουν σε έναν Microblaze να λειτουργήσει σε ταχύτητες έως και 210MHz.

Στο ενσωματωμένο σύστημα έχουν εισαχθεί δυο Microblaze, οι **microblaze_0** και **microblaze_1**, που λειτουργούν σε ταχύτητα 100MHz καθώς αυτή είναι η μέγιστη που μπορεί να υποστηρίξει η αναπτυξιακή πλακέτα που χρησιμοποιήθηκε. Οι απαιτήσεις που πρέπει να ικανοποιήσει το σύστημα κάνουν αναγκαία την ύπαρξη δυο επεξεργαστών καθώς μοιράζεται το φόρτο εργασιών και έτσι εξασφαλίζεται σταθερότητα και αποδοτικότητα.

Κάθε επεξεργαστής χρησιμοποιεί μνήμη BRAM (Block RAM) μεγέθους 32 Kbyte, στην οποία φορτώνεται ο κώδικας που θα εκτελεστεί, και η πρόσβαση σε αυτή γίνεται μέσω διαύλου LMB (Local Memory Bus). Ο **microblaze_0** πέραν της BRAM χρησιμοποιεί συμπληρωματικά και μνήμη SRAM καθώς το σύνολο του κώδικα που εκτελεί καταλαμβάνει αρκετά περισσότερο από 32 Kbyte.

Η επικοινωνία με τις περιφερειακές συσκευές πραγματοποιείται μέσα από ένα δίαυλο PLB (Processor Local Bus) ο οποίος είναι κοινός και για τους δυο επεξεργαστές. Το σύστημα ενσωματώνει επίσης ένα δίαυλο FSL (Fast Simplex Link) που επιτρέπει στους δύο Microblaze να ανταλλάξουν άμεσα δεδομένα μεταξύ τους.

4.3 Δίαυλοι (Buses) PLB, LMB και FSL

Ως δίαυλος χαρακτηρίζεται ένα υποσύστημα που επιτρέπει την μεταφορά δεδομένων μεταξύ περιφερειακών και εξαρτημάτων είτε αναφερόμαστε σε υπολογιστή ή σε ενσωματωμένο σύστημα. Στην ουσία πρόκειται για έναν αριθμό καλωδίων τα οποία βρίσκονται σε παραλληλία και επιτρέπουν την διέλευση δεδομένων μεταξύ συσκευών οι οποίες είναι συνδεδεμένες σε αυτά. Ένα σημαντικό χαρακτηριστικό των διαύλων είναι το μέγεθος τους το οποίο μετριέται σε bits και αναφέρεται στο μέγεθος των δεδομένων που μπορούν να μεταφερθούν παράλληλα ανά μονάδα χρόνου. Ένα δεύτερο χαρακτηριστικό είναι η ταχύτητα του διαύλου που μετριέται σε MHz. Οι δίαυλοι που έχουν χρησιμοποιηθεί σε αυτό το σύστημα έχουν μέγεθος 32 bit και ταχύτητα 100MHz.

4.3.1 Processor Local Bus (PLB)

Ο δίαυλος PLB αποτελεί το κύριο μέσο επικοινωνίας μεταξύ επεξεργαστών και περιφερειακών. Η αρχιτεκτονική του υποστηρίζει την ταυτόχρονη ύπαρξη πολλαπλών Masters και Slaves με τους επεξεργαστές να έχουν τον ρόλο των Masters ενώ οι περιφερειακές συσκευές να αποτελούν τους Slaves. Ο δίαυλος PLB δίνει την δυνατότητα για μεταφορές δεδομένων διαβάσματος και εγγραφής μεταξύ των Master και Slave συσκευών ενώ παρέχει ένα κεντρικό μηχανισμό διαιτησίας που παραχωρεί το δικαίωμα χρήσης του διαύλου μεταξύ των Masters. Χαρακτηριστικό του συγκεκριμένου μηχανισμού διαιτησίας είναι η ευελιξία που επιτρέπει την υλοποίηση ποικίλων μοντέλων προτεραιότητας.

Στο δίαυλο PLB αυτού του συστήματος είναι προσαρτημένοι και οι δύο επεξεργαστές λόγω της ανάγκης πρόσβασης σε κοινές περιφερειακές συσκευές. Ο δίαυλος προσφέρει το πλεονέκτημα στους επεξεργαστές να μπορούν να επικοινωνήσουν με όλες τις συσκευές, καθώς όλες είναι συνδεδεμένες στον ίδιο δίαυλο, αλλά επιλέγεται σκόπιμα κάποιες να είναι κοινές ενώ άλλες να επικοινωνούν μόνο με συγκεκριμένο επεξεργαστή. Η επικοινωνία μεταξύ των περιφερειακών του διαύλου γίνεται με την χρήση διευθύνσεων. Κάθε περιφερειακή συσκευή χαρακτηρίζεται από μία διεύθυνση και έτσι κάθε μεταφορά δεδομένων διαβάσματος ή εγγραφής κάνει χρήση της διεύθυνσης της συσκευής με την οποία πρόκειται να πραγματοποιηθεί η ανταλλαγή δεδομένων.

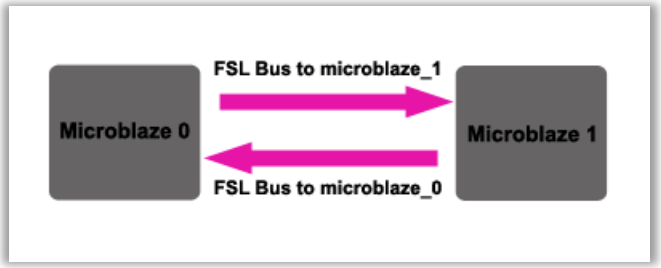
4.3.2 Local Memory Bus (LMB)

Ο δίαυλος LMB παρέχει την δυνατότητα στον επεξεργαστή να επικοινωνήσει με την μνήμη BRAM. Γενικότερα χαρακτηρίζεται ως τοπικός, γρήγορος δίαυλος που μπορεί να
δια-συνδέσει έναν Microblaze με περιφερειακά υψηλών ταχυτήτων, όπως στη συγκεκριμένη περίπτωση με την BRAM. Πρέπει να αναφερθεί πως σε έναν LMB υποστηρίζεται η ύπαρξη μόνο ενός Master, στη συγκεκριμένη υλοποίηση αυτός ο Master είναι ο Microblaze, ενώ τα δεδομένα διαβάσματος ή εγγραφής χρησιμοποιούν ξεχωριστούς διαύλους.

4.3.3 Fast Simplex Link (FSL)

Ως FSL ορίζεται ένας μονό-κατευθυντικός με επικοινωνία σημείου προς σημείο δίαυλος που προσφέρει γρήγορες ταχύτητες μεταξύ δυο στοιχείων μέσα σε ένα FPGA. Στην συγκεκριμένη υλοποίηση συνδέει τους δυο επεξεργαστές μεταξύ τους προσφέροντας την δυνατότητα άμεσης επικοινωνίας. Η εισαγωγή αυτού του είδους διαύλου είναι επιθυμητή γιατί παίζει καθοριστικό ρόλο στην βελτίωση της απόδοσης όλου του συστήματος. Ο δίαυλος PLB διαρρέεται από δεδομένα όλων των περιφερειακών, τα οποία δεν μπορούν να τον διατρέξουν ταυτόχρονα, με αποτέλεσμα να μειώνεται η απόδοση καθώς ο μηχανισμός διαιτησίας πρέπει να μοιράσει τον δίαυλο.

Το γεγονός ότι οι επεξεργαστές ανταλλάσουν πολύ συχνά δεδομένα μεταξύ τους κάνει επιθυμητή την χρήση του διαύλου FSL αφού έτσι δεν χρειάζεται να χρησιμοποιήσουν τον PLB για την μεταξύ τους επικοινωνία με αποτέλεσμα να μειωθεί η κίνηση στον δίαυλο. Λόγω του ότι ένας FSL δίαυλος, όπως προαναφέρθηκε, είναι μονό-κατευθυντικός γίνεται αναγκαία η χρήση δυο FSL διαύλων. Ο πρώτος από αυτούς, που εμφανίζεται με το όνομα **fsl_mb0_to_mb1**, χρησιμοποιείται για επικοινωνία από τον **microblaze_0** προς τον **microblaze_1**. Ο δεύτερος, που ονομάζεται **fsl_mb1_to_mb0**, χρησιμοποιείται αντίστροφα για επικοινωνία από τον **microblaze_1** προς τον **microblaze_0**.

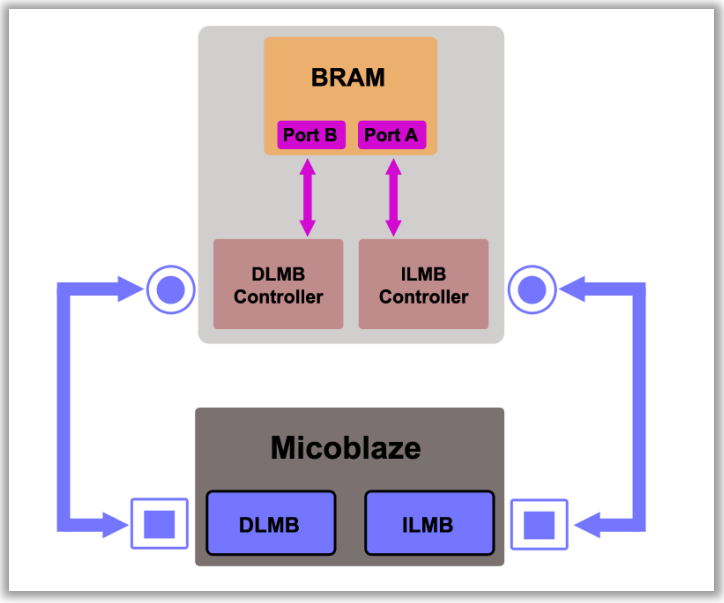


Σχήμα 4.2 Υλοποίηση διαύλων FSL

4.4 Μνήμη Block RAM (BRAM)

Η BRAM χαρακτηρίζεται ως ένα τροποποιήσιμο εξάρτημα μνήμης. Υποστηρίζει μεταφορές byte, half word, word και double word και περιλαμβάνει δυο θύρες, τις Port A και Port B, στις οποίες μπορούν να συνδεθούν ανεξάρτητοι ελεγκτές BRAM (BRAM Interface Controllers). Ως τμήμα της αρχιτεκτονικής του συστήματος που περιγράφεται αποτελεί την μνήμη στην οποία φορτώνεται ο κώδικας που θα εκτελέσει ο Microblaze. Έχει μέγεθος 32 Kbytes ενώ μπορεί να τροποποιηθεί ώστε να έχει μικρότερο ή και μεγαλύτερο μέγεθος πράγμα που εξαρτάται από την χωρητικότητα του FPGA.

Κάθε Microblaze χρησιμοποιεί δυο interfaces για επικοινωνία με την μνήμη BRAM. Το ένα interface διαβάζει ή γράφει τμήματα κώδικα που αφορούν τις εντολές (instructions) που εκτελούνται ενώ το άλλο διαβάζει ή γράφει τα δεδομένα (data) που αποτελούν και αυτά τμήματα του κώδικα. Αυτό προϋποθέτει την χρήση δυο διαύλων LMB ένας για τις εντολές (instructions), με όνομα ilmb, και ένας για τα δεδομένα (data), με όνομα dlmb. Κάθε ένας από αυτούς τους διαύλους συνδέεται με έναν BRAM Interface Controller ο οποίος καταλήγει σε μία από τις θύρες της μνήμης BRAM και φροντίζει να εξασφαλίσει την επικοινωνία με την μνήμη. Το αποτέλεσμα είναι ο δίαυλος ilmb να απευθύνεται στη θύρα A (Port A) ενώ ο dlmb να απευθύνεται στη θύρα B (Port B).



Σχήμα 4.3 Μνήμη Block RAM (BRAM)

4.5 Περιφερειακές συσκευές

Η ευελιξία ενός FPGA δίνει την δυνατότητα εισαγωγής περιφερειακών ως μέρος του ενσωματωμένου συστήματος. Τα περιφερειακά προσαρτώνται στον δίαυλο PLB και χαρακτηρίζονται από μια διεύθυνση που δίνει την δυνατότητα προσπέλασης τους από τους επεξεργαστές. Από την μεγάλη ποικιλία περιφερειακών στο συγκεκριμένο σύστημα ενσωματώνονται 10 τα οποία ανήκουν σε 5 κατηγορίες και αναλύονται στη συνέχεια.

4.5.1 UartLite Controller

Πρόκειται για μια απλοποιημένη (Lite) εκδοχή ενός UART (Universal Asynchronous Receiver Transmitter) Controller που συνδέεται στον δίαυλο PLB και παρέχει την δυνατότητα επικοινωνίας μέσω του σειριακού πρωτοκόλλου RS-232. Διαθέτει δυο κανάλια, ένα μετάδοσης και ένα λήψης (full duplex) και χρησιμοποιεί receive FIFO και transmit FIFO 64 χαρακτήρων κάθε μια. Ο UartLite Controller υλοποιεί παράλληλη σε σειριακή μετατροπή για χαρακτήρες που λαμβάνονται μέσω του διαύλου PLB (οι προς μετάδοση χαρακτήρες) και σειριακή σε παράλληλη μετατροπή για χαρακτήρες που λαμβάνονται από το σειριακό περιφερειακό (τα εισερχόμενα δεδομένα).

Μπορούν να τροποποιηθούν πολλά από τα χαρακτηριστικά του όπως ο αριθμός bit των μεταδιδόμενων ή ληφθέντων χαρακτήρων (8, 7, 6 ή 5 bit χαρακτήρες), ο ρυθμός μετάδοσης συμβόλων (Baud Rate),τα Stop Bits και το Parity (Odd, Even ή None). Οι τροποποιήσεις αυτές δεν μπορούν να πραγματοποιηθούν δυναμικά λόγω του ότι ο συγκεκριμένος controller αποτελεί μια απλοποιημένη εκδοχή ενός UART controller. Ένα άλλο χαρακτηριστικό, που είναι αποτέλεσμα της απλοποιημένης μορφής του, είναι ότι δεν διαθέτει σήματα ελέγχου, που υποστηρίζει το πρωτόκολλο RS-232, άλλα μόνο τα δυο κανάλια, ένα για την μετάδοση (TX) και ένα για την λήψη (RX).

Το σειριακό πρωτόκολλο που υλοποιείται στον UartLite Controller, και που ισχύει στα ενσωματωμένα συστήματα, μπορεί να χαρακτηριστεί ως Serial TTL. Αυτό σημαίνει πως υλοποιείται μεν το σειριακό πρωτόκολλο RS-232 αλλά με την ιδιαιτερότητα πως τα σήματα βρίσκονται μεταξύ των ορίων 0volt έως 5volt με το λογικό 1 να αντιστοιχεί στα 5Volt και το λογικό 0 να αντιστοιχεί στα 0Volt. Αυτή η λεπτομέρεια έχει ως αποτέλεσμα να είναι δυνατή η επικοινωνία με συσκευές που λειτουργούν στα ίδια επίπεδα Volt όπως μικροελεγκτές και GPS. Σε πολλές άλλες συσκευές, όπως οι υπολογιστές, τα επίπεδα των Volt λειτουργούν αναπαριστώντας το λογικό 0 με αρνητική τιμή και το λογικό 1 με θετική, παράδειγμα -13Volt και +13Volt. Σε αυτές τις περιπτώσεις είναι αναγκαία η χρήση ενός εξωτερικού μετατροπέα Volt, όπως ο MAX232, ώστε να είναι δυνατή η επικοινωνία μεταξύ του UartLite Controller και της εξωτερικής συσκευής.

Στο ενσωματωμένο σύστημα που περιγράφεται χρησιμοποιούνται τρία περιφερειακά που ανήκουν στην κατηγορία των UartLite Controllers:

- Το πρώτο από αυτά αναγνωρίζεται ως **RS232_Uart**. Μια από τις ανάγκες που πρέπει να ικανοποιηθούν είναι να ενημερώνεται ο διαχειριστής σχετικά με τις ενέργειες που πραγματοποιούνται μέσα στο σύστημα. Κατά την διάρκεια εκτέλεσης του κάθε επεξεργαστή δημιουργούνται μηνύματα τα οποία ενημερώνουν για την τρέχουσα κατάσταση και τις αλλαγές που πραγματοποιούνται. Με τη χρήση του ελεγκτή **RS232_Uart** και των κατάλληλων **print** εντολών δίνεται η δυνατότητα, και στους δυο microblaze, να τυπώνουν μέσω του σειριακού πρωτοκόλλου μηνύματα στον υπολογιστή σε οποιοδήποτε λογισμικό όπως το Hyper Terminal των Windows. Η αναπτυξιακή πλακέτα παρέχει μια υποδοχή RS-232 η οποία διαθέτει μετατροπέα για τις διαφορετικές διαβαθμίσεις των ρευμάτων. Η ιδιαιτερότητα του **RS232_Uart** είναι ότι συνδέεται σε αυτή την υποδοχή σε αντίθεση με τους άλλους controllers που συνδέονται σε εξωτερικά pins. Να σημειωθεί πως σε μελλοντική τροποποίηση τα μηνύματα θα τυπώνονται σε κάποιου είδους οθόνη και με την χρήση του ανάλογου controller.
- Το δεύτερο περιφερειακό αυτής της κατηγορίας είναι ο ελεγκτής **hspa_rs232** που χρησιμοποιείται κατεξοχήν ως το μέσο επικοινωνίας του **microblaze_0** με το HSPA modem. Όπως και οι άλλοι UartLite Controllers διαθέτει δυο κανάλια για ανταλλαγή δεδομένων, ένα για λήψη (RX) και ένα για μετάδοση (TX), τα οποία είναι συνδεδεμένα με τις αντίστοιχες υποδοχές του modem. Το modem υλοποιεί σειριακό πρωτόκολλο RS-232 και αυτό κάνει εφικτή την επικοινωνία με τον ελεγκτή **hspa_rs232**. Καθώς οι UartLite Controllers δεν μπορούν να τροποποιηθούν δυναμικά, ο **hspa_rs232** έχει ρυθμιστεί να χρησιμοποιεί το ίδιο baud rate με το modem, δηλαδή 921600. Σημαντικό χαρακτηριστικό του **hspa_rs232** είναι ότι χρησιμοποιεί σήματα interrupts προκειμένου να διακόψει την εκτέλεση του επεξεργαστή οποτεδήποτε υπάρχουν εισερχόμενα δεδομένα από το modem. Στην πραγματικότητα τα σήματα αυτά είναι συνδεδεμένα με έναν interrupt controller ο οποίος οποτεδήποτε λάβει σήματα interrupt από τον ελεγκτή **hspa_rs232** φροντίζει να ενημερώσει τον ενδιαφερόμενο επεξεργαστή. Η περιγραφή του interrupt controller θα δοθεί σε επόμενη ενότητα.

• Το τρίτο τέτοιο περιφερειακό είναι ο ελεγκτής **rs232_general_use**. Είναι συνδεδεμένος με τον **microblaze_1** και αποτελεί έναν γενικής χρήσης UartLite Controller που εξασφαλίζει την δυνατότητα επικοινωνίας του συστήματος, και συνεπώς του ενδιαφερόμενου διαχειριστή, με ποικίλες συσκευές που και αυτές υλοποιούν πρωτόκολλο RS-232. Όπως έχει αναφερθεί προηγουμένως, μερικές από αυτές τις συσκευές περιλαμβάνουν μικροελεγκτές, κάμερες, GPS και υπολογιστές. Όπως και ο **hspa_rs232** έτσι και ο ελεγκτής **rs232_general_use** χρησιμοποιεί σήματα interrupts και κάθε φορά που εισέρχονται δεδομένα από την εξωτερική συσκευή φροντίζει μέσα από έναν interrupt controller να ενημερώσει τον αντίστοιχο επεξεργαστή.

4.5.2 GPIO Interface

Ως GPIO (General Purpose Input/Output) ορίζεται ένα interface το οποίο διαθέτει έναν αριθμό ακροδεκτών που μπορούν να ελεγχθούν (προγραμματιστούν) δυναμικά από τον χρήστη. Αυτό σημαίνει πως οι ακροδέκτες μπορούν να μετατραπούν σε είσοδο ή έξοδο κατά τη διάρκεια εκτέλεσης ενός επεξεργαστή. Ένα GPIO Interface μπορεί να έχει μέγεθος έως και 32 bit που σημαίνει ότι μπορούν να υπάρχουν 32 ακροδέκτες, ένας για κάθε bit. Κάθε bit σε ένα GPIO Interface μπορεί να έχει δυο καταστάσεις, το λογικό 0 και το λογικό 1. Συνεπώς ένας επεξεργαστής μπορεί να γράψει ή να διαβάσει δεδομένα από αυτό όπου κάθε bit θα έχει μια από τις δυο αυτές τιμές.

Στο παρόν σύστημα υλοποιούνται τρία περιφερειακά που ανήκουν στην κατηγορία των GPIO interfaces:

• Το πρώτο από αυτά τα τρία GPIO interfaces είναι το **dtr_rts**. Έχει μέγεθος 2 bit και αποτελεί μια από τις περιφερειακές συσκευές του **microblaze_0**. Το modem χρησιμοποιεί σειριακό πρωτόκολλο RS-232 για επικοινωνία και εκτός από τα δυο κανάλια για μετάδοση και λήψη δεδομένων χρησιμοποιεί και σήματα ελέγχου ροής δεδομένων, όπως ορίζεται στο RS-232. Το γεγονός όμως πως ο **hspa_rs232** είναι απλοποιημένης μορφής Uart Controller και δεν υποστηρίζει σήματα ελέγχου, κάνει αναγκαία την χρήση του **dtr_rts** έτσι ώστε να υλοποιηθούν τα δυο από τα σήματα ελέγχου ροής που απαιτούνται από το modem. Τα δυο αυτά σήματα είναι τα DTR (Data Terminal Ready) και RTS (Ready To Send). Το σήμα RTS χρησιμεύει ώστε το modem να γνωρίζει πότε η άλλη μεριά, δηλαδή ο **microblaze_0**, είναι σε θέση να λάβει δεδομένα. Ο επεξεργαστής αλλάζοντας την λογική κατάσταση αυτού του σήματος, συνεπώς του ακροδέκτη, επιτρέπει ή αποτρέπει την λήψη δεδομένων από το modem, κάτι που εξαρτάται από το κατά πόσο ο πρώτος είναι σε θέση να λάβει δεδομένα. Το σήμα DTR ελέγχει την TCP σύνδεση με τον server. Τα δυο bit (συνεπώς οι ακροδέκτες) του **dtr_rts** ορίζονται εξαρχής ως έξοδοι. Όταν ο **microblaze_0** είναι σε θέση να λάβει δεδομένα αλλάζει την κατάσταση του σήματος RTS σε λογικό 0. Με αυτό τον τρόπο το modem ενημερώνεται ότι ο **microblaze_0** είναι διαθέσιμος και έτσι ξεκινάει την αποστολή δεδομένων. Αντίθετα, στέλνοντας λογικό 1 το modem αναμένει για διαθεσιμότητα από τον **microblaze_0**. Το σήμα DTR είναι μόνιμα σε κατάσταση λογικού 0. Η αλλαγή του σε κατάσταση λογικού 1 έχει ως αποτέλεσμα να τερματίσει το modem την σύνδεση του με τον server.

• Το δεύτερο περιφερειακό αυτής της κατηγορίας είναι το **dcd_dsr_cts**. Έχει μέγεθος 3 bit και, όπως και το προαναφερθέν interface, χρησιμοποιείται για τρία ακόμα από τα σήματα ελέγχου του modem. Τα σήματα αυτά είναι τα DCD, DSR και CTS. Οι τρείς ακροδέκτες αυτού του GPIO interface ορίζονται εξαρχής ως είσοδοι. Σκοπός είναι το σύστημα, διαβάζοντας την λογική τιμή του **dcd_dsr_cts**, να ενημερώνεται σχετικά με την τρέχουσα κατάσταση του modem. Το σήμα CTS χρησιμοποιείται από το modem για να δείξει κατά πόσο αυτό είναι σε ετοιμότητα αλληλεπίδρασης. Το σύστημα διαβάζοντας λογική τιμή 0 σε αυτό γνωρίζει ότι το modem βρίσκεται σε ετοιμότητα ενώ, αντίθετα, λογική τιμή 1 δείχνει ότι δεν είναι έτοιμο. Το σήμα DCD χρησιμοποιείται από το modem για να ενημερώσει σχετικά με την κατάσταση της σύνδεσης του δικτύου GPRS. Λογική τιμή 0 σε αυτό επιβεβαιώνει πως υπάρχει ενεργή σύνδεση στο δίκτυο GPRS ενώ λογική τιμή 1 το αντίθετο. Τέλος το σήμα DSR δείχνει το κατά πόσο υπάρχει ενεργή TCP σύνδεση μεταξύ του modem και του server. Αντίστοιχα και εδώ λογική τιμή 0 αυτού του σήματος συνεπάγεται ενεργή TCP σύνδεση. Σημαντικό χαρακτηριστικό αυτού του GPIO interface είναι πως μπορεί να προκαλέσει interrupts. Αυτό σημαίνει πως το **dcd_dsr_cts** είναι συνδεδεμένο με έναν interrupt controller και ως συνέπεια αν κάποιο από τρία προαναφερθέντα σήματα προκαλέσει διακοπή ο interrupt controller θα ενημερώσει τον **microblaze_0** ώστε ο τελευταίος να διαχειριστεί την αλλαγή των σημάτων, συνεπώς την αλλαγή στην κατάσταση του modem.

• Το τρίτο GPIO interface είναι το **gpio_16bit**. Όπως ορίζει και το όνομα του, έχει μέγεθος 16 bit, συνεπώς διαθέτει 16 ακροδέκτες, και είναι περιφερειακό του **microblaze_1**. Αντίθετα με τα δυο προαναφερθέντα interfaces**,** που χρησιμοποιούνται αποκλειστικά για την διαχείριση των σημάτων ελέγχου του modem, το **gpio_16bit** μπορεί να τροποποιήσει κάθε του ακροδέκτη ως είσοδο ή έξοδο και να συνδεθεί οποιαδήποτε συσκευή σε αυτό, ανάλογα με τι ανάγκες του διαχειριστή, όπως μικροελεγκτές, αισθητήρες, διακόπτες και leds.

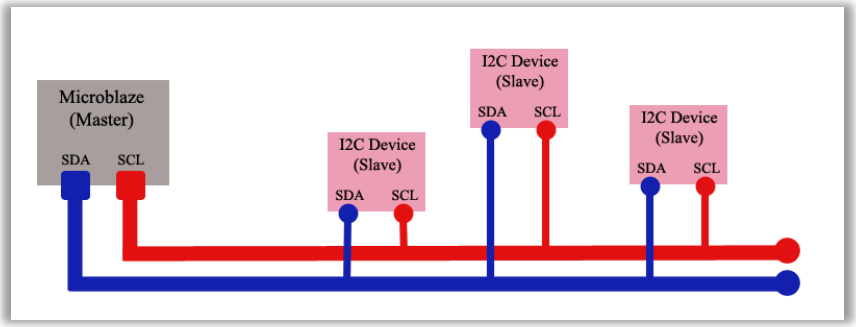
4.5.3 I²C Module

Το I²C (Inter-Intergrated Circuit) είναι ένα πρωτόκολλο επικοινωνίας που εισάχθηκε από την Phillips. Σχεδιάστηκε για να συνδέει περιφερειακά χαμηλής ταχύτητας με μητρικές πλακέτες, ενσωματωμένα συστήματα, κινητά, πληκτρολόγια και ποικίλες άλλες ηλεκτρονικές συσκευές. Το πρωτόκολλο αυτό επιτρέπει να συνδεθούν έως και 127 διαφορετικοί κόμβοι (συσκευές) σε έναν δίαυλο δυο γραμμών. Η πρώτη από αυτές τις γραμμές (SCL) χειρίζεται την ταχύτητα μετάδοσης δεδομένων ενώ η δεύτερη (SDA) αποτελεί το φυσικό μέσο αμφίδρομης μετάδοσης δεδομένων. Το I²C υποστηρίζει την ύπαρξη ενός ή πολλών masters (συνήθως μικροελεγκτές) και πολλών slaves που περιλαμβάνουν μεταξύ άλλων μικροελεγκτές και αισθητήρες. Στις περισσότερες περιπτώσεις χρησιμοποιείται ένας master.

Ο master είναι αυτός που συγχρονίζει την επικοινωνία μέσω της γραμμής SCL. Μεταφορά δεδομένων μπορεί να γίνει τόσο από τον master προς τον slave όσο και αντίστροφα αλλά η εκκίνηση μιας μεταφοράς γίνεται πάντα μόνο από τον master. Οι συσκευές slaves είναι αυτές που ανταποκρίνονται στον master.

Όταν ένας master, στην προκειμένη περίπτωση το ενσωματωμένο σύστημα, επιθυμεί να επικοινωνήσει με μια συσκευή slave στέλνει μια σειρά παλμών μέσω των γραμμών SDA και SCL. Τα δεδομένα που στέλνονται περιλαμβάνουν μια διεύθυνση που χαρακτηρίζει μια συγκεκριμένη συσκευή slave με την οποία πρόκειται ο master να αλληλεπιδράσει. Η διεύθυνση βρίσκεται στο πρώτο μεταδιδόμενο byte και αποτελείται από τα 7 πρώτα bits αυτού του byte. Το bit που απομένει καθορίζει το κατά πόσο ο master επιθυμεί να στείλει (write) ή να λάβει (read) δεδομένα από τη συσκευή slave. Το byte της διεύθυνσης ακολουθείται από ένα ή περισσότερα bytes που αποτελούν τα καθαρά δεδομένα. Στο σύνολο των δεδομένων που στέλνονται θα ανταποκριθεί μόνο η συσκευή που έχει την ίδια διεύθυνση με αυτήν που βρίσκεται στο πρώτο byte. Η συσκευή αυτή είτε θα λάβει τα δεδομένα από τον master είτε θα στείλει δεδομένα σε αυτόν, κάτι που εξαρτάται από το LSB bit του byte που περιλαμβάνει την διεύθυνση.

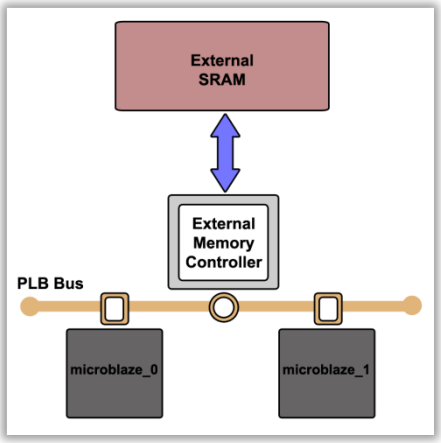
Στην συγκεκριμένη υλοποίηση έχει χρησιμοποιηθεί μόνο ένα I²C module το οποίο αναγνωρίζεται με το όνομα **i2c**. Το module αυτό, που ανήκει στα περιφερειακά αυτού του συστήματος, χρησιμοποιείται από τον **microblaze_1** για επικοινωνία με εξωτερικές συσκευές, όπως κατά κύριο λόγο αισθητήρες, οι οποίες είναι συνδεδεμένες στον δίαυλο I²C. Μπορεί να λειτουργήσει σε ταχύτητες των 100 KHz και 400 KHz ενώ διαθέτει ουρές FIFO μεγέθους 16 byte τόσο για την λήψη όσο και για την μετάδοση δεδομένων.



Σχήμα 4.4 Ο δίαυλος I²C

4.5.4 Μνήμη SRAM (External Memory Controller)

Η αναπτυξιακή πλακέτα έχει διαθέσιμη εξωτερική μνήμη SRAM μεγέθους 1 mb. Η SRAM αυτή δεν αποτελεί κάποιου είδους τροποποιήσιμη περιφερειακή συσκευή. Για να υπάρξει πρόσβαση σε αυτή χρειάζεται ένας ελεγκτής εξωτερικής μνήμης (External Memory Controller). Στην παρούσα υλοποίηση έχει προσαρτηθεί στον δίαυλο PLB ένας τέτοιος ελεγκτής, που στην ουσία αποτελεί μια από τις περιφερειακές συσκευές, και δίνει την δυνατότητα και στους δυο επεξεργαστές του συστήματος να έχουν πρόσβαση στην μνήμη SRAM. Η χρήση της μνήμης αυτής είναι αναγκαία καθώς σε αυτή φορτώνεται μεγάλο μέρος του κώδικα που θα εκτελέσει ο **microblaze_0** αλλά ταυτόχρονα αποτελεί και κοινή μνήμη για την ανταλλαγή δεδομένων μεταξύ των δυο microblaze.



Σχήμα 4.5 Πρόσβαση στην εξωτερική μνήμη SRAM

4.5.5 Interrupt controller (Ελεγκτής διακοπών)

Ως interrupt ορίζεται μια διακοπή που μπορεί να προκληθεί σε έναν επεξεργαστή από μια εξωτερική πηγή, όπως μια περιφερειακή συσκευή. Ένα interrupt ενδεικνύει μια συνθήκη υψηλής προτεραιότητας που απαιτεί την διακοπή του εκτελούμενου στον επεξεργαστή κώδικα προκειμένου να εξυπηρετηθεί ο πόρος που προκάλεσε την διακοπή. Αυτό που συμβαίνει είναι ο επεξεργαστής να θέτει σε αναμονή τις τρέχουσες δραστηριότητες του, να σώζει την κατάστασή του και να εκτελεί ένα μικρό κομμάτι κώδικα, που ονομάζεται interrupt handler, ώστε να διαχειριστεί το γεγονός. Η χρήση των interrupts είναι πολύ σημαντική καθώς ο επεξεργαστής δεν χρειάζεται να αναλώνεται ελέγχοντας συνεχώς τις περιφερειακές συσκευές για δεδομένα. Αντίθετα όταν προκληθεί κάποιο γεγονός θα είναι ενήμερος για να το διαχειριστεί.

Οι περισσότεροι επεξεργαστές, όπως και οι microblaze, υποστηρίζουν μια ή περισσότερες εισόδους για αιτήσεις interrupts που επιτρέπουν σε περιφερειακά να εξυπηρετηθούν. Στην παρούσα υλοποίηση οι microblaze παρέχουν μόνο μια τέτοια είσοδο. Το γεγονός ότι ένας επεξεργαστής επικοινωνεί με περισσότερα από ένα περιφερειακά κάνει αναγκαία την χρήση των interrupt controllers.

Ένας interrupt controller είναι ελεγκτής που μπορεί να διευρύνει τον αριθμό των πηγών που μπορούν να προκαλέσουν διακοπή σε έναν επεξεργαστή. Μπορεί να δεχτεί interrupts από έως και 32 διαφορετικές πηγές ενώ διαθέτει μια έξοδο προς τον microblaze. Αποτέλεσμα είναι ο επεξεργαστής να μπορεί να δεχτεί interrupts από περισσότερα από ένα περιφερειακά. Τα σήματα διακοπών των περιφερειακών είναι συγκεντρωμένα στον interrupt controller και οποτεδήποτε ένα από αυτά προκαλέσει διακοπή ο ελεγκτής διακόπτει, με την σειρά του, τον microblaze μέσα από την μια έξοδο προς αυτόν.

Κάθε ένας από τους δυο επεξεργαστές διαθέτει έναν τέτοιο ελεγκτή στον οποίο είναι συνδεδεμένα τα σήματα των interrupts που προκαλούν τα περιφερειακά με τα οποία κάθε ένας επικοινωνεί. Οι δυο ελεγκτές που προκύπτουν είναι οι **xps_intc_0** και **xps_intc_1** που είναι συνδεδεμένοι με τον **microblaze_0** και τον **microblaze_1** αντίστοιχα. Κάθε interrupt controller λαμβάνει σήματα από τις περιφερειακές συσκευές και ενημερώνει τον microblaze για το ποια συσκευή προκάλεσε το interrupt.

Ο ελεγκτής **xps_intc_0** έχει συνδεδεμένα τρία interrupt σήματα:

• Το πρώτο τέτοιο σήμα προέρχεται από τον ελεγκτή **hspa_rs232**. Ο ελεγκτής αυτός, όπως είναι ήδη γνωστό, επικοινωνεί με το HSPA modem και προκαλεί interrupt οποτεδήποτε λάβει δεδομένα από αυτό ώστε να τα διαχειριστεί ο **microblaze_0**.

• Το δεύτερο σήμα έχει πηγή τον δίαυλο **fsl_mb1_to_mb0** ο οποίος προκαλεί διακοπή οποτεδήποτε υπάρχουν δεδομένα στην SRAM από τον **microblaze_1** προς τον **microblaze_0**.

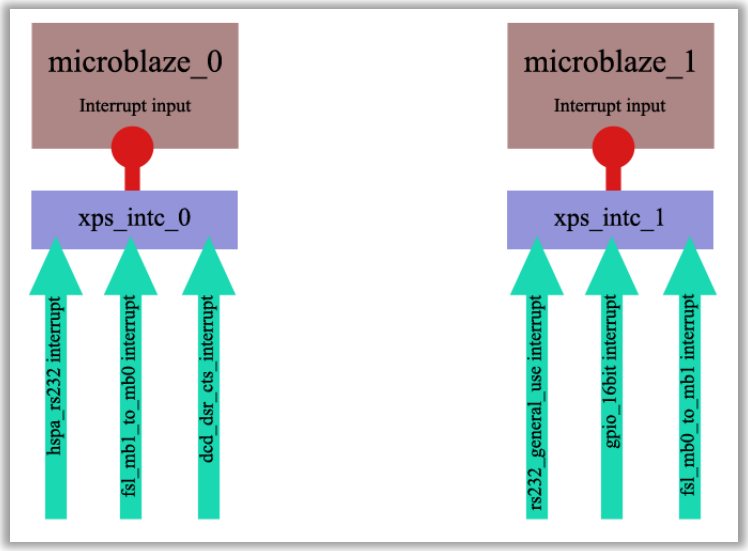
• Το τρίτο σήμα προέρχεται από το GPIO interface **dcd_dsr_cts**. Το interface αυτό διακόπτει τον **microblaze_0** αν υπάρξει αλλαγή σε ένα από τα DCD, DSR, CTS σήματα κάτι που συνεπάγεται ότι έχει αλλάξει η κατάσταση του modem. Σε μια τέτοια περίπτωση το σύστημα μπορεί να είναι πάντα ενήμερο και να διαχειρίζεται τις αλλαγές της κατάστασης του modem ώστε να αυτό να είναι πάντα σε σύνδεση με τον Server.

Ο **xps_intc_1** διαχειρίζεται, όπως και ο προηγούμενος interrupt controller, τρία interrupt σήματα:

• Το πρώτο από αυτά πηγάζει από τον ελεγκτή **rs232_general_use** που προκαλεί interrupt οποτεδήποτε ληφθούν δεδομένα από οποιαδήποτε συσκευή είναι συνδεδεμένη.

• Το δεύτερο σήμα προέρχεται από το GPIO interface **gpio_16bit** που στέλνει interrupt κάθε φορά που εξωτερικές πηγές αλλάζουν την λογική κατάσταση των σημάτων στους συνδεδεμένους ακροδέκτες σε αυτό.

• Το τρίτο σήμα προκαλείται από τον δίαυλο **fsl_mb0_to_mb1** ο οποίος, αντίστοιχα με τον προηγούμενο FSL δίαυλο, προκαλεί διακοπή οποτεδήποτε υπάρχουν δεδομένα στην SRAM από τον **microblaze_0** προς τον **microblaze_1**.



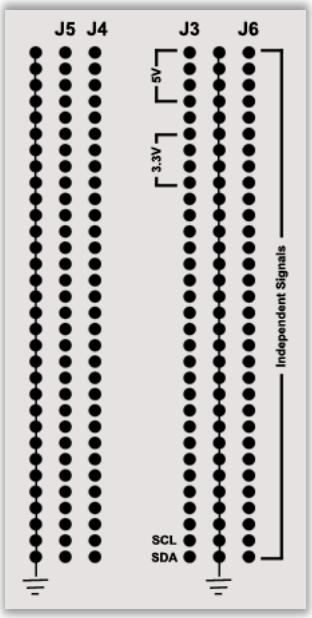
Σχήμα 4.6 Interrupt controllers και interrupts των περιφερειακών

4.6 Συνδεσμολογία των περιφερειακών

Ως τώρα έχει γίνει αναφορά στον τρόπο λειτουργίας των περιφερειακών αλλά και στον τρόπο που συνδέονται και επικοινωνούν μέσα στο σύστημα με τους επεξεργαστές. Σκοπός όμως πολλών από αυτά είναι να εξασφαλίσουν στους επεξεργαστές επικοινωνία με εξωτερικές συσκευές, είτε πρόκειται για το modem είτε για άλλο σύστημα. Συνεπώς χρειάζονται κάποιου είδους υποδοχές, όπως στη συγκεκριμένη περίπτωση ακροδέκτες, στις οποίες να καταλήγουν τα σήματα που διαθέτουν τα περιφερειακά για εξωτερική επικοινωνία.

Τα περιφερειακά που κάνουν χρήση ακροδεκτών είναι όλα όσα ανήκουν στις κατηγορίες των UartLite Controller, GPIO Interface και I^2C Module. Παρακάτω θα αναλυθούν οι ακροδέκτες της αναπτυξιακής πλακέτας που συνδέονται με αυτά τα περιφερειακά. Η συγκεκριμένη αναπτυξιακή πλακέτα διαθέτει 6 σειρές των 32 ακροδεκτών, όπου δυο από αυτές χρησιμοποιούνται αποκλειστικά για γείωση ενώ οι υπόλοιπες τέσσερεις, που διακρίνονται με τα ονόματα **J3**, **J4**, **J5** και **J6**, χρησιμοποιούνται για ποικίλους σκοπούς.

Η πλειοψηφία των περιφερειακών, με εξαίρεση το module **i2c** και τον ελεγκτή **RS232_Uart**, είναι συνδεδεμένα στους ακροδέκτες της επονομαζόμενης σειράς **J6**. Το module **i2c** έχει τους δυο ακροδέκτες του στη σειρά **J3**. Ο ελεγκτής **RS232_Uart** αποτελεί εξαίρεση καθώς τα σήματα του δεν είναι συνδεδεμένα σε κάποια από αυτές τις σειρές και για αυτό θα δοθεί ξεχωριστή περιγραφή για τον τρόπο που συνδέεται με εξωτερικές συσκευές.



Σχήμα 4.7 Οι σειρές ακροδεκτών της αναπτυξιακής πλακέτας

4.6.1 Συνδεσμολογία στη σειρά J6

Όπως προαναφέρθηκε, σε αυτή τη σειρά βρίσκονται συνδεδεμένα τα σήματα της πλειοψηφίας των περιφερειακών που επικοινωνούν με εξωτερικές συσκευές. Από τους ακροδέκτες αυτής της σειράς χρησιμοποιούνται οι πρώτοι 25.

- Το πρώτο περιφερειακό που καταλήγει στους ακροδέκτες της σειράς **J6** είναι το **gpio_16bit** το οποίο, όπως είναι ήδη γνωστό, έχει πλάτος 16 bit και καθώς κάθε bit αντιπροσωπεύεται από έναν ακροδέκτη χρησιμοποιούνται οι πρώτοι 16 ακροδέκτες (Pin 2 έως Pin 32) για αυτό το περιφερειακό. Καθώς ο ρόλος του **gpio_16bit** είναι να δώσει την δυνατότητα στον διαχειριστή να συνδέει ποικίλες συσκευές κατ' επιθυμία, οι ακροδέκτες αυτοί, ως υποδοχές, μπορούν να αλλάξουν δυναμικά και να χρησιμοποιηθούν τόσο ως είσοδοι όσο και ως έξοδοι.
- Το δεύτερο περιφερειακό κάνει χρήση των επόμενων δυο ακροδεκτών (Pin 34 και Pin36). Πρόκειται για τον ελεγκτή **rs232_general_use** που κάνει χρήση δυο σημάτων, ένα για αποστολή και ένα για λήψη δεδομένων. Ο πρώτος από τους δυο ακροδέκτες (Pin 34) χρησιμοποιείται, προφανώς, ως έξοδος για αποστολή δεδομένων (TX) ενώ ο δεύτερος (Pin 36) ως είσοδος για λήψη δεδομένων (RX).
- Το τρίτο περιφερειακό είναι το **hspa_rs232** το οποίο είναι συνδεδεμένο με τους ακροδέκτες Pin 38 και Pin 40. Οι δυο αυτοί ακροδέκτες αφορούν την αποστολή και λήψη δεδομένων προς και από το modem. Ο πρώτος (Pin 38) χρησιμοποιείται ως είσοδος για λήψη (RX) ενώ ο δεύτερος (Pin 40) χρησιμοποιείται ως έξοδος για αποστολή (TX).
- Το τέταρτο περιφερειακό χρησιμοποιεί τους ακροδέκτες Pin 42 και Pin 44. Πρόκειται για το GPIO interface **dtr_rts** το οποίο χρησιμοποιεί και τους δυο ακροδέκτες ως εξόδους ώστε να μπορεί να ελέγχει τα σήματα DTR και RTS του modem.
- Το πέμπτο και τελευταίο περιφερειακό για την σειρά **J6** είναι το **dcd_dsr_cts**. Κάνει χρήση των τελευταίων 3 ακροδεκτών από τους 25 (Pin 46, Pin 48 και Pin 50). Και οι τρεις ακροδέκτες λειτουργούν ως είσοδοι ώστε το σύστημα να λαμβάνει την κατάσταση των σημάτων DCD, DSR και CTS του modem.

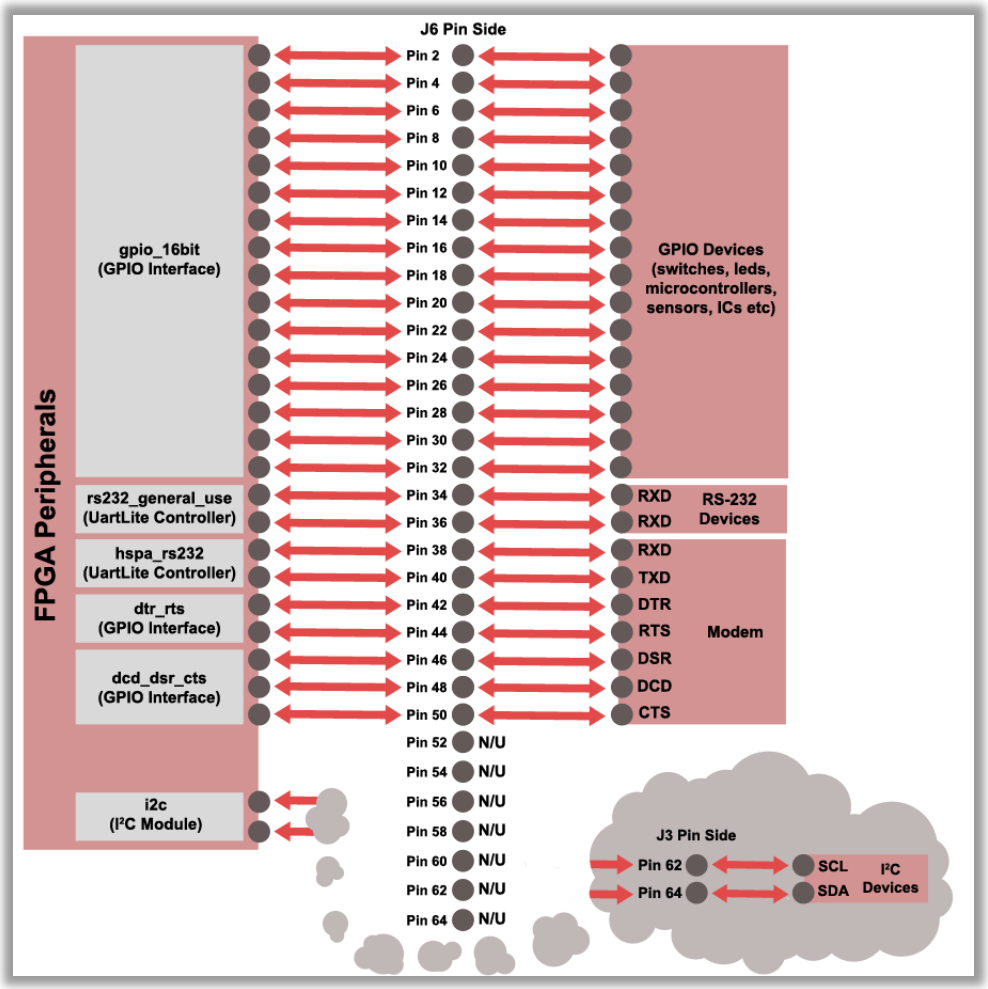
4.6.2 Συνδεσμολογία στη σειρά J3

Σε αυτή τη σειρά καταλήγουν οι ακροδέκτες του περιφερειακού **i2c**. Το συγκεκριμένο περιφερειακό χρειάζεται ειδικές αντιστάσεις (pull-up resistors) προκειμένου να επικοινωνήσει και είναι και ο λόγος που είναι συνδεδεμένο σε ακροδέκτες αυτής της σειράς καθώς η σειρά **J3** παρέχει τέτοιες αντιστάσεις ιδιαίτερα για αυτό το σκοπό. Συγκεκριμένα χρησιμοποιούνται οι ακροδέκτες Pin 62 και Pin 64 όπου ο πρώτος (Pin 62) είναι συνδεδεμένος με τη γραμμή SCL του I^2C Module ενώ ο δεύτερος (Pin 64) με τη γραμμή SDA.

Ένας επιπλέον λόγος που κάνει άξια προς αναφορά τη σειρά J3 είναι η δυνατότητα που παρέχει για άμεση τροφοδοσία εξωτερικών συσκευών σε τάσεις των 3,3V και 5V. Οι συσκευές αυτές λαμβάνουν την τροφοδοσία τους άμεσα από την αναπτυξιακή πλακέτα ενώ οι ανάγκες τους για γείωση καλύπτονται από τις δυο σειρές γείωσης.

4.6.3 Συνδεσμολογία του ελεγκτή RS232_Uart

Η ιδιαιτερότητα του ελεγκτή **RS232_Uart** είναι ότι χρησιμοποιείται από το ενσωματωμένο σύστημα για να τυπώσει μηνύματα σε μια εφαρμογή ενός τερματικού, όπως το Hyper Terminal, μέσω της σειριακής θύρας. Το πρόβλημα που εμφανίζεται είναι ότι το τερματικό χρησιμοποιεί διαφορετική τάση για επικοινωνία μέσω πρωτοκόλλου RS-232 από αυτή του ενσωματωμένου συστήματος. Για να λυθεί αυτό το πρόβλημα και να διευκολυνθεί η ανάγκη για επικοινωνία με το τερματικό η αναπτυξιακή πλακέτα παρέχει την κατάλληλη υποδοχή, γνωστή και ως DB9, η οποία συνοδεύεται από τον απαραίτητο μετατροπέα τάσης. Ο ελεγκτής **RS232_Uart** έχει συνδεδεμένα τα σήματα του με τον μετατροπέα τάσης και κατ' επέκταση με την υποδοχή DB9 συνεπώς γίνεται πολύ εύκολη η αποστολή μηνυμάτων προς την εφαρμογή του τερματικού με χρήση ενός καλωδίου σειριακού τύπου που ενώνει τα δυο συστήματα.



Σχήμα 4.8 Συνδεσμολογία των περιφερειακών με τους ακροδέκτες

4.7 HSPA Modem

Ο όρος HSPA (High Speed Packet Access) αναφέρεται σε πρωτόκολλο 3g κυψελοειδών δικτύων (κινητής τηλεφωνίας) που δίνει την δυνατότητα ασύρματης μετάδοσης δεδομένων σε υψηλές ταχύτητες. Αποτελεί ένα συνδυασμό των πρωτοκόλλων HSDPA (High Speed Downlink Packet Access) και HSUPA (High Speed Uplink Packet Access) που υποστηρίζουν μέγιστες ταχύτητες έως 14 mbit/sec και 5.8 mbit/sec αντίστοιχα.

Στη παρούσα υλοποίηση χρησιμοποιείται modem τεχνολογίας HSPA για να εξασφαλίσει τις ανάγκες του ενσωματωμένου συστήματος για ασύρματη επικοινωνία με τον χρήστη. Συγκεκριμένα χρησιμοποιείται το μοντέλο **MTSMC-H4**, προϊόν της εταιρείας Multi-Tech Systems, το οποίο είναι προσαρτημένο στην αναπτυξιακή πλακέτα **MTSMI-UDK** της ίδιας εταιρείας.

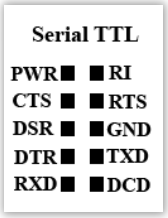
Σε αυτή την αναπτυξιακή πλακέτα μπορούν να προσαρμοστούν ποικίλες τεχνολογίες modem, προς πειραματισμό, και για αυτό το λόγο διατίθενται ποικίλοι τρόποι σύνδεσης με εξωτερικές πηγές ώστε να μπορεί ο χρήστης να προσαρμόσει τους πειραματισμούς στις ανάγκες του.

Η μόνη σύνδεση της αναπτυξιακής πλακέτας που ενδιαφέρει το σύστημα είναι αυτή που δίνει την δυνατότητα επικοινωνίας του ενσωματωμένου συστήματος με το HSPA modem μέσω πρωτοκόλλου Serial-TTL. Πρόκειται για μια υποδοχή δέκα ακροδεκτών που διαθέτει όλα τα απαραίτητα σήματα για την επιτυχή υλοποίηση του σειριακού πρωτοκόλλου. Για τις ανάγκες επικοινωνίας μεταξύ του modem και του ενσωματωμένου συστήματος χρησιμοποιούνται οι 8 από αυτούς τους ακροδέκτες που αναλύονται ακολούθως:

- CTS (Clear To Send): Το σήμα σε αυτόν τον ακροδέκτη είναι ένδειξη του κατά πόσο το modem είναι σε ετοιμότητα για οποιαδήποτε αλληλεπίδραση. Λογική τιμή 0 σε αυτόν συνεπάγεται ετοιμότητα του modem ενώ λογική τιμή 1 το αντίθετο.
- RXD (Receive Data): Ο ακροδέκτης αυτός χρησιμοποιείται για την λήψη δεδομένων, στην προκειμένη περίπτωση, από το ενσωματωμένο σύστημα.
- TXD (Transmit Data): Ο ακροδέκτης αυτός χρησιμεύει για αποστολή δεδομένων προς το ενσωματωμένο σύστημα.
- DTR (Data Terminal Ready): Πρόκειται για ένα σήμα το οποίο μπορεί να χρησιμοποιηθεί από το ενσωματωμένο σύστημα για έλεγχο λειτουργιών του modem. Συγκεκριμένα το ενσωματωμένο σύστημα αλλάζοντας την λογική τιμή αυτού του σήματος σε 1 μπορεί να τερματίσει μια TCP σύνδεση που έχει πραγματοποιήσει το modem με τον server.
- RTS (Ready to Send): Είναι το δεύτερο σήμα που χρησιμοποιείται από το ενσωματωμένο σύστημα προκειμένου να γίνει έλεγχος της ροής εισερχόμενων δεδομένων από το modem. Λογική τιμή 1 αποτρέπει το modem από το να αποστείλει δεδομένα προς το ενσωματωμένο σύστημα.
- DCD (Data Carrier Detect): Το σήμα σε αυτόν τον ακροδέκτη είναι ένδειξη του κατά πόσο το modem έχει συνδεθεί στο δίκτυο GPRS, κάτι που σημαίνει πως υπάρχει πρόσβαση στο internet. Λογική τιμή 0 αυτού του σήματος συνεπάγεται πως το modem έχει πραγματοποιήσει σύνδεση με το δίκτυο GPRS ενώ το αντίθετο προκύπτει με λογική τιμή 1.
- DSR (Data Set Ready): Το σήμα αυτό χρησιμοποιείται ως ένδειξη για την κατάσταση της TCP σύνδεσης μεταξύ του modem και του server. Ενεργή TCP σύνδεση συνεπάγεται λογική τιμή 0 σε αυτό. Σε περίπτωση που τερματιστεί αυτή η σύνδεση η λογική τιμή του σήματος αλλάζει σε 1.
- GND (Ground): Ο ακροδέκτης αυτός αποτελεί την απαιτούμενη γείωση μεταξύ του ενσωματωμένου συστήματος και του modem.

Η αλληλεπίδραση του ενσωματωμένου συστήματος με το modem μέσα από το σειριακό πρωτόκολλο γίνεται με χρήση ενός σετ εντολών που αναγνωρίζονται ως AT Commands. Είναι ένας κοινός τρόπος αλληλεπίδρασης με πολλές τεχνολογίες modem. Είναι στην ουσία μικρές συμβολοσειρές συγκεκριμένης μορφής που στέλνονται, στην προκειμένη περίπτωση, από το ενσωματωμένο σύστημα και στις οποίες ανταποκρίνεται το modem. Με χρήση τέτοιων εντολών μπορεί να γίνει, μεταξύ άλλων, έλεγχος και τροποποίηση των χαρακτηριστικών του modem, να ληφθούν πληροφορίες σχετικά με την κατάσταση του, να πραγματοποιηθούν κλήσεις ή συνδέσεις με απομακρυσμένα συστήματα. Παραδείγματα τέτοιων εντολών είναι η συμβολοσειρά "AT+CSQ" στην οποία το modem ανταποκρίνεται στέλνοντας ως απάντηση την κατάσταση του σήματος. Άλλο παράδειγμα είναι η συμβολοσειρά "AT#CONNECTIONSTART" στην οποία το modem ανταποκρίνεται προσπαθώντας να συνδεθεί στο δίκτυο GPRS.

Σε αυτό το σημείο θα δοθεί μια σύντομη περιγραφή σχετικά με τα βήματα που ακολουθούνται προκειμένου το modem να δημιουργήσει μια TCP σύνδεση με τον server. Μόλις το modem ενεργοποιηθεί πραγματοποιείται η αρχικοποίηση του. Όταν ολοκληρωθεί αυτό το βήμα αλλάζει η λογική τιμή του σήματος CTS σε 0 ως ένδειξη ότι το modem είναι έτοιμο για αλληλεπίδραση. Στη συνέχεια στέλνεται από το ενσωματωμένο σύστημα η εντολή "AT#CONNECTIONSTART". Το modem ανταποκρίνεται επιχειρώντας να συνδεθεί στο δίκτυο GPRS και μόλις πραγματοποιηθεί αυτή η σύνδεση αλλάζει η λογική κατάσταση του σήματος DCD σε 0 πράγμα που για το ενσωματωμένο σύστημα σημαίνει πως το modem εκτέλεσε με επιτυχία στην τελευταία εντολή. Σε αυτό το σημείο αποστέλλεται η εντολή "AT#OTCP=1" που αναγκάζει το modem να επιχειρήσει TCP σύνδεση με τον server. Αν ο server αποδεχτεί την σύνδεση το modem θα αλλάξει την λογική τιμή του σήματος DSR σε 0. Με την τελευταία ενέργεια από μέρος του modem το ενσωματωμένο σύστημα γνωρίζει πως πλέον είναι εφικτή η ανταλλαγή δεδομένων μεταξύ αυτού και του server.



Σχήμα 4.9 Η υποδοχή Serial TTL στο modem

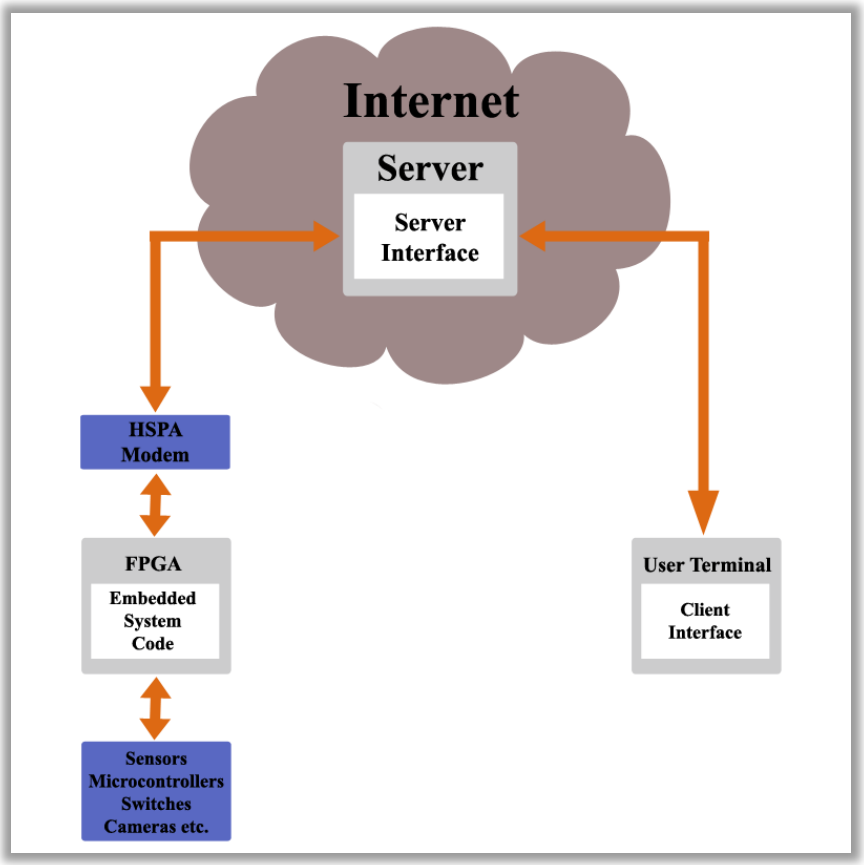
5. Ανάλυση του λογισμικού(Software)

Στο κεφάλαιο αυτό θα αναλυθεί το λογισμικό, ως σύνολο, που αναπτύχθηκε για τις ανάγκες του ενσωματωμένου συστήματος. Ξεκινώντας θα γίνει μια εισαγωγική αναφορά σε αυτό αλλά και στους λόγους για τους οποίους χρειάστηκε να παραχθεί. Στη συνέχεια θα προσεγγιστούν σε μεγαλύτερο βάθος τα επιμέρους τμήματα που το συνθέτουν.

5.1 Γενική αναφορά στο λογισμικό

Προκειμένου να αναλυθεί το λογισμικό και οι λόγοι για τους οποίους δημιουργήθηκε είναι αναγκαίο να γίνει μια πολύ σύντομη περιγραφή του τρόπου λειτουργίας του συστήματος αλλά και του τρόπου επικοινωνίας του με τον χρήστη.

Ο λόγος ανάπτυξης αυτού του συστήματος είναι να δώσει την δυνατότητα στον οποιοδήποτε χρήστη να αντλήσει απομακρυσμένες πληροφορίες και να ελέγξει απομακρυσμένα ψηφιακά εξαρτήματα μέσω διαδικτύου και με την χρήση ασύρματης τεχνολογίας. Για να επιτευχθεί αυτό, δημιουργήθηκε ένα ενσωματωμένο σύστημα στο οποίο προσαρτώνται τα προαναφερθέντα ψηφιακά εξαρτήματα και το οποίο, με τη χρήση modem ασύρματης τεχνολογίας, μπορεί να επικοινωνήσει διαδικτυακά με έναν χρήστη και να ανταλλάξει δεδομένα που αφορούν αυτά τα εξαρτήματα. Αυτός ο χρήστης από τη μεριά του χρειάζεται μια διεπαφή για να «μιλήσει» με το απομακρυσμένο σύστημα. Αυτή η διεπαφή είναι στην ουσία μια εφαρμογή που εκτελείται στο τερματικό του χρήστη και κάνει δυνατή την πιο πάνω απαίτηση. Το γεγονός ότι γίνεται χρήση του διαδικτύου για τη μεταξύ τους επικοινωνία δημιουργεί επιπλοκές καθώς σε πολλές περιπτώσεις δεν είναι δυνατή η σύνδεση σημείου προς σημείο ανάμεσα σε δυο οποιεσδήποτε συσκευές. Έτσι προκύπτει η ανάγκη για ένα μηχανισμό που θα λειτουργήσει ως μεσολαβητής ώστε να είναι εφικτή αυτή η επικοινωνία. Η λύση δόθηκε με την χρήση ενός server που εκτελεί μια εφαρμογή και φροντίζει να μεταβιβάσει τα δεδομένα μεταξύ του χρήστη και του συστήματος.



Σχήμα 5.1.1 Εισαγωγική αναφορά στο λογισμικό

Από την πιο πάνω περιγραφή είναι ξεκάθαρο πως πρέπει να αναπτυχθεί λογισμικό για τρία ανεξάρτητα υπολογιστικά συστήματα, το ενσωματωμένο σύστημα, το τερματικό του χρήστη (υπολογιστής) και τον server.

Το λογισμικό στον server διαδραματίζει τον ρόλο του μεσολαβητή μεταξύ του χρήστη και του ενσωματωμένου συστήματος. Αποτελεί ένα απλό μηχανισμό που μεταβιβάζει τα δεδομένα που ο server λαμβάνει από τον χρήστη προς το ενσωματωμένο σύστημα και αντίστροφα.

Το λογισμικό στον χρήστη αποτελεί μια διεπαφή που εκτελείται στο τερματικό. Πρόκειται ουσιαστικά για ένα γραφικό περιβάλλον που δίνει την δυνατότητα διάδρασης του χρήστη με το απομακρυσμένο σύστημα. Μπορεί να στέλνει δεδομένα κατά απαίτηση του χρήστη αλλά και να μεταφράζει τα εισερχόμενα δεδομένα ώστε να είναι φιλικά προς αυτόν. Αυτό που συμβαίνει στην πραγματικότητα είναι ότι η διεπαφή επικοινωνεί με τον server και όχι με το απομακρυσμένο σύστημα. Τα δεδομένα της διεπαφής αποστέλλονται, όπως προαναφέρθηκε, στον server ο οποίος στη συνέχεια θα τα μεταβιβάσει στον παραλήπτη.

Το λογισμικό στο ενσωματωμένο σύστημα, το οποίο είναι μοιρασμένο στους δυο επεξεργαστές, είναι υπεύθυνο για την αρχικοποίηση και διαχείριση του συστήματος και των περιφερειακών, την επικοινωνία με το modem (και κατ' επέκταση με τον server) και τις ψηφιακές συσκευές αλλά και την επεξεργασία των δεδομένων που στέλνονται και λαμβάνονται.

5.2 Λογισμικό της εφαρμογής στον Server

Το λογισμικό που περιγράφεται σε αυτό το υποκεφάλαιο έχει αναπτυχθεί σε γλώσσα java με χρήση της πλατφόρμας Netbeans 7.1. Φιλοξενείται σε έναν εικονικό Server που βασίζεται σε λειτουργικό σύστημα των Windows, πιο συγκεκριμένα στα Windows Server 2008 R2. Αποτελεί μια εφαρμογή με διαδραστικό γραφικό περιβάλλον που υλοποιεί έναν απλό μηχανισμό μεταβίβασης δεδομένων μεταξύ υπολογιστικών συστημάτων.

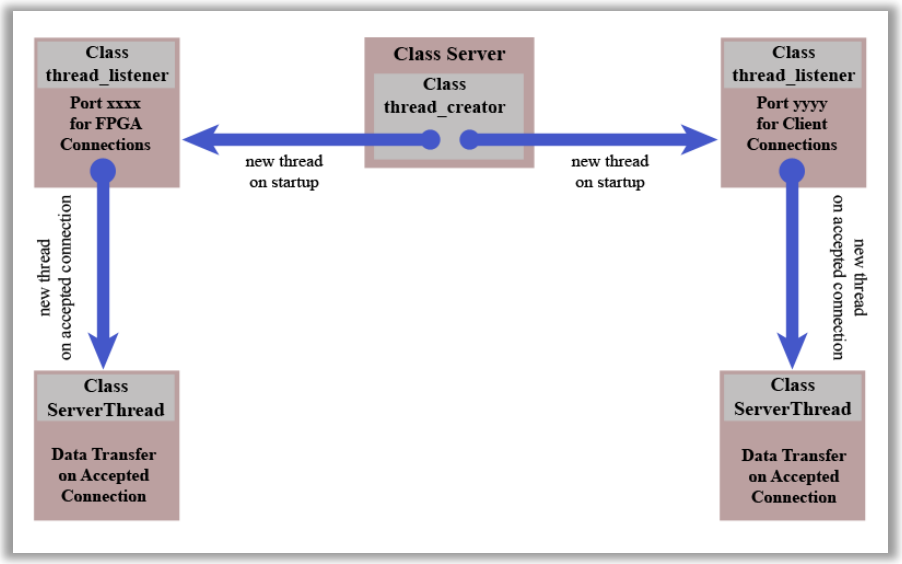
5.2.1 Γενική αναφορά στο λογισμικό του Server

Όπως αναφέρθηκε πιο πάνω το λογισμικό του Server είναι μια πολύ βασική εφαρμογή με συγκεκριμένες λειτουργίες καθώς στην παρούσα φάση στόχος είναι η διασφάλιση επιτυχούς μεταβίβασης δεδομένων μεταξύ δυο υπολογιστικών συστημάτων. Ο τρόπος επικοινωνίας αυτής της εφαρμογής με τα υπολογιστικά συστήματα είναι διαδικτυακά με χρήση του πρωτοκόλλου TCP. Ο βασικός λόγος επιλογής του συγκεκριμένου πρωτοκόλλου είναι ότι εξασφαλίζει την επιτυχή παράδοση των πακέτων δεδομένων. Για να επιτευχθεί επικοινωνία μεταξύ της εφαρμογής και των υπολογιστικών συστημάτων όμως είναι αναγκαίο πρώτα κάθε ένα από αυτά να εγκαθιδρύσει TCP σύνδεση με την εφαρμογή. Αφού πραγματοποιηθεί μια επιτυχής σύνδεση η μεταξύ τους ανταλλαγή δεδομένων είναι πλέον εφικτή.

Κατά την εκκίνηση της εφαρμογής στον Server δημιουργούνται δυο νήματα(threads) από την εφαρμογή. Κάθε ένα από αυτά τα νήματα εκτελεί έναν Listener που στην ουσία είναι ένα κομμάτι κώδικα το οποίο αναμένει για αποδοχή νέων TCP συνδέσεων. Η διαφορά μεταξύ των δυο αυτών νημάτων είναι ότι αναμένουν συνδέσεις σε διαφορετικές θύρες. Η μια από αυτές τις θύρες αφορά συνδέσεις από τον χρήστη προς τον Server ενώ η άλλη από το ενσωματωμένο σύστημα προς τον Server.

Όταν ένας από τους προαναφερθέντες ζητήσει να συνδεθεί με τον Server τότε η εφαρμογή εγκαθιδρύει μια σύνδεση μεταξύ τους ανοίγοντας μια νέα υποδοχή (socket). Ταυτόχρονα δημιουργεί ένα νέο νήμα το οποίο εκτελεί μια ρουτίνα που θα εξυπηρετήσει τη νέα σύνδεση. Οι δυο βασικές λειτουργίες του νέου νήματος είναι η λήψη των εισερχόμενων δεδομένων και στη συνέχεια η προώθηση τους. Στην περίπτωση που ένα νήμα, ας υποθέσουμε αυτό που εξυπηρετεί τη σύνδεση μεταξύ του χρήστη και του Server, λάβει στην υποδοχή του εισερχόμενα δεδομένα πρέπει να γνωρίζει την υποδοχή του νήματος που εξυπηρετεί την «απέναντι πλευρά» προκειμένου να μεταβιβάσει τα δεδομένα. Συνεπώς, για να είναι αυτό εφικτό, κάθε φορά που δημιουργείται μια νέα σύνδεση, το νήμα που την εξυπηρετεί γνωστοποιεί την υποδοχή αυτής της σύνδεσης.

Για να είναι τελικά επιτυχής η επικοινωνία μεταξύ του χρήστη και του συστήματος και οι δυο πλευρές ζητούν σύνδεση με τον Server κάθε μια στον αντίστοιχο Listener στην θύρα που αυτός «ακούει». Ως αποτέλεσμα θα ανοίξουν δυο νέες υποδοχές για τις δυο συνδέσεις και θα δημιουργηθούν δυο νέα νήματα για κάθε μια από τις συνδέσεις. Κάθε ένα από αυτά τα νήματα θα γνωστοποιήσει την υποδοχή της σύνδεσης που έχει αναλάβει και πλέον αν κάποιο νήμα λάβει δεδομένα θα τα αποστείλει στην άλλη υποδοχή πετυχαίνοντας έτσι την επιθυμητή μεταβίβαση. Σε αυτό το σημείο είναι σημαντικό να αναφερθεί πως στην περίπτωση που μόνο η μια πλευρά έχει συνδεθεί με τον Server τότε τα δεδομένα που πιθανώς λαμβάνονται από αυτή τη σύνδεση αγνοούνται καθώς δεν υπάρχει παραλήπτης για να μεταβιβαστούν.



Σχήμα 5.2.1 Κλάσεις και Νήματα

5.2.2 Το γραφικό περιβάλλον της εφαρμογής Server

Το γραφικό περιβάλλον της εφαρμογής αναπτύχθηκε με τρόπο τέτοιο ώστε να προσφέρει έναν φιλικό και κατανοητό τρόπο αλληλεπίδρασης με τον διαχειριστή. Ο διαχειριστής είναι σε θέση να ξεκινάει ή να σταματάει τον Server, να ρυθμίζει τις θύρες των listeners, να παρατηρεί τις ενέργειες του server, του FPGA αλλά και του χρήστη και να ενημερώνεται σχετικά με τα δεδομένα που μεταβιβάζονται.

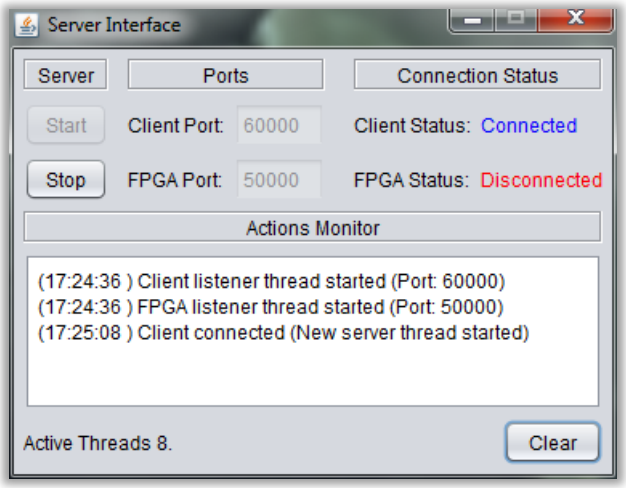
Το γραφικό περιβάλλον συνθέτουν τα κουμπιά (buttons) **Start**, **Stop** και **Clear**, δυο πεδία εισαγωγής κειμένου (textfields), ένα πεδίο προβολής κειμένου (textarea) και ένας αριθμός, σε κάποιες περιπτώσεις δυναμικών, ετικετών.

Στην περιοχή "Server", το κουμπί **Start** είναι αυτό που εκκινεί τον server. Με το πάτημα του δημιουργούνται τα δυο νήματα που αναφέρθηκαν στο υποκεφάλαιο 5.2.1 και τα οποία εκτελούν τους listeners. Το κουμπί **Stop** σταματάει την εκτέλεση του server. Με το πάτημα του καταστρέφονται τα δυο νήματα των listeners αλλά και τα νήματα που εξυπηρετούν τις ενεργές συνδέσεις εφόσον έχει δημιουργηθεί κάποια σύνδεση. Το κουμπί **Clear** χρησιμοποιείται για να καθαρίσει το πεδίο προβολής κειμένου.

Στην περιοχή "Ports", τα δυο πεδία εισαγωγής κειμένου είναι τα σημεία στα οποία θα εισάγει ο διαχειριστής τις επιθυμητές θύρες στις οποίες θα αναμένουν οι δυο listeners για νέες συνδέσεις. Η ετικέτα "Client Port" υποδεικνύει πως το πρώτο πεδίο εισαγωγής κειμένου αφορά συνδέσεις από την εφαρμογή στο τερματικό του χρήστη ενώ η ετικέτα "FPGA Port" υποδεικνύει πως το δεύτερο πεδίο εισαγωγής κειμένου αφορά συνδέσεις από το ενσωματωμένο σύστημα.

Στην περιοχή "Connection Status", βρίσκονται δυο δυναμικές ετικέτες που άλλοτε προβάλλουν το μήνυμα "Disconnected" και άλλοτε το μήνυμα "Connected" κάτι που εξαρτάται από το αν υπάρχουν ενεργές συνδέσεις. Η πρώτη δυναμική ετικέτα αφορά την κατάσταση σύνδεσης με το τερματικό του χρήστη όπως υποδεικνύει και η ετικέτα "Client Status". Η δεύτερη δυναμική ετικέτα, όπως υποδεικνύει και η ετικέτα "FPGA Status", αφορά συνδέσεις μεταξύ του Server και του ενσωματωμένου συστήματος.

Στην περιοχή "Actions Monitor" προβάλλονται πληροφορίες προς ενημέρωση του διαχειριστή. Τα μηνύματα που προβάλλονται ενημερώνουν σχετικά με την δημιουργία νέων νημάτων, τις συνδέσεις που δημιουργούνται ή διακόπτονται αλλά και τα δεδομένα που μεταβιβάζονται. Στο χαμηλότερο σημείο της ίδιας περιοχής βρίσκεται μια ακόμα δυναμική ετικέτα που προβάλλει συνεχώς τα ενεργά νήματα και ανανεώνεται κάθε ένα δευτερόλεπτο. Κατά την εκκίνηση της εφαρμογής εκτελούνται στο παρασκήνιο πέντε νήματα. Αυτός ο αριθμός αυξομειώνεται ανάλογα με το αν δημιουργούνται ή καταστρέφονται νήματα που εκτελούν listeners ή νήματα που εξυπηρετούν νέες συνδέσεις.



Εικόνα 5.2.1 Το γραφικό περιβάλλον της εφαρμογής Server

5.2.3 Ανάλυση του κώδικα στον Server

Σε αυτό το υποκεφάλαιο θα αναλυθεί ο κώδικας που εκτελείται στα παρασκήνια της εφαρμογής του server. Ακολουθώντας την ροή εκτέλεσης του κώδικα, από της εκκίνηση της εφαρμογής και καθόλη τη διάρκεια εξέλιξης της, θα παρουσιαστούν τα σημαντικά στοιχεία του είτε πρόκειται για κλάσεις, μεθόδους, απλές εντολές ή μεταβλητές. Σε αρκετές περιπτώσεις θα δοθεί ιδιαίτερη σημασία στο ρόλο που επιτελούν αυτά τα στοιχεία, στο πως δημιουργούνται και λειτουργούν αλλά και στο πως συνδέονται μεταξύ τους.

Η κύρια κλάση, στην εφαρμογή που αναλύεται, είναι η **server_interface** η οποία εμφωλεύει όλες τις κλάσεις, μεθόδους και μεταβλητές της εφαρμογής. Οι σημαντικότερες οντότητες στις οποίες θα γίνει αναφορά είναι η **μέθοδος κατασκευαστής server_interface** που προφανώς έχει το ίδιο όνομα με την κύρια κλάση **server_interface**, η κλάση **check_active_threads**, η μέθοδος **thread_creator**, η κλάση **listener_thread** και η κλάση **server_thread**. Επίσης θα γίνει αναφορά στη μέθοδο **start_buttonActionPerformed**, στη μέθοδο **stop_buttonActionPerformed** και στη μέθοδο **clear_buttonActionPerformed** που αποτελούν χειριστές γεγονότων και καλούνται οποτεδήποτε ο διαχειριστής αλληλεπιδράσει με τα κουμπιά.

5.2.3.1 Η μέθοδος main

Όπως συμβαίνει και στις περισσότερες περιπτώσεις στον προγραμματισμό, σημείο εκκίνησης της εφαρμογής αποτελεί η κύρια μέθοδος **main** που βρίσκεται εσωτερικά της κύριας κλάσης **server_interface.** Η μέθοδος **main**, αρχικά, δημιουργεί ένα αντίγραφο της κλάσης **server_interface** (εικόνα 5.2.2 γραμμή 558)**.**

Το αντίγραφο αυτό, το οποίο αποτελεί και το πρώτο νήμα, κατά τη δημιουργία του θα εμφανίσει το παραθυρικό περιβάλλον της εφαρμογής και θα εκτελέσει αυτόματα, ως απαράβατος κανόνας της java, τον κώδικα που βρίσκεται εσωτερικά της μεθόδου κατασκευαστή **server_interface**.

```
557        public void run() {
558            new server_interface().setVisible(true);
```

Εικόνα 5.2.2 Κύρια μέθοδος main

5.2.3.2 Η μέθοδος κατασκευαστής server_interface

Ο κώδικας που θα εκτελεστεί σε αυτή τη μέθοδο, στο πρώτο βήμα, αρχικοποιεί το παραθυρικό περιβάλλον και τα στοιχεία που το συνθέτουν (εικόνα 5.2.3 γραμμή 42). Στη συνέχεια, με χρήση της εντολής **setLocationRelativeTo** , τοποθετεί το παράθυρο στο κέντρο της οθόνης (εικόνα 5.2.3 γραμμή 43). Σε επόμενο βήμα μηδενίζεται η μεταβλητή **actions_monitor_text** που συγκρατεί το κείμενο που εμφανίζεται στην περιοχή "Actions Monitor" (εικόνα 5.2.3 γραμμή 44). Στις δυο εντολές που ακολουθούν ορίζεται το κείμενο των δυο πεδίων εισαγωγής κειμένου να είναι αυτό που ορίζουν οι μεταβλητές **client_port** και **fpga_port** (εικόνα 5.2.3 γραμμές 45 και 46). Οι δυο αυτές μεταβλητές περιέχουν τις θύρες στις οποίες συνδέονται ο χρήστης και το ενσωματωμένο σύστημα αντίστοιχα και ως αποτέλεσμα τα δυο προαναφερθέντα πεδία κατά την εμφάνιση του παραθύρου θα προβάλλουν αυτές τις θύρες. Τέλος δημιουργείται ένας μετρητής με όνομα **timer** (εικόνα 5.2.3 γραμμή 47), που στην επόμενη εντολή (εικόνα 5.2.3 γραμμή 48) ρυθμίζεται να καλεί την κλάση **check_active_threads** κάθε ένα δευτερόλεπτο καλώντας την για πρώτη φορά ένα δευτερόλεπτο αφότου ξεκινήσει ο μετρητής. Ο μετρητής **timer** αποτελεί ένα από τα νήματα που εκτελούνται παράλληλα. Μόλις η μέθοδος κατασκευαστής **server_interface** ολοκληρώσει την εκτέλεση της, η εφαρμογή θα αναμένει για οποιαδήποτε ενέργεια μπορεί να προκαλέσει ο διαχειριστής με το πάτημα ενός κουμπιού.

```
42          initComponents();
43          this.setLocationRelativeTo(null);
44          actions_monitor_text = "";
45          client_port_textfield.setText(client_port);
46          fpga_port_textfield.setText(fpga_port);
47          timer = new Timer();
48          timer.schedule(new check_active_threads(), 1000, 1000);
```

Εικόνα 5.2.3 Η μέθοδος κατασκευαστής server_interface

5.2.3.3 Η κλάση check_active_threads

Η κλάση **check_active_threads** όπως αναφέρεται παραπάνω είναι αυτή που καλείται από τον μετρητή **timer** κάθε ένα δευτερόλεπτο. Σκοπός της είναι να ανανεώνει την ετικέτα που προβάλει τα ενεργά νήματα ώστε να ενημερώνεται ο διαχειριστής για την τρέχουσα κατάσταση της εφαρμογής. Σε κάθε ανανέωση προστίθεται μια τελεία στο τέλος του κειμένου της ετικέτας έως ότου υπάρξουν τρείς τελείες οπότε και αρχίζει πάλι από την μία τελεία.

Σε κάθε κλήση της κλάσης αυτής εκτελούνται τέσσερεις συνθήκες **if** που εξετάζουν την μεταβλητή τύπου ακεραίου **active_threads_counter**. Η μεταβλητή αυτή περιέχει έναν αριθμό σύμφωνα με τον οποίο αποφασίζεται ποιο κείμενο θα τυπωθεί στην ετικέτα ενεργών νημάτων. Ο αριθμός αυτός αρχικά είναι μηδέν και σε κάθε κλήση της κλάσης αυξάνεται κατά ένα. Αν ο αριθμός είναι 0 θα τυπωθεί το κείμενο συνοδευόμενο από μια τελεία, αν ο αριθμός είναι 1 θα τυπωθεί το κείμενο με δυο τελείες ενώ αν ο αριθμός είναι 2 θα τυπωθεί το κείμενο με τρεις τελείες. Ο αριθμός της μεταβλητής όμως θα συνεχίζει να αυξάνεται και αυτό μπορεί να δημιουργήσει πρόβλημα αφού μόλις πραγματοποιηθεί η τρίτη επανάληψη είναι επιθυμητό να εμφανιστεί πάλι το κείμενο συνοδευόμενο από μια τελεία. Συνεπώς, για να αποφευχθεί οποιοδήποτε πρόβλημα, η πρώτη συνθήκη **if** (εικόνα 5.2.4 γραμμή 61) εξετάζει σε κάθε κλήση της κλάσης αν η μεταβλητή **active_threads_counter** έχει τιμή 3 κάτι που σημαίνει ότι έχει ήδη εμφανιστεί το κείμενο με τρείς τελείες οπότε πρέπει να ξεκινήσει η επανάληψη από την αρχή. Για αυτό τον λόγο αν η πρώτη συνθήκη βρεθεί αληθής μηδενίζεται η μεταβλητή **active_threads_counter** (εικόνα 5.2.4 γραμμή 62).

Η δεύτερη συνθήκη **if** (εικόνα 5.2.4 γραμμή 65) ελέγχει αν η μεταβλητή **active_threads_counter** έχει τιμή 0. Αν βρεθεί αληθής αρχικά αποθηκεύεται το μήνυμα, που πρόκειται να τυπωθεί, στην μεταβλητή τύπου String **active_threads** (εικόνα 5.2.4 γραμμή 66). Το μήνυμα αυτό αποτελείται από το κείμενο "Active Threads" συνοδευόμενο από τον αριθμό των νημάτων και μια τελεία. Ο αριθμός των νημάτων εξάγεται από την εντολή **Thread.activeCount** (εικόνα 5.2.4 γραμμή 66). Στη συνέχεια καλείται η εντολή **setText** (εικόνα 5.2.4 γραμμή 67) που ορίζει το κείμενο που θα τυπωθεί στην ετικέτα ενεργών νημάτων να είναι αυτό της μεταβλητής **active_threads**.

Η τρίτη συνθήκη **if** (εικόνα 5.2.4 γραμμή 69) εξετάζει αν η μεταβλητή **active_threads_counter** έχει τιμή 1. Σε αυτή την περίπτωση στην μεταβλητή **active_threads** αποθηκεύεται το προηγούμενο κείμενο αλλά αυτή την φορά με δυο τελείες (εικόνα 5.2.4 γραμμή 70). Στη συνέχεια καλείται πάλι η εντολή **setText** (εικόνα 5.2.4 γραμμή 71) που ορίζει το κείμενο που θα τυπωθεί στην ετικέτα ενεργών νημάτων να είναι αυτό της μεταβλητής **active_threads.**

Η τέταρτη συνθήκη **if** (εικόνα 5.2.4 γραμμή 73) εξετάζει αυτή τη φορά αν η μεταβλητή **active_threads_counter** έχει τιμή 2. Αν βρεθεί αληθής θα ακολουθηθεί η ίδια διαδικασία με τις προηγούμενες περιπτώσεις με το κείμενο τώρα να συνοδεύεται από τρείς τελείες (εικόνα 5.2.4 γραμμή 74). Έπειτα θα κληθεί η εντολή **setText** (εικόνα 5.2.4 γραμμή 75) που ορίζει το κείμενο που θα τυπωθεί στην ετικέτα ενεργών νημάτων να είναι πάλι αυτό της μεταβλητής **active_threads.**

Μόλις ολοκληρωθούν και οι τέσσερεις συνθήκες αυξάνεται η μεταβλητή **active_threads_counter** κατά ένα (εικόνα 5.2.4 γραμμή 77) ώστε την επόμενη φορά που θα κληθεί η κλάση **check_active_threads** να τυπωθεί το κατάλληλο μήνυμα της αλληλουχίας στην ετικέτα ενεργών νημάτων.

```
61  if (active_threads_counter == 3) {
62      active_threads_counter = 0;
63  }
64
65  if (active_threads_counter == 0) {
66      active_threads = "Active Threads " + Thread.activeCount() + ".";
67      active_threads_label.setText(active_threads);
68  }
69  if (active_threads_counter == 1) {
70      active_threads = "Active Threads " + Thread.activeCount() + "..";
71      active_threads_label.setText(active_threads);
72  }
73  if (active_threads_counter == 2) {
74      active_threads = "Active Threads " + Thread.activeCount() + "...";
75      active_threads_label.setText(active_threads);
76  }
77  active_threads_counter++;
```

Εικόνα 5.2.4 Η κλάση check_active_threads

5.2.3.4 Η μέθοδος start_buttonActionPerformed

Η μέθοδος αυτή καλείται οποτεδήποτε προκληθεί γεγονός από το πάτημα του κουμπιού **Start**. Βασικός σκοπός της είναι να εκκινήσει τα δυο νήματα των listeners στις θύρες που έχει καθορίσει ο διαχειριστής.

Αρχικά στην μέθοδο αυτή θα οριστεί η boolean μεταβλητή **server_on** ως **true** (εικόνα 5.2.5 γραμμή 466). Ο ρόλος της μεταβλητής αυτής θα εξηγηθεί στην κλάση **listener_thread**, όπου και χρησιμοποιείται. Στα δυο επόμενα βήματα θα αποθηκευτούν οι θύρες, οι οποίες εξάγονται από τα δυο πεδία εισαγωγής κειμένου, στις αντίστοιχες μεταβλητές **client_port** και **fpga_port** με χρήση της εντολής **getText** (εικόνα 5.2.5 γραμμές 467 και 468). Στη συνέχεια ορίζεται το κουμπί **Start** ως μη ενεργό (εικόνα 5.2.5 γραμμή 469) ενώ το κουμπί **Stop** ως ενεργό (εικόνα 5.2.5 γραμμή 470). Κάτι τέτοιο είναι επιθυμητό καθώς ο διαχειριστής δεν πρέπει να είναι σε θέση να επιλέξει ξανά το κουμπί Start εφόσον το έχει ήδη πατήσει και εκτελούνται τα νήματα των listeners. Από την άλλη το κουμπί **Stop** πρέπει να είναι ενεργό ώστε οποιαδήποτε στιγμή να είναι δυνατή η ακύρωση εκτέλεσης του server. Για τους ίδιους λόγους απενεργοποιούνται και τα δυο πεδία εισαγωγής κειμένου ώστε να μην επιτρέπεται η αλλαγή των θυρών ενόσω βρίσκονται σε λειτουργία τα παραπάνω νήματα (εικόνα 5.2.5 γραμμές 471 και 472). Πριν ολοκληρωθεί η εκτέλεση αυτής της μεθόδου καλείται η μέθοδος **thread_creator** (εικόνα 5.2.5 γραμμή 473) η οποία θα εκκινήσει τα δυο νήματα των listeners.

```
466  server_on = true;
467  client_port = client_port_textfield.getText();
468  fpga_port = fpga_port_textfield.getText();
469  start_button.setEnabled(false);
470  stop_button.setEnabled(true);
471  client_port_textfield.setEnabled(false);
472  fpga_port_textfield.setEnabled(false);
473  thread_creator();
```

Εικόνα 5.2.5 Η μέθοδος start_buttonActionPerformed

5.2.3.5 Η μέθοδος thread_creator

Η μέθοδος **thread_creator** δημιουργεί δυο νέα νήματα κάθε ένα από τα οποία εκτελεί ένα αντίγραφο της κλάσης **listener_thread**. Αποτελεί, στην ουσία, το σημείο εκκίνησης του server ο οποίος αναμένει για αποδοχή νέων συνδέσεων. Για κάθε ένα αντίγραφο της κλάσης **listener_thread** η μέθοδος **thread_creator** δίνει ως παραμέτρους έναν ακέραιο που αντιπροσωπεύει μια TCP θύρα και μια συμβολοσειρά που αντιπροσωπεύει το όνομα του νέου νήματος (εικόνα 5.2.6). Οι θύρες αντλούνται από τις προαναφερθείσες μεταβλητές **client_port** και **fpga_port**.

```
52          new listener_thread(Integer.parseInt(client_port), "Client").start();
53          new listener_thread(Integer.parseInt(fpga_port), "FPGA").start();
```

Εικόνα 5.2.6 Η μέθοδος thread_creator

5.2.3.6 Η κλάση listener_thread

Η κλάση **listener_thread** αποτελεί ένα τμήμα κώδικα που εκτελεί έναν listener ο οποίος αναμένει καθόλη τη διάρκεια λειτουργίας της εφαρμογής για αποδοχή νέων TCP συνδέσεων. Όπως έχει προαναφερθεί η μέθοδος **thread_creator** δημιουργεί δυο αντίγραφα αυτής της κλάσης. Κατά την εκκίνηση εκτέλεσης του κάθε αντίγραφου αποθηκεύεται η θύρα και το όνομα του νέου νήματος, που έχουν προέλθει ως ορίσματα από την κλάση **thread_creator**, στις τοπικές μεταβλητές **serverPort** και **thread_name** (εικόνα 5.2.7 γραμμές 90 και 91).

```
90    this.serverPort = serverPort;
91    this.thread_name = thread_name;
```

Εικόνα 5.2.7 Αποθήκευση των ορισμάτων στις μεταβλητές serverPort και thread_name

Παρακάτω δημιουργείται μια νέα υποδοχή (socket) που ταυτίζεται με την θύρα της μεταβλητής **serverPort**. Η υποδοχή αυτή θα αποθηκευτεί σε μια εξωτερική μεταβλητή τύπου **ServerSocket**. Καθώς, όμως, η κλάση **listener_thread** αναμένει συνδέσεις τόσο από το τερματικό του χρήστη όσο και από το FPGA έχουν δημιουργηθεί δυο εξωτερικές μεταβλητές τύπου **ServerSocket** με όνομα **client_listener_socket** και **fpga_listener_socket.** Για να αποφασιστεί τελικά σε ποια από αυτές τις δυο θα αποθηκευτεί η νέα υποδοχή εκτελούνται δυο συνθήκες **if** που εξετάζουν το όνομα του νήματος που ορίζει η μεταβλητή **thread_name.**

Η πρώτη συνθήκη **if** (εικόνα 5.2.8 γραμμή 98) εξετάζει αν η **thread_name** περιέχει το όνομα "Client". Σε αυτή την περίπτωση η νέα υποδοχή θα αποθηκευτεί στη μεταβλητή **client_listener_socket** (εικόνα 5.2.8 γραμμή 99). Η δεύτερη συνθήκη **if** (εικόνα 5.2.8 γραμμή 101) εξετάζει αν η **thread_name** περιέχει το όνομα "FPGA" που θα έχει ως αποτέλεσμα η νέα υποδοχή να αποθηκευτεί στη μεταβλητή **fpga_listener_socket** (εικόνα 5.2.8 γραμμή 102). Σε αυτό το σημείο έχει δημιουργηθεί και αποθηκευτεί μια νέα υποδοχή.

```
98    if (thread_name.equals("Client")) {
99    client_listener_socket = new ServerSocket(serverPort);
100   }
101   if (thread_name.equals("FPGA")) {
102   fpga_listener_socket = new ServerSocket(serverPort);
103   }
```

Εικόνα 5.2.8 Δημιουργία των υποδοχών

Παρακάτω ξεκινάει να επαναλαμβάνεται ένας βρόγχος **while** (εικόνα 5.29 γραμμή 118) του οποίου ο κώδικας είναι το σημείο που αναμένονται οι νέες συνδέσεις. Ο βρόγχος αυτός θα επαναλαμβάνεται όσο η μεταβλητή **server_on** έχει τιμή **true**. Σκοπός είναι να σταματήσει ο βρόγχος, επομένως και η αναμονή για νέες συνδέσεις, όταν η προηγούμενη μεταβλητή αποκτήσει τιμή **false** κάτι που μπορεί να γίνει μόνο με την κλήση της μεθόδου **stop_buttonActionPerformed**. Αρχικά μέσα στον βρόγχο δημιουργείται μια νέα κενή υποδοχή με όνομα socket (εικόνα 5.2.9 γραμμή 119). Στο επόμενο βήμα πρόκειται να κληθεί η εντολή **accept** που θα δεχτεί τη νέα σύνδεση. Προκειμένου να αποφασιστεί για ποια υποδοχή, μεταξύ των **client_listener_socket** και **fpga_listener_socket**, θα γίνει η αποδοχή της σύνδεσης εκτελούνται και εδώ δυο συνθήκες **if** που εξετάζουν την μεταβλητή **thread_name** για το όνομα του νήματος.

Η πρώτη συνθήκη (εικόνα 5.29 γραμμή 121) εξετάζει αν η μεταβλητή **thread_name** περιέχει το όνομα "Client". Σε αυτή την περίπτωση η εντολή **accept** θα αναμένει σύνδεση στην υποδοχή **client_listener_socket**. Μόλις υπάρξει νέα σύνδεση η **client_listener_socket** θα εκχωρηθεί στη μεταβλητή **socket** (εικόνα 5.2.9 γραμμή 122). Η δεύτερη συνθήκη **if** (εικόνα 5.2.9 γραμμή 124) ελέγχει αν η μεταβλητή **thread_name** περιέχει το όνομα "FPGA". Αν βρεθεί αληθής θα ακολουθηθεί η ίδια διαδικασία με την διαφορά ότι μόλις υπάρξει νέα σύνδεση θα εκχωρηθεί η **fpga_listener_socket** στη μεταβλητή **socket** (εικόνα 5.2.9 γραμμή 125). Μόλις έχει υπάρξει νέα σύνδεση εκτελείται μια τρίτη συνθήκη **if** (εικόνα 5.2.9 γραμμή 128) που, αφού επαληθεύσει πως η μεταβλητή **server_on** έχει τιμή **true**, δημιουργεί και εκκινεί ένα αντίγραφο της κλάσης **server_thread** (εικόνα 5.2.9 γραμμή 129). Το αντίγραφο αυτό θα λάβει ως ορίσματα την υποδοχή **socket** που πρόκειται να εξυπηρετήσει και το όνομα του νήματος.

Με την προϋπόθεση πως η μεταβλητή **server_on** συνεχίζει να έχει τιμή **true**, ο βρόγχος **while** θα επαναληφθεί και θα συνεχίσει να αναμένει για νέα σύνδεση.

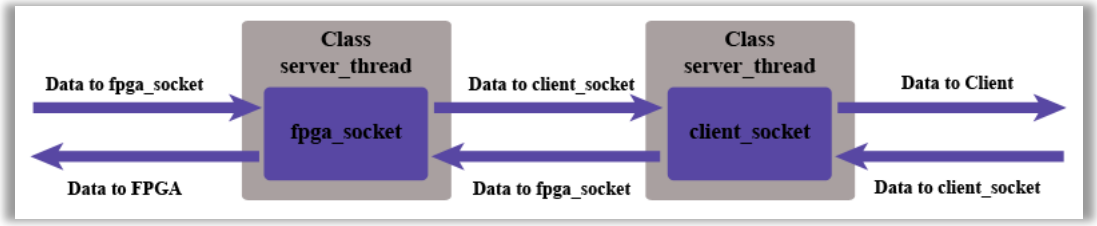
```
118  while (server_on == true) {
119  Socket socket = null;
120  try {
121      if (thread_name.equals("Client")) {
122      socket = client_listener_socket.accept();
123      }
124      if (thread_name.equals("FPGA")) {
125      socket = fpga_listener_socket.accept();
126      }
127
128      if (server_on == true) {
129      new server_thread(socket, thread_name).start();
130      }
```

Εικόνα 5.2.9 Αποδοχή νέας σύνδεσης

5.2.3.7 Η κλάση server_thread

Η κλάση **server_thread** καλείται για να εξυπηρετήσει την νέα TCP σύνδεση. Καθόλη τη διάρκεια εκτέλεσής της αναμένει για αποδοχή εισερχόμενων δεδομένων τα οποία προωθεί στιγμιαία στον παραλήπτη. Σε αυτό το σημείο θα πρέπει να αναφερθεί ότι από τις δυο κλάσεις **listener_thread** η μία αναμένει αιτήσεις σύνδεσης αποκλειστικά από το FPGA ενώ η άλλη από την εφαρμογή στο τερματικό του απομακρυσμένου χρήστη. Επομένως στην περίπτωση που και οι δυο κλάσεις **listener_thread** αποδεχτούν νέες συνδέσεις θα δημιουργήσουν δυο νέες κλάσεις **server_thread** όπου η μία θα εξυπηρετεί το FPGA και η άλλη τον απομακρυσμένο χρήστη.

Όταν κάποιο από τα αντίγραφα **server_thread** λάβει στην υποδοχή του εισερχόμενα δεδομένα τα προωθεί στην υποδοχή του άλλου αντίγραφου το οποίο φροντίζει να τα στείλει στο υπολογιστικό σύστημα (FPGA ή χρήστης) που βρίσκεται στην άλλη άκρη της σύνδεσης που εξυπηρετεί (Σχήμα 5.2.2). Για να γίνει όμως αυτό κάθε αντίγραφο **server_thread** πρέπει να γνωρίζει την υποδοχή του άλλου αλλά και να γνωστοποιεί την δική του υποδοχή. Όπως αναφέρθηκε στην προηγούμενη παράγραφο η κλάση **server_thread** δέχεται ως παραμέτρους το socket της νέας σύνδεσης που θα εξυπηρετήσει και μια συμβολοσειρά με το όνομα του νήματος **listener_thread** που τη δημιούργησε.



Σχήμα 5.2.2 Υποδοχές και μεταβίβαση δεδομένων

Λαμβάνοντας υπόψη τα προαναφερθέντα όταν ξεκινάει να εκτελείται η κλάση **server_thread** αποθηκεύει το πρώτο όρισμα, τύπου Socket, στην τοπική μεταβλητή **socket** (εικόνα 5.2.10 γραμμή 150) και το δεύτερο όρισμα, τύπου String, στην τοπική μεταβλητή **thread_name** (εικόνα 5.2.10 γραμμή 151). Η μεταβλητή **socket** περιέχει την νέα υποδοχή, που δημιουργήθηκε στην κλάση **listener_thread**, η οποία θα εξυπηρετήσει τη νέα σύνδεση ενώ η μεταβλητή **thread_name** περιέχει το όνομα της κλάσης **listener_thread** από την οποία κλήθηκε η κλάση **server_thread**.

```
150   this.socket = socket;
151   this.thread_name = thread_name;
```

Εικόνα 5.2.10 Αποθήκευση των ορισμάτων στις μεταβλητές socket και thread_name

Στη συνέχεια εκτελούνται δυο συνθήκες **if** οι οποίες ελέγχουν το όνομα που περιέχεται στη μεταβλητή **thread_name.**

Η πρώτη συνθήκη **if** (εικόνα 5.2.11 γραμμή 155) ελέγχει αν η προηγούμενη μεταβλητή περιέχει το όνομα "Client". Κάτι τέτοιο σημαίνει πως η κλάση **server_thread** έχει δημιουργηθεί και κληθεί από την κλάση **listener_thread** με όνομα "Client". Αν η συνθήκη αυτή βρεθεί αληθής αρχικά θα αλλάξει το κείμενο της ετικέτας, που προβάλλει την κατάσταση σύνδεσης του χρήστη, σε "Connected" με χρήση της εντολής **setText** (εικόνα 5.2.11 γραμμή 156). Στο επόμενο βήμα το χρώμα του κειμένου αυτού, με χρήση της **setForeground**, θα αλλάξει σε μπλε (εικόνα 5.2.11 γραμμή 157). Με αυτό τον τρόπο ο διαχειριστής της εφαρμογής ενημερώνεται για την ύπαρξη νέας σύνδεσης με το τερματικό του χρήστη. Στη συνέχεια η υποδοχή που βρίσκεται στη τοπική μεταβλητή **socket** θα καταχωρηθεί στην εξωτερική μεταβλητή **client_socket** (εικόνα 5.2.11 γραμμή 158). Έτσι το συγκεκριμένο νήμα έχει ενημερώσει σχετικά με το ποια είναι η υποδοχή του και το πιθανό νήμα που εξυπηρετεί την άλλη πλευρά (δηλαδή το FPGA) μπορεί, γνωρίζοντας πλέον την προαναφερθείσα υποδοχή, να μεταβιβάσει δεδομένα προς το τερματικό του χρήστη. Παρακάτω εκτελείται μια εμφωλευμένη **if** (εικόνα 5.2.11 γραμμή 160) η οποία εξετάζει κατά πόσο η μεταβλητή **fpga_socket** είναι κενή. Καθώς η κλάση αυτή εξυπηρετεί την σύνδεση με τον χρήστη θεωρείται δεδομένο πως η υποδοχή της άλλης πλευράς θα βρίσκεται στην μεταβλητή **fpga_socket** η οποία, με την προϋπόθεση πως δεν είναι κενή, θα καταχωρηθεί στην μεταβλητή **otherside_socket** (εικόνα 5.2.11 γραμμή 161) ώστε να γνωρίζει η **server_thread** που να προωθήσει τυχόν εισερχόμενα δεδομένα.

Η δεύτερη συνθήκη **if** λειτουργεί ακριβώς με τον ίδιο τρόπο με την διαφορά ότι εξετάζεται αν η μεταβλητή **thread_name** περιέχει το όνομα "FPGA". Σε αυτή την περίπτωση η κλάση εξυπηρετεί το ενσωματωμένο σύστημα επομένως η τοπική μεταβλητή **socket** αποθηκεύεται στην εξωτερική μεταβλητή **fpga_socket**. Επίσης η υποδοχή της άλλης πλευράς είναι αυτή που βρίσκεται στην εξωτερική μεταβλητή **client_socket** και είναι αυτή που καταχωρείται στην μεταβλητή **otherside_socket.**

```
155   if (thread_name.equals("Client")) {
156       client_status_label.setText("Connected");
157       client_status_label.setForeground(Color.BLUE);
158       client_socket = this.socket;
159
160       if (fpga_socket != null) {
161           otherside_socket = fpga_socket;
```

Εικόνα 5.2.11 Γνωστοποίηση υποδοχών

Αφού πλέον έχει υπάρξει ενημέρωση σχετικά με τις υποδοχές ακολουθεί ένα βρόγχος **while** του οποίου ο κώδικας είναι υπεύθυνος για την λήψη και την προώθηση των δεδομένων (εικόνα 5.2.12 γραμμή 183). Ο βρόγχος αυτός θα επαναλαμβάνεται για όσο η εντολή **socket.isClosed** επιστρέφει τιμή **false** (εικόνα 5.2.12 γραμμή 183). Αν η παραπάνω εντολή επιστρέψει τιμή **true** θα τερματιστεί η επανάληψη του βρόγχου και αυτό θα έχει ως αποτέλεσμα να καταστραφεί και το νήμα **server_thread**.Κατά την εκκίνηση του βρόγχου καλείται η εντολή **input.readLine** η οποία μόλις λάβει δεδομένα τα αποθηκεύει στην μεταβλητή, τύπου String, **receivedData** (εικόνα 5.2.12 γραμμή 186). Τα δεδομένα αυτά πρέπει να προωθηθούν στην απέναντι πλευρά.

```
183   while (socket.isClosed() == false) {
184
185   Date date = new Date();
186   String receivedData = input.readLine();
```

Εικόνα 5.2.12 Λήψη δεδομένων μέσα στον βρόγχο while

Ακολουθεί η εκτέλεση δυο συνθηκών **if** πανομοιότυπων με τις δυο που αναφέρθηκαν σε προηγούμενη παράγραφο. Όπως και οι προηγούμενες έτσι και αυτές χρησιμοποιούνται για να εντοπίσουν αν υπάρχει διαθέσιμη υποδοχή στην απέναντι πλευρά πριν προβούν στις απαραίτητες ενέργειες για την προώθηση των δεδομένων. Στη συνέχεια εκτελείται μια ακόμα συνθήκη **if** η οποία αυτή τη φορά εξετάζει αν η υποδοχή της μεταβλητής **otherside_socket** (εικόνα 5.2.13 γραμμή 221), δηλαδή της απέναντι πλευράς, δεν είναι κενή και ταυτόχρονα εξετάζει αν η μεταβλητή **receivedData** περιέχει δεδομένα. Αν επαληθευτούν και οι δύο απαιτήσεις σημαίνει πως υπάρχει συνδεδεμένος παραλήπτης στην απέναντι πλευρά και ταυτόχρονα υπάρχουν δεδομένα προς παράδοση. Συνεπώς καλείται η εντολή **println** (εικόνα 5.2.13 γραμμή 223) που αποστέλλει τα περιεχόμενα της μεταβλητής **receivedData** και έτσι επιτυγχάνεται η επιθυμητή προώθηση των εισερχόμενων δεδομένων.

```
221  if (otherside_socket != null && receivedData != null) {
222      PrintWriter share = new PrintWriter(otherside_socket.getOutputStream(), true);
223      share.println(receivedData);
```

Εικόνα 5.2.13 Προώθηση των δεδομένων με χρήση της εντολής println

Όσον αφορά την κλάση **server_thread**, ως αυτό το σημείο έχουν περιγραφεί οι σημαντικότερες ενέργειες που πραγματοποιούνται για την λήψη και την προώθηση των δεδομένων. Η κλάση αυτή όμως είναι, επί των πλείστων, υπεύθυνη για την ενημέρωση του διαχειριστή σχετικά με πιθανή αποσύνδεση είτε του χρήστη είτε του ενσωματωμένου συστήματος. Αποσύνδεση μπορεί να προκληθεί είτε από απρόσμενο γεγονός, όπως η επανεκκίνηση του modem ή του FPGA, είτε ύστερα από αίτηση της εφαρμογής στο τερματικό του χρήστη που αποστέλλει το μήνυμα "ccr" (Close Connection Request). Άσχετα από τον λόγο που προέκυψε μια αποσύνδεση εκτελείται συγκεκριμένος κώδικας που φροντίζει να ενημερώσει τον διαχειριστή.

Αρχικά σε αυτό το τμήμα κώδικα αποθηκεύεται στην εξωτερική μεταβλητή **actions_monitor_text** το κείμενο που πρόκειται να προβληθεί το οποίο αναφέρει ότι προέκυψε αποσύνδεση και επιπλέον ενημερώνει ποια πλευρά αποσυνδέθηκε (εικόνα 5.2.14 γραμμή 260). Στην συνέχεια εκτελείται η εντολή **setText** (εικόνα 5.2.14 γραμμή 261) που θα προβάλλει το κείμενο της προηγούμενης μεταβλητής στο πεδίο προβολής κειμένου της περιοχής "Actions Monitor". Σε αυτό το σημείο πρόκειται να τροποποιηθεί μια από τις δυο δυναμικές ετικέτες που αφορούν την κατάσταση σύνδεσης. Συνεπώς, ακολουθούν δυο συνθήκες **if** (εικόνα 5.2.14 γραμμές 262 και 266) που εξετάζουν το όνομα που περιέχεται στη μεταβλητή **thread_name**. Σε περίπτωση που περιέχεται το όνομα "Client" θα κληθεί η εντολή **setText** (εικόνα 5.2.14 γραμμή 263) που θα αλλάξει το κείμενο της πρώτης ετικέτας σε "Disconnected" ενώ στο επόμενο βήμα η εντολή **setForeground** (εικόνα 5.2.14 γραμμή 264) θα αλλάξει το χρώμα του κειμένου της ετικέτας αυτής σε κόκκινο. Τα αντίστοιχα θα εκτελεστούν αν η μεταβλητή **thread_name** περιέχει το όνομα "FPGA" με την διαφορά ότι θα τροποποιηθεί η δεύτερη δυναμική ετικέτα.

```
260      actions_monitor_text = actions_monitor_text + "(" + dat
261      actions_monitor_textarea.setText(actions_monitor_text);
262      if (thread_name.equals("Client")) {
263          client_status_label.setText("Disconnected");
264          client_status_label.setForeground(Color.RED);
265      }
266      if (thread_name.equals("FPGA")) {
267          fpga_status_label.setText("Disconnected");
268          fpga_status_label.setForeground(Color.RED);
269      }
```

Εικόνα 5.2.14 Ενημέρωση του διαχειριστή για αποσύνδεση

5.2.3.8 Η μέθοδος stop_buttonActionPerformed

Η **stop_buttonActionPerformed** εκτελείται υστέρα από γεγονός που θα προκληθεί με το πάτημα του κουμπιού **Stop**. Σκοπός της είναι να τερματίσει τον server επομένως εκτελεί τις απαραίτητες ενέργειες που θα καταστρέψουν τα νήματα των listeners αλλά και τα νήματα που εξυπηρετούν τις πιθανές ενεργές συνδέσεις.

Για να τερματιστούν τα δυο νήματα που εκτελούν τις κλάσεις **listener_thread** πρέπει να σταματήσει να επαναλαμβάνεται ο βρόγχος while. Έτσι στο πρώτο βήμα εκτέλεσης αυτής της μεθόδου αλλάζει η τιμή της μεταβλητής **server_on** σε false (εικόνα 5.2.15 γραμμή 479) με αποτέλεσμα να σταματήσει η επανάληψη των βρόγχων. Παρόλα αυτά δεν σταματάει η εκτέλεση των νημάτων, όπως αναμενόταν, καθώς η εντολή **accept** εσωτερικά των βρόγχων συνεχίζει να αναμένει για νέες συνδέσεις. Συνεπώς, για να καταστραφούν τελικά τα δυο νήματα πρέπει να σταματήσει η εκτέλεση της εντολής **accept.**

Έτσι στα δυο επόμενα βήματα γίνεται προσπάθεια αποθήκευσης νέων υποδοχών στις εξωτερικές μεταβλητές **client_listener_socket** και **fpga_listener_socket** (εικόνα 5.2.15 γραμμές 482 και 484) κάτι που προκαλεί σφάλμα και δημιουργείται μια εξαίρεση (exception) που τερματίζει την εντολή **accept** και προφανώς καταστρέφει τα δυο νήματα.

Στη συνέχεια θα τερματιστούν τα νήματα **server_thread** που εξυπηρετούν τις ενεργές συνδέσεις, εφόσον υπάρχουν τέτοιες. Για να γίνει αυτό θα πρέπει να κλείσουν οι υποδοχές των συνδέσεων, κάτι που θα τερματίσει τους βρόγχους **while** εσωτερικά των κλάσεων **server_thread** και κατά συνέπεια τις ίδιες τις κλάσεις. Συνεπώς, πρώτα εκτελείται μια συνθήκη **if** (εικόνα 5.2.15 γραμμή 491) που επαληθεύεται αν η υποδοχή της μεταβλητής **client_socket**, δηλαδή η υποδοχή του τερματικού του χρήστη, δεν είναι κενή και εφόσον ισχύει κάτι τέτοιο καλείται η εντολή **close** (εικόνα 5.2.15 γραμμή 492) που κλείνει την συγκεκριμένη υποδοχή και κατά συνέπεια τερματίζει την σύνδεση. Ακολουθεί μια δεύτερη **if** (εικόνα 5.2.15 γραμμή 494) η οποία με τον ίδιο τρόπο εξετάζει και κλείνει (εικόνα 5.2.15 γραμμή 495) την υποδοχή **fpga_socket**, δηλαδή την υποδοχή του ενσωματωμένου συστήματος.

```
479   server_on = false;
480
481   try {
482       new Socket(client_listener_socket.getInetAddress(),
483               client_listener_socket.getLocalPort()).close();
484       new Socket(fpga_listener_socket.getInetAddress(),
485               fpga_listener_socket.getLocalPort()).close();
486       } catch (IOException e) {
              e.printStackTrace();
488       }
489
490   try {
491       if (client_socket != null) {
492          client_socket.close();
493       }
494       if (fpga_socket != null) {
495          fpga_socket.close();
496       }
497       } catch (IOException e) {
              e.printStackTrace();
499       }
```

Εικόνα 5.2.15 Τερματισμός των νημάτων

Πριν ολοκληρωθεί η εκτέλεση αυτής της μεθόδου απενεργοποιείται το κουμπί **Stop** (εικόνα 5.2.16 γραμμή 509) ενώ ενεργοποιείται το κουμπί **Start** (εικόνα 5.2.16 γραμμή 510). Με αυτό τον τρόπο ο διαχειριστής είναι σε θέση, πατώντας το κουμπί **Start**, να εκκινήσει εκ νέου τον server ενώ ταυτόχρονα αποφεύγεται το πάτημα του κουμπιού **Stop** αφού ο server είναι ήδη τερματισμένος. Στη συνέχεια ενεργοποιούνται τα δυο πεδία εισαγωγής κειμένου (εικόνα 5.2.16 γραμμές 511 και 512) ώστε να μπορούν να τροποποιηθούν οι θύρες πριν γίνει νέα απόπειρα για εκκίνηση του server.

```
509   stop_button.setEnabled(false);
510   start_button.setEnabled(true);
511   client_port_textfield.setEnabled(true);
512   fpga_port_textfield.setEnabled(true);
```

Εικόνα 5.2.16 Ενεργοποίηση/Απενεργοποίηση στοιχείων του γραφικού περιβάλλοντος

5.2.3.9 Η μέθοδος clear_buttonActionPerformed

Η μέθοδος αυτή καλείται με το πάτημα του κουμπιού **Clear**. Σκοπός της είναι να καθαρίσει το πεδίο προβολής κειμένου από το κείμενο που έχει ήδη τυπωθεί. Στο πρώτο βήμα εκτέλεσης του κώδικα θα μηδενιστούν τα περιεχόμενα της μεταβλητής **actions_monitor_text** (εικόνα 5.2.17 γραμμή 513) η οποία συγκρατεί το κείμενο που πρόκειται να τυπωθεί. Στη συνέχεια καλείται η εντολή **setText** (εικόνα 5.2.17 γραμμή 514) που ορίζει το κείμενο που θα προβληθεί να είναι αυτό της πιο πάνω μεταβλητής και καθώς αυτή έχει μηδενιστεί θα έχει ως αποτέλεσμα τον καθαρισμό του πεδίου από το κείμενο.

```
513           actions_monitor_text = "";
514           actions_monitor_textarea.setText(actions_monitor_text);
```

Εικόνα 5.2.17 Καθαρισμός του πεδίου προβολής κειμένου στην περιοχή "Actions Monitor"

5.3 Λογισμικό στον χρήστη (Client)

Όπως και η εφαρμογή που εκτελείται στον Server έτσι και η εφαρμογή του χρήστη αναπτύχθηκε σε γλώσσα java στην πλατφόρμα Netbeans 7.1. Σχεδιάστηκε να εκτελείται στο τερματικό του χρήστη και αποτελείται από ένα διαδραστικό γραφικό περιβάλλον, που κρύβει τις πολυπλοκότητες και δίνει την δυνατότητα πρόσβασης σε απομακρυσμένα ψηφιακά συστήματα.

Σε αυτό το υποκεφάλαιο αρχικά θα γίνει μια γενική αναφορά στο λογισμικό που αναπτύχθηκε για τις ανάγκες ενός επιτυχημένου συστήματος από τη μεριά του χρήστη. Στη συνέχεια θα αναλυθεί ο κώδικας που εκτελείται στα παρασκήνια σύμφωνα με την διάκριση τριών επιπέδων.

5.3.1 Γενική αναφορά στο λογισμικό του χρήστη

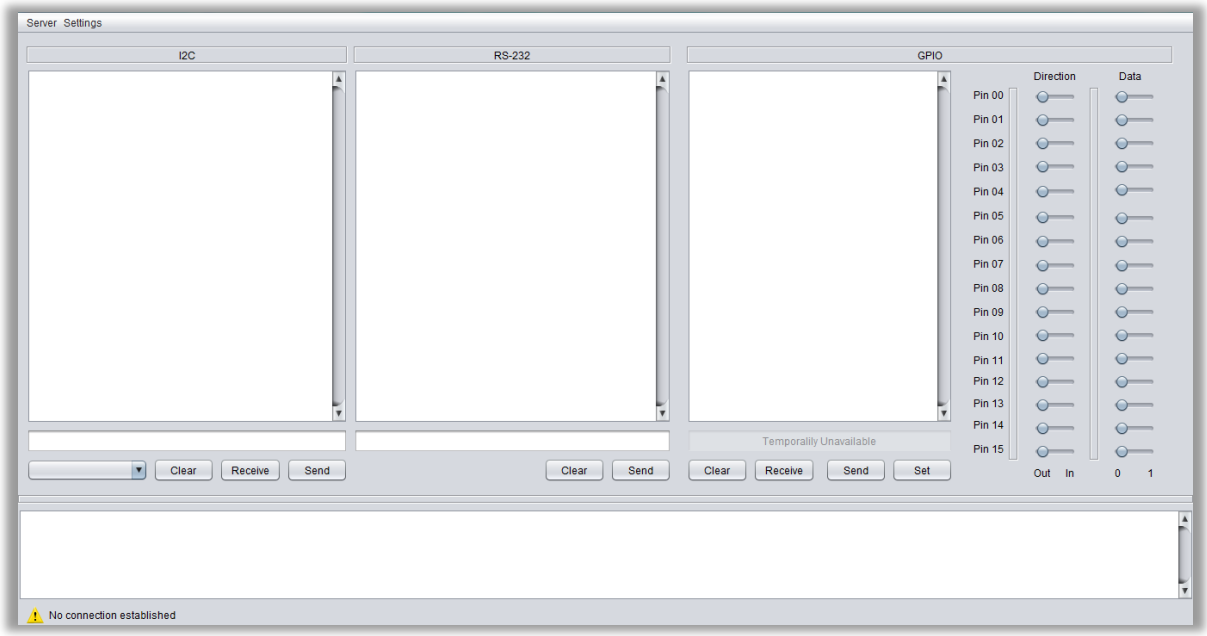
Όπως αναφέρεται παραπάνω η εφαρμογή που περιγράφεται αποτελεί το μέσο που επιτρέπει στον χρήστη, μέσα από ένα διαδραστικό γραφικό περιβάλλον (εικόνα 5.3.1), να αποκτήσει πρόσβαση σε απομακρυσμένα ψηφιακά συστήματα.

Ο χρήστης μέσα από την εφαρμογή είναι σε θέση να αλληλεπιδράσει με ψηφιακές συσκευές που μπορεί να είναι σε σύνδεση με το FPGA. Όπως είναι ήδη γνωστό (κεφάλαιο 4) το FPGA μπορεί να επικοινωνήσει με τέτοιες συσκευές εφόσον αυτές χρησιμοποιούν I^2C, RS-232 ή GPIO πρωτόκολλα επικοινωνίας και για αυτό το λόγο η εφαρμογή διαθέτει επιλογές που λαμβάνουν υπόψη τις ιδιαιτερότητες κάθε πρωτοκόλλου και επιτρέπουν την πρόσβαση σε αυτές τις συσκευές.

Κατά την εκκίνηση της εφαρμογής ο χρήστης επιχειρεί να δημιουργήσει TCP σύνδεση με τον Server, ο οποίος λόγω της περιπλοκότητας του διαδικτύου αποτελεί τον μηχανισμό που αναλαμβάνει την μεταβίβαση δεδομένων μεταξύ του χρήστη και του FPGA. Μόλις υπάρξει επιτυχής σύνδεση, ο χρήστης μπορεί πλέον μέσα από το γραφικό περιβάλλον της εφαρμογής να δημιουργήσει και να στείλει δεδομένα. Τα δεδομένα αυτά, που έχουν ως ενδιαφερόμενο παραλήπτη το FPGA, αρχικά θα αποσταλούν στην εφαρμογή του Server η οποία στη συνέχεια θα τα προωθήσει στο FPGA.

Εκτός από τα δεδομένα που αφορούν τις προσαρτημένες συσκευές στο FPGA, η εφαρμογή μπορεί να στείλει δεδομένα που αφορούν αποκλειστικά το FPGA και που μπορούν να τροποποιήσουν τη συμπεριφορά του. Παράδειγμα μπορεί να είναι η ενεργοποίηση ή απενεργοποίηση των interrupts όπως και η αλλαγή των ακροδεκτών του FPGA σε είσοδο ή έξοδο.

Η δημιουργία των δεδομένων, ανεξάρτητα από τη φύση τους, γίνεται ύστερα από την αλληλεπίδραση του χρήστη με την εφαρμογή. Παραδείγματα αλληλεπίδρασης αποτελούν το πάτημα κουμπιών, η εισαγωγή κειμένου, η αλλαγή θέσης των διακοπτών αλλά και η αλλαγή ρυθμίσεων στο μενού.

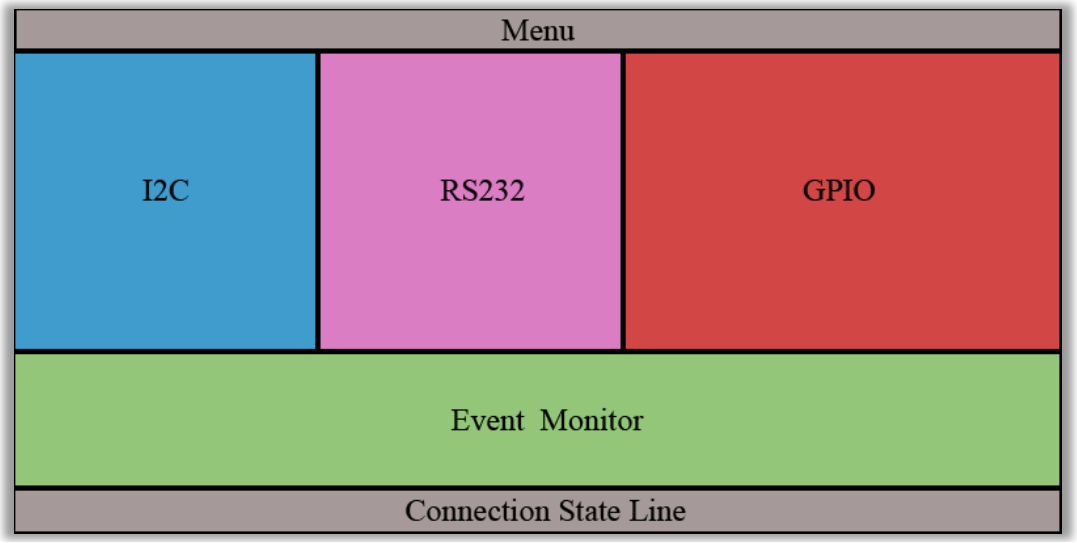


Εικόνα 5.3.1 Το γραφικό περιβάλλον της εφαρμογής

5.3.2 Το γραφικό περιβάλλον της εφαρμογής του χρήστη

Το γραφικό περιβάλλον της εφαρμογής με το οποίο αλληλεπιδράει ο χρήστης αποτελεί μια σύνθεση γραφικών στοιχείων μεταξύ των οποίων είναι κουμπιά (buttons), πεδία εισαγωγής και προβολής κειμένου (textfields και textareas), μενού, ετικέτες (labels) και διακόπτες (sliders). Μπορεί να θεωρηθεί ότι χωρίζεται σε έξι περιοχές με βάση τις ιδιαιτερότητες του συστήματος αλλά και τις ανάγκες διευκόλυνσης του χρήστη.

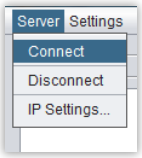
Στο υψηλότερο σημείο έχει εισαχθεί η περιοχή με τη γραμμή μενού της εφαρμογής. Στο κέντρο βρίσκονται τρεις από τις έξι περιοχές. Ξεκινώντας από τα αριστερά προς τα δεξιά πρώτα είναι τοποθετημένη η περιοχή που αφορά την αλληλεπίδραση με το πρωτόκολλο I^2C. Ακολουθεί αυτή που αφορά το πρωτόκολλο RS232 και τέλος βρίσκεται η περιοχή για το GPIO. Κάτω από το κέντρο βρίσκεται η περιοχή παρακολούθησης των γεγονότων της εφαρμογής και στο χαμηλότερο σημείο υπάρχει η γραμμή κατάστασης σύνδεσης.



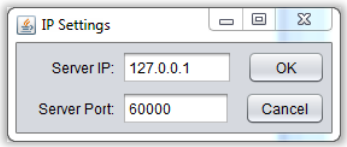
Σχήμα 5.3.1 Περιοχές Γραφικού Περιβάλλοντος

5.3.2.1 Γραμμή μενού

Η γραμμή μενού διαθέτει δυο μενού, τα **Server** και **Settings**. Το μενού **Server** (εικόνα 5.3.2) διαθέτει επιλογές σχετικά με την TCP σύνδεση που πρέπει να δημιουργήσει η εφαρμογή με τον Server ώστε να γίνει εφικτή η αποστολή και λήψη δεδομένων. Ο χρήστης σε αυτό το μενού μπορεί να πραγματοποιήσει σύνδεση ή αποσύνδεση με τον Server αλλά και να τροποποιήσει την IP και την θύρα στην οποία αναμένει ο Server για σύνδεση (εικόνα 5.3.3).

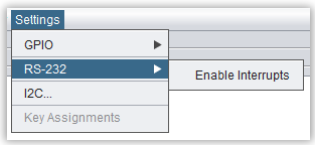


Εικόνα 5.3.2 Μενού Server



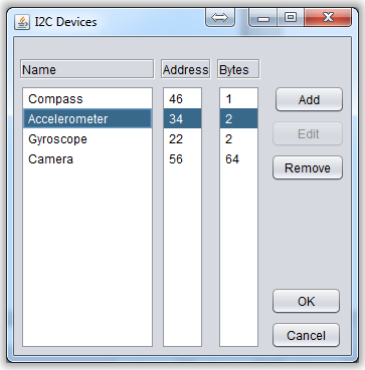
Εικόνα 5.3.3 Τροποποίηση IP και Θύρας

Το μενού **Settings** (εικόνα 5.3.4) διαθέτει επιλογές σχετικές με τα τρία πρωτόκολλα επικοινωνίας I^2C, RS-232 και GPIO. Όσον αφορά τα RS-232 και GPIO δίνεται η δυνατότητα στον χρήστη να ενεργοποιήσει ή όχι τα interrupts που προκαλούν οι ψηφιακές συσκευές στο FPGA. Μια ψηφιακή συσκευή, έστω πως είναι ένας μικροελεγκτής, θα προκαλέσει στο FPGA interrupt στην περίπτωση που πρόκειται να του στείλει δεδομένα. Το FPGA είναι σχεδιασμένο να αποστέλλει αυτά τα δεδομένα αυτόματα στην εφαρμογή του χρήστη, εφόσον ο τελευταίος έχει ενεργοποιήσει τα interrupts.



Εικόνα 5.3.4 Μενού Settings

Λαμβάνοντας υπόψη τις ιδιαιτερότητες του πρωτοκόλλου I^2C (Κεφάλαιο 4) ο χρήστης που θέλει να επικοινωνήσει με συσκευές που χρησιμοποιούν αυτό το πρωτόκολλο πρέπει να γνωρίζει την διεύθυνση της συσκευής όπως επίσης και τον αριθμό των bytes που πρόκειται να λάβει σε περίπτωση που ζητήσει δεδομένα από αυτή. Για το λόγο αυτό το μενού επιλογών **Settings** δίνει τη δυνατότητα προσθήκης τέτοιων συσκευών που χαρακτηρίζονται από μια διεύθυνση και έναν αριθμό bytes (εικόνα 5.3.5). Στη συνέχεια ο χρήστης μπορεί να επιλέξει μέσα από αυτές τις συσκευές και να στείλει ή να λάβει δεδομένα.



Εικόνα 5.3.5 Προσθήκη συσκευών I^2C

5.3.2.2 Περιοχή επικοινωνίας I^2C

Η περιοχή I^2C (εικόνα 5.3.6) δίνει τη δυνατότητα επικοινωνίας με τις συσκευές που είναι προσαρτημένες στο FPGA και χρησιμοποιούν το πρωτόκολλο I^2C. Η περιοχή αυτή διαθέτει ένα πεδίο εισαγωγής κειμένου (textfield), ένα πεδίο για προβολή κειμένου (textarea), τρία κουμπιά (buttons) και ένα κουτί επιλογής (combobox). Στο πεδίο εισαγωγής κειμένου ο χρήστης εισάγει τα δεδομένα σε μορφή κειμένου τα οποία μπορεί να αποστείλει πατώντας το κουμπί **Send**. Λόγω της φύσης του συγκεκριμένου πρωτοκόλλου η λήψη δεδομένων μπορεί να γίνει μόνο κατά απαίτηση του χρήστη πατώντας το κουμπί **Receive**. Το πεδίο προβολής κειμένου χρησιμεύει στο να καταγράφει τα δεδομένα που στέλνονται και λαμβάνονται καθόλη τη διάρκεια λειτουργίας της εφαρμογής και το οποίο μπορεί κάθε στιγμή να καθαριστεί από τα δεδομένα που καταγράφονται πατώντας το κουμπί **Clear**. Τελευταίο αλλά εξίσου σημαντικό είναι το κουτί επιλογής (combobox). Όπως έχει προαναφερθεί, το μενού **Settings** δίνει την δυνατότητα προσθήκης συσκευών μέσα από τις οποίες ο χρήστης θα επιλέξει να επικοινωνήσει. Η επιλογή της ενεργής συσκευής για επικοινωνία γίνεται με το κουτί επιλογής (combobox) το οποίο ενημερώνεται με τις διαθέσιμες συσκευές που προστέθηκαν από τον χρήστη.

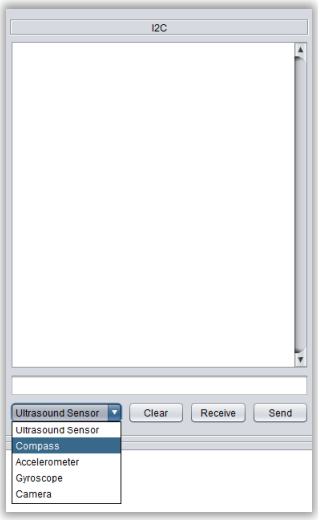
5.3.2.3 Περιοχή επικοινωνίας RS-232

Αντίστοιχα με την περιοχή I^2C η περιοχή RS-232 χρησιμοποιείται για επικοινωνία με τις συσκευές που εφαρμόζουν πρωτόκολλο RS-232. Διαθέτει ένα πεδίο εισαγωγής κειμένου (textfield), ένα πεδίο για προβολή κειμένου (textarea) και δυο κουμπιά. Το ένα κουμπί είναι το **Send** που χρησιμοποιείται για την αποστολή των πιθανών δεδομένων που βρίσκονται στο πεδίο εισαγωγής κειμένου. Το δεύτερο κουμπί είναι το **Clear** που καθαρίζει το πεδίο προβολής κειμένου από τις καταγραφές εισερχόμενων και εξερχόμενων δεδομένων. Χαρακτηριστικό αυτής της περιοχής είναι ότι δεν διαθέτει κουμπί Receive καθώς τα δεδομένα δεν λαμβάνονται κατά απαίτηση του χρήστη. Το FPGA είναι σχεδιασμένο ώστε στη περίπτωση που λάβει δεδομένα από προσαρτημένη συσκευή, που χρησιμοποιεί πρωτόκολλο RS-232, να τα αποστέλλει αυτόματα στην εφαρμογή του χρήστη. Τα δεδομένα μόλις ληφθούν από την εφαρμογή θα τυπωθούν στο πεδίο προβολής κειμένου.

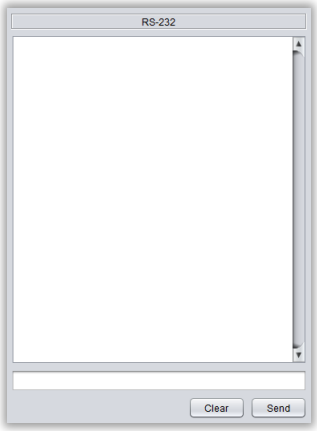
5.3.2.4 Περιοχή επικοινωνίας GPIO

Η περιοχή του GPIO υποστηρίζει την επικοινωνία με συσκευές που συνδέονται στους 16 ακροδέκτες GPIO της αναπτυξιακής πλακέτας. Ένα χαρακτηριστικό του GPIO, όπως αναφέρεται στο κεφάλαιο 4, είναι η δυνατότητα να μετατρέπει τους ακροδέκτες δυναμικά σε είσοδο ή έξοδο. Ένα δεύτερο χαρακτηριστικό , που αφορά τους ακροδέκτες, είναι ότι μπορούν να εναλλάσσουν την κατάστασή τους μεταξύ λογικού μηδέν και λογικού ένα. Λαμβάνοντας υπόψη τα παραπάνω ήταν αναγκαία η χρήση διακοπτών (Sliders) με τους οποίους ο χρήστης μπορεί να θέτει την κατεύθυνση των ακροδεκτών σε είσοδο ή έξοδο αλλά και να εναλλάσσει την κατάσταση του σήματος σε αυτούς μεταξύ λογικού μηδέν και ένα. Συνεπώς η περιοχή του GPIO διαθέτει δυο σειρές 16 διακοπτών όπου η μια διαχειρίζεται την κατεύθυνση των ακροδεκτών και η άλλη τα σήματα σε αυτούς.

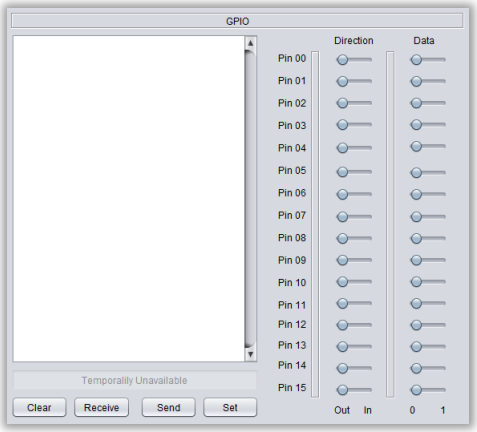
Τα δεδομένα που στέλνονται και λαμβάνονται καταγράφονται, όπως και στις περιοχές I^2C και RS232, σε ένα πεδίο προβολής κειμένου (textarea) το οποίο μπορεί να καθαριστεί με τη χρήση του αντίστοιχου κουμπιού **Clear.** Η περιοχή GPIO διαθέτει τρία επιπλέον κουμπιά. Ένα από αυτά είναι το **Send** το οποίο θα συλλέξει και θα στείλει ως δεδομένα την λογική κατάσταση των σημάτων και των 16 ακροδεκτών όπως έχουν οριστεί από τον χρήστη. Το κουμπί **Set** στέλνει δεδομένα που τροποποιούν την κατεύθυνση των ακροδεκτών στην αναπτυξιακή πλακέτα. Το τελευταίο κουμπί είναι το **Receive** με το οποίο ο χρήστης μπορεί να λάβει στιγμιαία την τρέχουσα κατάσταση των σημάτων στους ακροδέκτες. Σε αυτό το σημείο είναι σημαντικό να αναφερθεί ότι ο χρήστης, ως πρόσθετη δυνατότητα, μπορεί να λαμβάνει αυτόματα την τελευταία κατάσταση των σημάτων, εφόσον υπάρξει κάποια αλλαγή σε αυτά, ενεργοποιώντας τα GPIO interrupts στο μενού **Settings**.



Εικόνα 5.3.6 Η περιοχή I2C



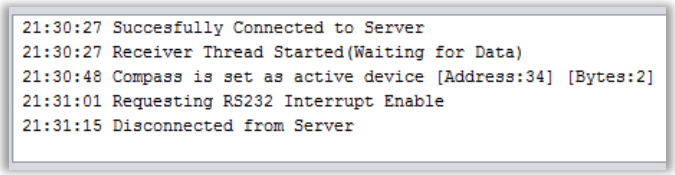
Εικόνα 5.3.7 Η περιοχή RS232



Εικόνα 5.3.8 Η περιοχή GPIO

5.3.2.5 Περιοχή παρακολούθησης γεγονότων (Event Monitor)

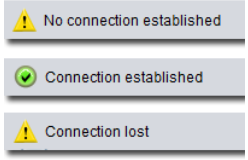
Η περιοχή παρακολούθησης γεγονότων προβάλει σημαντικές πληροφορίες σχετικά με τις ενέργειες του χρήστη και την συμπεριφορά της εφαρμογής. Παράδειγμα αποτελεί η ενημέρωση για την επιτυχία ή αποτυχία εγκαθίδρυσης σύνδεσης με τον Server. Ένα δεύτερο από τα αρκετά παραδείγματα που θα μπορούσαν να αναφερθούν είναι η εμφάνιση των αιτήσεων που κάνει ο χρήστης για ενεργοποίηση ή απενεργοποίηση των interrupts στο FPGA.



Εικόνα 5.3.9 Παράδειγμα μηνυμάτων στην περιοχή παρακολούθησης γεγονότων

5.3.2.6 Γραμμή κατάστασης σύνδεσης

Όπως προδίδει και το όνομά της, χρησιμεύει στο να ενημερώνει το χρήστη σχετικά με την κατάσταση της TCP σύνδεσης που πρέπει να πραγματοποιεί η εφαρμογή με τον Server για την ανταλλαγή δεδομένων. Κατά την εκκίνηση της εφαρμογής η γραμμή κατάστασης σύνδεσης προβάλει το μήνυμα **No connection established**. Με αυτό τον τρόπο ο χρήστης μπορεί έγκαιρα να προβεί στις απαραίτητες ενέργειες ώστε να συνδεθεί με τον Server. Στην περίπτωση που υπάρξει επιτυχής σύνδεση το μήνυμα αλλάζει σε **Connection established** ενώ αν η σύνδεση διακοπεί από τον χρήστη ή από αποτυχία του **Server** προβάλλεται το μήνυμα **Connection Lost**.



Εικόνα 5.3.10 Μηνύματα στη γραμμή κατάστασης σύνδεσης

5.3.3 Ανάλυση του κώδικα (Διάκριση τριών επιπέδων)

Ο κώδικας που κρύβεται πίσω από την εφαρμογή του χρήστη παρουσιάζει αρκετή πολυπλοκότητα καθώς διαχειρίζεται την προβολή του γραφικού περιβάλλοντος, τα γεγονότα (events) που προκαλεί ο χρήστης αλλά και την επικοινωνία με τα απομακρυσμένα ψηφιακά συστήματα και την επεξεργασία των δεδομένων.

Μπορεί να θεωρηθεί ότι διακρίνεται σε τρία επίπεδα. Το πρώτο επίπεδο, που θα μπορούσε να χαρακτηριστεί ως το ανώτερο, περιλαμβάνει όλα τα τμήματα κώδικα που χειρίζονται την προβολή του γραφικού περιβάλλοντος. Το δεύτερο επίπεδο ή ενδιάμεσο διαθέτει τμήματα κώδικα που εξυπηρετούν τα γεγονότα που προκαλεί ο χρήστης κατά την αλληλεπίδραση του με το πρώτο επίπεδο. Το τρίτο επίπεδο και κατώτερο, που συνθέτει τον κώδικα που εκτελείται στο παρασκήνιο, έχει ως ρόλο την επεξεργασία των δεδομένων αλλά και την επικοινωνία με το απομακρυσμένο σύστημα. Το δεύτερο επίπεδο χαρακτηρίζεται ως το ενδιάμεσο καθώς στην ουσία αποτελεί το μέσο σύνδεσης μεταξύ του ανώτερου επιπέδου της διάδρασης του χρήστη και του κατώτερου επιπέδου των εντολών του παρασκηνίου.

Μια πιο απλή εξήγηση όσον αφορά την διάκριση του κώδικα σε τρία επίπεδα είναι η ακόλουθη: ο χρήστης αλληλεπιδράει με το γραφικό περιβάλλον που παράγεται από τον κώδικα του πρώτου επιπέδου και κατά την αλληλεπίδραση του προβαίνει σε ενέργειες όπως το πάτημα κουμπιών και η εισαγωγή κειμένου. Οι ενέργειες αυτές, κατά κύριο λόγο, παραθέτουν σε εκτέλεση εντολών που βρίσκονται στο τρίτο επίπεδο και οι οποίες αφορούν την επεξεργασία δεδομένων ή την διαχείριση της επικοινωνίας της εφαρμογής με το απομακρυσμένο σύστημα. Σε αυτό το σημείο μπορεί να φανεί η ιδιαίτερη σημασία του δεύτερου επιπέδου καθώς την στιγμή που ο χρήστης προβεί σε κάποια ενέργεια, όπως το πάτημα κουμπιού, τα τμήματα κώδικα του δεύτερου επιπέδου (χειριστές γεγονότων ή event handlers) εντοπίζουν το γεγονός που προκλήθηκε και φροντίζουν να το παραθέσουν στην αντίστοιχη εντολή του τρίτου επιπέδου.

Είναι σημαντικό να εξηγηθεί ότι η φυσική τοποθέτηση του κώδικα αυτής της εφαρμογής μπορεί να διαφέρει πολύ από την εικονική διάκριση των τριών επιπέδων. Στην ουσία ο κώδικας αποτελείται από επτά κλάσεις με σημείο εκκίνησης την κλάση **java_interface** η οποία δημιουργεί αντίγραφα των υπολοίπων είτε κατά την εκκίνηση της εφαρμογής είτε ύστερα από ενέργεια του χρήστη.

Η **java_interface**, που είναι και η κύρια κλάση, περιλαμβάνει κατά κύριο λόγο κώδικες για την ανάπτυξη του βασικού γραφικού περιβάλλοντος όπως και τους χειριστές γεγονότων (event handlers) του περιβάλλοντος αυτού. Τέσσερεις ακόμα κλάσεις, οι **server_settings**, **i2c_settings**, **i2c_device_element** και **dialog_box** δημιουργούν νέα γραφικά περιβάλλοντα (αναδυόμενα παράθυρα) που προκύπτουν κατά την αλληλεπίδραση του χρήστη. Αντίστοιχα και αυτές διαθέτουν κώδικες για την δημιουργία των δικών τους γραφικών στοιχείων αλλά και κώδικες για την διαχείριση των δικών τους γεγονότων. Μια ακόμα κλάση είναι η **third_level_code** που, όπως προδίδει το όνομα της, συγκρατεί το μεγαλύτερο ποσοστό του κώδικα που ανήκει στο τρίτο επίπεδο. Τελευταία κλάση είναι η **data_bean** που διαθέτει το μεγαλύτερο μέρος των μεταβλητών της εφαρμογής, με άλλα λόγια τα απαραίτητα δεδομένα που διαχειρίζεται η εφαρμογή.

5.3.3.1 Το πρώτο επίπεδο (Ο κώδικας του γραφικού περιβάλλοντος)

Όπως έχει προαναφερθεί, το πρώτο επίπεδο αφορά αποκλειστικά την παραγωγή του γραφικού περιβάλλοντος. Ο συνδυασμός της ανάγκης για διευκόλυνση και του περιορισμένου χρόνου είχε ως αποτέλεσμα την χρήση των σχεδιαστικών εργαλείων που παρέχει η πλατφόρμα Netbeans για τον **drag and drop** σχεδιασμό του γραφικού περιβάλλοντος. Ως συνέπεια ο κώδικας σε αυτό το επίπεδο έχει δημιουργηθεί αυτοματοποιημένα μέσω της πλατφόρμας Netbeans και χρησιμοποιεί τη βιβλιοθήκη **Swing** της Java.

Είναι ήδη γνωστό ότι πέντε από τις κλάσεις της εφαρμογής δημιουργούν γραφικά περιβάλλοντα πράγμα που σημαίνει ότι και οι πέντε διαθέτουν κώδικα που ανήκει στο πρώτο επίπεδο. Αυτές είναι οι **java_interface**, **server_settings**, **i2c_settings**, **i2c_device_element** και **dialog_box.** Ο κώδικας του πρώτου επιπέδου σε όλες τις κλάσεις έχει δημιουργηθεί αυτοματοποιημένα και πρέπει να τονιστεί ότι δεν θα αναλυθεί καθώς δεν αποτελεί σημείο ενδιαφέροντος. Παρακάτω θα γίνει μια απλή αναφορά στις πέντε αυτές κλάσεις όσον αφορά την λειτουργία τους και τον τρόπο που καλούνται.

5.3.3.1.1 Η κλάση java_interface

Η κλάση **java_interface**, κατά την εκκίνηση της, πέραν των άλλων εκτελεί τις απαραίτητες ρουτίνες του πρώτου επιπέδου προκειμένου να παράγει το βασικό παράθυρο της εφαρμογής που αποτελεί το μέσο αλληλεπίδρασης του χρήστη. Λεπτομέρειες σχετικά με το τι περιλαμβάνει το βασικό γραφικό περιβάλλον αναφέρονται στο υποκεφάλαιο 5.3.2.

5.3.3.1.2 Η κλάση server_settings

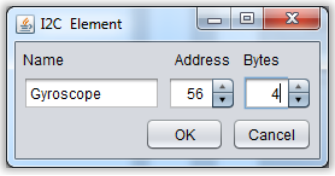
Ο κώδικας πρώτου επιπέδου της κλάσης **server_settings** παράγει ένα αναδυόμενο παράθυρο (εικόνα 5.3.3) που δίνει την δυνατότητα στον χρήστη να ρυθμίσει την IP και την θύρα του server στον οποίο πρόκειται να συνδεθεί η εφαρμογή. Καλείται ύστερα από ενέργεια του χρήστη πάνω στο βασικό γραφικό περιβάλλον πατώντας την επιλογή **IP_Settings** που βρίσκεται στο μενού **Server**. Το παράθυρο αυτό περιλαμβάνει δυο περιοχές εισαγωγής κειμένου (textfield) για την προσθήκη της επιθυμητής IP και θύρας του server. Επίσης περιλαμβάνει ένα κουμπί με όνομα **OK** για επιβεβαίωση των επιλογών και ένα κουμπί **Cancel** για ακύρωση της ενέργειας.

5.3.3.1.3 Η κλάση i2c_settings

Η κλάση **i2c_settings** καλείται από τον χρήστη πατώντας την επιλογή **I2C** στο μενού **Settings** του βασικού παραθύρου. Κατά την κλήση της εμφανίζει ένα αναδυόμενο παράθυρο (εικόνα 5.3.5) που επιτρέπει την προσθήκη συσκευών που χρησιμοποιούν πρωτόκολλο I^2C για επικοινωνία με το FPGA και κατά επέκταση με το χρήστη. Το παράθυρο που δημιουργεί η κλάση **i2c_settings** διαθέτει μεταξύ άλλων τρεις λίστες στις οποίες προβάλλονται οι συσκευές που έχουν προστεθεί. Στη πρώτη λίστα τυπώνεται το όνομα της κάθε συσκευής, στη δεύτερη η διεύθυνση της και στη τρίτη ο αριθμός των bytes τον οποίο πρόκειται να λάβει το FPGA από αυτή τη συσκευή. Επιπλέον το παράθυρο περιλαμβάνει ένα κουμπί με όνομα **Add** για την προσθήκη νέων συσκευών, ένα κουμπί **Remove** για αφαίρεση μιας επιλεγμένης συσκευής από τη λίστα όπως και τα αντίστοιχα κουμπιά **OK** και **Cancel** για επιβεβαίωση και ακύρωση.

5.3.3.1.4 Η κλάση i2c_device_element

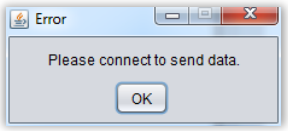
Η κλάση **i2c_device_element** καλείται πατώντας το κουμπί **Add** του παραθύρου της προηγούμενης παραγράφου (εικόνα 5.3.5). Κατά την κλήση της δημιουργεί ένα νέο παράθυρο (εικόνα 5.3.11) με το οποίο ο χρήστης προσθέτει μια νέα συσκευή εισάγοντας τα χαρακτηριστικά της. Το παράθυρο περιλαμβάνει τρεις περιοχές εισαγωγής κειμένου όπου στην πρώτη εισάγεται το όνομα της συσκευής, στη δεύτερη η διεύθυνση της και στην τρίτη ο αριθμός των bytes που μπορούν να ληφθούν. Τέλος προφανώς υπάρχει ένα κουμπί **OK** για επιβεβαίωση της προσθήκης νέας συσκευής και ένα **Cancel** για ακύρωση της.



Εικόνα 5.3.11 Εισαγωγή νέας συσκευής

5.3.3.1.5 Η κλάση dialog_box

Τελευταία αναφέρεται η κλάση **dialog_box** η οποία αναδύει ένα παράθυρο (εικόνα 5.3.12), προειδοποιητικού χαρακτήρα, οποτεδήποτε ο χρήστης επιχειρήσει να στείλει δεδομένα πάνω από μη εγκαθιδρυμένη TCP σύνδεση. Τα μόνα γραφικά στοιχεία σε αυτό το παράθυρο είναι ένα προειδοποιητικό μήνυμα και ένα κουμπί **OK.**



Εικόνα 5.3.12 Προειδοποιητικό μήνυμα

5.3.3.2 Το δεύτερο επίπεδο (Η διαχείριση των γεγονότων)

Το δεύτερο επίπεδο χαίρει ιδιαίτερης σημασίας καθώς όπως έχει ήδη αναφερθεί παραπέμπει τις ενέργειες του χρήστη, κατά την αλληλεπίδραση με το πρώτο επίπεδο, στον αντίστοιχο κώδικα του τρίτου επιπέδου.

Στο δεύτερο επίπεδο ανήκει μια ομάδα μεθόδων (διαχειριστές γεγονότων ή event handlers) οι οποίες ενεργούν οποτεδήποτε ο χρήστης προκαλέσει κάποιο γεγονός (event) με τρόπους όπως το πάτημα κουμπιών (buttons) ή τη μετακίνηση διακοπτών (sliders). Κάθε γραφικό στοιχείο της εφαρμογής το οποίο είναι σε θέση να προκαλέσει γεγονός διαθέτει το δικό του διαχειριστή γεγονότων. Από άποψη φυσικής τοποθέτησης του κώδικα οι διαχειριστές γεγονότων βρίσκονται στις ίδιες κλάσεις που διαθέτουν κώδικα πρώτου επιπέδου. Λαμβάνοντας αυτό υπόψη οι πέντε κλάσεις που αναφέρθηκαν στην προηγούμενη ενότητα περιλαμβάνουν και κώδικα δευτέρου επιπέδου, τους διαχειριστές γεγονότων. Κάθε κλάση φιλοξενεί διαχειριστές γεγονότων μόνο για γραφικά στοιχεία που η ίδια παράγει και που μπορούν να προκαλέσουν γεγονός.

Στην ενότητα αυτή θα αναλυθούν οι διαχειριστές γεγονότων των κλάσεων **java_interface**, **server_settings**, **i2c_settings**, **i2c_device_element**. Θα εξηγηθεί η λειτουργία για κάθε έναν από αυτούς αλλά και ποια γεγονότα γραφικών στοιχείων προκαλούν την εκτέλεση τους.

5.3.3.2.1 Η κλάση java_interface

Το γραφικό περιβάλλον που παράγει αυτή η κλάση διαιρείται σε έξι περιοχές (υποκεφάλαιο 5.3.2) συνεπώς η περιγραφή των διαχειριστών γεγονότων θα γίνει ανά περιοχή σε μια προσπάθεια για μεγαλύτερη κατανόηση.

Στην περιοχή του μενού (σχήμα 5.3.1) τόσο το μενού Server όσο και το Settings περιέχουν στοιχεία που προκαλούν γεγονότα ύστερα από ενέργεια του χρήστη. Στο μενού Server τέτοια στοιχεία είναι τα Connect, Disconnect και IP Settings ενώ στο μενού Settings τα στοιχεία Enable Interrupts του υπό-μενού GPIO, Enable Interrupts του υπό-μενού RS-232 και το στοιχείο I2C. Παρακάτω θα παρουσιαστούν οι διαχειριστές γεγονότων αυτών των στοιχείων.

I. Η μέθοδος server_connectActionPerformed

Η μέθοδος **server_connectActionPerformed** καλείται όταν προκληθεί γεγονός από το στοιχείο Connect του μενού Server. Σκοπός αυτής της μεθόδου είναι η σύνδεση με τον server και έτσι κατά την εκτέλεση της καλεί την μέθοδο **start_client** που εμφωλεύεται στην κλάση **third_level_code** και η οποία επιχειρεί να εγκαθιδρύσει την TCP σύνδεση **(**εικόνα 5.3.13 γραμμή 1606).

```
1605    private void server_connectActionPerformed(java.awt.event.ActionEvent evt) {
1606        third_level_code.start_client();
1607    }
```

Εικόνα 5.3.13 Διαχειριστής γεγονότος του στοιχείου Connect (Μενού Server)

II. Η μέθοδος server_disconnectActionPerformed

Η μέθοδος **server_disconnectActionPerformed** καλείται από το στοιχείο Disconnect του μενού Server και σκοπός της είναι προφανώς η αποσύνδεση από τον Server. Αρχικά αυτή η μέθοδος αποστέλλει τη συμβολοσειρά **«ccr»** στην εφαρμογή του server (εικόνα 5.3.14 γραμμή 1613) ενημερώνοντας τον με αυτό τον τρόπο ότι πρόκειται να αποσυνδεθεί. Στη συνέχεια κλείνει την υποδοχή (socket) που είχε δημιουργηθεί κατά την σύνδεση της εφαρμογής με τον server (εικόνα 5.3.14 γραμμή 1614 έως 1616) και καλεί την μέθοδο **message_printer** (εικόνα 5.3.14 γραμμή 1617) που βρίσκεται στην κλάση **third_level_code** και η οποία φροντίζει να τυπώσει το μήνυμα **«Disconnected from Server»** στην περιοχή παρακολούθησης γεγονότων (Event Monitor). Τέλος αλλάζει την κατάσταση της boolean μεταβλητής **connected_state**, που βρίσκεται στην κλάση **data_bean**, σε false. Η σημασία της μεταβλητής **connected_state** όπως και άλλων μεταβλητών θα εξηγηθεί στην επόμενη ενότητα του τρίτου επιπέδου.

```
1612            PrintWriter output = new PrintWriter(data_bean.socket.getOutputStream(), true);
1613            output.println("ccr");
1614            data_bean.socket.shutdownInput();
1615            data_bean.socket.shutdownOutput();
1616            data_bean.socket.close();
1617            third_level_code.message_printer(" Disconnected from Server\n", event_monitor_textarea);
1618            data_bean.connected_state = false;
```

Εικόνα 5.3.14 Διαχειριστής γεγονότος του στοιχείου Disconnect (Μενού Server)

III. Η μέθοδος ip_settingsActionPerformed

Η μέθοδος **ip_settingsActionPerformed** καλείται από το στοιχείο IP Settings του μενού Server. Η μέθοδος αυτή δημιουργεί ένα νέο στιγμιότυπο της κλάσης **server_settings** και ορίζοντας την κλάση αυτή ως visible (εικόνα 5.3.15 γραμμή 1680) αναδύεται ένα νέο παράθυρο για τις ρυθμίσεις της IP και θύρας του server (βλέπε ενότητα 5.3.3.1).

```
1679    private void ip_settingsActionPerformed(java.awt.event.ActionEvent evt) {
1680        new server_settings().setVisible(true);
1681    }
```

Εικόνα 5.3.15 Διαχειριστής γεγονότος του στοιχείου IP Settings (Μενού Server)

IV. Η μέθοδος enable_gpio_interruptsActionPerformed

Η μέθοδος **enable_gpio_interruptsActionPerformed** καλείται οποτεδήποτε προκαλέσει γεγονός το στοιχείο Enable Interrupts που βρίσκεται στο υπό-μενού GPIO του μενού Settings. Η μέθοδος αυτή ως πρώτο βήμα ελέγχει την κατάσταση του στοιχείου Enable Interrupts του υπό-μενού GPIO το οποίο στον κώδικα εμφανίζεται με όνομα **enable_gpio_interrupts** (εικόνα 5.3.16 γραμμή 1077). Σε περίπτωση που το στοιχείο βρίσκεται σε κατάσταση true η μέθοδος καλεί την μέθοδο **message_sender**, που βρίσκεται εσωτερικά τη κλάσης **third_level_code**, η οποία θα αποστείλει τον χαρακτήρα **«a»** (εικόνα 5.3.16 γραμμή 1078) ζητώντας έτσι από το FPGA να ενεργοποιήσει τα interrupts για το module GPIO. Στη συνέχεια καλώντας την **message_printer** τυπώνεται στη περιοχή παρακολούθησης γεγονότων το μήνυμα **«Requesting GPIO Interrupt Enable»**. Σε αντίθετη περίπτωση που το στοιχείο **enable_gpio_interrupts** βρεθεί σε κατάσταση false ακολουθείται η ίδια διαδικασία μόνο που αυτή τη φορά αποστέλλεται ο χαρακτήρας **«b»** (εικόνα 5.3.16 γραμμή 1081), ζητώντας έτσι την απενεργοποίηση των interrupts, και τυπώνεται το μήνυμα **«Requesting GPIO Interrupt Disable»**.

```
1076    private void enable_gpio_interruptsActionPerformed(java.awt.event.ActionEvent evt) {
1077        if (enable_gpio_interrupts.getState() == true) {
1078            third_level_code.message_sender("a");
1079            third_level_code.message_printer(" Requesting GPIO Interrupt Enable\n", event_monitor_textarea);
1080        } else {
1081            third_level_code.message_sender("b");
1082            third_level_code.message_printer(" Requesting GPIO Interrupt Disable\n", event_monitor_textarea);
```

Εικόνα 5.3.16 Διαχειριστής γεγονότος του στοιχείου GPIO > Enable Interrupts (Μενού Settings)

V. Η μέθοδος enable_rs232_interruptsActionPerformed

Η μέθοδος **enable_rs232_interruptsActionPerformed** καλείται από το στοιχείο Enable Interrupts του υπό-μενού RS-232 του μενού Settings. Σε αυτή τη μέθοδο ακολουθείται η ίδια διαδικασία, όπως και στην μέθοδο της προηγούμενης παραγράφου, μόνο που αυτή τη φορά ελέγχεται η κατάσταση του στοιχείου Enable Interrupts του υπό-μενού RS-232 με το στοιχείο να εμφανίζεται στον κώδικα ως **enable_rs232_interrupts** (εικόνα 5.3.17 γραμμή 1068). Στην περίπτωση που το στοιχείο βρίσκεται σε κατάσταση true αποστέλλεται ο χαρακτήρας **«f»** (εικόνα 5.3.17 γραμμή 1069) ώστε το FPGA να ενεργοποιήσει τα interrupts για το πρωτόκολλο RS232 ενώ αν η κατάσταση βρεθεί να είναι false αποστέλλεται ο χαρακτήρας **«g»** (εικόνα 5.3.17 γραμμή 1072) για απενεργοποίηση των interrupts. Για κάθε περίπτωση τυπώνονται στην περιοχή παρακολούθησης γεγονότων αντίστοιχα τα μηνύματα **«Requesting RS232 Interrupt Enable»** και **«Requesting RS232 Interrupt Disable»**.

```
1067    private void enable_rs232_interruptsActionPerformed(java.awt.event.ActionEvent evt) {
1068        if (enable_rs232_interrupts.getState() == true) {
1069            third_level_code.message_sender("f\n");
1070            third_level_code.message_printer(" Requesting RS232 Interrupt Enable\n", event_monitor_textarea);
1071        } else {
1072            third_level_code.message_sender("g");
1073            third_level_code.message_printer(" Requesting RS232 Interrupt Disable\n", event_monitor_textarea);
```

Εικόνα 5.3.17 Διαχειριστής γεγονότος του στοιχείου RS-232 > Enable Interrupts (Μενού Settings)

VI. Η μέθοδος i2c_menuActionPerformed

Η μέθοδος **i2c_menuActionPerformed** καλείται από το στοιχείο I2C του μενού Settings. Η μέθοδος αυτή, αντίστοιχα με τη μέθοδο **ip_settingsActionPerformed**, δημιουργεί ένα νέο στιγμιότυπο της κλάσης **i2c_settings** και ορίζοντας την κλάση αυτή ως visible (εικόνα 5.3.18 γραμμή 1635) αναδύεται ένα νέο παράθυρο στο οποίο ο χρήστης είναι σε θέση να προσθέσει νέες συσκευές που επικοινωνούν με το πρωτόκολλο I^2C(βλέπε ενότητα 5.3.3.1).

```
1634    private void i2c_menuActionPerformed(java.awt.event.ActionEvent evt) {
1635        new i2c_settings().setVisible(true);
```

Εικόνα 5.3.18 Διαχειριστής γεγονότος του στοιχείου I2C (Μενού Settings)

Στην περιοχή επικοινωνίας I^2C (σχήμα 5.3.1) τα γραφικά στοιχεία που μπορούν να προκαλέσουν γεγονός και κατά συνέπεια διαθέτουν διαχειριστές γεγονότων είναι τα κουμπιά **Clear**, **Receive**, **Send** όπως επίσης και το **κουτί επιλογής** (**combobox**).

VII. Η μέθοδος i2c_clear_buttonActionPerformed

Η μέθοδος **i2c_clear_buttonActionPerformed** καλείται ύστερα από πάτημα του κουμπιού **Clear** στην περιοχή επικοινωνίας I^2C. Κατά την κλήση της η μέθοδος διαγράφει τα δεδομένα της String μεταβλητής **i2c_text** (εικόνα 5.3.19 γραμμή 1184). Η μεταβλητή **i2c_text** βρίσκεται στην κλάση **data_bean** και συγκρατεί το κείμενο που εμφανίζεται στην περιοχή προβολής κειμένου του **I^2C**. Στη συνέχεια ορίζει το κείμενο που τυπώνεται στη περιοχή προβολής κειμένου **I^2C**, με όνομα στον κώδικα **i2c_textarea**, να είναι αυτό της μεταβλητής **i2c_text** (εικόνα 5.3.19 γραμμή 1185). Το αποτέλεσμα σε αυτή την περίπτωση είναι ο επιθυμητός καθαρισμός της περιοχής προβολής κειμένου καθώς η μεταβλητή **i2c_text** δεν έχει πλέον κείμενο αποθηκευμένο.

```
1183    private void i2c_clear_buttonActionPerformed(java.awt.event.ActionEvent evt) {
1184        data_bean.i2c_text = "";
1185        i2c_textarea.setText(data_bean.i2c_text);
```

Εικόνα 5.3.19 Διαχειριστής γεγονότος για το κουμπί Clear (Περιοχή επικοινωνίας I2C)

VIII. Η μέθοδος i2c_receive_buttonActionPerformed

Η μέθοδος **i2c_receive_buttonActionPerformed** καλείται όταν προκληθεί γεγονός από το κουμπί **Receive** της περιοχής I^2C. Σκοπός του κουμπιού Receive και κατά επέκταση αυτής της μεθόδου είναι να σταλεί στο FPGA ένα μήνυμα με το οποίο ο χρήστης ζητάει να αντλήσει δεδομένα από μια συσκευή τύπου I^2C. Προκειμένου όμως το FPGA να αντλήσει αυτά τα δεδομένα χρειάζεται να γνωρίζει την διεύθυνση της συσκευής αλλά και τον αριθμό των bytes που πρέπει να λάβει από αυτή. Συνεπώς η εφαρμογή πρέπει να στείλει ένα μήνυμα συγκεκριμένης δομής που παρέχει στο FPGA τις παραπάνω πληροφορίες. Αρχικά η μέθοδος αυτή διαβάζει την ακέραια τιμή της μεταβλητής **i2c_active_address**, που βρίσκεται στην κλάση **data_bean** και αντιπροσωπεύει την διεύθυνση της επιθυμητής συσκευής I^2C, και με casting την μετατρέπει στον αντίστοιχο χαρακτήρα ascii (εικόνα 5.3.20 γραμμή 1172). Ο χαρακτήρας στη συνέχεια μετατρέπεται σε συμβολοσειρά και αποθηκεύεται στη προσωρινή String μεταβλητή **receive_address** (εικόνα 5.3.20 γραμμή 1172). Η ίδια διαδικασία ακολουθείται για τον αριθμό των bytes με τη διαφορά ότι η συμβολοσειρά που αντιπροσωπεύει τα bytes αποθηκεύεται στην String μεταβλητή **receive_bytes** (εικόνα 5.3.20 γραμμή 1173). Σε επόμενο βήμα (εικόνα 5.3.20 γραμμή 1174) η μέθοδος συνθέτει το συγκεκριμένης δομής μήνυμα και καλεί την μέθοδο **message_sender** που θα το στείλει προς το FPGA. Η δομή του μηνύματος περιλαμβάνει τον χαρακτήρα **«j»**, ως κεφαλίδα, ακολουθούμενο από τη συμβολοσειρά της διεύθυνσης και τη συμβολοσειρά των bytes. Η κεφαλίδα παίζει πολύ σημαντικό ρόλο καθώς όταν το FPGA διαβάζει τον χαρακτήρα **«j»** γνωρίζει ότι ο χρήστης ζητάει να λάβει δεδομένα από μια συσκευή I^2C.

```
1171    private void i2c_receive_buttonActionPerformed(java.awt.event.ActionEvent evt) {
1172        String receive_address = Character.toString((char) data_bean.i2c_active_address);
1173        String receive_bytes = Character.toString((char) data_bean.i2c_active_bytes);
1174        third_level_code.message_sender("j" + receive_address + receive_bytes);
1175        if (data_bean.connected_state == true) {
1176            third_level_code.message_printer(" Client-> Requesting I2C Data\n", event_monitor_textarea);
```

Εικόνα 5.3.20 Διαχειριστής γεγονότος για το κουμπί Receive (Περιοχή επικοινωνίας I2C)

IX. Η μέθοδος i2c_send_buttonActionPerformed

Η μέθοδος **i2c_send_buttonActionPerformed** καλείται με το πάτημα του κουμπιού Send της περιοχής I^2C. Σκοπός της είναι να στείλει δεδομένα προς μια συσκευή τύπου I^2C που βρίσκεται προσαρτημένη στο FPGA. Όπως και στη προηγούμενη παράγραφο για να στείλει ο χρήστης δεδομένα σε μια τέτοια συσκευή πρέπει να γνωρίζει την διεύθυνσή της, συνεπώς προκύπτει και πάλι η ανάγκη το μήνυμα που θα σταλεί προς το FPGA να έχει συγκεκριμένη δομή. Η μέθοδος αυτή στο πρώτο βήμα εκτέλεσης της ελέγχει το μήκος του κειμένου δεδομένων που έχει εισαχθεί στο πεδίο εισαγωγής κειμένου, με όνομα στον κώδικα **i2c_textfield** (εικόνα 5.3.21 γραμμή 1160). Αν το μήκος του κειμένου δεν είναι μηδέν, συνεπώς υπάρχουν δεδομένα προς αποστολή, καλείται η μέθοδος **message_sender** να στείλει τα δεδομένα τα οποία εμφωλεύονται στο προηγούμενο μήνυμα συγκεκριμένης δομής (εικόνα 5.3.21 γραμμή 1161). Η δομή του μηνύματος περιλαμβάνει τον χαρακτήρα **«i»** ως κεφαλίδα ακολουθούμενο από την διεύθυνση της συσκευής και τα δεδομένα που αντλούνται από το πεδίο εισαγωγής κειμένου **i2c_textfield**. Η κεφαλίδα «i» ενημερώνει το FPGA πως στο μήνυμα υπάρχουν δεδομένα που πρέπει να παραδοθούν σε συσκευή τύπου I^2C. Στη συνέχεια καλείται η μέθοδος **message_printer** που τυπώνει τα δεδομένα που στάλθηκαν στο πεδίο προβολής κειμένου της περιοχής I^2C (εικόνα 5.3.21 γραμμή 1163). Στο τελευταίο βήμα (εικόνα 5.3.21 γραμμή 1164) καθαρίζεται το πεδίο εισαγωγής κειμένου ώστε να μπορούν μελλοντικά να εισαχθούν νέα δεδομένα.

```
1159    private void i2c_send_buttonActionPerformed(java.awt.event.ActionEvent evt) {
1160    if (i2c_textfield.getText().length() > 0) {
1161       third_level_code.message_sender("i"+(char)data_bean.i2c_active_address+i2c_textfield.getText().toString());
1162       if (data_bean.connected_state == true) {
1163            third_level_code.message_printer(" Sent: " + i2c_textfield.getText() + "\n", i2c_textarea);         }
1164            i2c_textfield.setText("");
```

Εικόνα 5.3.21 Διαχειριστής γεγονότος για το κουμπί Send (Περιοχή επικοινωνίας I2C)

X. Η μέθοδος i2c_device_comboboxActionPerformed

Η μέθοδος **i2c_device_comboboxActionPerformed** καλείται ύστερα από γεγονός που προκαλείται από αλληλεπίδραση του χρήστη με το κουτί επιλογής (combobox) της περιοχής I^2C. Σκοπός αυτού του κουτιού είναι να μπορεί ο χρήστης να επιλέξει μεταξύ συσκευών, που έχει ήδη προσθέσει, για το ποια θα είναι η ενεργή συσκευή με την οποία επικοινωνεί. Δίνεται δε η δυνατότητα, οποιαδήποτε στιγμή, να μπορεί να οριστεί μια άλλη συσκευή ως η ενεργή για επικοινωνία. Ο κώδικας σε αυτή την περίπτωση δεν θα αναλυθεί καθώς παρουσιάζει αρκετή πολυπλοκότητα, αλλά θα δοθεί μια σύντομη περιγραφή της λειτουργίας αυτής της μεθόδου. Όσον αφορά τις συσκευές I^2C υπάρχουν τρεις λίστες στις οποίες αποθηκεύονται το όνομα, η διεύθυνση και τα bytes των συσκευών αντίστοιχα. Κάθε φορά που ο χρήστης προσθέτει ή αφαιρεί συσκευές οι τρεις αυτές λίστες ενημερώνονται. Το κουτί επιλογής αναπαράγει τα ονόματα από τη λίστα ονομάτων . Όταν ο χρήστης επιλέξει ένα από αυτά τότε η μέθοδος συλλέγει από τις άλλες δυο λίστες την διεύθυνση και τα bytes που αντιστοιχούν στο επιλεγμένο όνομα. Στη συνέχεια αποθηκεύει την διεύθυνση στη μεταβλητή **i2c_active_address** και τα bytes στη μεταβλητή **i2c_active_bytes** και με αυτό τον τρόπο δημιουργείται μια ενεργή συσκευή. Αυτές τις μεταβλητές τις χειρίζονται πλέον οι μέθοδοι των κουμπιών **Send** και **Receive** και έτσι τα δεδομένα στέλνονται προς την επιθυμητή συσκευή I^2C.

Τα γραφικά στοιχεία στην περιοχή επικοινωνίας RS-232 που μπορούν να δεχτούν κάποια ενέργεια από τον χρήστη, και κατά συνέπεια διαθέτουν διαχειριστές γεγονότων είναι τα κουμπιά **Clear** και **Send** (εικόνα 5.3.7).

XI. Η μέθοδος rs232_clear_buttonActionPerformed

Η μέθοδος **rs232_clear_buttonActionPerformed** λειτουργεί ακριβώς όπως και η μέθοδος **i2c_clear_buttonActionPerformed**. Κατά την κλήση της διαγράφει τα δεδομένα της String μεταβλητής **rs232_text** (εικόνα 5.3.22 γραμμή 1153). Η μεταβλητή **rs232_text** βρίσκεται στην κλάση **data_bean** και συγκρατεί το κείμενο που εμφανίζεται στο πεδίο προβολής κειμένου του **RS-232**. Στη συνέχεια ορίζει το κείμενο που τυπώνεται στη περιοχή προβολής κειμένου **rs232**, με όνομα στον κώδικα **rs232_textarea**, να είναι αυτό της μεταβλητής **rs232_text** (εικόνα 5.3.22 γραμμή 1154). Το αποτέλεσμα και σε αυτή την περίπτωση είναι ο επιθυμητός καθαρισμός της περιοχής προβολής κειμένου καθώς η μεταβλητή **rs232_text** δεν έχει πλέον κείμενο αποθηκευμένο.

```
1152    private void rs232_clear_buttonActionPerformed(java.awt.event.ActionEvent evt) {
1153        data_bean.rs232_text = "";
1154        rs232_textarea.setText(data_bean.rs232_text);
```

Εικόνα 5.3.22 Διαχειριστής γεγονότος για το κουμπί Clear (Περιοχή επικοινωνίας RS-232)

XII. Η μέθοδος rs232_send_buttonActionPerformed

Η μέθοδος **rs232_send_buttonActionPerformed** καλείται με το πάτημα του κουμπιού **Send** της περιοχής επικοινωνίας RS-232. Σκοπός αυτής της μεθόδου είναι η αποστολή δεδομένων προς οποιαδήποτε συσκευή είναι προσαρτημένη στο FPGA και επικοινωνεί με αυτό με χρήση του πρωτοκόλλου RS-232. Ένα πακέτο δεδομένων που στέλνεται προς μια συσκευή RS-232, όπως και στην περίπτωση του I^2C, έχει συγκεκριμένη δομή καθώς έτσι το FPGA γνωρίζει πως τα εισερχόμενα δεδομένα αφορούν την συσκευή RS-232. Το πακέτο αποτελείται από τον χαρακτήρα **«h»,** ως κεφαλίδα, ακολουθούμενο από τα δεδομένα που βρίσκονται στο πεδίο εισαγωγής κειμένου. Αρχικά κατά την εκτέλεση του κώδικα της μεθόδου, όπως και με το αντίστοιχο κουμπί Send της περιοχής επικοινωνίας I^2C, ελέγχεται το πεδίο εισαγωγής κειμένου (εικόνα 5.3.23 γραμμή 1141). Αν το μήκος του κειμένου δεν είναι μηδέν, συνεπώς υπάρχουν δεδομένα, καλείται η μέθοδος **message_sender** για να στείλει το πακέτο δεδομένων (εικόνα 5.3.23 γραμμή 1142). Στη συνέχεια καλείται η μέθοδος **message_printer** που τυπώνει το απεσταλμένο μήνυμα στο πεδίο προβολής κειμένου **rs232_textarea** της περιοχής επικοινωνίας RS-232 (εικόνα 5.3.23 γραμμή 1144). Στο τελευταίο βήμα αφαιρούνται τα απεσταλμένα δεδομένα από το πεδίο εισαγωγής κειμένου **rs232_textfield**, της ίδιας περιοχής, ώστε να μπορούν να εισαχθούν νέα δεδομένα (εικόνα 5.3.23 γραμμή 1145).

```
1140    private void rs232_send_buttonActionPerformed(java.awt.event.ActionEvent evt) {
1141        if (rs232_textfield.getText().length() > 0) {
1142            third_level_code.message_sender("h" + rs232_textfield.getText().toString());
1143            if (data_bean.connected_state == true) {
1144                third_level_code.message_printer(" Sent: " + rs232_textfield.getText() + "\n", rs232_textarea); }
1145            rs232_textfield.setText("");
```

Εικόνα 5.3.23 Διαχειριστής γεγονότος για το κουμπί Send (Περιοχή επικοινωνίας RS-232)

Η περιοχή επικοινωνίας GPIO (εικόνα 5.3.8) συγκεντρώνει το μεγαλύτερο ποσοστό γραφικών στοιχείων που μπορούν να προκαλέσουν κάποιο γεγονός και κατά συνέπεια και το μεγαλύτερο ποσοστό διαχειριστών γεγονότων. Συγκεκριμένα τα στοιχεία αυτά είναι τέσσερα κουμπιά (Clear, Receive, Send και Set) και δυο σειρές με δεκαέξι διακόπτες (sliders) για κάθε μια, όπου η μια σειρά αφορά τα δεδομένα και η άλλη την κατεύθυνση των ακροδεκτών. Όσον αφορά τους διακόπτες θα αναλυθεί ένας από κάθε σειρά καθώς οι υπόλοιποι ακολουθούν την ίδια διαδικασία.

XIII. Η μέθοδος gpio_clear_buttonActionPerformed

Η μέθοδος **gpio_clear_buttonActionPerformed** καλείται με το πάτημα του κουμπιού **Clear** και λειτουργεί όπως και τα κουμπιά **Clear** των άλλων δυο περιοχών. Σκοπός αυτής της μεθόδου είναι να καθαριστεί το πεδίο προβολής κειμένου της περιοχής επικοινωνίας GPIO. Κατά την εκτέλεση της διαγράφει τα δεδομένα της String μεταβλητής **gpio_text** που βρίσκεται στην κλάση **data_bean** (εικόνα 5.3.24 γραμμή 1133). Η μεταβλητή **gpio_text**, αντίστοιχα με τις **rs232_text** και **i2c_text**, συγκρατεί το κείμενο που τυπώνεται στο πεδίο προβολής κειμένου της περιοχής GPIO από την εκκίνηση μέχρι τον τερματισμό της εφαρμογής. Στη συνέχεια ορίζεται το κείμενο που θα τυπωθεί στο πεδίο προβολής κειμένου, με όνομα στον κώδικα **gpio_textarea**, να είναι αυτό της μεταβλητής **gpio_text** και καθώς αυτή η μεταβλητή δεν έχει πλέον δεδομένα το αποτέλεσμα είναι να καθαριστεί το πεδίο προβολής κειμένου (εικόνα 5.3.24 γραμμή 1134).

```
1132    private void gpio_clear_buttonActionPerformed(java.awt.event.ActionEvent evt) {
1133        data_bean.gpio_text = "";
1134        gpio_textarea.setText(data_bean.gpio_text);
```

Εικόνα 5.3.24 Διαχειριστής γεγονότος για το κουμπί Clear (Περιοχή επικοινωνίας GPIO)

XIV. Η μέθοδος gpio_receive_buttonActionPerformed

Η μέθοδος **gpio_receive_buttonActionPerformed** καλείται οποτεδήποτε ο χρήστης ενεργήσει στο κουμπί **Receive** της περιοχής επικοινωνίας GPIO. Σκοπός χρήσης αυτού του κουμπιού, και κατά συνέπεια της μεθόδου, είναι να ζητήσει και να λάβει ο χρήστης δεδομένα από τους ακροδέκτες GPIO που βρίσκονται στην αναπτυξιακή πλακέτα του FPGA. Όπως και σε προηγούμενες περιπτώσεις, αποστέλλεται ένα μήνυμα συγκεκριμένης δομής προς το FPGA. Το μήνυμα αυτό αποτελείται μόνο από τον χαρακτήρα **«e»** που λειτουργεί ως κεφαλίδα. Στην περίπτωση που το FPGA λάβει ένα τέτοιο μήνυμα δημιουργεί ένα πακέτο με τα δεδομένα των ακροδεκτών GPIO και το στέλνει στην εφαρμογή του χρήστη. Στο πρώτο βήμα εκτέλεσης του κώδικα καλείται η μέθοδος **message_sender** που στέλνει το μήνυμα με την κεφαλίδα (εικόνα 5.3.25 γραμμή 1126). Τελικά καλείται η μέθοδος **message_printer** που τυπώνει το μήνυμα **«Requesting GPIO Data»** στην περιοχή παρακολούθησης γεγονότων (εικόνα 5.3.25 γραμμή 1128).

```
1125    private void gpio_receive_buttonActionPerformed(java.awt.event.ActionEvent evt) {
1126        third_level_code.message_sender("e");
1127        if (data_bean.connected_state == true) {
1128            third_level_code.message_printer(" Requesting GPIO Data\n", event_monitor_textarea);
```

Εικόνα 5.3.25 Διαχειριστής γεγονότος για το κουμπί Receive (Περιοχή επικοινωνίας GPIO)

XV. Η μέθοδος gpio_send_buttonActionPerformed

Η μέθοδος **gpio_send_buttonActionPerformed** καλείται με χρήση του κουμπιού **Send** της περιοχής επικοινωνίας GPIO. Έχει ως σκοπό την αποστολή των δεδομένων, που βρίσκονται στο πεδίο εισαγωγής κειμένου, προς το FPGA με τελικό παραλήπτη τους ακροδέκτες GPIO. Το τελικό πακέτο που αποστέλλεται αποτελείται από τον χαρακτήρα «d», ως κεφαλίδα, ακολουθούμενο από τα δεδομένα που πρέπει να παραδοθούν στους ακροδέκτες. Τα δεδομένα αυτά είναι αποθηκευμένα στην ακέραια μεταβλητή **gpio_data_output_message**. Η μεταβλητή αυτή, που βρίσκεται στην κλάση **data_bean**, χρησιμοποιείται και τροποποιείται από την σειρά με τους ακροδέκτες δεδομένων οποτεδήποτε αυτοί προκαλέσουν κάποια αλλαγή της κατάστασης τους. Συνεπώς όταν στέλνονται τα δεδομένα αυτής της μεταβλητής, τα σήματα στους ακροδέκτες μεταβάλλονται σύμφωνα με το πώς έχει χειριστεί ο χρήστης τους διακόπτες της περιοχής επικοινωνίας GPIO στην εφαρμογή. Κατά την εκτέλεση του πρώτου βήματος της μεθόδου αντλούνται τα δεδομένα από την μεταβλητή **gpio_data_output_message**, μετατρέπονται από ακέραιο σε συμβολοσειρά, συνενώνονται με την κεφαλίδα **«d»** και καλείται η μέθοδος **message_sender** για να αποστείλει το σύνθετο μήνυμα (εικόνα 5.3.26 γραμμή 1118). Τελικά καλείται η μέθοδος message_printer που τυπώνει το σταλμένο μήνυμα στο πεδίο προβολής κειμένου **gpio_textarea** της περιοχής επικοινωνίας GPIO (εικόνα 5.3.26 γραμμή 1120).

```
1117    private void gpio_send_buttonActionPerformed(java.awt.event.ActionEvent evt) {
1118        third_level_code.message_sender("d" + Integer.toString(data_bean.gpio_data_output_message));
1119        if (data_bean.connected_state == true) {
1120        third_level_code.message_printer(" Sent: "+Integer.toString(data_bean.gpio_data_output_message)+"\n",gpio_textarea);
```

Εικόνα 5.3.26 Διαχειριστής γεγονότος για το κουμπί Send (Περιοχή επικοινωνίας GPIO)

XVI. Η μέθοδος gpio_set_buttonActionPerformed

Η μέθοδος **gpio_set_buttonActionPerformed** εκτελείται με το πάτημα του κουμπιού **Set** της περιοχής επικοινωνίας GPIO. Η μέθοδος αυτή λειτουργεί όπως και αυτή της προηγούμενης παραγράφου με τη διαφορά ότι ο σκοπός της είναι να τροποποιηθεί η κατεύθυνση των ακροδεκτών και όχι τα σήματα τους. Το πακέτο που θα αποσταλθεί αποτελείται από την κεφαλίδα **«c»** και τα δεδομένα της νέας κατεύθυνσης τα οποία είναι αποθηκευμένα στην ακέραια μεταβλητή **gpio_direction_output_message**. Το FPGA αφού λάβει ένα τέτοιο πακέτο, και ύστερα από ανάγνωση της κεφαλίδας, γνωρίζει πως τα δεδομένα αφορούν την αλλαγή της κατεύθυνσης των ακροδεκτών. Η μεταβλητή **gpio_direction_output_message**, όπως και η αντίστοιχη της προηγούμενης παραγράφου, χρησιμοποιείται και τροποποιείται από την σειρά με τους ακροδέκτες κατεύθυνσης οποτεδήποτε αυτοί προκαλέσουν κάποια αλλαγή της κατάστασης τους. Κατά την κλήση της μεθόδου αντλούνται τα δεδομένα από την μεταβλητή **gpio_direction_output_message**, μετατρέπονται από ακέραιο σε συμβολοσειρά, συνενώνονται με την κεφαλίδα **«c»** και καλείται η μέθοδος **message_sender** για να αποστείλει το σύνθετο μήνυμα (εικόνα 5.3.27 γραμμή 1110). Στο τελευταίο βήμα καλείται η μέθοδος **message_printer** που τυπώνει την κατεύθυνση που στάλθηκε στην περιοχή παρακολούθησης γεγονότων (εικόνα 5.3.27 γραμμή 1112).

```
1109    private void gpio_set_buttonActionPerformed(java.awt.event.ActionEvent evt) {
1110    third_level_code.message_sender("c" + Integer.toString(data_bean.gpio_direction_output_message));
1111        if (data_bean.connected_state == true) {
1112            third_level_code.message_printer(" Sending Direction:"+data_bean.gpio_direction_output_message+"\n",event_monitor_
```

Εικόνα 5.3.27 Διαχειριστής γεγονότος για το κουμπί Set (Περιοχή επικοινωνίας GPIO)

Σε αυτό το σημείο πρέπει να γίνει μια ιδιαίτερη αναφορά στους διακόπτες (sliders) της περιοχής GPIO. Είναι γνωστό από προηγούμενη παράγραφο πως υπάρχουν δυο σειρές διακοπτών με δεκαέξι σε κάθε μια σειρά. Οι σειρές αυτές αντικατοπτρίζουν τους δεκαέξι ακροδέκτες που βρίσκονται στην αναπτυξιακή πλακέτα. Η πρώτη σειρά επιτρέπει την αλλαγή της κατεύθυνσης (είσοδος ή έξοδος) των ακροδεκτών ενώ η δεύτερη την αλλαγή των σημάτων τους (λογικό μηδέν ή λογικό ένα). Στον κώδικα ο πρώτος διακόπτης της σειράς κατεύθυνσης ονομάζεται **direction_0** και ο τελευταίος **direction_15** ενώ οι ενδιάμεσοι έχουν ονομασίες με τις αντίστοιχες ενδιάμεσες τιμές. Το ίδιο ισχύει για τους διακόπτες της σειράς δεδομένων με τον πρώτο να ονομάζεται **data_0** και τον τελευταίο **data_15**. Όλοι οι διακόπτες διαθέτουν μεθόδους (διαχειριστές γεγονότων) που εκτελούνται σε κάθε αλλαγή της θέσης τους, καθώς τότε σηματοδοτείται νέο γεγονός.

Τα δεδομένα που αφορούν την κατεύθυνση των ακροδεκτών αποθηκεύονται στην ακέραια μεταβλητή **gpio_direction_output_message** και τα δεδομένα που αφορούν τα σήματα αποθηκεύονται στη μεταβλητή **gpio_data_output_message**. Οι διακόπτες τροποποιούν τις τιμές που βρίσκονται σε αυτές τις μεταβλητές οι οποίες στην ουσία είναι αριθμοί που δεν ξεπερνούν τα δεκαέξι bit. Κάθε διακόπτης αντιστοιχεί σε ένα bit της μεταβλητής επομένως κάθε φορά που ένας από αυτούς αλλάζει κατάσταση η μέθοδός του τροποποιεί το αντίστοιχο bit στην αντίστοιχη μεταβλητή. Παρακάτω θα αναλυθεί μια μέθοδος από κάθε σειρά διακοπτών καθώς όλοι λειτουργούν με τον ίδιο ακριβώς τρόπο.

XVII. Η μέθοδος direction_0StateChanged

Η μέθοδος του πρώτου διακόπτη κατεύθυνσης ονομάζεται **direction_0StateChanged** και καλείται, προφανώς, οποτεδήποτε πραγματοποιηθεί αλλαγή της θέσης του διακόπτη **direction_0**. Ο διακόπτης στη θέση **Out** αντιστοιχεί σε τιμή μηδέν ενώ στη θέση **In** σε τιμή ένα. Καθώς ο συγκεκριμένος διακόπτης είναι ο πρώτος μεταξύ αυτών της σειράς κατεύθυνσης μια αλλαγή σε αυτόν θα επηρεάσει μόνο το πρώτο bit στην μεταβλητή **gpio_direction_output_message**. Στην αρχή εκτέλεσης του κώδικα αυτής της μεθόδου ελέγχεται η τιμή που απέκτησε ο διακόπτης (εικόνα 5.3.28 γραμμή 1188). Αν βρεθεί να είναι ένα τότε γίνεται η λογική πράξη OR μεταξύ της μεταβλητής **gpio_direction_output_message** και του αριθμού 1 και η νέα τιμή αποθηκεύεται πίσω στην ίδια μεταβλητή (εικόνα 5.3.28 γραμμή 1189). Το αποτέλεσμα, λαμβάνοντας υπόψη τις λογικές πράξεις, είναι πως το πρώτο bit της μεταβλητής θα αποκτήσει τιμή 1, σε περίπτωση που προηγουμένως ήταν 0, ενώ θα παραμείνει ως έχει σε περίπτωση που ήταν ήδη 1. Τα υπόλοιπα bit με αυτό τον τρόπο δεν θα επηρεαστούν. Στην περίπτωση που ο διακόπτης έχει αποκτήσει τιμή 0 πραγματοποιείται η λογική πράξη AND μεταξύ της μεταβλητής **gpio_direction_output_message** και του αριθμού 65534 και η νέα τιμή αποθηκεύεται πίσω στην ίδια μεταβλητή (εικόνα 5.3.28 γραμμή 1191). Το αποτέλεσμα, αυτή τη φορά, είναι πως το πρώτο bit της μεταβλητής θα αποκτήσει τιμή 0, σε περίπτωση που προηγουμένως ήταν 1, ενώ θα παραμείνει ως έχει σε περίπτωση που ήταν ήδη 0. Με τον ίδιο τρόπο αλλάζουν και τα υπόλοιπα bit με κάθε διακόπτη, της σειράς κατεύθυνσης, να καλεί τη δική του μέθοδο για να τροποποιήσει το δικό του bit. Η μόνη διαφορά είναι πως κάθε μέθοδος πραγματοποιεί τις λογικές πράξεις με διαφορετικούς αριθμούς. Για παράδειγμα η λογική πράξη OR στον διακόπτη **direction_1** γίνεται μεταξύ της μεταβλητής και του αριθμού 2 ενώ η πράξη AND μεταξύ της μεταβλητής και του αριθμού 65533. Στον διακόπτη **direction_2** οι πράξεις γίνονται με τους αριθμούς 4 και 65531 αντίστοιχα και ούτω καθεξής.

```
1187    private void direction_0StateChanged(javax.swing.event.ChangeEvent evt) {
1188        if (direction_0.getValue() == 1) {
1189            data_bean.gpio_direction_output_message = data_bean.gpio_direction_output_message | 1;
1190        } else {
1191            data_bean.gpio_direction_output_message = data_bean.gpio_direction_output_message & 65534;
```

Εικόνα 5.3.28 Διαχειριστής γεγονότος για τον διακόπτη direction_0 (Περιοχή επικοινωνίας GPIO)

XVIII. Η μέθοδος data_0StateChanged

Η μέθοδος του πρώτου διακόπτη σημάτων ονομάζεται **data_0StateChanged** και καλείται ύστερα από αλλαγή της θέσης του διακόπτη **data_0**. Ο διακόπτης στη θέση **0** αντιστοιχεί σε τιμή μηδέν ενώ στη θέση **1** σε τιμή ένα. Και εδώ ο διακόπτης είναι ο πρώτος μεταξύ αυτών της σειράς κατεύθυνσης και έτσι μια αλλαγή σε αυτόν θα επηρεάσει μόνο το πρώτο bit στην μεταβλητή **gpio_data_output_message**. Ο κώδικας αυτής της μεθόδου λειτουργεί με τον ίδιο τρόπο όπως και αυτός της **direction_0StateChanged** και συνεπώς δεν χρειάζεται περαιτέρω διευκρίνιση. Σημαντικό είναι να τονιστεί πως αυτή η μέθοδος, όπως και οι υπόλοιπες των διακοπτών της σειράς δεδομένων, χρησιμοποιούν και τροποποιούν από κοινού την μεταβλητή **gpio_data_output_message**.

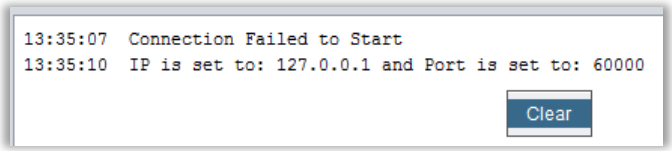
```
1392    private void data_0StateChanged(javax.swing.event.ChangeEvent evt) {
1393        if (data_0.getValue() == 1) {
1394            data_bean.gpio_data_output_message = data_bean.gpio_data_output_message | 1;
1395        } else {
1396            data_bean.gpio_data_output_message = data_bean.gpio_data_output_message & 65534;
```

Εικόνα 5.3.29 Διαχειριστής γεγονότος για τον διακόπτη data_0 (Περιοχή επικοινωνίας GPIO)

Για την κλάση java_interface η περιοχή παρακολούθησης γεγονότων είναι η τελευταία που διαθέτει κώδικα δεύτερου επιπέδου. Σε αυτή την περιοχή το μόνο στοιχείο που μπορεί να προκαλέσει γεγονός είναι η επιλογή **Clear** του μενού που εμφανίζεται με το πάτημα του δεξιού κλικ.

XIX. Η μέθοδος clear_menu_itemMouseReleased

Η περιοχή αυτή έχει, ως μόνο γραφικό στοιχείο, ένα πεδίο προβολής κειμένου. Το πάτημα του δεξιού κλικ οπουδήποτε μέσα σε αυτό το πεδίο έχει ως αποτέλεσμα την εμφάνιση ενός μενού με μοναδικό στοιχείο την επιλογή **Clear** (εικόνα 5.3.30). Σκοπός του **Clear** είναι να καθαριστεί το κείμενο που τυπώνεται στο πεδίο προβολής κειμένου και συνεπώς επιλέγοντας το προκαλείται γεγονός που καλεί την μέθοδο **clear_menu_itemMouseReleased** ως διαχειριστή γεγονότων.



Εικόνα 5.3.30 Μενού Clear με δεξί κλικ (Περιοχή παρακολούθησης γεγονότων)

Η μέθοδος αυτή λειτουργεί όπως και οι μέθοδοι όλων των κουμπιών **Clear** αυτής της εφαρμογής. Αρχικά διαγράφει τα δεδομένα της String μεταβλητής **event_monitor_text** (εικόνα 5.3.31 γραμμή 1623). Η μεταβλητή **event_monitor_text** βρίσκεται στην κλάση **data_bean** και συγκρατεί το κείμενο που εμφανίζεται στο πεδίο προβολής κειμένου της περιοχής παρακολούθησης γεγονότων. Το πεδίο αυτό στον κώδικα εμφανίζεται με το όνομα **event_monitor_textarea**. Στο δεύτερο βήμα του κώδικα ορίζεται το κείμενο που θα τυπωθεί στο πεδίο **event_monitor_textarea** να είναι αυτό της μεταβλητής **event_monitor_text**. Λόγω του ότι αυτή η μεταβλητή δεν έχει πλέον δεδομένα το πεδίο προβολής κειμένου καθαρίζεται από το κείμενο που είχε τυπωθεί νωρίτερα.

```
1622    private void clear_menu_itemMouseReleased(java.awt.event.MouseEvent evt) {
1623        data_bean.event_monitor_text = "";
1624        event_monitor_textarea.setText(data_bean.event_monitor_text);
```

Εικόνα 5.3.31 Διαχειριστής γεγονότος για το μενού Clear (Περιοχή παρακολούθησης γεγονότων)

5.3.3.2.2 Η κλάση server_settings

Ο κώδικας πρώτου επιπέδου αυτής της κλάσης αναδύει ένα νέο παράθυρο (ενότητα 5.3.3.1) στο οποίο μπορεί να οριστεί η IP και η θύρα του server. Τα στοιχεία αυτού του παραθύρου που μπορούν να προκαλέσουν γεγονός, και για αυτό τον λόγο ενσωματώνεται κώδικας δεύτερου επιπέδου στην κλάση, είναι τα κουμπιά **OK** και **Cancel** (εικόνα 5.3.3).

I. Η μέθοδος ip_settings_ok_buttonActionPerformed

Η μέθοδος **ip_settings_ok_buttonActionPerformed** καλείται πατώντας το κουμπί **OK** στο παράθυρο που δημιουργεί η κλάση **server_settings**. Σκοπός αυτής της μεθόδου είναι να αποθηκευτούν οι νέες ρυθμίσεις που πραγματοποίησε ο χρήστης. Η IP και η θύρα αποθηκεύονται στις μεταβλητές **remote_ip** και **remote_port** αντίστοιχα. Οι μεταβλητές αυτές βρίσκονται στην κλάση **data_bean** όπως και οι περισσότερες μεταβλητές της εφαρμογής. Στο πρώτο βήμα του κώδικα αντλείται η IP από το πρώτο πεδίο εισαγωγής κειμένου και αποθηκεύεται στη μεταβλητή **remote_ip** (εικόνα 5.3.32 γραμμή 108). Στο δεύτερο βήμα αντλείται η θύρα από το δεύτερο πεδίο εισαγωγής κειμένου και αποθηκεύεται στη μεταβλητή **remote_port** (εικόνα 5.3.32 γραμμή 109). Στη συνέχεια καλείται η μέθοδος **message_printer** που τυπώνει, προς ενημέρωση του χρήστη, την IP και την θύρα στην περιοχή παρακολούθησης γεγονότων (εικόνα 5.3.32 γραμμή 110). Στο τελευταίο βήμα, αφού έχουν αποθηκευτεί οι νέες ρυθμίσεις, ορίζεται το παράθυρο να μην είναι πλέον ορατό (εικόνα 5.3.32 γραμμή 111).

```
107     private void ip_settings_ok_buttonActionPerformed(java.awt.event.ActionEvent evt) {
108         data_bean.remote_ip=server_ip_textfield.getText();
109         data_bean.remote_port=server_port_textfield.getText();
110         third_level_code.message_printer(" IP is set to: "+data_bean.remote_ip+" and Por
111         this.setVisible(false);
```

Εικόνα 5.3.32 Διαχειριστής γεγονότος για το κουμπί OK (Κλάση server_settings)

II. Η μέθοδος ip_settings_cancel_buttonActionPerformed

Η μέθοδος **ip_settings_cancel_buttonActionPerformed** καλείται από γεγονός που προκαλεί το κουμπί **Cancel** στο ίδιο παράθυρο της κλάσης **server_settings.** Η έννοια του κουμπιού **Cancel** είναι να μην πραγματοποιηθεί καμία αλλαγή και έτσι δεν τροποποιούνται οι μεταβλητές που αποθηκεύουν την IP και την θύρα. Αυτή η μέθοδος έχει ως μόνο σκοπό να κλείσει το παράθυρο που δημιούργησε η κλάση **server_settings** ορίζοντάς το ως μη ορατό (εικόνα 5.3.33), όπως και στον κώδικα της προηγούμενης μεθόδου.

```
115     private void ip_settings_cancel_buttonActionPerformed(java.awt.event.ActionEvent evt) {
116         this.setVisible(false);
```

Εικόνα 5.3.33 Διαχειριστής γεγονότος για το κουμπί Cancel (Κλάση server_settings)

5.3.3.2.3 Η κλάση i2c_settings

Το παράθυρο που εμφανίζει αυτή η κλάση χρησιμεύει στην προσθήκη αλλά και στην αφαίρεση συσκευών που επικοινωνούν με πρωτόκολλο I^2C. Τα γραφικά στοιχεία που κάνουν χρήση κώδικα δεύτερου επιπέδου είναι τα κουμπιά **Add**, **Remove**, **OK** και **Cancel**.

I. Η μέθοδος i2c_addActionPerformed

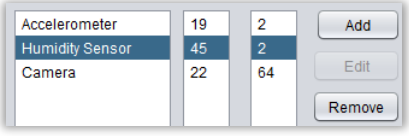
Η μέθοδος **i2c_addActionPerformed** καλείται με το πάτημα του κουμπιού **Add**. Σκοπός αυτής της μεθόδου είναι να προστεθεί νέα συσκευή στη λίστα συσκευών του παραθύρου που δημιουργεί η κλάση **i2c_settings**. Με την κλήση της δημιουργεί ένα νέο αντίγραφο της κλάσης **i2c_device_element** (εικόνα 5.3.34). Το αντίγραφο της κλάσης αναδύει ένα νέο παράθυρο στο οποίο εισάγεται το όνομα και τα χαρακτηριστικά της νέας συσκευής που προστίθεται.

```
202     private void i2c_addActionPerformed(java.awt.event.ActionEvent evt) {
203         new i2c_device_element().setVisible(true);
```

Εικόνα 5.3.34 Διαχειριστής γεγονότος για το κουμπί Add (Κλάση i2c_settings)

II. Η μέθοδος i2c_removeActionPerformed

Η μέθοδος **i2c_removeActionPerformed** εκτελείται ύστερα από ενέργεια του χρήστη στο κουμπί **Remove** και χρησιμεύει στην αφαίρεση συσκευών από την λίστα. Ο χρήστης αρχικά επιλέγει μια συσκευή από την λίστα. Πατώντας το κουμπί **Remove** η μέθοδος ελέγχει ποιο στοιχείο της λίστας έχει επιλεγεί και το αφαιρεί από αυτή. Ο κώδικας αυτής της μεθόδου δεν θα δειχθεί.



Εικόνα 5.3.35 Επιλογή και αφαίρεση συσκευής από τη λίστα

III. Η μέθοδος i2c_okActionPerformed

Η μέθοδος **i2c_okActionPerformed** εκτελείται με το πάτημα του κουμπιού **OK**. Κατά την κλήση της αναβαθμίζει το κουτί επιλογών (combobox) της περιοχής I^2C με τα ονόματα των συσκευών που βρίσκονται στη λίστα (εικόνα 5.3.36 γραμμή 191). Στη συνέχεια ορίζει το παράθυρο ως μη ορατό. Σε περίπτωση που μελλοντικά ο χρήστης κάνει νέες αλλαγές (προσθήκη ή αφαίρεση συσκευών) με το πάτημα του κουμπιού **OK** το κουτί επιλογών θα αναβαθμιστεί με τις νέες τροποποιήσεις.

```
190     private void i2c_okActionPerformed(java.awt.event.ActionEvent evt) {
191         java_interface.i2c_device_combobox.setModel(new DefaultComboBoxModel(model_name.toArray()));
192         this.setVisible(false);
```

Εικόνα 5.3.36 Διαχειριστής γεγονότος για το κουμπί OK (Κλάση i2c_settings)

IV. Η μέθοδος i2c_cancelActionPerformed

Η μέθοδος **i2c_cancelActionPerformed** καλείται με το πάτημα του κουμπιού **Cancel**. Η μέθοδος αυτή λειτουργεί όπως και η αντίστοιχη του κουμπιού **Cancel** στην κλάση **server_settings** και απλά ορίζει το παράθυρο ως μη ορατό.

5.3.3.2.4 Η κλάση i2c_device_element

Η κλάση αυτή καλείται, όπως αναφέρθηκε προηγουμένως, από τη μέθοδο **i2c_addActionPerformed** της κλάσης **i2c_settings** και καθώς περιλαμβάνει κώδικα πρώτου επιπέδου δημιουργεί ένα νέο παράθυρο. Στο παράθυρο αυτό μπορεί να εισαχθεί το όνομα, η διεύθυνση και τα bytes για τη νέα συσκευή I^2C. Το μόνο στοιχείο που θα αναλυθεί σε αυτή την κλάση είναι το κουμπί **OK** το οποίο, με το πάτημα του, καλεί την μέθοδο **i2c_element_ok_buttonActionPerformed**.

I. Η μέθοδος i2c_element_ok_buttonActionPerformed

Στο πρώτο βήμα εκτέλεσης της μεθόδου **i2c_element_ok_buttonActionPerformed** αντλείται το όνομα της συσκευής από το πεδίο εισαγωγής κειμένου **i2c_element_name_textfield** και αποθηκεύεται σε μια νέα String μεταβλητή με όνομα **device_name** (εικόνα 5.3.37 γραμμή 120). Ακολουθείται η ίδια διαδικασία για την αποθήκευση της διεύθυνσης στη μεταβλητή **device_address** και του αριθμού των bytes στη μεταβλητή **device_bytes** (εικόνα 5.3.37 γραμμές 121 και 122). Στα επόμενα τρία βήματα προστίθενται τα στοιχεία αυτών των μεταβλητών στις αντίστοιχες λίστες που συγκρατούν το όνομα, την διεύθυνση και τα bytes της νέας συσκευής (εικόνα 5.3.37 γραμμές 123-125).

```
119     private void i2c_element_ok_buttonActionPerformed(java.awt.event.ActionEvent evt) {
120         String device_name=i2c_device_element.i2c_element_name_textfield.getText();
121         String device_address=i2c_device_element.i2c_element_address_spinner.getValue().toString();
122         String device_bytes=i2c_device_element.i2c_element_bytes_spinner.getValue().toString();
123         i2c_settings.model_name.add(0,device_name);
124         i2c_settings.model_address.add(0,device_address);
125         i2c_settings.model_bytes.add(0,device_bytes);
126         this.setVisible(false);
```

Εικόνα 5.3.37 Διαχειριστής γεγονότος για το κουμπί OK (Κλάση i2c_device_element)

5.3.3.3 Το τρίτο επίπεδο (Ο κώδικας του παρασκηνίου)

Στον κώδικα τρίτου επιπέδου ανήκουν οι κλάσεις **third_level_code** και **data_bean** που δημιουργούνται στη μέθοδο **main** της κλάσης **java_interface** κατά την εκκίνηση της εφαρμογής. Η κλάση **third_level_code** εμφωλεύει κλάσεις και μεθόδους που καλούνται, κατά κύριο λόγο, από τους διαχειριστές γεγονότων του δεύτερου επιπέδου. Η κλάση **data_bean** συγκρατεί το μεγαλύτερο ποσοστό των μεταβλητών με τα δεδομένα που χειρίζεται η μεταβλητή.

5.3.3.3.1 Η κλάση third_level_code

Η κλάση αυτή δημιουργείται στη μέθοδο main της κλάσης **java_interface** κατά την εκκίνηση της εφαρμογής. Από άποψη φυσικής τοποθέτησης αποτελεί το χώρο στον οποίο έχει εισαχθεί το μεγαλύτερο μέρος του κώδικα τρίτου επιπέδου. Παρακάτω θα περιγραφούν οι οκτώ κλάσεις και μέθοδοι που εμφωλεύονται στη κλάση **third_level_code**.

I. Η μέθοδος third_level_code

Η μέθοδος **third_level_code** αποτελεί τον κατασκευαστή (constructor) της κλάσης **third_level_code** και συνέπεια αυτού είναι πως ο κώδικας αυτής της μεθόδου εκτελείται αυτόματα με την εκκίνηση του νέου αντίγραφου της κλάσης **third_level_code**. Αρχικά η μέθοδος δημιουργεί ένα μετρητή (timer) (εικόνα 5.3.38 γραμμή 75). Ο μετρητής αυτός είναι στην ουσία ένα νέο νήμα το οποίο εκτελείται παράλληλα με την εφαρμογή. Στο επόμενο βήμα ορίζεται το νήμα να δημιουργεί και να εκτελεί κάθε ένα δευτερόλεπτο ένα αντίγραφο της κλάσης **check_connection** (εικόνα 5.3.38 γραμμή 76).

```
74      public third_level_code() {
75          timer = new Timer();
76          timer.schedule(new third_level_code.check_connection(), 3000, 1000);
```

Εικόνα 5.3.38 Η μέθοδος κατασκευαστής της κλάσης third_level_code

II. Η κλάση check_connection

Η κλάση **check_connection** καλείται κάθε ένα δευτερόλεπτο από τον μετρητή που αναφέρεται στην προηγούμενη παράγραφο. Σκοπός της είναι να ενημερώνει τον χρήστη σχετικά με την κατάσταση της TCP σύνδεσης, με τον server, με ένδειξη που εμφανίζεται στην γραμμή κατάστασης σύνδεσης (υποκεφάλαιο 5.3.2). Σημαντικό ρόλο σε αυτή την κλάση παίζουν οι boolean μεταβλητές **connected_state** και **previously_connected**. Στις μεταβλητές αυτές έχουν πρόσβαση μέθοδοι που χειρίζονται την TCP σύνδεση οι οποίες τροποποιούν τις μεταβλητές ανάλογα με την κατάσταση της σύνδεσης. Η μεταβλητή **connected_state** έχει τιμή true όταν υπάρχει σύνδεση με τον server ενώ αποκτάει τιμή false σε περίπτωση που είτε δεν έχει γίνει προσπάθεια για σύνδεση είτε αυτή έχει διακοπεί. Η μεταβλητή **previously_connected** έχει τιμή false όσο δεν έχει γίνει σύνδεση από την αρχή εκτέλεσης της εφαρμογής ενώ αλλάζει σε true αν έχει δημιουργηθεί σύνδεση έστω και μια φορά. Η κλάση αυτής της παραγράφου αποκτάει πρόσβαση σε αυτές τις μεταβλητές προκειμένου να ενημερωθεί σχετικά με την τρέχουσα κατάσταση της σύνδεσης.

Η ένδειξη που εμφανίζεται στη γραμμή κατάστασης σύνδεσης αποτελείται από ένα εικονίδιο και ένα μήνυμα κειμένου που εμφανίζεται ως ετικέτα (label). Οι τρεις πρώτες δηλώσεις if καθορίζουν ποιο θα είναι το εικονίδιο και το μήνυμα που εμφανίζεται ανάλογα με την κατάσταση τη σύνδεσης εξετάζοντας τις τιμές των μεταβλητών **connected_state** και **previously_connected.**

Αν και οι δυο μεταβλητές έχουν τιμή false προκύπτει ότι δεν υπάρχει ενεργή σύνδεση αλλά ταυτόχρονα δεν έχει επιχειρηθεί να γίνει σύνδεση προηγουμένως. Σε αυτή την περίπτωση εκτελείται ο κώδικας που εσωκλείεται στην πρώτη δήλωση if (εικόνα 5.3.39 γραμμή 45). Αυτός ο κώδικας ορίζει να εμφανίζεται το εικονίδιο **attention.png** και το μήνυμα **«No connection established»** (εικόνα 5.3.39 γραμμές 46 και 47).

Αν η μεταβλητή **previously_connected** βρεθεί να έχει τιμή true και ταυτόχρονα η **connected_state** έχει τιμή false συμπεραίνεται ότι δεν είναι σύνδεση ενεργή στην παρούσα φάση αλλά έχει υπάρξει επιτυχής σύνδεση στο παρελθόν. Σε μια τέτοια περίπτωση εκτελείται ο κώδικας της δεύτερης δήλωσης if (εικόνα 5.3.39 γραμμή 49) που ορίζει το εικονίδιο να είναι πάλι το **attention.png** αλλά το μήνυμα αυτή τη φορά είναι το **«Connection lost»** (εικόνα 5.3.39 γραμμές 50 και 51).

Αν η μεταβλητή **connected_state** έχει τιμή true είναι προφανές ότι υπάρχει ενεργή σύνδεση και εκτελείται ο κώδικας της τρίτης δήλωσης if (εικόνα 5.3.39 γραμμή 53). Ορίζεται η εμφάνιση του εικονιδίου **green_blink.png** και του μηνύματος **«Connection established»** (εικόνα 5.3.39 γραμμές 54 και 55).

Αφού εξεταστούν οι τρεις πρώτες δηλώσεις if ακολουθεί η εκτέλεση της δήλωσης if-else. Η δήλωση αυτή εξετάζει την τιμή της boolean μεταβλητής **blink** που βρίσκεται στην κλάση **data_bean.** Αυτή η μεταβλητή χρησιμεύει στο να αναβοσβήνει η ένδειξη που έχει οριστεί νωρίτερα από τις τρεις δηλώσεις if. Αν η τιμή της μεταβλητής **blink** βρεθεί να είναι false (εικόνα 5.3.39 γραμμή 58) ορίζεται τόσο το εικονίδιο όσο και το μήνυμα να είναι ορατό (εικόνα 5.3.39 γραμμές 59 και 60). Στη συνέχεια ορίζεται η τιμή της μεταβλητής ως true (εικόνα 5.3.39 γραμμή 61).

Το επόμενο δευτερόλεπτο που θα εξεταστεί και πάλι η μεταβλητή **blink** θα βρεθεί να έχει τιμή true με αποτέλεσμα να εκτελεστεί ο κώδικας που εσωκλείεται στην δήλωση else. Ο κώδικας αυτός θα ορίσει το εικονίδιο και το μήνυμα ως μη ορατά ενώ η τιμή της μεταβλητής θα γίνει false. Είναι προφανές ότι καθόλη τη διάρκεια λειτουργίας της εφαρμογής η ένδειξη θα αναβοσβήνει παραμένοντας ορατή για ένα δευτερόλεπτο και μη ορατή το επόμενο δευτερόλεπτο.

```
42      class check_connection extends TimerTask {
            @Override
44          public void run() {
45              if (data_bean.previously_connected == false && data_bean.connected_state == false) {
46                  java_interface.indicator.setIcon(new javax.swing.ImageIcon("C:\\attention.png"));
47                  java_interface.indicator_label.setText("No connection established");
48              }
49              if (data_bean.previously_connected == true && data_bean.connected_state == false) {
50                  java_interface.indicator.setIcon(new javax.swing.ImageIcon("C:\\attention.png"));
51                  java_interface.indicator_label.setText("Connection lost");
52              }
53              if (data_bean.connected_state == true) {
54                  java_interface.indicator.setIcon(new javax.swing.ImageIcon("C:\\green_blink.png"));
55                  java_interface.indicator_label.setText("Connection established");
56              }
57
58              if (data_bean.blink == false) {
59                  java_interface.indicator.setVisible(true);
60                  java_interface.indicator_label.setVisible(true);
61                  data_bean.blink = true;
62              } else {
63                  java_interface.indicator.setVisible(false);
64                  java_interface.indicator_label.setVisible(false);
65                  data_bean.blink = false;
```

Εικόνα 5.3.39 Η κλάση check_connection

III. Η μέθοδος message_printer

Η μέθοδος **message_printer** χρησιμοποιείται για την τύπωση μηνυμάτων, προς ενημέρωση του χρήστη, στα τέσσερα πεδία προβολής κειμένου της εφαρμογής. Καλείται από μεθόδους τόσο του δεύτερου όσο και του τρίτου επιπέδου όταν πρόκειται να τυπωθεί χρήσιμη πληροφορία. Τα πεδία προβολής κειμένου είναι των περιοχών επικοινωνίας I^2C, RS232, GPIO και της περιοχής παρακολούθησης γεγονότων.

Η μέθοδος αυτή δέχεται δυο ορίσματα με πρώτο μια String μεταβλητή, με όνομα **message**, που μεταφέρει το μήνυμα που πρόκειται να τυπωθεί. Το δεύτερο όρισμα είναι μια μεταβλητή τύπου JTextarea, με όνομα **textarea**, που δείχνει σε ποιο πεδίο προβολής κειμένου πρόκειται να τυπωθεί το μήνυμα του πρώτου ορίσματος. Κατά την κλήση της η μέθοδος λαμβάνει την τρέχουσα ώρα και την αποθηκεύει σε μια Date μεταβλητή με όνομα **date** (εικόνα 5.3.40 γραμμή 79). Η μεταβλητή αυτή τυπώνεται μαζί με το πρώτο όρισμα ώστε να δείξει την ώρα που τυπώθηκε το νέο μήνυμα. Στη συνέχεια εκτελεί τέσσερεις δηλώσεις if οι οποίες εξετάζουν το δεύτερο όρισμα της μεθόδου. Αν αυτό είναι ίδιο με το πεδίο προβολής κειμένου **event_monitor_textarea** σημαίνει πως το μήνυμα πρέπει να τυπωθεί στην περιοχή παρακολούθησης γεγονότων και εκτελείται ο κώδικας της πρώτης δήλωσης if (εικόνα 5.3.40 γραμμή 81).

Αρχικά στην εκτέλεση της δήλωσης if προστίθενται στην μεταβλητή **event_monitor_text**, η οποία συγκρατεί το κείμενο που τυπώνεται στην περιοχή παρακολούθησης γεγονότων, η ώρα της μεταβλητής **date** και το μήνυμα του πρώτου ορίσματος (εικόνα 5.3.40 γραμμή 82). Στη συνέχεια ορίζεται το κείμενο που τυπώνεται στο πεδίο **event_monitor_textarea** να είναι αυτό της μεταβλητής **event_monitor_text** (εικόνα 5.3.40 γραμμή 83).

Με αυτό τον τρόπο ανανεώνεται το πεδίο προβολής κειμένου στο οποίο τώρα τυπώνεται το κείμενο που ήδη υπήρχε αλλά και η ώρα και το νέο μήνυμα. Οι άλλες τρεις δηλώσεις if λειτουργούν με τον ίδιο τρόπο με την διαφορά ότι εξετάζονται τα πεδία προβολής κειμένου **i2c_textarea**, **rs232_textarea** και **gpio_textarea** ενώ η αποθήκευση του μηνύματος και της ώρας γίνεται στις αντίστοιχες μεταβλητές **i2c_text**, **rs232_text** και **gpio_text**.

```
77      public static void message_printer(String message, JTextArea textarea) {
78
79          Date date = new Date();
80
81          if (textarea.equals(java_interface.event_monitor_textarea)) {
82              data_bean.event_monitor_text = data_bean.event_monitor_text + data_bean.dateFormat.format(date) + message;
83              textarea.setText(data_bean.event_monitor_text);
84          }
85
86          if (textarea.equals(java_interface.i2c_textarea)) {
87              data_bean.i2c_text = data_bean.i2c_text + data_bean.dateFormat.format(date) + message;
88              textarea.setText(data_bean.i2c_text);
89          }
90
91          if (textarea.equals(java_interface.rs232_textarea)) {
92              data_bean.rs232_text = data_bean.rs232_text + data_bean.dateFormat.format(date) + message;
93              textarea.setText(data_bean.rs232_text);
94          }
95
96          if (textarea.equals(java_interface.gpio_textarea)) {
97              data_bean.gpio_text = data_bean.gpio_text + data_bean.dateFormat.format(date) + message;
98              textarea.setText(data_bean.gpio_text);
99          }
```

Εικόνα 5.3.40 Η μέθοδος message_printer

IV. Η μέθοδος start_client

Η μέθοδος **start_client** καλείται από τη μέθοδο **server_connectActionPerformed** (ενότητα 5.3.3.2), του δεύτερου επιπέδου, όταν ο χρήστης πατήσει το κουμπί **Connect** στο μενού **Server**. Σκοπός αυτής της μεθόδου είναι να εγκαθιδρύσει σύνδεση με τον server. Αρχικά η εξωτερική δήλωση if εξετάζει την boolean μεταβλητή **connected_state** (εικόνα 5.3.41 γραμμή 312). Αν η τιμή της είναι false αυτομάτως σημαίνει πως δεν υπάρχει ενεργή σύνδεση επομένως εκτελείται ο κώδικας εσωτερικά της δήλωσης if ώστε να δημιουργηθεί μια νέα σύνδεση. Αν, αντίθετα, η μεταβλητή έχει τιμή true υπάρχει ήδη ενεργή σύνδεση επομένως αποτρέπεται η προσπάθεια νέας σύνδεσης. Σε αυτή την περίπτωση εκτελείται ο κώδικας της δήλωσης else που καλεί την μέθοδο **message_printer** για να τυπώσει το μήνυμα **«Client is already Connected to Server»** στην περιοχή παρακολούθησης γεγονότων (εικόνα 5.3.41 γραμμή 336). Επιστρέφοντας στον κώδικα της εξωτερικής δήλωσης if στο πρώτο βήμα εκτέλεσης του αποθηκεύει την IP στην μεταβλητή τύπου InetAddress, με όνομα **serverAddress**, την οποία αντλεί από την String μεταβλητή **remote_ip** (εικόνα 5.3.41 γραμμή 314). Στο επόμενο βήμα επιχειρείται να δημιουργηθεί νέα σύνδεση με IP αυτή της μεταβλητής **serverAddress** και θύρα αυτή της μεταβλητής **remote_port** (εικόνα 5.3.41 γραμμή 319). Με την επιτυχή σύνδεση δημιουργείται μια υποδοχή (socket), που εξυπηρετεί την επικοινωνία μέσα από αυτή τη σύνδεση, η οποία αποθηκεύεται στην μεταβλητή **socket** που βρίσκεται στην κλάση **data_bean**. Στη συνέχεια καλείται η μέθοδος **message_printer** που τυπώνει το μήνυμα **«Succesfully Connected to Server»** στην περιοχή παρακολούθησης γεγονότων και αλλάζει η τιμή των μεταβλητών **connected_state** και **previously_connected** σε true (εικόνα 5.3.41 γραμμές 320, 322 και 323). Στο τελευταίο βήμα εξετάζεται, στην εσωτερική δήλωση if, αν υπάρχει ενεργή σύνδεση μέσα από την υποδοχή **socket** (εικόνα 5.3.41 γραμμή 324). Στην περίπτωση που κάτι τέτοιο επαληθευτεί με τιμή true δημιουργείται και εκκινείται ένα νέο αντίγραφο της κλάσης **Receiver** (εικόνα 5.3.41 γραμμές 325 και 326).

```
310     public static void start_client() {
311
312         if (data_bean.connected_state == false) {
313             try {
314                 data_bean.serverAddress = InetAddress.getByName(data_bean.remote_ip);
315             } catch (UnknownHostException e) {
                    e.printStackTrace();
317             }
318             try {
319                 data_bean.socket = new Socket(data_bean.serverAddress, Integer.parseInt(data_bean.remote_port));
320                 message_printer(" Succesfully Connected to Server\n", java_interface.event_monitor_textarea);
321
322                 data_bean.connected_state = true;
323                 data_bean.previously_connected = true;
324                 if (data_bean.socket.isConnected() == true) {
325                     Receiver receiver = new Receiver();
326                     receiver.start();
327                 }
328             } catch (UnknownHostException e) {
329                 //e.printStackTrace();
330                 message_printer(" Connection Failed to Start\n", java_interface.event_monitor_textarea);
331             } catch (Exception e) {
332                 //e.printStackTrace();
333                 message_printer(" Connection Failed to Start\n", java_interface.event_monitor_textarea);
334             }
335         } else {
336             message_printer(" Client is already Connected to Server\n", java_interface.event_monitor_textarea);
```

Εικόνα 5.3.41 Η μέθοδος start_client

V. Η κλάση Receiver

Η κλάση **Receiver** δημιουργείται στη μέθοδο **start_client** μόλις επιτευχθεί TCP σύνδεση με τον server. Η κλάση αυτή είναι ένα νέο νήμα που εκτελείται παράλληλα με τα άλλα τμήματα της εφαρμογής και αναμένει για λήψη εισερχόμενων δεδομένων. Κατά την εκκίνηση εκτέλεσης αυτού του νήματος καλείται η μέθοδος **message_printer** που τυπώνει το μήνυμα **«Receiver Thread Started (Waiting for Data)»** στην περιοχή παρακολούθησης γεγονότων (εικόνα 5.3.42 γραμμή 288). Στη συνέχεια δημιουργείται μια νέα είσοδος τύπου BufferedReader, με όνομα **input**, που λαμβάνει τα δεδομένα μέσα από την υποδοχή **socket** (εικόνα 5.3.42 γραμμή 290). Σε επόμενο βήμα εκκινείται ένας βρόγχος while που εκτελείται όσο η μεταβλητή **connected_state** έχει τιμή true. Κατά την διάρκεια εκτέλεσης του βρόγχου διαβάζεται η είσοδος **input** για εισερχόμενα δεδομένα τα οποία μόλις ληφθούν αποθηκεύονται σε μια String μεταβλητή με όνομα **receivedData** (εικόνα 5.3.42 γραμμή 293). Στο τελευταίο βήμα καλείται η μέθοδος **received_message_analyzer** με όρισμα την μεταβλητή **receivedData** ώστε να αναλύσει τα εισερχόμενα δεδομένα (εικόνα 5.3.42 γραμμή 294).

```
284     public static class Receiver extends Thread {
285
            @Override
287         public void run() {
288             message_printer(" Receiver Thread Started(Waiting for Data)\n", java_interface.event_monitor_textarea);
289             try {
290                BufferedReader input = new BufferedReader(new InputStreamReader(data_bean.socket.getInputStream()));
291
292                 while (data_bean.connected_state == true) {
293                     String receivedData = input.readLine();
294                     received_message_analyzer(receivedData);
295                 }
```

Εικόνα 5.3.42 Η κλάση Receiver

VI. Η μέθοδος message_sender

Η μέθοδος **message_sender** καλείται από τις μεθόδους του δεύτερου επιπέδου οποτεδήποτε πρόκειται να σταλούν δεδομένα προς τον server και κατά επέκταση προς το FPGA. Η μέθοδος αυτή αρχικά ελέγχει την τιμή της μεταβλητής **connected_state** (εικόνα 5.3.43 γραμμή 28). Στην περίπτωση που η τιμή της είναι true δημιουργείται μια έξοδος τύπου PrintWriter, με όνομα **output**, που θα αποστείλει τα δεδομένα μέσα από την υποδοχή **socket** (εικόνα 5.3.43 γραμμές 30 και 31). Στην αντίθετη περίπτωση που η τιμή της μεταβλητής είναι false αποτρέπεται η αποστολή δεδομένων ενώ δημιουργείται και εμφανίζεται ένα παράθυρο διαλόγου με το μήνυμα **«Please connect to send data»** (εικόνα 5.3.43 γραμμή 36).

```
26      public static void message_sender(String message) {
27
28          if (data_bean.connected_state == true) {
29              try {
30                  PrintWriter output = new PrintWriter(data_bean.socket.getOutputStream(), true);
31                  output.println(message);
32              } catch (IOException e) {
                    e.printStackTrace();
34              }
35          } else {
36              new dialog_box("Please connect to send data.").setVisible(true);
37          }
```

Εικόνα 5.3.43 Η μέθοδος message_sender

VII. Η μέθοδος received_message_analyzer

Η μέθοδος **received_message_analyzer** καλείται οποτεδήποτε το νήμα της κλάσης **Receiver** λάβει δεδομένα. Η μέθοδος αυτή έχει ως όρισμα μια String μεταβλητή με όνομα **message** στην οποία βρίσκονται τα εισερχόμενα δεδομένα που πρόκειται να αναλυθούν. Ο κώδικας σε αυτή τη μέθοδο αποτελείται από τρεις δηλώσεις if που εξετάζουν την κεφαλίδα του ληφθέντος μηνύματος.

Αν η κεφαλίδα του μηνύματος, δηλαδή ο χαρακτήρας στην πρώτη θέση της συμβολοσειράς, αποτελείται από τον χαρακτήρα **«d»** (εικόνα 5.3.44 γραμμή 258) το μήνυμα προέρχεται από το GPIO module του FPGA και συνεπώς αφορά, στην εφαρμογή, την περιοχή επικοινωνίας GPIO. Σε αυτή την περίπτωση εκτελείται ο κώδικας της πρώτης δήλωσης if. Στο πρώτο βήμα αφαιρείται η κεφαλίδα ώστε να μείνει το καθαρό μήνυμα το οποίο στη συνέχεια αποθηκεύεται στη μεταβλητή **filtered_message** (εικόνα 5.3.44 γραμμή 259). Ακολουθεί η κλήση της μεθόδου **message_printer** που τυπώνει το φιλτραρισμένο μήνυμα στο πεδίο **gpio_textarea** της περιοχής επικοινωνίας GPIO (εικόνα 5.3.44 γραμμή 260). Η εκτέλεση των εντολών αυτής της δήλωσης if τελειώνει με την κλήση της μεθόδου **gpio_received_data_analyzer** που δέχεται ως όρισμα το φιλτραρισμένο μήνυμα, το αναλύει σε bit και δημιουργεί οπτική αναπαράσταση μετακινώντας τους διακόπτες της σειράς δεδομένων.

Αν ο χαρακτήρας της κεφαλίδας είναι το **«h»** (εικόνα 5.3.44 γραμμή 263) το ληφθέν μήνυμα αφορά την περιοχή επικοινωνίας RS-232 και εκτελείται ο κώδικας εσωτερικά της δεύτερης δήλωσης if. Και σε αυτή την περίπτωση αφαιρείται η κεφαλίδα, το φιλτραρισμένο μήνυμα αποθηκεύεται στη μεταβλητή **filtered_message** και στη συνέχεια τυπώνεται στο πεδίο **rs232_textarea** (εικόνα 5.3.44 γραμμές 264 και 265). Στην περίπτωση που η κεφαλίδα είναι ο χαρακτήρας **«i»** ακολουθείται η ίδια διαδικασία με τη διαφορά ότι το φιλτραρισμένο μήνυμα αφορά την περιοχή επικοινωνίας I^2C και συνεπώς τυπώνεται στο πεδίο **i2c_textarea** (εικόνα 5.3.44 γραμμές 268 και 269).

```
255     public static void received_message_analyzer(String message) {
256         String filtered_message;
257
258         if (message.charAt(0) == 'd') {
259             filtered_message = message.substring(1);
260             message_printer(" Received: " + filtered_message + "\n", java_interface.gpio_textarea);
261             gpio_received_data_analyzer(filtered_message);
262         }
263         if (message.charAt(0) == 'h') {
264             filtered_message = message.substring(1);
265             message_printer(" Received: " + filtered_message + "\n", java_interface.rs232_textarea);
266         }
267         if (message.charAt(0) == 'i') {
268             filtered_message = message.substring(1);
269             message_printer(" Received: " + filtered_message + "\n", java_interface.i2c_textarea);
```

Εικόνα 5.3.44 Η μέθοδος received_message_analyzer

VIII. Η μέθοδος gpio_received_data_analyzer

Η μέθοδος **gpio_received_data_analyzer** καλείται από την **received_message_analyzer** οποτεδήποτε υπάρχουν εισερχόμενα δεδομένα που προέρχονται από το GPIO module του FPGA και πρέπει να αναλυθούν. Τα δεδομένα αυτά είναι στην ουσία ένας αριθμός ο οποίος δεν ξεπερνάει τα δεκαέξι bit με κάθε bit να αντιπροσωπεύει το σήμα του αντίστοιχου ακροδέκτη. Σκοπός αυτής της μεθόδου είναι να εντοπίσει ποια bit έχουν τιμή μηδέν και ποια ένα ώστε να μετακινήσει τους αντίστοιχους διακόπτες της σειράς δεδομένων δίνοντας έτσι οπτική αναπαράσταση των σημάτων.

Αρχικά στην εκτέλεση του κώδικα γίνεται μετατροπή της String μεταβλητής **gpio_received_data** σε ακέραιο και αποθηκεύεται στη μεταβλητή **gpio_data_output_message** (εικόνα 5.3.45 γραμμή 106). Συνέπεια αυτού είναι πως πλέον τα δεδομένα έχουν την μορφή αριθμού που μπορεί να επεξεργαστεί η μέθοδος με λογικές πράξεις. Στο επόμενο βήμα δημιουργείται ένα αντίγραφο της προηγούμενης μεταβλητής το οποίο αποθηκεύεται στη ακέραια μεταβλητή **temp** (εικόνα 5.3.45 γραμμή 107). Η επεξεργασία του αριθμού στη συνέχεια θα γίνει με χρήση της **temp** ώστε να μην επηρεαστεί η **gpio_data_output_message**. Ακολουθεί η λογική πράξη AND μεταξύ της μεταβλητής **temp** και του αριθμού 1 (εικόνα 5.3.45 γραμμή 109). Το αποτέλεσμα είναι να απομονωθεί μόνο το πρώτο bit του αριθμού, το οποίο αντιστοιχεί στον πρώτο ακροδέκτη και κατά συνέπεια στον πρώτο διακόπτη. Ο νέος αριθμός της μεταβλητής θα έχει τιμή 0 ή 1. Εξετάζεται στην δήλωση if η μεταβλητή **temp** (εικόνα 5.3.45 γραμμή 110). Αν η τιμή της είναι 1 ο πρώτος διακόπτης, με όνομα **data_0**, τοποθετείται στη θέση 1 (εικόνα 5.3.45 γραμμή 111) ενώ αν η τιμή είναι 0 ο διακόπτης τοποθετείται στη θέση 0 (εικόνα 5.3.45 γραμμή 113). Στο τελευταίο βήμα η μεταβλητή temp αποκτά πάλι την αρχική της τιμή, που αντλείται από την μεταβλητή **gpio_data_output_message** (εικόνα 5.3.45 γραμμή 116), ώστε να γίνουν οι ανάλογες λογικές πράξεις για το επόμενο bit.

Ο κώδικας μεταξύ των γραμμών 109 και 116 της παρακάτω εικόνας επαναλαμβάνεται άλλες δεκαπέντε φορές για κάθε ένα από τα υπόλοιπα δεκαπέντε bit. Για το επόμενο bit η λογική πράξη AND γίνεται μεταξύ της temp και του 2. Το αποτέλεσμα είναι να απομονωθούν τα δυο πρώτα bit και ο νέος αριθμός θα έχει τιμή 2 ή 0. Προφανώς αν η τιμή της **temp** είναι 2 ο διακόπτης **data_1** θα τοποθετηθεί στη θέση 1 ενώ με τιμή 0 θα τοποθετηθεί στη θέση 0. Η ίδια λογική ακολουθείται για τα υπόλοιπα bit.

```
103     public static void gpio_received_data_analyzer(String gpio_received_data) {
104
105         int temp;
106         data_bean.gpio_data_output_message = Integer.parseInt(gpio_received_data);
107         temp = data_bean.gpio_data_output_message;
108
109         temp = temp & 1;
110         if (temp == 1) {
111             java_interface.data_0.setValue(1);
112         } else {
113             java_interface.data_0.setValue(0);
114         }
115
116         temp = data_bean.gpio_data_output_message;
```

Εικόνα 5.3.45 Η μέθοδος gpio_received_data_analyzer

5.3.3.3.2 Η κλάση data_bean

Η κλάση αυτή δημιουργείται στη μέθοδο **main** της κλάσης **java_interface** κατά την εκκίνηση της εφαρμογής. Περιέχει το μεγαλύτερο ποσοστό των μεταβλητών που χρησιμοποιούνται από τις περισσότερες μεθόδους και κλάσεις είτε για άντληση των δεδομένων είτε για τροποποίηση αυτών.

Οι String μεταβλητές **event_monitor_text**, **gpio_text**, **rs232_text** και **i2c_text** (εικόνα 5.3.46 γραμμές 19 έως 23) χρησιμοποιούνται για την αποθήκευση των κειμένων που προβάλλονται στην περιοχή παρακολούθησης γεγονότων και τις περιοχές επικοινωνίας GPIO, RS-232 και I^2C αντίστοιχα. Πρόσβαση σε αυτές τις μεταβλητές αποκτούν οι μέθοδοι των κουμπιών **Clear** αλλά και η **message_printer** για την τύπωση των δεδομένων τους, την αναβάθμιση ή την διαγραφή τους.

Η μεταβλητή **date_format** (εικόνα 5.3.46 γραμμή 23) καθορίζει την μορφή που θα έχει η ώρα που τυπώνεται μαζί με το κείμενο στα πεδία προβολής κειμένου. Καθορίζεται να είναι τύπου String και να αποτελείται από δυο ψηφία για την ώρα, δύο για τα λεπτά και δυο για τα δευτερόλεπτα. Η μεταβλητή αυτή χρησιμοποιείται από τη μέθοδο **message_printer**.

Όταν πρόκειται να τυπωθεί η τρέχουσα ώρα, που είναι αποθηκευμένη στη μεταβλητή date, καλείται η αντίστοιχη εντολή που της δίνει την μορφή που ορίζει η **date_format** (εικόνα 5.3.40 γραμμές 82, 87, 92 και 97).

Οι boolean μεταβλητές **connected_state** και **previously_connected** (εικόνα 5.3.46 γραμμές 24 και 25) χρησιμοποιούνται για ενημέρωση όσον αφορά την κατάσταση της σύνδεσης. Η πρώτη δείχνει αν υπάρχει ενεργή σύνδεση ενώ η δεύτερη αν έχει υπάρξει επιτυχής σύνδεση στο παρελθόν. Και οι δυο μεταβλητές έχουν αρχικά τιμή false καθώς δεν υπάρχει τρέχουσα ενεργή σύνδεση αλλά ούτε έχει υπάρξει από την εκκίνηση της εφαρμογής. Χρησιμοποιούνται και ενημερώνονται από τις μεθόδους σύνδεσης ή αποσύνδεσης με τον server σχετικά με την ισχύουσα κατάσταση της σύνδεσης. Άλλες μέθοδοι εκμεταλλεύονται αυτές τις μεταβλητές καθώς η τρέχουσα κατάσταση της σύνδεσης μπορεί να επιτρέψει ή όχι την αποστολή δεδομένων.

Η μεταβλητή **serverAddress**, που είναι τύπου InetAddress συγκρατεί την IP που αντλεί από τη μεταβλητή **remote_ip**. Είναι η μεταβλητή που χρησιμοποιεί η μέθοδος **start_client** όταν πρόκειται να συνδεθεί με τον server (εικόνα 5.3.41 γραμμή 319).

Οι μεταβλητές **remote_ip** και **remote_port** συγκρατούν την IP και την θύρα του server αντίστοιχα. Χρησιμοποιούνται από τις μεθόδους δεύτερου επιπέδου της κλάσης **server_settings** όταν πρόκειται να τροποποιηθούν από τον χρήστη. Χρησιμοποιούνται επίσης από τη μέθοδο **start_client** για την σύνδεση με τον server. Η IP που βρίσκεται στην **remote_ip** πριν χρησιμοποιηθεί σε μια προσπάθεια σύνδεσης μετατρέπεται σε τύπο InetAddress και αποθηκεύεται στη **serverAddress**.

Η μεταβλητή **socket** είναι ο χώρος που αποθηκεύεται η νέα υποδοχή κατά τη δημιουργία μιας TCP σύνδεσης. Με πρόσβαση στην μεταβλητή **socket**, η κλάση **Receiver** χρησιμοποιεί την υποδοχή αυτή για την λήψη δεδομένων ενώ η μέθοδος **message_sender** την χρησιμοποιεί για αποστολή δεδομένων. Η μέθοδος **server_disconnectActionPerformed** της επιλογής **Disconnect** στο μενού **Server** μέσα από τη μεταβλητή **socket** μπορεί να τερματίσει την σύνδεση.

Οι μεταβλητές **gpio_data_output_message** και **gpio_direction_output_message** (εικόνα 5.3.46 γραμμές 30 και 31) αποτελούν τους χώρους αποθήκευσης των δεδομένων που αφορούν τα σήματα αλλά και την κατεύθυνση των ακροδεκτών του FPGA. Χρησιμοποιούνται κυρίως από τις μεθόδους δευτέρου επιπέδου. Κάποιες από αυτές τις μεθόδους καλούνται με την αλλαγή της θέσης των διακοπτών και κατά την εκτέλεση τους τροποποιούν τις τιμές των μεταβλητών. Σε άλλες περιπτώσεις καλούνται οι μέθοδοι των κουμπιών **Send** και **Set** που αντλούν τα δεδομένα των **gpio_data_output_message** και **gpio_direction_output_message** αντίστοιχα ώστε να τα αποστείλουν προς το GPIO module του FPGA. Η μεταβλητή **gpio_data_output_message** χρησιμοποιείται, επιπλέον, από την μέθοδο **gpio_received_data_analyzer**, του τρίτου επιπέδου, η οποία αποθηκεύει τα δεδομένα που λήφθηκαν από το GPIO module στην προαναφερθείσα μεταβλητή. Με αυτό το τρόπο η εφαρμογή μπορεί να είναι ενημερωμένη με την τρέχουσα κατάσταση των σημάτων των ακροδεκτών.

Η boolean μεταβλητή **blink** (εικόνα 5.3.46 γραμμή 32) χρησιμοποιείται από την κλάση **check_connection** καθώς καθορίζει, ανάλογα με την τιμή της, το αν θα είναι ή όχι ορατή η ένδειξη στην γραμμή κατάστασης σύνδεσης. Αρχικά έχει τιμή false ενώ κάθε δευτερόλεπτο αλλάζει αποκτώντας τιμή αντίθετη από αυτή που διέθετε την τελευταία φορά που τροποποιήθηκε. Με αυτό τον τρόπο η ένδειξη παραμένει ορατή για ένα δευτερόλεπτο και μη ορατή το επόμενο.

Οι μεταβλητές **i2c_active_address** και **i2c_active_bytes** (εικόνα 5.3.46 γραμμές 35 και 36) συγκρατούν την διεύθυνση και τα bytes της τρέχουσας συσκευής I^2C. Τροποποιούνται από τη μέθοδο δευτέρου επιπέδου του κουτιού επιλογής (combobox), της περιοχής επικοινωνίας I^2C, οποτεδήποτε αλλάζει η τρέχουσα συσκευή και πρέπει να αποθηκευτούν η νέα διεύθυνση και τα νέα bytes. Οι μεταβλητές αυτές χρησιμοποιούνται επίσης από τις μεθόδους των κουμπιών **Receive** και **Send** καθώς η διεύθυνση και τα bytes πρέπει να συμπεριληφθούν στα σύνθετα μηνύματα που αποστέλλονται από τα παραπάνω κουμπιά της περιοχής I^2C.

```
17  public class data_bean {
18
19      public static String event_monitor_text = "";
20      public static String gpio_text = "";
21      public static String rs232_text = "";
22      public static String i2c_text = "";
23      public static DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss ");
24      public static boolean connected_state = false;
25      public static boolean previously_connected = false;
26      public static InetAddress serverAddress;
27      public static String remote_ip = "127.0.0.1";
28      public static String remote_port = "60000";
29      public static Socket socket;
30      public static int gpio_data_output_message = 0;
31      public static int gpio_direction_output_message = 0;
32      public static boolean blink = false;
33      public static boolean set_direction_on_change_status = false;
34      public static boolean send_data_on_change_status = false;
35      public static int i2c_active_address = 0;
36      public static int i2c_active_bytes = 0;
37  }
```

Εικόνα 5.3.46 Η κλάση data_bean

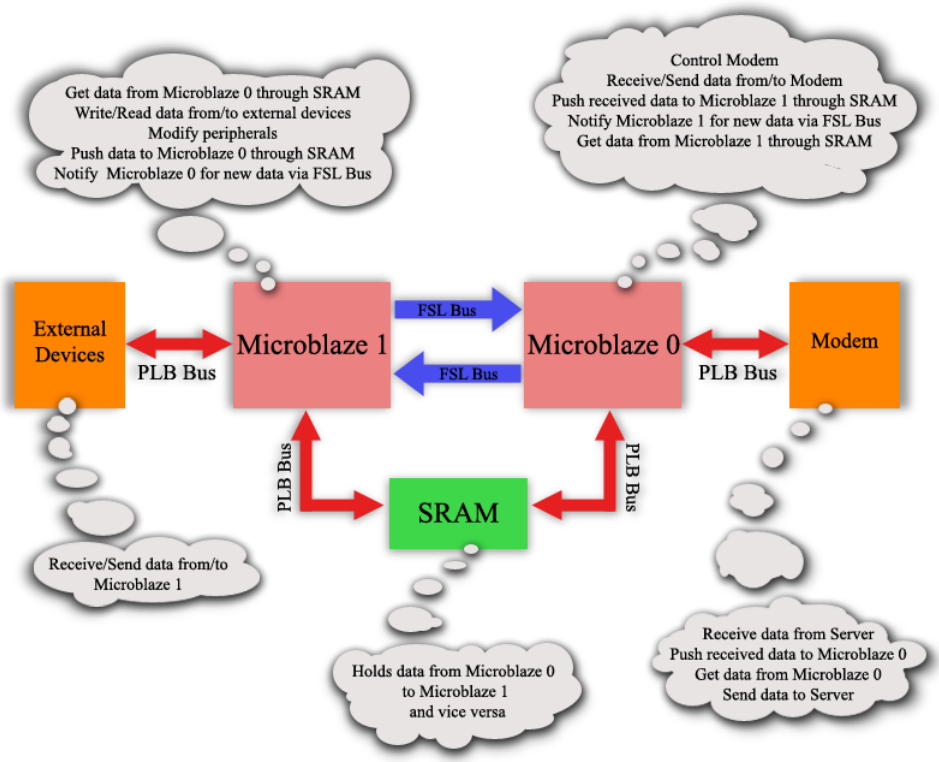
5.4 Λογισμικό στο Ενσωματωμένο Σύστημα (FPGA)

Το λογισμικό που εκτελείται στο ενσωματωμένο σύστημα αναπτύχθηκε σε γλώσσα C με χρήση του Xilinx Platform Studio. Είναι υπεύθυνο για την αρχικοποίηση και διαχείριση του συστήματος και των περιφερειακών, την επικοινωνία με το modem (και κατ' επέκταση με τον **Server**) και τις ψηφιακές συσκευές αλλά και την επεξεργασία των δεδομένων που στέλνονται και λαμβάνονται.

5.4.1 Γενική αναφορά στο λογισμικό του Ενσωματωμένου Συστήματος

Είναι ήδη γνωστό (Κεφάλαιο 4) πως το ενσωματωμένο σύστημα αποτελείται από δυο συνεργαζόμενους επεξεργαστές που μοιράζονται το φόρτο των εργασιών που πρέπει να διευθετηθούν. Συνεπώς έχει αναπτυχθεί λογισμικό ανεξάρτητα για κάθε επεξεργαστή. Ο πρώτος επεξεργαστής, με όνομα **Microblaze 0**, διαχειρίζεται την επικοινωνία με το modem που κατ' επέκταση δίνει την δυνατότητα επικοινωνίας με τον **Server** και τελικά με την εφαρμογή στο τερματικό του χρήστη. Ταυτόχρονα μεταβιβάζει τα εισερχόμενα δεδομένα από το modem προς τον δεύτερο επεξεργαστή (για περαιτέρω επεξεργασία) και αντίστροφα προωθεί τα εξερχόμενα δεδομένα προς το modem. Ο δεύτερος επεξεργαστής, με όνομα **Microblaze 1**, επεξεργάζεται τα δεδομένα που λαμβάνει από τον **Microblaze 0**. Τα δεδομένα αυτά είτε πρέπει να παραδοθούν προς κάποια από τις προσαρτημένες συσκευές είτε ζητούν κάποιου είδους τροποποίηση στο σύστημα (πχ ενεργοποίηση ή απενεργοποίηση των interrupts στο GPIO Module). Άλλη ευθύνη του **Microblaze 1** είναι η λήψη δεδομένων από αυτές τις συσκευές και η προώθηση τους προς τον **Microblaze 0.**

Σε αυτό το σημείο θα δοθεί μια σύντομη περιγραφή για τις ενέργειες που πραγματοποιούνται σε κάθε επεξεργαστή τόσο κατά την εκκίνηση τους όσο και κατά την υπόλοιπη λειτουργία τους. Κάθε επεξεργαστής περιλαμβάνει λογισμικό που, μεταξύ άλλων, διαχειρίζεται τις περιφερειακές συσκευές που είναι προσαρτημένες στον δίαυλο PLB (Peripheral Local Bus). Σημαντικό επίσης είναι να υπενθυμισθεί η δυνατότητα άμεσης επικοινωνίας μεταξύ των δυο επεξεργαστών μέσω του διαύλου FSL(Fast Simplex Link).



Σχήμα 5.4.1 Γενική Αναφορά στο Λογισμικό του Ενσωματωμένου Συστήματος

Ο **Microblaze 0** κατά την εκκίνηση του αρχικοποιεί τα περιφερειακά του και ενεργοποιεί τα απαραίτητα interrupts. Τα περιφερειακά αυτά είναι η σειριακή θύρα, που χρησιμοποιείται για την επικοινωνία μεταξύ του επεξεργαστή και του modem, τα GPIO Modules, που διαχειρίζονται τα σήματα του modem, και ο δίαυλος FSL, μέσα από τον οποίο υπάρχει άμεση επικοινωνία με τον **Microblaze 1**. Ενεργοποιούνται επίσης και τα αντίστοιχα interrupts που προκαλούνται από αυτά τα περιφερειακά ώστε να είναι σε θέση ο επεξεργαστής να διαχειριστεί οποιοδήποτε γεγονός, όπως η λήψη δεδομένων από τη σειριακή θύρα αλλά και η γνωστοποίηση της κατάστασης του modem (συνδεδεμένο, σε ετοιμότητα κλπ) μέσω των σημάτων του.

Στη συνέχεια του κώδικα ελέγχεται η κατάσταση του modem διαβάζοντας τα σήματα που λαμβάνουν τα GPIO Modules. Ανάλογα με την κατάσταση του modem, ο επεξεργαστής προβαίνει στις απαραίτητες ενέργειες, στέλνοντας τις κατάλληλες εντολές μέσω της σειριακή θύρας στο modem, ώστε να πραγματοποιηθεί σύνδεση με τον **Server**. Σε αυτό το σημείο το σύστημα αναμένει να προκληθεί κάποιο interrupt, μέσω των σημάτων του modem, που επιβεβαιώνει ότι έχει δημιουργηθεί επιτυχής σύνδεση. Μόλις επιτευχθεί σύνδεση ο Microblaze 0 αναμένει για interrupts είτε από τη σειριακή θύρα, δηλαδή εισερχόμενα δεδομένα από τον Server, είτε από τον δίαυλο FSL ,δηλαδή δεδομένα που πρέπει να σταλθούν προς τον Server. Σε περίπτωση που προκληθεί interrupt από την σειριακή θύρα ξεκινάει μια ρουτίνα που λαμβάνει τα δεδομένα από αυτή. Στη συνέχεια τα αντιγράφει σε συγκεκριμένη θέση στη μνήμη SRAM και ενημερώνει μέσω του διαύλου FSL τον Microblaze 1 ώστε να τα διαβάσει. Σε περίπτωση που ο Microblaze 0 δεχτεί interrupt από τον δίαυλο FSL αντιγράφει δεδομένα από τη μνήμη SRAM (τα οποία έχουν σταλεί από τον Microblaze 1) και σε επόμενο βήμα τα γράφει στη σειριακή θύρα ώστε να τα λάβει το modem και να τα στείλει στον Server.Καθόλη τη διάρκεια λειτουργίας του επεξεργαστή υπάρχει η πιθανότητα να προκληθεί interrupt από τα σήματα του modem. Αυτό συνεπάγεται με κάποια αλλαγή στην κατάσταση του, όπως αποσύνδεση από τον Server ή απρόσμενη αποτυχία που θέτει το modem μη έτοιμο. Σε τέτοιες περιπτώσεις εξετάζεται η κατάσταση του modem ώστε να εκτελεστούν οι απαραίτητες ενέργειες για την αυτόματη επανασύνδεση του συστήματος με τον Server.

Αντίστοιχα και ο **Microblaze 1** κατά την εκκίνηση του αρχικοποιεί τα περιφερειακά που χρησιμοποιεί. Σε αυτά ανήκει ένας ελεγκτής σειριακού πρωτοκόλλου RS-232, ένα GPIO Module και ένας ελεγκτής επικοινωνίας I^2C τα οποία αφορούν οποιεσδήποτε συσκευές προσαρτώνται στο σύστημα και επικοινωνούν με ένα από τα πρωτόκολλα που υποστηρίζουν τα περιφερειακά αυτά.

Αρχικοποιείται επίσης ο δίαυλος FSL. Ο **Microblaze 1** μπορεί να δεχτεί interrupts από το GPIO Module τον ελεγκτή σειριακού πρωτοκόλλου RS-232 και από τον δίαυλο FSL. Κατά τη εκκίνηση του, βέβαια, ενεργοποιούνται τα interrupts μόνο του διαύλου καθώς είναι ζωτικής σημασίας για την επικοινωνία με τον **Microblaze 0**. Τα άλλα δυο interrupts μπορούν να ενεργοποιηθούν μόνο κατά επιλογή του χρήστη.

Στη συνέχεια ο **Microblaze 1** αναμένει να προκληθεί γεγονός από τον δίαυλο FSL που σηματοδοτεί την ύπαρξη εισερχόμενων δεδομένων προς επεξεργασία. Σε περίπτωση που υπάρξει κάποιο interrupt εκτελείται μια ρουτίνα που αντιγράφει τα δεδομένα από την SRAM σε έναν πίνακα χαρακτήρων. Στη συνέχεια διαβάζεται η κεφαλίδα του μηνύματος και αποφασίζεται τι πρέπει να εκτελεστεί στο επόμενο βήμα. Σύμφωνα με τις απαιτήσεις τις κεφαλίδας, μπορεί να γραφτούν τα καθαρά δεδομένα του μηνύματος (χωρίς την κεφαλίδα) σε συγκεκριμένη προσαρτημένη συσκευή ή να αντληθούν δεδομένα από αυτή. Μπορεί να τροποποιηθεί η κατάσταση των ακροδεκτών του GPIO Module από είσοδο σε έξοδο και αντίστροφα ή να ενεργοποιηθούν/απενεργοποιηθούν τα interrupts που προαναφέρθηκαν στην προηγούμενη παράγραφο.

Μόλις η ρουτίνα ολοκληρώσει την εκτέλεση της ο επεξεργαστής θα συνεχίσει να αναμένει για νέα interrupts. Αν αυτή την φορά έχουν ενεργοποιηθεί επιπλέον interrupts θα αναμένει και για αυτά και με ένα νέο γεγονός θα εκτελέσει την αντίστοιχη ρουτίνα να τα εξυπηρετήσει. Σε περίπτωση που προκληθεί γεγονός είτε από το GPIO Module είτε από τον ελεγκτή RS-232, αυτόματα, συλλέγονται τα δεδομένα από αυτά, αντιγράφονται στη μνήμη SRAM και στη συνέχεια ενημερώνεται ο **Microblaze 0** μέσω του διαύλουFSL ώστε να τα προωθήσει μέσω του modem στον Server.

5.4.2 Ανάλυση του κώδικα στο Ενσωματωμένο Σύστημα

Σε αυτό το υποκεφάλαιο θα αναλυθεί διεξοδικά ο κώδικας τόσο του **Microblaze 0** όσο και του **Microblaze 1**. Θα εξηγηθεί ο ρόλος κάθε συνάρτησης και μεταβλητής σαν υποσύνολο της εκτέλεσης από την εκκίνηση του επεξεργαστή και καθόλη τη διάρκεια λειτουργίας του. Αρχικά θα αναλυθεί ο **Microblaze 0** και στη συνέχεια ο **Microblaze 1.**

5.4.2.1 Ο κώδικας στον Microblaze 0

Ο **Microblaze 0** ξεκινάει την εκτέλεση από την συνάρτηση **main**. Καλείται μέσα από αυτή η συνάρτηση **wireless_interface** η οποία με την σειρά της καλεί την **setup_system**. Αυτές είναι τρεις από τις συναρτήσεις που θα περιγραφούν παρακάτω. Μετά την εκτέλεση αυτών ο επεξεργαστής αναμένει για οποιοδήποτε γεγονός μπορεί να προκαλέσει interrupts και ανάλογα με την πηγή που το προκάλεσε καλείται η αντίστοιχη συνάρτηση μεταξύ των **rs232_interrupt_handler**, **fsl_incoming_handler** και **dcd_dsr_cts_interrupt_handler** για να εξυπηρετήσει το γεγονός. Αυτές είναι τρεις ακόμα από τις σημαντικές συναρτήσεις που θα αναλυθούν. Πέραν αυτών θα περιγραφούν και οι **sender** και **modem_handler** που ολοκληρώνουν το σύνολο των συναρτήσεων που απαρτίζουν τον **Microblaze 0**.

5.4.2.1.1 Η συνάρτηση main

Ο κώδικας όπως ορίζει η C εκκινείται από την συνάρτηση **main**. Η **main** χρησιμοποιείται, μόνο, για να καλέσει την συνάρτηση **wireless_interface** προς εκτέλεση.

5.4.2.1.2 Η συνάρτηση wireless_interface

Η **wireless_interface**, αρχικά, τυπώνει το μήνυμα "Starting System" με χρήση της εντολής **print** (εικόνα 5.4.1 γραμμή 235), υποδηλώνοντας την εκκίνηση του συστήματος . Στη συνέχεια καλείται η συνάρτηση **setup_system** (εικόνα 5.4.1 γραμμή 236), η οποία θα εξηγηθεί στην επόμενη παράγραφο, που αρχικοποιεί τα περιφερειακά του συστήματος και ενεργοποιεί τα interrupts. Σε επόμενο βήμα διαβάζεται η λογική κατάσταση των ακροδεκτών του GPIO Module, με όνομα **dcd_dsr_cts**, και αποθηκεύεται ως ακέραιος αριθμός στη μεταβλητή **read_dcd_dsr_cts** (εικόνα 5.4.1 γραμμή 238). Η μεταβλητή αυτή, έπειτα, εξετάζεται από τέσσερεις συνθήκες **if** που, ανάλογα με τη τιμή της, παραπέμπει σε ανάλογη εντολή. Το συγκεκριμένο **GPIO Module** έχει πλάτος 3 bit και είναι συνδεδεμένο με τρία σήματα του modem τα οποία ενδεικνύουν την κατάσταση του modem. Συνεπώς, η τιμή που συγκρατεί η πιο πάνω μεταβλητή δίνει πληροφορίες όσον αφορά το αν το modem είναι έτοιμο, αν είναι συνδεδεμένο στο δίκτυο GPRS (συνεπώς συνδεδεμένο στο internet), ή αν έχει ήδη συνδεθεί με τον Server. Τα σήματα αυτά είναι τα CTS, DCD, και DSR. Το CTS πληροφορεί για την ετοιμότητα του modem για επικοινωνία. Το ενεργό DCD δείχνει αν το modem είναι συνδεδεμένο στο GPRS δίκτυο και το DSR αν έχει συνδεθεί με τον Server.

Στην πρώτη συνθήκη **if** (εικόνα 5.4.1 γραμμή 240) εξετάζεται αν η μεταβλητή έχει τιμή 7 πράγμα που σημαίνει πως και τα τρία σήματα είναι ανενεργά. Σε μια τέτοια περίπτωση το modem, προφανώς, δεν είναι σε ετοιμότητα. Έτσι, αν αυτή η συνθήκη είναι αληθής, τυπώνεται αρχικά το μήνυμα "Modem is not ready.Waiting for Modem…" αντίστοιχα με χρήση της εντολής **print** (εικόνα 5.4.1 γραμμή 242). Στη συνέχεια αποθηκεύεται η τιμή 7 στη μεταβλητή **get_dcd_dsr_cts_logic_previous_state** (εικόνα 5.4.1 γραμμή 243), η οποία συγκρατεί την τελευταία κατάσταση του modem και της οποίας η χρησιμότητα θα εξηγηθεί παρακάτω.

Στην δεύτερη συνθήκη **if** (εικόνα 5.4.1 γραμμή 245) εξετάζεται αν η μεταβλητή έχει τιμή 6. Σε αυτή την περίπτωση είναι ενεργό μόνο το σήμα CTS που σημαίνει πως το modem είναι σε ετοιμότητα. Αν εκτελεστεί ο κώδικας εσωτερικά αυτής της **if** αρχικά θα τυπωθεί το μήνυμα "Modem is ready.Attempting to attach on GPRS network" (εικόνα 5.4.1 γραμμή 247). Έπειτα καλείται η συνάρτηση **sender** (εικόνα 5.4.1 γραμμή 248) που θα στείλει τα περιεχόμενα της μεταβλητής **gprs_connection_start** μέσω της σειριακής θύρας στο modem. Η μεταβλητή αυτή περιέχει την συμβολοσειρά **"AT#CONNECTIONSTART"** που στην ουσία είναι μια εντολή, αναγνωρίσιμη από το modem, που εκκινεί τις απαραίτητες λειτουργίες για να συνδεθεί το modem στο δίκτυο GPRS. Στο επόμενο βήμα η μεταβλητή **get_dcd_dsr_cts_logic_previous_state** αποκτάει τιμή 6 (εικόνα 5.4.1 γραμμή 249).

Η τρίτη συνθήκη **if** (εικόνα 5.4.1 γραμμή 251) εξετάζει αν η μεταβλητή **read_dcd_dsr_cts** έχει τιμή 2. Κάτι τέτοιο σημαίνει πως είναι ενεργό τόσο το σήμα CTS όσο και το DCD συνεπώς το modem είναι σε ετοιμότητα και έχει ήδη συνδεθεί στο δίκτυο GPRS. Κατά την εκτέλεση του κώδικα αυτής της **if** στην πρώτη εντολή τυπώνεται το μήνυμα "Modem is already attached on GPRS Network. Attempting to establish server connection…" (εικόνα 5.4.1 γραμμή 253). Στη συνέχεια καλείται η συνάρτηση **sender** (εικόνα 5.4.1 γραμμή 254) που αυτή τη φορά θα στείλει την συμβολοσειρά **"AT#OTCP=1"** της μεταβλητής **server_connect**. Η συμβολοσειρά αυτή, ως εντολή, αναγκάζει το modem να επιχειρήσει σύνδεση με τον server. Έπειτα η μεταβλητή **get_dcd_dsr_cts_logic_previous_state** αποκτάει τιμή 2 (εικόνα 5.4.1 γραμμή 255).

Στην τέταρτη συνθήκη **if** (εικόνα 5.4.1 γραμμή 257) εξετάζεται αν η μεταβλητή **read_dcd_dsr_cts** έχει τιμή 0. Σε μια τέτοια περίπτωση και τα τρία σήματα είναι ενεργά επομένως το modem είναι σε ετοιμότητα και έχει ήδη συνδεθεί τόσο στο δίκτυο GPRS όσο και στον Server. Αυτό σημαίνει πως είναι εφικτή η ανταλλαγή δεδομένων μεταξύ του ενσωματωμένου συστήματος και του Server. Αν η συνθήκη είναι αληθής θα τυπωθεί αρχικά το μήνυμα "Modem is already connected to Server. System is ready for data transfer" (εικόνα 5.4.1 γραμμή 259). Στη συνέχεια θα αποθηκευτεί η τιμή 0 στη μεταβλητή **get_dcd_dsr_cts_logic_previous_state** (εικόνα 5.4.1 γραμμή 260).

Αφού εξεταστούν και οι τέσσερεις συνθήκες **if**, και αφού βρεθεί οπωσδήποτε μια και μόνο μια αληθής, η συνάρτηση **wireless_interface** θα εκκινήσει έναν βρόγχο **while** (εικόνα 5.4.1 γραμμή 263) που θα εκτελείται επ' άπειρον (στην πραγματικότητα καθόλη τη διάρκεια λειτουργίας του ενσωματωμένου συστήματος.

```
235  print("Starting System\r\n");
236  setup_system();
237
238  read_dcd_dsr_cts=XGpio_DiscreteRead(&dcd_dsr_cts, 1);
239
240  if(read_dcd_dsr_cts==7)
241     {
242     print("Modem is not ready\r\nWaiting for Modem...\r\n");
243     get_dcd_dsr_cts_logic_previous_state=7;
244     }
245  if(read_dcd_dsr_cts==6)
246     {
247     print("Modem is ready\r\nAttempting to attach on GPRS network...\r\n");
248     sender(gprs_connection_start,strlen(gprs_connection_start));
249     get_dcd_dsr_cts_logic_previous_state=6;
250     }
251  if(read_dcd_dsr_cts==2)
252     {
253     print("Modem is already attached on GPRS Network\r\nAttempting to establish server connection...\r\n");
254     sender(server_connect,strlen(server_connect));
255     get_dcd_dsr_cts_logic_previous_state=2;
256     }
257  if(read_dcd_dsr_cts==0)
258     {
259     print("Modem is already connected to Server\r\nSystem is ready for data transfer\r\n");
260     get_dcd_dsr_cts_logic_previous_state=0;
261     }
262
263  while(1)
264  {
265
266  }
```

Εικόνα 5.4.1 Η συνάρτηση wireless_interface

5.4.2.1.3 Η συνάρτηση setup_system

Όπως έχει προαναφερθεί η **setup_system** καλείται από την **wireless_interface**. Σκοπός της είναι η αρχικοποίηση των περιφερειακών και η ενεργοποίηση των interrupts. Αρχικά εκτελούνται οι εντολές αρχικοποίησης των τεσσάρων περιφερειακών του **Microblaze 0** (εικόνα 5.4.2). Πρώτα αρχικοποιείται ο ελεγκτής των interrupts (**xps_intc_0**) από την εντολή **XIntc_Initialize**. Στη συνέχεια ο ελεγκτής του σειριακού πρωτοκόλλου RS-232 (**hspa_rs232**), που χρησιμοποιείται για την επικοινωνία με το modem ,με χρήση της **XUartlite_Initialize**. Έπειτα αρχικοποιείται το GPIO Module (**dtr_rts**), πλάτους 2 bit, που χειρίζεται τα σήματα DTR και RTS αλλά και το GPIO Module (**dcd_dsr_cts**) που εξετάζει τα σήματα CTS, DCD και DCR. Και τα δυο τελευταία περιφερειακά αρχικοποιούνται από την εντολή **XGpio_Initialize**. Σε αυτό το σημείο να υπενθυμισθεί πως μέσω του σήματος RTS το σύστημα μπορεί να επιτρέψει ή αποτρέψει την είσοδο δεδομένων από το modem σε περιπτώσεις που το πρώτο δεν είναι σε θέση να δεχτεί δεδομένα, ενώ με το σήμα DTR μπορεί να τερματίσει μια σύνδεση με τον Server.

```
182  XIntc_Initialize(&InterruptController, XPAR_XPS_INTC_0_DEVICE_ID);
183  XUartLite_Initialize(&uartLite,XPAR_HSPA_RS232_DEVICE_ID);
184  XGpio_Initialize(&dtr_rts, XPAR_DTR_RTS_DEVICE_ID);
185  XGpio_Initialize(&dcd_dsr_cts, XPAR_DCD_DSR_CTS_DEVICE_ID);
```

Εικόνα 5.4.2 Εντολές αρχικοποίησης των περιφερειακών του Microblaze 0

Στη συνέχεια εκτέλεσης του κώδικα εσωτερικά της **setup_system** καλείται η εντολή **XIntc_RegisterHandler** που ορίζει για κάθε περιφερειακό που μπορεί να προκαλέσει interrupt ποια συνάρτηση θα εξυπηρετήσει το αντίστοιχο interrupt. Για το περιφερειακό **hspa_rs232** η συνάρτηση που θα κληθεί ύστερα από κάποιο γεγονός είναι η **rs232_interrupt_handler** (εικόνα 5.4.3 γραμμή 187). Για το GPIO Module **dcd_dsr_cts** η αντίστοιχη συνάρτηση είναι η **dcd_dsr_cts_interrupt_handler** (εικόνα 5.4.3 γραμμή 192) ενώ για τον δίαυλο FSL η **fsl_incoming_handler** (εικόνα 5.4.3 γραμμή 197).

```
187  XIntc_RegisterHandler(XPAR_XPS_INTC_0_BASEADDR,
188                        XPAR_XPS_INTC_0_HSPA_RS232_INTERRUPT_INTR,
189                        rs232_interrupt_handler,
190                        (void *)XPAR_HSPA_RS232_BASEADDR);
191
192  XIntc_RegisterHandler(XPAR_XPS_INTC_0_BASEADDR,
193                        XPAR_XPS_INTC_0_DCD_DSR_CTS_IP2INTC_IRPT_INTR,
194                        dcd_dsr_cts_interrupt_handler,
195                        (void *)XPAR_DCD_DSR_CTS_BASEADDR);
196
197  XIntc_RegisterHandler(XPAR_XPS_INTC_0_BASEADDR,
198                        XPAR_XPS_INTC_0_FSL_MB1_TO_MB0_FSL_HAS_DATA_INTR,
199                        fsl_incoming_handler,(void *)0);
```

Εικόνα 5.4.3 Ορισμός των συναρτήσεων που εξυπηρετούν τα interrupts στον Microblaze 0

Αφού ολοκληρωθεί η εκτέλεση των εντολών **XIntc_RegisterHandler** καλείται η **XUartLite_ResetFifos** που καθαρίζει την ουρά **fifo** του ελεγκτή **hspa_rs232** (εικόνα 5.4.4 γραμμή 201). Έπειτα με χρήση βρόγχου **for** (εικόνα 5.4.4 γραμμή 203) μηδενίζονται οι πίνακες **receive_buffer** και **send_buffer** που συγκρατούν τα δεδομένα που λαμβάνονται και στέλνονται μέσω της σειριακής θύρας. Ακολουθεί η ενεργοποίηση των interrupts (εικόνα 5.4.4 γραμμές 209 έως 215). Στην περίπτωση του **Microblaze 0** είναι αναγκαίο να έχουν ενεργοποιηθεί όλα τα interrupts έτσι ώστε το σύστημα να είναι σε θέση να λάβει άμεσα δεδομένα από την σειριακή θύρα, να μπορεί να γνωρίζει την κατάσταση των σημάτων στο modem οποτεδήποτε αυτή αλλάζει αλλά και να μπορεί να ενημερώνεται μέσω του διαύλου FSL για δεδομένα από τον **Microblaze 1**. Στη συνέχεια ορίζονται οι ακροδέκτες του GPIO Module **dtr_rts** σε έξοδο και του **dcd_dsr_cts** σε είσοδο (εικόνα 5.4.4 γραμμές 218 και 219). Ορίζεται επίσης η κατάσταση των σημάτων στο **dtr_rts** σε λογικό 0 (εικόνα 5.4.4 γραμμή 220) πληροφορώντας με αυτόν τον τρόπο το modem πως το σύστημα είναι σε ετοιμότητα να δεχτεί δεδομένα από αυτό. Πριν ολοκληρωθεί η εκτέλεση της **setup_system** τυπώνεται το μήνυμα “System is Initialized” ενημερώνοντας έτσι τον χειριστή του ενσωματωμένου συστήματος πως έχει ολοκληρωθεί η αρχικοποίηση των περιφερειακών (εικόνα 5.4.4 γραμμή 222).

```
201  XUartLite_ResetFifos(&uartLite);
202
203  for (reset_count = 0; reset_count < 128; reset_count++)
204     {
205     receive_buffer[reset_count] = 0;
206     send_buffer[reset_count] = 0;
207     }
208  XGpio_WriteReg(XPAR_DCD_DSR_CTS_BASEADDR, XGPIO_IER_OFFSET, XGPIO_IR_CH1_MASK) ;
209  XGpio_InterruptEnable(&dcd_dsr_cts, XPAR_DCD_DSR_CTS_IP2INTC_IRPT_MASK);
210  XGpio_InterruptGlobalEnable(&dcd_dsr_cts);
211  XUartLite_EnableIntr(XPAR_HSPA_RS232_BASEADDR);
212  interrupt_mask=XPAR_HSPA_RS232_INTERRUPT_MASK+XPAR_FSL_MB1_TO_MB0_FSL_HAS_DATA_MA
213  XIntc_EnableIntr(XPAR_XPS_INTC_0_BASEADDR,interrupt_mask);
214  XIntc_MasterEnable(XPAR_XPS_INTC_0_BASEADDR);
215  microblaze_enable_interrupts();
216  XIntc_Start(&InterruptController, XIN_REAL_MODE);
217
218  XGpio_SetDataDirection(&dtr_rts, 1, 0);
219  XGpio_SetDataDirection(&dcd_dsr_cts, 1, 7);
220  XGpio_DiscreteWrite(&dtr_rts, 1, 0);
221
222  print("System is Initialized\r\n");
```

Εικόνα 5.4.4 Ενεργοποίηση των interrupts στον Microblaze 0

5.4.2.1.4 Η συνάρτηση sender

Η συνάρτηση αυτή, που καλείται πρώτη φορά μέσα στην **wireless_interface**, έχει ως σκοπό την αποστολή δεδομένων μέσω της σειριακής θύρας στο modem. Η φύση των δεδομένων μπορεί να διαφέρει, καθώς, άλλοτε πρόκειται για συμβολοσειρές που λειτουργούν ως εντολές που πρέπει να εκτελέσει το modem και άλλοτε πρόκειται για δεδομένα που πρέπει να παραδοθούν στον Server. Παρόλα αυτά, η συνάρτηση **sender** αρκείται στην αποστολή των δεδομένων χωρίς να χρειάζεται να γνωρίζει τον σκοπό τους. Η συνάρτηση αυτή δέχεται ως ορίσματα έναν δείκτη τύπου χαρακτήρα, που παραπέμπει στη συμβολοσειρά που πρέπει να αποσταλεί, και έναν ακέραιο που ενημερώνει σχετικά με το μήκος της συμβολοσειράς.

Μόλις κληθεί η **sender** εκκινεί έναν βρόγχο for (εικόνα 5.2.5 γραμμή 53). Ο βρόγχος αυτός σε κάθε επανάληψή του καλεί την εντολή **XUartLite_SendByte** (εικόνα 5.2.5 γραμμή 55) που στέλνει έναν προς έναν τους χαρακτήρες της συμβολοσειράς έως ότου αποσταλθούν όλοι οι χαρακτήρες.

```
53  for(i=0;i<length;i++)
54      {
55      XUartLite_SendByte(XPAR_HSPA_RS232_BASEADDR,buffer[i]);
56      }
```

Εικόνα 5.4.5 Η συνάρτηση sender

5.4.2.1.5 Η συνάρτηση dcd_dsr_cts_interrupt_handler

Η συνάρτηση αυτή καλείται οποτεδήποτε προκληθεί interrupt από το GPIO Module **dcd_dsr_cts** που σηματοδοτεί ότι έχει υπάρξει κάποια αλλαγή στην κατάσταση του modem. Σκοπός λοιπόν αυτής της συνάρτησης είναι να χειριστεί ανάλογα την αλλαγή στην κατάσταση του modem ώστε αυτό να συνδεθεί και πάλι με τον Server. Έτσι μόλις κληθεί αρχικά απενεργοποιεί τα interrupts του συγκεκριμένου GPIO Module (εικόνα 5.4.6 γραμμή 107) και στη συνέχεια εκτελεί την συνάρτηση **modem_handler** (εικόνα 5.4.6 γραμμή 109). Στην ουσία η **modem_handler** είναι αυτή που θα χειριστεί το modem. Η απενεργοποίηση των interrupts στο GPIO Module είναι καθοριστικής σημασίας καθώς προστατεύει την συνάρτηση **modem_handler** από ανεπιθύμητη διακοπή που μπορεί να προκληθεί κατά την διάρκεια εκτέλεσης της. Αφού η εκτέλεση του επεξεργαστή επιστρέψει στην συνάρτηση **dcd_dsr_cts_interrupt_handler** στο επόμενο βήμα θα ενεργοποιηθούν και πάλι τα interrupts (εικόνα 5.4.6 γραμμή 113) για το GPIO Module **dcd_dsr_cts**.

```
107 XGpio_InterruptDisable(&dcd_dsr_cts, XPAR_DCD_DSR_CTS_IP2INTC_IRPT_MASK);
108
109 modem_handler();
110
111 (void)XGpio_InterruptClear(&dcd_dsr_cts, XPAR_DCD_DSR_CTS_IP2INTC_IRPT_MASK);
112 XGpio_WriteReg(XPAR_DCD_DSR_CTS_BASEADDR, XGPIO_IER_OFFSET, XGPIO_IR_CH1_MASK) ;
113 XGpio_InterruptEnable(&dcd_dsr_cts, XPAR_DCD_DSR_CTS_IP2INTC_IRPT_MASK);
```

Εικόνα 5.4.6 Η συνάρτηση dcd_dsr_cts_interrupt_handler

5.4.2.1.6 Η συνάρτηση rs232_interrupt_handler

Η συνάρτηση αυτή καλείται οποτεδήποτε προκαλείται interrupt από εισερχόμενα δεδομένα στην ουρά του ελεγκτή RS-232. Σκοπός της είναι να λάβει τα εισερχόμενα δεδομένα που προέρχονται από το modem και στη συνέχεια να προβεί στις καταλληλότερες ενέργειες ανάλογα με το είδος των δεδομένων.

Κατά την κλήση της **rs232_interrupt_handler** ξεκινάει ένας βρόγχος **while** (εικόνα 5.4.7 γραμμή 149) ο οποίος επαναλαμβάνεται όσο η εντολή **XUartLite_IsReceiveEmpty** παραμένει ψευδής, με άλλα λόγια όσο ο ελεγκτής RS-232 συνεχίζει να λαμβάνει χαρακτήρες. Το σύνολο του κώδικα που εκτελείται σε αυτή τη συνάρτηση βρίσκεται εσωτερικά του βρόγχου αυτού. Αρχικά κατά την εκτέλεση του βρόγχου λαμβάνεται ένας χαρακτήρας με χρήση της εντολής **XUartLite_RecvByte** και αποθηκεύεται στη μεταβλητή **c** τύπου χαρακτήρα (εικόνα 5.4.7 γραμμή 151). Στη συνέχεια ο χαρακτήρας που βρίσκεται στην μεταβλητή **c** αποθηκεύεται στην τρέχουσα θέση του πίνακα χαρακτήρων **receive_buffer** (εικόνα 5.4.7 γραμμή 152) η οποία καθορίζεται από την ακέραια μεταβλητή count. Έπειτα η μεταβλητή **count** αυξάνεται κατά ένα (εικόνα 5.4.7 γραμμή 153) ώστε την επόμενη φορά να δείξει στην επόμενη θέση του πίνακα. Μετά την ολοκλήρωση των τριών τελευταίων βημάτων εκτελείται μια συνθήκη **if** (εικόνα 5.4.7 γραμμή 154) που εξετάζει αν ο τελευταίος χαρακτήρας που βρίσκεται στη μεταβλητή **c** αντιστοιχεί στον δέκατο χαρακτήρα του κώδικα ASCII. Ο δέκατος χαρακτήρας σηματοδοτεί την ολοκλήρωση μιας συμβολοσειράς. Όσο επαναλαμβάνεται αυτός ο βρόγχος εκτελούνται κάθε φορά τα 4 βήματα που αναφέρθηκαν ως τώρα. Με αυτό τον τρόπο η συνάρτηση συνεχίζει να λαμβάνει χαρακτήρες, συνεπώς τα εισερχόμενα δεδομένα, τους οποίους διαδοχικά τους αποθηκεύει στον πίνακα **receive_buffer**. Μόλις ληφθεί ο δέκατος χαρακτήρας του κώδικα ASCII η συνθήκη **if** γίνεται αληθής, σηματοδοτείται η ολοκλήρωση της ληφθείσας συμβολοσειράς και πλέον μπορεί να εκτελεστεί ο κώδικας εσωτερικά της **if.**

Αρχικά, εσωτερικά της συνθήκης **if** καλείται η εντολή **XGpio_DiscreteWrite** (εικόνα 5.4.7 γραμμή 156) που αλλάζει την κατάσταση του σήματος RTS σε λογικό ένα απενεργοποιώντας έτσι το σήμα RTS. Με αυτό τον τρόπο το ενσωματωμένο σύστημα αποτρέπει το modem από το να στείλει νέα δεδομένα έως ότου το σύστημα ολοκληρώσει την επεξεργασία των δεδομένων που έχουν ήδη ληφθεί. Στο επόμενο βήμα μηδενίζεται η τιμή της μεταβλητής count ώστε την επόμενη φορά που θα κληθεί η συνάρτηση **rs232_interrupt_handler** να αποθηκευτούν σωστά τα δεδομένα στον πίνακα **receive_buffer**. Παρακάτω θα εκτελεστεί μια εμφωλευμένη συνθήκη **if else if** που εξετάζει τα περιεχόμενα του πίνακα **receive_buffer.**

Συγκεκριμένα, η συνθήκη **if** (εικόνα 5.4.7 γραμμή 159) εξετάζει αν ο πρώτος χαρακτήρας αυτού του πίνακα βρίσκεται μεταξύ των χαρακτήρων 97 και 106 του κώδικα ASCII δηλαδή ανήκει μεταξύ των χαρακτήρων "**a**" και "**j**". Αυτό αυτομάτως σημαίνει πως τα δεδομένα του πίνακα πρέπει να παραδοθούν στον **Microblaze 1**. Συνεπώς αν αυτή η συνθήκη είναι αληθής, αρχικά, θα τυπωθεί, με χρήση της εντολής **xil_printf**, το μήνυμα "Received data from Client:" συνοδευόμενο από τα ληφθέντα δεδομένα (εικόνα 5.4.7 γραμμή 161). Στη συνέχεια αντιγράφονται τα δεδομένα του πίνακα (εικόνα 5.4.7 γραμμή 162), με χρήση της εντολής **strcpy**, σε συγκεκριμένη διεύθυνση της μνήμης SRAM. Έπειτα καλείται η εντολή **putfsl** (εικόνα 5.4.7 γραμμή 163) που στέλνει μέσω του διαύλου FSL προς τον **Microblaze 1** τον χαρακτήρα "**o**". Σκοπός εκτέλεσης της **putfsl** είναι να προκαλέσει interrupt στον **Microblaze 1** ώστε να γνωρίζει ο τελευταίος πως υπάρχουν νέα δεδομένα στη μνήμη SRAM προς επεξεργασία.

Η συνθήκη **else if** (εικόνα 5.4.7 γραμμή 165) εξετάζει αν τα δεδομένα του πίνακα είναι ένα προειδοποιητικό μήνυμα λάθους από το modem μεταξύ των "**NO CARRIER**", "**ERROR**" και "**#CME ERROR: 38016**". Οποιαδήποτε από τις τρεις αυτές περιπτώσεις συνεπάγεται ότι έχει υπάρξει κάποια αποτυχία εκτέλεσης στο modem είτε ύστερα από αποτυχημένη προσπάθεια σύνδεσης με το δίκτυο GPRS είτε ύστερα από αποτυχία σύνδεσης με τον Server. Αν αυτή η συνθήκη βρεθεί αληθής καλείται η συνάρτηση **modem_handler** (εικόνα 5.4.7 γραμμή 167) που θα χειριστεί αναλόγως την αποτυχία του modem.

Αφότου ολοκληρωθεί η συνθήκη **if else if** εκτελείται ένας βρόγχος **for** (εικόνα 5.4.7 γραμμή 169) με σκοπό να μηδενίσει τα περιεχόμενα του πίνακα **receive_buffer** ώστε να αποφευχθούν ατέλειες ή λάθη την επόμενη φορά που θα ληφθούν δεδομένα από το modem. Στο τελευταίο βήμα, πριν ολοκληρωθεί ο κύκλος της συνάρτησης **rs232_interrupt_handler**, καλείται η εντολή **XGpio_DiscreteWrite** (εικόνα 5.4.7 γραμμή 171) που αλλάζει την κατάσταση του σήματος RTS σε λογικό μηδέν ενεργοποιώντας έτσι το σήμα RTS. Με αυτό τον τρόπο το σύστημα ενημερώνει το modem πως πλέον είναι σε ετοιμότητα να δεχτεί νέα δεδομένα.

```
149  while (!XUartLite_IsReceiveEmpty(XPAR_HSPA_RS232_BASEADDR))
150     {
151     c = XUartLite_RecvByte(XPAR_HSPA_RS232_BASEADDR);
152     receive_buffer[count]=c;
153     count++;
154     if(c==10)
155        {
156        XGpio_DiscreteWrite(&dtr_rts, 1, 1);
157        count=0;
158        number=strlen(receive_buffer);
159        if(receive_buffer[0]>=97 && receive_buffer[0]<=106)
160           {
161           xil_printf("Received data from Client:%s",receive_buffer);
162           strcpy(data_to_mb1,receive_buffer);
163           putfsl('o',0);
164           }
165        else if(strcmp(receive_buffer,no_carrier)==0 || strcmp(receiv
166           {
167           modem_handler();
168           }
169        for(j=0;j<number+5;j++)
170           receive_buffer[j]=0;
171        XGpio_DiscreteWrite(&dtr_rts, 1, 0);
172        }
```

Εικόνα 5.4.7 Η συνάρτηση rs232_interrupt_handler

5.4.2.1.7 Η συνάρτηση modem_hander

Η **modem_hander** καλείται είτε από την συνάρτηση **dcd_dsr_cts_interrupt_handler** είτε από την **rs232_interrupt_handler**. Σκοπός της είναι να χειριστεί οποιαδήποτε αποτυχία προκληθεί στο modem ώστε το τελευταίο να μπορέσει αυτόματα να δημιουργήσει επιτυχή επανασύνδεση με τον server.

Αρχικά εκτελείται η εντολή **XGpio_DiscreteRead** (εικόνα 5.4.8 γραμμή 61) που αντλεί την λογική κατάσταση των σημάτων στο GPIO Module **dcd_dsr_cts** και την αποθηκεύει στην μεταβλητή **get_dcd_dsr_cts_logic_state**. Στη συνέχεια εκτελούνται τέσσερεις συνθήκες **if** που εξετάζουν αυτή τη μεταβλητή και κατά συνέπεια εξετάζουν την κατάσταση του modem.

Η πρώτη συνθήκη **if** ελέγχει αν η μεταβλητή **get_dcd_dsr_cts_logic_state** έχει τιμή 7. Σε μια τέτοια περίπτωση και τα τρία σήματα του modem έχουν γίνει ανενεργά ύστερα από κάποια αναπάντεχη αποτυχία όπως ξαφνική επανεκκίνηση του modem. Αν εκτελεστεί αυτή η συνθήκη θα τυπωθεί το μήνυμα "Error: Modem is not ready due to unexpected failure. Waiting for Modem…" ενημερώνοντας έτσι τον διαχειριστή για την αποτυχία (εικόνα 5.4.8 γραμμή 65). Στη συνέχεια αποθηκεύεται η τιμή 7 στη μεταβλητή **get_dcd_dsr_cts_logic_previous_state** (εικόνα 5.4.8 γραμμή 66). Όταν το modem επανέρθει σε κατάσταση ετοιμότητας θα ενεργοποιηθεί το σήμα CTS με αποτέλεσμα να δημιουργηθεί νέο interrupt που τελικά θα καλέσει ξανά την συνάρτηση **modem_handler** να προβεί στις απαραίτητες ενέργειες.

```
61  get_dcd_dsr_cts_logic_state=XGpio_DiscreteRead(&dcd_dsr_cts, 1);
62
63  if(get_dcd_dsr_cts_logic_state==7)
64     {
65     print("Error: Modem is not ready due to unexpected failure\r\nWaiting for Modem...\r\n");
66     get_dcd_dsr_cts_logic_previous_state=7;
```

Εικόνα 5.4.8 Η πρώτη συνθήκη if (συνάρτηση modem_handler)

Η δεύτερη συνθήκη **if** εξετάζει αν η μεταβλητή **get_dcd_dsr_cts_logic_state** έχει τιμή 6 (εικόνα 5.4.9 γραμμή 68). Αυτό σημαίνει πως μόνο το σήμα CTS είναι ενεργό επομένως το modem είναι σε ετοιμότητα και βρίσκεται στην αρχική του κατάσταση (δεν είναι συνδεδεμένο). Σε αυτό το σημείο είναι σημαντικό να γνωρίζει το σύστημα ποια ήταν η προηγούμενη κατάσταση του modem και εδώ θα φανεί η χρησιμότητα της μεταβλητής **get_dcd_dsr_cts_logic_previous_state**. Υπάρχουν δυο ενδεχόμενα σε αυτή την περίπτωση. Το modem στην προηγούμενη κατάστασή του είχε και τα τρία σήματα ανενεργά καθώς προέκυψε αναπάντεχη επανεκκίνηση του, επομένως η μεταβλητή **get_dcd_dsr_cts_logic_previous_state** έχει τιμή 7. Σε άλλη περίπτωση το modem στην προηγούμενη κατάσταση του ήταν συνδεδεμένο στο δίκτυο GPRS και είχε ενεργά τα σήματα CTS και DCD που σημαίνει πως η μεταβλητή **get_dcd_dsr_cts_logic_previous_state** είχε τιμή 2. Απρόσμενα έχασε την σύνδεση του και το σήμα DCD έγινε ανενεργό. Από τα παραπάνω προκύπτει ότι το modem μπορεί να βρεθεί σε κατάσταση ετοιμότητας από δυο προηγούμενες καταστάσεις. Αυτό απαιτεί για κάθε προηγούμενη κατάσταση συγκεκριμένο χειρισμό επομένως προκύπτουν δυο εμφωλευμένες συνθήκες **if** που εξετάζουν την τιμή της μεταβλητής **get_dcd_dsr_cts_logic_previous_state**. Η πρώτη εμφωλευμένη συνθήκη **if** (εικόνα 5.4.9 γραμμή 70). εξετάζει αν αυτή η μεταβλητή έχει τιμή 7 και αν αυτό βρεθεί αληθές τυπώνεται το μήνυμα "Modem is ready.Re-Attempting to attach". Αν η μεταβλητή **dcd_dsr_cts_logic_previous_state** έχει τιμή 2 θα εκτελεστεί η δεύτερη εμφωλευμένη συνθήκη **if** (εικόνα 5.4.9 γραμμή 74). που θα τυπώσει το μήνυμα "Error: Modem lost GPRS network connection or failed to attach.Re-attempting to attach". Αφού ελεγχθούν και οι δυο συνθήκες καλείται η συνάρτηση **sender** (εικόνα 5.4.9 γραμμή 78). που θα στείλει τα περιεχόμενα του πίνακα **gprs_connection_start** μέσω της σειριακής θύρας στο modem. Όπως έχει προαναφερθεί στην περιγραφή της συνάρτησης **wireless_interface** τα περιεχόμενα του πίνακα **gprs_connection_start** είναι η συμβολοσειρά "**AT#CONNECTIONSTART**" που, ως εντολή, αναγκάζει το modem να επιχειρήσει σύνδεση με το δίκτυο GPRS. Στη συνέχεια αποθηκεύεται η τιμή 6 στη μεταβλητή **dcd_dsr_cts_logic_previous_state**(εικόνα 5.4.9 γραμμή 79) ώστε να είναι το σύστημα πάντα ενήμερο για την προηγούμενη κατάσταση του modem.

```
68  if(get_dcd_dsr_cts_logic_state==6)
69     {
70     if(get_dcd_dsr_cts_logic_previous_state==7)
71        {
72        print("Modem is ready\r\nRe-Attempting to attach\r\n");
73        }
74     if(get_dcd_dsr_cts_logic_previous_state==2)
75        {
76        print("Error: Modem lost GPRS network connection or failed to attach\r\nRe-Attempting to attach\r\n");
77        }
78     sender(gprs_connection_start,strlen(gprs_connection_start));
79     get_dcd_dsr_cts_logic_previous_state=6;
```

Εικόνα 5.4.9 Η δεύτερη συνθήκη if (συνάρτηση modem_handler)

Η τρίτη συνθήκη **if** ελέγχει αν η μεταβλητή **get_dcd_dsr_cts_logic_state** έχει τιμή 2 (εικόνα 5.4.10 γραμμή 81) επομένως το modem είναι ήδη συνδεδεμένο στο δίκτυο GPRS και είναι ενεργά τα σήματα CTS και DCD. Σε αυτή την περίπτωση το επόμενο βήμα είναι να πραγματοποιηθεί σύνδεση με τον Server. Όμως όπως και στη δεύτερη συνθήκη **if** διακρίνονται δυο πιθανές προηγούμενες καταστάσεις που απαιτούν συγκεκριμένο χειρισμό. Έτσι εκτελούνται και εδώ δυο εμφωλευμένες συνθήκες **if** που, προφανώς, ελέγχουν την μεταβλητή **dcd_dsr_cts_logic_previous_state**. Η πρώτη εμφωλευμένη **if** (εικόνα 5.4.10 γραμμή 83) ελέγχει αν η μεταβλητή έχει τιμή 6. Αυτό σημαίνει πως το modem προηγουμένως ήταν σε ετοιμότητα και στην αρχική του κατάσταση επομένως τυπώνεται το μήνυμα "Succesfully attached on GPRS network. Attempting to establish server connection" (εικόνα 5.4.10 γραμμή 85). Αν η τιμή της **dcd_dsr_cts_logic_previous_state** είναι 0 συνεπάγεται πως στην προηγούμενη κατάσταση του το modem είχε και τα τρία σήματα ενεργά και προφανώς ήταν συνδεδεμένο με τον Server. Σε αυτή την περίπτωση θα εκτελεστεί η δεύτερη εμφωλευμένη **if** (εικόνα 5.4.10 γραμμή 87) που θα τυπώσει το μήνυμα "Server connection lost. Re-attempting to connect" (εικόνα 5.4.10 γραμμή 89). Στη συνέχεια, αφού εκτελεστούν και οι δυο εμφωλευμένες **if** καλείται η συνάρτηση **sender** (εικόνα 5.4.10 γραμμή 91) που θα στείλει τα περιεχόμενα του πίνακα **server_connect**. Ο πίνακας αυτός περιέχει την συμβολοσειρά "**AT#OTCP=1**" που αναγκάζει το modem να επιχειρήσει σύνδεση με τον server. Πριν ολοκληρώσει τον κύκλο της η τρίτη συνθήκη **if** αποθηκεύει την τιμή 2 στην μεταβλητή **dcd_dsr_cts_logic_previous_state** (εικόνα 5.4.10 γραμμή 92).

```
81  if(get_dcd_dsr_cts_logic_state==2)
82     {
83     if(get_dcd_dsr_cts_logic_previous_state==6)
84        {
85        print("Succesfully attached on GPRS network\r\nAttempting to establish server connection\r\n");
86        }
87     if(get_dcd_dsr_cts_logic_previous_state==0)
88        {
89        print("Server connection lost\r\nRe-attempting to connect\r\n");
90        }
91     sender(server_connect,strlen(server_connect));
92     get_dcd_dsr_cts_logic_previous_state=2;
```

Εικόνα 5.4.10 Η τρίτη συνθήκη if (συνάρτηση modem_handler)

Η τέταρτη συνθήκη **if** ελέγχει αν η μεταβλητή **get_dcd_dsr_cts_logic_state** έχει τιμή 0 (εικόνα 5.4.11 γραμμή 94) που συνεπάγεται πως όλα τα σήματα είναι ενεργά και το modem είναι συνδεδεμένο με το GPRS δίκτυο αλλά και με τον Server. Σε αυτή την περίπτωση δεν πραγματοποιούνται περαιτέρω ενέργειες καθώς έχει ήδη ικανοποιηθεί η απαίτηση για σύνδεση, έτσι απλώς τυπώνεται το μήνυμα "Succesfully connected to server"(εικόνα 5.4.11 γραμμή 96) και αποθηκεύεται η τιμή 0 στην μεταβλητή **dcd_dsr_cts_logic_previous_state** (εικόνα 5.4.11 γραμμή 97).

```
94  if(get_dcd_dsr_cts_logic_state==0)
95     {
96     print("Succesfully connected to server\r\n");
97     get_dcd_dsr_cts_logic_previous_state=0;
```

Εικόνα 5.4.11 Η τέταρτη συνθήκη if (συνάρτηση modem_handler)

5.4.2.1.8 Η συνάρτηση fsl_incoming_handler

Η **fsl_incoming_handler** καλείται οποτεδήποτε υπάρξει interrupt από τον **Microblaze 1** μέσω του διαύλου FSL. Ο λόγος που χρησιμοποιείται ο δίαυλος είναι η ενημέρωση του **Microblaze 0** για δεδομένα που πρέπει να αντλήσει από την SRAM και τα οποία έχουν σταλεί από τον **Microblaze 1**. Σε αυτό το σημείο πρέπει να εξηγηθεί μια ιδιαιτερότητα σχετικά με αυτά τα δεδομένα. Τα δεδομένα που στέλνει ο Microblaze 1 μέσω της SRAM αντλούνται από το GPIO Module, τον δίαυλο I^2C και τον ελεγκτή RS-232. Η ιδιαιτερότητά τους είναι ότι σε αντίθεση με τον ελεγκτή RS-232 τα δεδομένα που προέρχονται από το GPIO Module και τον δίαυλο I^2C δεν περιλαμβάνουν στο τέλος τους τον δέκατο χαρακτήρα του ASCII που σηματοδοτεί την ολοκλήρωση μιας συμβολοσειράς. Για να μεταδοθούν όμως επιτυχώς μέσω του modem αυτά τα δεδομένα είναι αναγκαία η ύπαρξη του δέκατου χαρακτήρα επομένως απαιτείται ειδικός χειρισμός από την συνάρτηση **fsl_incoming_handler** με χρήση δυο συνθηκών **if** που εξετάζουν την προέλευση των δεδομένων.

Αρχικά, κατά την κλήση της συνάρτησης εκτελείται η εντολή **getfsl_interruptible** (εικόνα 5.4.12 γραμμή 121) που λαμβάνει έναν χαρακτήρα από τον δίαυλο FSL και τον αποθηκεύει στην μεταβλητή **fsl_incoming_data**. Ο χαρακτήρας αυτός προέρχεται από τον **Microblaze 1** και ενημερώνει σχετικά με την προέλευση των δεδομένων. Στη συνέχεια εκτελούνται οι δυο **if** που προαναφέρθηκαν και οι οποίες εξετάζουν τον χαρακτήρα της μεταβλητής **fsl_incoming_data**.

Η πρώτη συνθήκη **if** (εικόνα 5.4.12 γραμμή 123) ελέγχει αν η μεταβλητή **fsl_incoming_data** περιέχει είτε τον χαρακτήρα "**i**" είτε τον "**g**". Κάτι τέτοιο σημαίνει πως τα δεδομένα προέρχονται είτε από τον δίαυλο I^2C είτε από τον GPIO Module. Σε αυτή την περίπτωση αν η συνθήκη είναι αληθής θα εκτελεστεί η εντολή **sprintf** (εικόνα 5.4.12 γραμμή 125) η οποία θα προσθέσει τον δέκατο χαρακτήρα του κώδικα ASCII στο τέλος της συμβολοσειράς που βρίσκεται στην SRAM και έπειτα θα αντιγράψει την ολοκληρωμένη συμβολοσειρά στον πίνακα **send_buffer.**

Αν η μεταβλητή **fsl_incoming_data** περιέχει τον χαρακτήρα "**r**" τα δεδομένα προέρχονται από τον ελεγκτή RS-232. Επομένως εκτελείται η δεύτερη συνθήκη **if** (εικόνα 5.4.12 γραμμή 127) και, απλά, γίνεται αντιγραφή των δεδομένων από την SRAM στον πίνακα **send_buffer** με χρήση της εντολής **strcpy** (εικόνα 5.4.12 γραμμή 129).

Αφού, πλέον, ο πίνακας **send_buffer** έχει αποκτήσει δεδομένα καλείται η συνάρτηση **sender** (εικόνα 5.4.12 γραμμή 131) που τα στέλνει μέσω της σειριακής θύρας στο modem και κατ' επέκταση στον Server. Έπειτα, προς ενημέρωση του διαχειριστή, τυπώνεται το μήνυμα "Sent data to Client:" συνοδευόμενο από τα σταλμένα δεδομένα (εικόνα 5.4.12 γραμμή 132). Τέλος επαναλαμβάνεται ένας βρόγχος **for** (εικόνα 5.4.12 γραμμή 134) που μηδενίζει τόσο την SRAM όσο και τον πίνακα **send_buffer** από τα τελευταία δεδομένα.

```
121  getfsl_interruptible(fsl_incoming_data, 1);
122
123  if(fsl_incoming_data=='i' || fsl_incoming_data=='g')
124     {
125     sprintf(send_buffer,"%s%s",data_from_mb1,"\r\n");
126     }
127  if(fsl_incoming_data=='r')
128     {
129     strcpy(send_buffer,data_from_mb1);
130     }
131  sender(send_buffer,strlen(send_buffer));
132  xil_printf("Sent data to Client:%s",send_buffer);
133
134  for(incr=0;incr<128;incr++)
135     {
136     data_from_mb1[incr]=0;
137     send_buffer[incr]=0;
138     }
```

Εικόνα 5.4.12 Η συνάρτηση fsl_incoming_handler

5.4.2.2 Ο κώδικας στον Microblaze 1

Ο **Microblaze 1** ξεκινάει την εκτέλεση από την συνάρτηση **main** η οποία θα καλέσει την συνάρτηση **setup_system**. Σημαντικό είναι να τονιστεί, προς αποφυγή σφαλμάτων, ότι συνάρτηση **setup_system** διαθέτουν και οι δυο επεξεργαστές. Μετά την εκτέλεση της **setup_system** ο επεξεργαστής αναμένει για οποιοδήποτε γεγονός μπορεί να προκαλέσει interrupts και ανάλογα με την πηγή που το προκάλεσε καλείται η αντίστοιχη συνάρτηση μεταξύ των **fsl_interrupt_handler**, **rs232_interrupt_handler** και **gpio_interrupt_handler**. Πέραν αυτών τέσσερεις ακόμα συναρτήσεις που θα περιγραφούν είναι οι **enable_rs232_int**, **enable_gpio_int**, **disable_rs232_int** και **disable_gpio_int**.

5.4.2.2.1 Η συνάρτηση main

Ο κώδικας όπως ορίζει η C εκκινείται και εδώ από την συνάρτηση **main**. Η **main** χρησιμοποιείται, μόνο, για να καλέσει την συνάρτηση **setup_system** προς εκτέλεση.

5.4.2.2.2 Η συνάρτηση setup_system

Όπως έχει προαναφερθεί η **setup_system** καλείται από την **main.** Σκοπός της είναι η αρχικοποίηση των περιφερειακών και η ενεργοποίηση των interrupts. Αρχικά εκτελούνται οι εντολές αρχικοποίησης των τριών περιφερειακών του **Microblaze 1**.Πρώτα αρχικοποιείται ο ελεγκτής των interrupts (**xps_intc_1**) από την εντολή **XIntc_Initialize** (εικόνα 5.4.13 γραμμή 219). Στη συνέχεια, με χρήση της **XUartlite_Initialize** (εικόνα 5.4.13 γραμμή 220),αρχικοποιείται ο ελεγκτής του σειριακού πρωτοκόλλου RS-232 (**rs232_general_use**) και τέλος, με την εντολή **XGpio_Initialize** (εικόνα 5.4.13 γραμμή 221)**,** το GPIO Module (**gpio_16bit**), πλάτους 16 bit.

```
219  XIntc_Initialize(&InterruptController, XPAR_XPS_INTC_1_DEVICE_ID);
220  XUartLite_Initialize(&uartLite,XPAR_RS232_GENERAL_USE_DEVICE_ID);
221  XGpio_Initialize(&gpio_in_out, XPAR_GPIO_16BIT_DEVICE_ID);
```

Εικόνα 5.4.13 Αρχικοποίηση των περιφερειακών του Microblaze 1

Στη συνέχεια εκτέλεσης του κώδικα εσωτερικά της **setup_system** καλείται η εντολή **XIntc_RegisterHandler** που ορίζει για κάθε περιφερειακό ή δίαυλο που μπορεί να προκαλέσει interrupt ποια συνάρτηση θα εξυπηρετήσει το αντίστοιχο interrupt. Για τον δίαυλο FSL η συνάρτηση που θα κληθεί ύστερα από κάποιο γεγονός είναι η **fsl_interrupt_handler** (εικόνα 5.4.14 γραμμή 223). Για τον ελεγκτή **rs232_general_use** η αντίστοιχη συνάρτηση είναι η **rs232_interrupt_handler** (εικόνα 5.4.14 γραμμή 228) ενώ για το GPIO Module **gpio_16bit** η συνάρτηση **gpio_interrupt_handler** (εικόνα 5.4.14 γραμμή 233). Και εδώ πρέπει να τονιστεί πως συνάρτηση **rs232_interrupt_handler** έχουν και οι δυο επεξεργαστές.

```
223  XIntc_RegisterHandler(XPAR_XPS_INTC_1_BASEADDR,
224                        XPAR_XPS_INTC_1_FSL_MB0_TO_MB1_FSL_HAS_DATA_INTR,
225                        fsl_interrupt_handler,
226                        (void *)0);
227
228  XIntc_RegisterHandler(XPAR_XPS_INTC_1_BASEADDR,
229                        XPAR_XPS_INTC_1_RS232_GENERAL_USE_INTERRUPT_INTR,
230                        rs232_interrupt_handler,
231                        (void *)XPAR_RS232_GENERAL_USE_BASEADDR);
232
233  XIntc_RegisterHandler(XPAR_XPS_INTC_1_BASEADDR,
234                        XPAR_XPS_INTC_1_GPIO_16BIT_IP2INTC_IRPT_INTR,
235                        gpio_interrupt_handler,
236                        (void *)XPAR_GPIO_16BIT_BASEADDR);
```

Εικόνα 5.4.14 Ορισμός των συναρτήσεων που εξυπηρετούν τα interrupts στον Microblaze 1

Αφού ολοκληρωθεί η εκτέλεση των εντολών **XIntc_RegisterHandler** καλείται η **XUartLite_ResetFifos** που καθαρίζει την ουρά **fifo** του ελεγκτή **rs232_general_use** (εικόνα 5.4.15 γραμμή 238). Έπειτα με χρήση βρόγχου **for** (εικόνα 5.4.15 γραμμή 240) μηδενίζεται ο πίνακας **rs232_receive_buffer** που συγκρατεί τα δεδομένα που λαμβάνονται από την σειριακή θύρα. Ακολουθεί η αρχικοποίηση του διαύλου I^2C με την εντολή **i2c_setup** (εικόνα 5.4.15 γραμμή 245) και η ενεργοποίηση των interrupts (εικόνα 5.4.15 γραμμές 249 έως 254).

Στην περίπτωση του **Microblaze 1** αρχικά ενεργοποιούνται μόνο τα interrupts του διαύλου FSL έτσι ώστε να ενημερώνεται για δεδομένα από τον **Microblaze 0**. Τα interrupts από τον ελεγκτή **rs232_general_use** και το GPIO Module **gpio_16bit** ενεργοποιούνται κατ' επιλογή του χρήστη. Στη συνέχεια ορίζονται οι ακροδέκτες του GPIO Module **gpio_16bit** σε έξοδο με χρήση της εντολής **XGpio_SetDataDirection** (εικόνα 5.4.15 γραμμή 256).

```
238  XUartLite_ResetFifos(&uartLite);
239
240  for (local_counter = 0; local_counter < 128; local_counter++)
241     {
242     rs232_receive_buffer[local_counter] = 0;
243     }
244
245  i2c_setup();
246
247  interrupt_mask=XPAR_FSL_MB0_TO_MB1_FSL_HAS_DATA_MASK/*+XPAR_GPIO_16BIT_IP2INTC_IRPT_MASK*/;
248  XGpio_WriteReg(XPAR_GPIO_16BIT_BASEADDR, XGPIO_IER_OFFSET, XGPIO_IR_CH1_MASK) ;
249  XGpio_InterruptEnable(&gpio_in_out, XPAR_GPIO_16BIT_IP2INTC_IRPT_MASK);
250  XGpio_InterruptGlobalEnable(&gpio_in_out);
251  XUartLite_EnableIntr(XPAR_RS232_GENERAL_USE_BASEADDR);
252  XIntc_EnableIntr(XPAR_XPS_INTC_1_BASEADDR,interrupt_mask);
253  XIntc_MasterEnable(XPAR_XPS_INTC_1_BASEADDR);
254  microblaze_enable_interrupts();
255  XIntc_Start(&InterruptController, XIN_REAL_MODE);
256  XGpio_SetDataDirection(&gpio_in_out, 1,0);
```

Εικόνα 5.4.15 Ενεργοποίηση των interrupt στον Microblaze 1

5.4.2.2.3 Η συνάρτηση fsl_interrupt_handler

Η συνάρτηση αυτή καλείται σε περίπτωση που προκληθεί interrupt από τον δίαυλο FSL. Κάτι τέτοιο συνεπάγεται πως υπάρχουν δεδομένα προς επεξεργασία από τον **Microblaze 0**. Αρχικά εκτελείται η εντολή **getfsl_interruptible** (εικόνα 5.4.16 γραμμή 138) που λαμβάνει έναν χαρακτήρα, από τον **Microblaze 0,** τον οποίο αποθηκεύει στη μεταβλητή **fsl_receive**. Ακολουθεί μια συνθήκη **if** (εικόνα 5.4.16 γραμμή 140) που εξετάζει αν η μεταβλητή **fsl_receive** έχει αποθηκευμένο τον χαρακτήρα "**o**". Η συνθήκη αυτή βγαίνει πάντα αληθής καθώς ο **Microblaze 0** στέλνει πάντα τον συγκεκριμένο χαρακτήρα επομένως ξεκινάει να εκτελείται ο κώδικας που βρίσκεται στο εσωτερικό της. Πρώτα καλείται η εντολή **strcpy** (εικόνα 5.4.16 γραμμή 142) που αντιγράφει τα δεδομένα από την μνήμη SRAM στον πίνακα **to_process**. Ακολουθούν δέκα εμφωλευμένες συνθήκες if που εξετάζουν τον χαρακτήρα που βρίσκεται στη πρώτη θέση του πίνακα **to_process**. Ο πρώτος χαρακτήρας αποτελεί την κεφαλίδα των ληφθέντων δεδομένων και καθορίζει το πώς πρέπει να διαχειριστεί το σύστημα τα δεδομένα αυτά.

```
138   getfsl_interruptible(fsl_receive, 0);
139
140   if(fsl_receive=='o')
141      {
142      strcpy(to_process,sram_data_get);
```

Εικόνα 5.4.16 Εκκίνηση της συνάρτησης fsl_interrupt_handler

Η πρώτη εμφωλευμένη συνθήκη **if** (εικόνα 5.4.17 γραμμή 144) εξετάζει κατά πόσο στην πρώτη θέση του πίνακα **to_process** βρίσκεται ο χαρακτήρας "**a**". Κάτι τέτοιο παραπέμπει στην ενεργοποίηση των interrupts για το GPIO Module **gpio_16bit**. Συνεπώς αν η συνθήκη αυτή βρεθεί αληθής καλείται η συνάρτηση **enable_gpio_int** (εικόνα 5.4.17 γραμμή 146), της οποίας η περιγραφή θα δοθεί παρακάτω, για να ενεργοποιήσει τα interrupts για το **gpio_16bit.**

Η δεύτερη εμφωλευμένη συνθήκη **if** (εικόνα 5.4.17 γραμμή 148) ελέγχει αν στην πρώτη θέση του πίνακα **to_process** βρίσκεται ο χαρακτήρας "**b**" που συνεπάγεται πως πρέπει να απενεργοποιηθούν τα interrupts για το GPIO Module **gpio_16bit**. Επομένως θα κληθεί η συνάρτηση **disable_gpio_int** (εικόνα 5.4.17 γραμμή 150) που θα ικανοποιήσει αυτή την απαίτηση.

Η τρίτη εμφωλευμένη συνθήκη **if** (εικόνα 5.4.17 γραμμή 152) αναζητάει στην πρώτη θέση του πίνακα **to_process** τον χαρακτήρα "**c**". Σε μια τέτοια περίπτωση το σύστημα πρέπει να τροποποιήσει τους ακροδέκτες του GPIO Module **gpio_16bit** σε είσοδο ή έξοδο όπως ορίζουν τα καθαρά δεδομένα (χωρίς την κεφαλίδα) που βρίσκονται στον πίνακα **to_process.**

Η τροποποίηση των ακροδεκτών γίνεται από έναν ακέραιο αριθμό. Έτσι, αρχικά, καλείται η εντολή **atoi** που μετατρέπει την συμβολοσειρά, που βρίσκεται από την δεύτερη θέση του πίνακα και μετά (**to_process+1**), σε ακέραιο(εικόνα 5.4.17 γραμμή 154). Στη συνέχεια εκτελείται η εντολή **XGpio_SetDataDirection** (εικόνα 5.4.17 γραμμή 154) που τελικά πραγματοποιεί την αλλαγή των ακροδεκτών σύμφωνα με τον προηγούμενο ακέραιο. Έπειτα τυπώνεται το μήνυμα "Setting Pin Direction on GPIO Module as:" συνοδευόμενο από τα δεδομένα που προκάλεσαν την αλλαγή (εικόνα 5.4.17 γραμμή 155).

Η τέταρτη εμφωλευμένη συνθήκη **if** (εικόνα 5.4.17 γραμμή 157) ελέγχει αν στην πρώτη θέση του πίνακα **to_process** είναι ο χαρακτήρας "**d**". Ο χαρακτήρας αυτός παραπέμπει στην αποστολή δεδομένων στους ακροδέκτες του GPIO Module **gpio_16bit** που στην ουσία τα δεδομένα αυτά αλλάζουν την λογική κατάσταση των ακροδεκτών σε μηδέν ή ένα. Αν αυτή η συνθήκη είναι αληθής καλείται και εδώ η εντολή atoi που θα μετατρέψει τα καθαρά δεδομένα του πίνακα σε ακέραιο (εικόνα 5.4.17 γραμμή 159). Στη συνέχεια εκτελείται η εντολή **XGpio_DiscreteWrite** (εικόνα 5.4.17 γραμμή 159) που θα προωθήσει τον ακέραιο αυτό στους ακροδέκτες αλλάζοντας την λογική κατάσταση των σημάτων τους. Έπειτα θα κληθεί η **xil_printf** που θα τυπώσει το μήνυμα " Writing Data on GPIO Module:" συνοδευόμενο από τα καθαρά δεδομένα (εικόνα 5.4.17 γραμμή 160).

Η πέμπτη εμφωλευμένη συνθήκη **if** (εικόνα 5.4.17 γραμμή 162) εξετάζει αν είναι αποθηκευμένος ο χαρακτήρας "**e**" στην πρώτη θέση του πίνακα **to_process**. Με αυτή την κεφαλίδα ζητείται από το σύστημα να διαβαστεί η λογική κατάσταση των ακροδεκτών στο GPIO Module **gpio_16bit** και τα δεδομένα από αυτούς να σταλθούν πίσω στο τερματικό του χρήστη. Αρχικά στην εκτέλεση αυτής της συνθήκης γίνεται προετοιμασία των δεδομένων που πρέπει να σταλθούν στον χρήστη. Καταχωρείται στην πρώτη θέση του πίνακα **get_gpio_data** ο χαρακτήρας "**d**" (εικόνα 5.4.17 γραμμή 164) που θα αποτελέσει την κεφαλίδα που θα διαβάσει η εφαρμογή στο τερματικό του χρήστη. Στο επόμενο βήμα καλείται η εντολή **XGpio_DiscreteRead** (εικόνα 5.4.17 γραμμή 165) που θα διαβάσει την λογική κατάσταση των ακροδεκτών και θα επιστρέψει έναν ακέραιο. Ο ακέραιος αυτός θα μετατραπεί σε συμβολοσειρά με χρήση της εντολής **itoa** και θα αποθηκευτεί στον πίνακα **get_gpio_data** ξεκινώντας από τη δεύτερη θέση του (εικόνα 5.4.17 γραμμή 165). Με αυτό τον τρόπο έχει προετοιμαστεί ένα πακέτο που περιλαμβάνει την κεφαλίδα "**d**" και τα καθαρά δεδομένα. Στη συνέχεια τυπώνεται το μήνυμα "Reading Data from GPIO Module:" συνοδευόμενο από τα δεδομένα που διαβάστηκαν (εικόνα 5.4.17 γραμμή 166). Ύστερα καλείται η εντολή strcpy (εικόνα 5.4.17 γραμμή 167), που αντιγράφει τα περιεχόμενα του πίνακα **get_gpio_data** στη μνήμη SRAM. Στο τελευταίο βήμα της πέμπτης εμφωλευμένης συνθήκης **if** καλείται η **putfsl** (εικόνα 5.4.17 γραμμή 168), που θα ειδοποιήσει τον Microblaze 0 για νέα δεδομένα, στέλνοντας τον χαρακτήρα "**g**" μέσω του διαύλου FSL.

```
142    strcpy(to_process,sram_data_get);
143
144    if(to_process[0]=='a')
145       {
146       enable_gpio_int();
147       }
148    if(to_process[0]=='b')
149       {
150       disable_gpio_int();
151       }
152    if(to_process[0]=='c')
153       {
154       XGpio_SetDataDirection(&gpio_in_out, 1, atoi(to_process+1));
155       xil_printf("Setting Pin Direction on GPIO Module as:%d\r\n",atoi(to_process+1));
156       }
157    if(to_process[0]=='d')
158       {
159       XGpio_DiscreteWrite(&gpio_in_out, 1,atoi(to_process+1));
160       xil_printf("Writing Data on GPIO Module:%d\r\n",atoi(to_process+1));
161       }
162    if(to_process[0]=='e')
163       {
164       get_gpio_data[0]='d';
165       itoa(XGpio_DiscreteRead(&gpio_in_out, 1),get_gpio_data+1);
166       xil_printf("Reading Data from GPIO Module:%d\r\n",XGpio_DiscreteRead(&gpio_in_out, 1));
167       strcpy(sram_data_put,get_gpio_data);
168       putfsl('g',1);
169       }
```

Εικόνα 5.4.17 Οι εμφωλευμένες συνθήκες if που αφορούν το GPIO Module gpio_16bit

Η έκτη εμφωλευμένη συνθήκη **if** (εικόνα 5.4.18 γραμμή 170) αναζητάει στην πρώτη θέση του πίνακα **to_process** τον χαρακτήρα "**f**" που παραπέμπει σε ενεργοποίηση των interrupts για τον ελεγκτή **rs232_general_use**. Συνεπώς καλείται η συνάρτηση **enable_rs232_int** (εικόνα 5.4.18 γραμμή 172) για να πραγματοποιήσει αυτή την ενεργοποίηση.

Η έβδομη εμφωλευμένη συνθήκη **if** (εικόνα 5.4.18 γραμμή 174) θα εκτελεστεί αν βρεθεί ο χαρακτήρας "**g**" στην πρώτη θέση του πίνακα **to_process**. Ο χαρακτήρας αυτός απαιτεί την απενεργοποίηση των interrupts για τον ελεγκτή **rs232_general_use** και έτσι καλείται η συνάρτηση **disable_rs232_int** (εικόνα 5.4.18 γραμμή 176).

Η όγδοη εμφωλευμένη συνθήκη **if** (εικόνα 5.4.18 γραμμή 178) εξετάζει αν στην πρώτη θέση του πίνακα **to_process** βρίσκεται ο χαρακτήρας "**h**". Κάτι τέτοιο σημαίνει πως τα ληφθέντα δεδομένα πρέπει να γραφτούν μέσω του ελεγκτή **rs232_general_use** σε οποιαδήποτε συσκευή είναι συνδεδεμένη στη σειριακή θύρα. Αρχικά κατά την εκτέλεση της συνθήκης αυτής τυπώνεται το μήνυμα "Writing data to Usart:" συνοδευόμενο από τα δεδομένα που πρόκειται να γραφτούν (εικόνα 5.4.18 γραμμή 180). Στη συνέχεια επαναλαμβάνεται ένας βρόγχος **for** (εικόνα 5.4.18 γραμμή 181) που σε κάθε επανάληψή του καλεί την εντολή **XUartLite_SendByte** (εικόνα 5.4.18 γραμμή 183) η οποία κάθε φορά μεταδίδει έναν χαρακτήρα από τον πίνακα **to_process** μέσα από τη σειριακή θύρα. Ο βρόγχος **for** θα συνεχίζει να επαναλαμβάνεται έως ότου μεταδοθούν όλα τα δεδομένα τα οποία ξεκινούν από τη δεύτερη θέση του πίνακα.

```
170     if(to_process[0]=='f')
171        {
172        enable_rs232_int();
173        }
174     if(to_process[0]=='g')
175        {
176        disable_rs232_int();
177        }
178     if(to_process[0]=='h')
179        {
180        xil_printf("Writing Data to Usart:%s",to_process+1);
181        for(local_counter=1;local_counter<strlen(to_process);local_counter++)
182           {
183           XUartLite_SendByte(XPAR_RS232_GENERAL_USE_BASEADDR,to_process[local_counter]);
184           }
```

Εικόνα 5.4.18 Οι εμφωλευμένες συνθήκες if που αφορούν τον ελεγκτή RS-232 rs232_general_use

Η ένατη εμφωλευμένη συνθήκη **if** (εικόνα 5.4.19 γραμμή 187) ελέγχει αν στην πρώτη θέση του πίνακα **to_process** βρίσκεται ο χαρακτήρας "**i**". Σε αυτή την περίπτωση τα ληφθέντα δεδομένα πρέπει να παραδοθούν στον δίαυλο I^2C. Αν αυτή η συνθήκη βρεθεί αληθής, αρχικά, θα εκτελεστεί η εντολή **strcpy** (εικόνα 5.4.19 γραμμή 189) που θα αντιγράψει τα καθαρά δεδομένα που ξεκινούν από την τρίτη θέση πίνακα **to_process** στον πίνακα **i2c_send_buffer**. Έπειτα θα τυπωθεί το μήνυμα "Writing Data on I2C Bus:" συνοδευόμενο από τα δεδομένα που πρόκειται να παραδοθούν (εικόνα 5.4.19 γραμμή 190). Στο επόμενο βήμα καλείται η εντολή **i2c_send** (εικόνα 5.4.19 γραμμή 191) που θα στείλει τα δεδομένα. Η εντολή αυτή θα τυπώσει μήνυμα επιτυχίας ή αποτυχίας ανάλογα με το αν τα δεδομένα παραδόθηκαν ή όχι. Τέλος εκτελείται ένας βρόγχος **for** (εικόνα 5.4.19 γραμμή 192) που μηδενίζει τα περιεχόμενα του πίνακα **i2c_send_buffer**.

Η δέκατη και τελευταία εμφωλευμένη συνθήκη **if** (εικόνα 5.4.19 γραμμή 197) εξετάζει αν βρίσκεται ο χαρακτήρας "**j**" στην πρώτη θέση του πίνακα **to_process**. Κάτι τέτοιο παραπέμπει στην λήψη δεδομένων από τον δίαυλο I^2C και αποστολή τους στο τερματικό του χρήστη. Αρχικά και εδώ θα γίνει προετοιμασία του μηνύματος που θα σταλθεί στον χρήστη έτσι στο πρώτο βήμα καταχωρείται ο χαρακτήρας "**i**" στην πρώτη θέση του πίνακα **i2c_receive_buffer** (εικόνα 5.4.19 γραμμή 199). Στη συνέχεια καλείται η εντολή **i2c_receive** (εικόνα 5.4.19 γραμμή 200) που θα λάβει δεδομένα από τον δίαυλο I^2C και θα τα αποθηκεύσει στον πίνακα **i2c_receive_buffer** ξεκινώντας από την δεύτερη θέση του. Με αυτό τον τρόπο έχει προετοιμαστεί ένα πακέτο που περιλαμβάνει την κεφαλίδα "**i**" και το καθαρό μήνυμα. Αφού πλέον έχει προετοιμαστεί ένα ολοκληρωμένο μήνυμα καλείται η εντολή **strcpy** (εικόνα 5.4.19 γραμμή 201) που αντιγράφει τα περιεχόμενα του πίνακα **i2c_receive_buffer**, δηλαδή το ολοκληρωμένο μήνυμα, στη μνήμη SRAM. Έπειτα καλείται η **putfsl** (εικόνα 5.4.19 γραμμή 202), που θα ειδοποιήσει τον Microblaze 0 για νέα δεδομένα, στέλνοντας αυτή την φορά τον χαρακτήρα "**i**" μέσω του διαύλου FSL. Τέλος επαναλαμβάνεται ένας βρόγχος **for** (εικόνα 5.4.19 γραμμή 202) που μηδενίζει τα περιεχόμενα του πίνακα **i2c_receive_buffer**.

```
187     if(to_process[0]=='i')
188        {
189        strcpy(i2c_send_buffer,to_process+2);
190        xil_printf("Writing Data on I2C Bus:%s",to_process+1);
191        i2c_send(to_process[1],i2c_send_buffer,strlen(i2c_send_buffer));
192        for(local_counter=0;local_counter<strlen(i2c_send_buffer);local_counter++)
193           {
194           i2c_send_buffer[local_counter]=0;
195           }
196        }
197     if(to_process[0]=='j')
198        {
199        i2c_receive_buffer[0]='i';
200        i2c_receive(to_process[1],to_process[2],i2c_receive_buffer+1);
201        strcpy(sram_data_put,i2c_receive_buffer);
202        putfsl('i',1);
203        for(local_counter=0;local_counter<strlen(i2c_receive_buffer);local_counter++)
204           {
205           i2c_receive_buffer[local_counter]=0;
206           }
```

Εικόνα 5.4.19 Οι εμφωλευμένες συνθήκες if που αφορούν τον δίαυλο I^2C

5.4.2.2.4 Η συνάρτηση rs232_interrupt_handler

Η συνάρτηση αυτή καλείται οποτεδήποτε προκαλείται interrupt από εισερχόμενα δεδομένα στην ουρά του ελεγκτή **rs232_general_use**. Σκοπός της είναι να λάβει τα εισερχόμενα δεδομένα που προέρχονται από οποιαδήποτε συσκευή είναι συνδεδεμένη στη σειριακή θύρα και στη συνέχεια να τα προωθήσει στον **Microblaze 0** μέσω της SRAM.

Κατά την κλήση της **rs232_interrupt_handler** ξεκινάει ένας βρόγχος **while** (εικόνα 5.4.20 γραμμή 108) ο οποίος επαναλαμβάνεται όσο η εντολή **XUartLite_IsReceiveEmpty** παραμένει ψευδής, με άλλα λόγια όσο ο ελεγκτής RS-232 συνεχίζει να λαμβάνει χαρακτήρες. Το σύνολο του κώδικα που εκτελείται σε αυτή τη συνάρτηση βρίσκεται εσωτερικά του βρόγχου αυτού. Αρχικά κατά την εκτέλεση του βρόγχου λαμβάνεται ένας χαρακτήρας με χρήση της εντολής **XUartLite_RecvByte** και αποθηκεύεται στη μεταβλητή **c** τύπου χαρακτήρα (εικόνα 5.4.20 γραμμή 110). Στη συνέχεια ο χαρακτήρας που βρίσκεται στην μεταβλητή **c** αποθηκεύεται στην τρέχουσα θέση του πίνακα χαρακτήρων **rs232_receive_buffer** (εικόνα 5.4.20 γραμμή 111) η οποία καθορίζεται από την ακέραια μεταβλητή **rs232_counter**. Έπειτα η μεταβλητή **rs232_counter** αυξάνεται κατά ένα (εικόνα 5.4.20 γραμμή 112) ώστε την επόμενη φορά να δείξει στην επόμενη θέση του πίνακα. Μετά την ολοκλήρωση των τριών τελευταίων βημάτων εκτελείται μια συνθήκη **if** (εικόνα 5.4.20 γραμμή 113) που εξετάζει αν ο τελευταίος χαρακτήρας που βρίσκεται στη μεταβλητή **c** αντιστοιχεί στον δέκατο χαρακτήρα του κώδικα ASCII. Ο δέκατος χαρακτήρας σηματοδοτεί, όπως είναι ήδη γνωστό, την ολοκλήρωση μιας συμβολοσειράς. Όσο επαναλαμβάνεται αυτός ο βρόγχος εκτελούνται κάθε φορά τα 4 βήματα που αναφέρθηκαν ως τώρα. Με αυτό τον τρόπο η συνάρτηση συνεχίζει να λαμβάνει χαρακτήρες, συνεπώς τα εισερχόμενα δεδομένα, τους οποίους διαδοχικά τους αποθηκεύει στον πίνακα **rs232_receive_buffer**. Σημαντικό είναι να σημειωθεί πως τα δεδομένα που λαμβάνονται αποθηκεύονται ξεκινώντας από την δεύτερη θέση του πίνακα καθώς στην πρώτη θέση μελλοντικά θα τοποθετηθεί η κεφαλίδα. Μόλις ληφθεί ο δέκατος χαρακτήρας του κώδικα ASCII η συνθήκη **if** γίνεται αληθής, σηματοδοτείται η ολοκλήρωση της ληφθείσας συμβολοσειράς και πλέον μπορεί να εκτελεστεί ο κώδικας εσωτερικά της **if**.

Μόλις η **if** βρεθεί αληθής, αρχικά, θα μηδενιστεί η μεταβλητή **rs232_counter** (εικόνα 5.4.20 γραμμή 115) ώστε να αποθηκευτούν από την αρχή του πίνακα **rs232_receive_buffer** τα δεδομένα που θα ληφθούν την επόμενη φορά. Στο επόμενο βήμα, όπως αναφέρθηκε και στη προηγούμενη παράγραφο, τοποθετείται η κεφαλίδα "**h**" στην πρώτη θέση του πίνακα **rs232_receive_buffer** (εικόνα 5.4.20 γραμμή 116). Στη συνέχεια τυπώνεται το μήνυμα "Reading Data from Usart:" συνοδευόμενο, όπως και σε άλλες περιπτώσεις, από τα ληφθέντα δεδομένα (εικόνα 5.4.20 γραμμή 117). Έπειτα αντιγράφονται τα δεδομένα από τον πίνακα **rs232_receive_buffer** στη μνήμη SRAM (εικόνα 5.4.20 γραμμή 119) και καλείται η εντολή putfsl (εικόνα 5.4.20 γραμμή 120) που στέλνει τον χαρακτήρα "**r**" μέσω του διαύλου FSL με σκοπό να ενημερώσει τον **Microblaze 0** ώστε να τα παραλάβει. Τέλος επαναλαμβάνεται ένας βρόγχος **for** (εικόνα 5.4.20 γραμμή 122)που μηδενίζει τον πίνακα.

```
108  while (!XUartLite_IsReceiveEmpty(XPAR_RS232_GENERAL_USE_BASEADDR))
109     {
110     c = XUartLite_RecvByte(XPAR_RS232_GENERAL_USE_BASEADDR);
111     rs232_receive_buffer[rs232_counter+1]=c;
112     rs232_counter++;
113     if(c==10)
114        {
115        rs232_counter=0;
116        rs232_receive_buffer[0]='h';
117        xil_printf("Reading Data from Usart:%s",rs232_receive_buffer+1);
118        number=strlen(rs232_receive_buffer);
119        strcpy(sram_data_put,rs232_receive_buffer);
120        putfsl('r',1);
121
122        for(j=0;j<number+5;j++)
123           {
124           rs232_receive_buffer[j]=0;
125           }
126        }
127     }
```

Εικόνα 5.4.20 Η συνάρτηση rs232_interrupt_handler στον Microblaze 1

5.4.2.2.5 Η συνάρτηση gpio_interrupt_handler

Η συνάρτηση αυτή θα κληθεί οποτεδήποτε προκληθεί γεγονός από το GPIO Module **gpio_16bit** κάτι που σημαίνει πως έχει υπάρξει αλλαγή στη λογική κατάσταση των σημάτων στους ακροδέκτες. Σκοπός της **gpio_interrupt_handler** είναι να λάβει τα νέα δεδομένα που προκύπτουν από αυτή την αλλαγή και να τα προωθήσει στον **Microblaze 0**, μέσω της SRAM, με τελικό παραλήπτη την εφαρμογή στο τερματικό του χρήστη.

Σε περίπτωση που κληθεί αυτή η συνάρτηση, αρχικά, απενεργοποιούνται τα interrupts (εικόνα 5.4.21 γραμμή 89) ώστε να μην προκληθούν άλλα έως ότου εξυπηρετηθεί αυτή η ρουτίνα. Έπειτα καλείται η εντολή **XGpio_DiscreteRead** που διαβάζει την λογική κατάσταση των σημάτων και την αποθηκεύει, ως ακέραιο, στη μεταβλητή **num** (εικόνα 5.4.21 γραμμή 90). Στη συνέχεια τυπώνεται το μήνυμα "Reading Data from GPIO Module" συνοδευόμενο από τα δεδομένα που μόλις διάβασε ο **Microblaze 1** (εικόνα 5.4.21 γραμμή 91). Στο επόμενο βήμα ξεκινάει η προετοιμασία του μηνύματος που θα σταλθεί στον χρήστη τοποθετώντας την κεφαλίδα "**d**" στην πρώτη θέση του πίνακα **get_gpio_data** (εικόνα 5.4.21 γραμμή 92). Ακολουθεί η κλήση της εντολής **itoa** (εικόνα 5.4.21 γραμμή 93), που θα μετατρέψει τον ακέραιο **num** σε συμβολοσειρά, και θα τον αποθηκεύσει στον πίνακα **get_gpio_data** ξεκινώντας από την δεύτερη θέση του. Σε αυτό το σημείο έχει προετοιμαστεί ένα ολοκληρωμένο πακέτο δεδομένων, που περιλαμβάνει την κεφαλίδα και το καθαρό μήνυμα, συνεπώς στο επόμενο βήμα καλείται η εντολή **strcpy** (εικόνα 5.4.21 γραμμή 94) που αντιγράφει τα περιεχόμενα του πίνακα **get_gpio_data** στην μνήμη SRAM. Ακολουθεί η **putfsl** (εικόνα 5.4.21 γραμμή 95) που στέλνει τον χαρακτήρα "**g**" μέσω του διαύλου FSL στον **Microblaze 0** με σκοπό να τον ενημερώσει για νέα δεδομένα στην SRAM. Τέλος μηδενίζονται οι καταχωρητές των interrupts (εικόνα 5.4.21 γραμμή 97) και ενεργοποιούνται εκ νέου τα interrupts (εικόνα 5.4.21 γραμμή 99) για το **gpio_16bit** .

```
89  XGpio_InterruptDisable(&gpio_in_out, XPAR_GPIO_16BIT_IP2INTC_IRPT_MASK);
90  num=XGpio_DiscreteRead(&gpio_in_out, 1);
91  xil_printf("Reading Data from GPIO Module:%d\r\n",num);
92  get_gpio_data[0]='d';
93  itoa(num,get_gpio_data+1);
94  strcpy(sram_data_put,get_gpio_data);
95  putfsl('g',1);
96
97  (void)XGpio_InterruptClear(&gpio_in_out, XPAR_GPIO_16BIT_IP2INTC_IRPT_MASK);
98  XGpio_WriteReg(XPAR_GPIO_16BIT_BASEADDR, XGPIO_IER_OFFSET, XGPIO_IR_CH1_MASK) ;
99  XGpio_InterruptEnable(&gpio_in_out, XPAR_GPIO_16BIT_IP2INTC_IRPT_MASK);
```

Εικόνα 5.4.21 Η συνάρτηση gpio_interrupt_handler στον Microblaze 1

5.4.2.2.6 Η συνάρτηση enable_rs232_int

Η συνάρτηση αυτή είναι μια από τις τέσσερεις που ενεργοποιούν ή απενεργοποιούν interrupts κατ' επιλογή του χρήστη. Η συγκεκριμένη ενεργοποιεί τα interrupt για τον ελεγκτή **rs232_general_use** και καλείται μέσα από τη συνάρτηση **fsl_interrupt_handler** όταν τα εισερχόμενα δεδομένα έχουν ως κεφαλίδα τον χαρακτήρα "**f**".

Μόλις κληθεί η **enable_rs232_int**, αρχικά, εκτελεί μια συνθήκη **if** (εικόνα 5.4.22 γραμμή 36) που εξετάζει αν η μεταβλητή **rs232_status** έχει αποθηκευμένο τον χαρακτήρα "**d**". Η μεταβλητή αυτή ενημερώνει σχετικά με το αν τα interrupts για τον ελεγκτή **rs232_general_use** είναι ήδη ενεργά ή ανενεργά και μπορεί να περιέχει είτε τον χαρακτήρα "**d**" είτε τον "**e**". Συνεπώς, αν η παραπάνω συνθήκη βρεθεί αληθής συνεπάγεται πως τα interrupts είναι ανενεργά και προφανώς πρέπει να ενεργοποιηθούν οπότε θα εκτελεστεί ο κώδικας εσωτερικά της συνθήκης. Σε περίπτωση που η συνθήκη δεν ήταν αληθής, είναι προφανές πως η μεταβλητή **rs232_status** περιέχει άλλο χαρακτήρα επομένως τα interrupts είναι ήδη ενεργά και δεν χρειάζεται να πραγματοποιηθεί κάποια ενέργεια. Αν τελικά εκτελεστεί ο κώδικας της **if** αρχικά μηδενίζεται η ουρά FIFO του ελεγκτή **rs232_general_use** με χρήση της εντολής **XUartLite_ResetFifos** (εικόνα 5.4.22 γραμμή 38). Στη συνέχεια ενεργοποιούνται τα interrupts (εικόνα 5.4.22 γραμμές 39 και 40) και στη μεταβλητή **rs232_status** αποθηκεύεται ο χαρακτήρας "**e**" (εικόνα 5.4.22 γραμμή 41) ώστε την επόμενη φορά το σύστημα να γνωρίζει πως τα interrupts είναι ήδη ενεργά. Τέλος τυπώνεται το μήνυμα "RS232 Interrupt (Enabled)" ώστε να ενημερωθεί ο διαχειριστής του συστήματος για την επιτυχημένη ενεργοποίηση.

```
36  if(rs232_status=='d')
37     {
38     XUartLite_ResetFifos(&uartLite);
39     interrupt_mask+=XPAR_RS232_GENERAL_USE_INTERRUPT_MASK;
40     XIntc_EnableIntr(XPAR_XPS_INTC_1_BASEADDR,interrupt_mask);
41     rs232_status='e';
42     print("RS232 Interrupt(Enabled)\r\n");
```

Εικόνα 5.4.22 Η συνάρτηση enable_rs232_int στον Microblaze 1

5.4.2.2.7 Η συνάρτηση disable_rs232_int

Η **disable_rs232_int** σε αντίθεση με την προαναφερθείσα απενεργοποιεί τα interrupts για τον ελεγκτή **rs232_general_use**. Καλείται και αυτή μέσα από τη συνάρτηση **fsl_interrupt_handler** όταν τα εισερχόμενα δεδομένα έχουν ως κεφαλίδα τον χαρακτήρα "**g**".

Κατά την κλήση της εκτελείται μια συνθήκη **if** (εικόνα 5.4.23 γραμμή 62) που και εδώ εξετάζει την μεταβλητή **rs232_status** αλλά αυτή την φορά θα βρεθεί αληθής αν η προηγούμενη μεταβλητή περιέχει τον χαρακτήρα "**e**". Κάτι τέτοιο σημαίνει πως τα interrupts είναι ήδη ενεργά και θα εκτελεστεί ο κώδικας εσωτερικά της συνθήκης για να τα απενεργοποιήσει. Αρχικά θα απενεργοποιηθούν τα interrupts (εικόνα 5.4.23 γραμμές 64 και 65) και στη συνέχεια στην μεταβλητή **rs232_status** αποθηκεύεται ο χαρακτήρας "**d**" (εικόνα 5.4.23 γραμμή 66) ώστε την επόμενη φορά το σύστημα να γνωρίζει πως τα interrupts είναι ήδη ανενεργά. Τέλος τυπώνεται το μήνυμα "RS232 Interrupt (Disabled)" (εικόνα 5.4.23 γραμμή 67).

```
62  if(rs232_status=='e')
63     {
64     interrupt_mask-=XPAR_RS232_GENERAL_USE_INTERRUPT_MASK;
65     XIntc_EnableIntr(XPAR_XPS_INTC_1_BASEADDR,interrupt_mask);
66     rs232_status='d';
67     print("RS232 Interrupt(Disabled)\r\n");
```

Εικόνα 5.4.23 Η συνάρτηση disable_rs232_int στον Microblaze 1

5.4.2.2.8 Η συνάρτηση enable_gpio_int

Η συνάρτηση αυτή καλείται μέσα από τη συνάρτηση **fsl_interrupt_handler** όταν τα εισερχόμενα δεδομένα έχουν ως κεφαλίδα τον χαρακτήρα "**a**". Σκοπός της είναι να ενεργοποιήσει τα interrupts για το GPIO Module **gpio_16bit**.

Αρχικά η **enable_gpio_int** εξετάζει, με χρήση συνθήκης **if** (εικόνα 5.4.24 γραμμή 48), αν η μεταβλητή **gpio_status** περιέχει τον χαρακτήρα "**d**". Η μεταβλητή αυτή έχει τον ίδιο ρόλο με την **rs232_status** με την διαφορά ότι αφορά τα interrupts για το **gpio_16bit**. Αν η συνθήκη βρεθεί αληθής συνεπάγεται πως τα interrupts είναι ήδη ανενεργά και επομένως στα δυο επόμενα βήματα ενεργοποιούνται (εικόνα 5.4.24 γραμμές 50 και 51). Στη συνέχεια αποθηκεύεται ο χαρακτήρας "**e**" στη μεταβλητή **gpio_status** (εικόνα 5.4.24 γραμμή 52) ώστε το σύστημα να γνωρίζει την πιο πρόσφατη κατάσταση των interrupts, που σε αυτή την περίπτωση συνεπάγεται πως είναι ενεργά. Τελικά τυπώνεται το μήνυμα "GPIO Interrupt (Enabled)" προς ενημέρωση του διαχειριστή (εικόνα 5.4.24 γραμμή 53).

```
48  if(gpio_status=='d')
49     {
50     interrupt_mask+=XPAR_GPIO_16BIT_IP2INTC_IRPT_MASK;
51     XIntc_EnableIntr(XPAR_XPS_INTC_1_BASEADDR,interrupt_mask);
52     gpio_status='e';
53     print("GPIO Interrupt(Enabled)\r\n");
```

Εικόνα 5.4.24 Η συνάρτηση enable_gpio_int στον Microblaze 1

5.4.2.2.9 Η συνάρτηση disable_gpio_int

Η **disable_gpio_int** καλείται μέσα από τη συνάρτηση **fsl_interrupt_handler** όταν τα εισερχόμενα δεδομένα έχουν ως κεφαλίδα τον χαρακτήρα "**b**". Η συνάρτηση αυτή θα απενεργοποιήσει τα interrupts για το GPIO Module **gpio_16bit**.

Όπως και στην συνάρτηση **disable_rs232_int** εκτελείται μια συνθήκη **if** (εικόνα 5.4.25 γραμμή 73) που εδώ εξετάζει την μεταβλητή **gpio_status** αλλά αυτή την φορά θα βρεθεί αληθής αν η προηγούμενη μεταβλητή περιέχει τον χαρακτήρα "**e**". Σε μια τέτοια περίπτωση στα πρώτα δυο βήματα θα απενεργοποιηθούν τα interrupts (εικόνα 5.4.25 γραμμές 75 και 76). Στη συνέχεια θα αποθηκευτεί ο χαρακτήρας "**d**" στην μεταβλητή **gpio_status** (εικόνα 5.4.25 γραμμή 77) και τελικά θα τυπωθεί το μήνυμα "GPIO Interrupt (Disabled)" (εικόνα 5.4.25 γραμμή 78).

```
73  if(gpio_status=='e')
74     {
75     interrupt_mask-=XPAR_GPIO_16BIT_IP2INTC_IRPT_MASK;
76     XIntc_EnableIntr(XPAR_XPS_INTC_1_BASEADDR,interrupt_mask);
77     gpio_status='d';
78     print("GPIO Interrupt(Disabled)\r\n");
```

Εικόνα 5.4.25 Η συνάρτηση disable_gpio_int στον Microblaze 1

6. Μελλοντικές επεκτάσεις και βελτιστοποιήσεις-Συμπεράσματα

Σε αυτό το κεφάλαιο, αρχικά, θα γίνει αναφορά σε τρόπους με τους οποίους το σύστημα που υλοποιήθηκε σε αυτή την εργασία θα μπορούσε να επεκταθεί αλλά και να βελτιστοποιηθεί. Στη συνέχεια θα δοθούν τα συμπεράσματα που προκύπτουν από την εκπόνηση αυτής της εργασίας.

6.1 Μελλοντικές επεκτάσεις και βελτιστοποιήσεις

Σε αυτή την εργασία υλοποιήθηκε ένα ολοκληρωμένο σύστημα που παρέχει σε έναν χρήστη την δυνατότητα να έχει διαδικτυακή πρόσβαση σε απομακρυσμένες συσκευές. Το σύστημα αυτό παραδίδει υπηρεσίες διαδικτυακής επικοινωνίας ενός χρήστη με πολλούς τύπους απομακρυσμένων συσκευών με αξιοπιστία και ευελιξία.

Παρ' όλες τις δυνατότητες που παρέχονται υπάρχει πάντα περιθώριο τόσο επέκτασης του συστήματος όσο και βελτιστοποίησης του. Σε αυτή την ενότητα θα γίνει αναφορά σε μερικές από αυτές τις πολλές επεκτάσεις και βελτιστοποιήσεις, που θα μπορούσαν να υλοποιηθούν μελλοντικά.

6.1.1 Επεκτάσεις και βελτιστοποιήσεις στο ενσωματωμένο σύστημα

Όσον αφορά το ενσωματωμένο σύστημα θα μπορούσαν να υπάρξουν αλλαγές τόσο στο υλικό όσο και στο λογισμικό.

Ένα παράδειγμα αλλαγής θα ήταν η ενσωμάτωση στο υλικό ενός ethernet controller για υποστήριξη πρωτοκόλλου ethernet. Μια τέτοια επέκταση θα ήταν χρήσιμη σε περιπτώσεις όπου ο ιδιοκτήτης ενός τέτοιου συστήματος έχει την δυνατότητα να το συνδέσει διαδικτυακά με χρήση καλωδίου UTP. Ως συνέπεια, θα ήταν αναγκαία η ανάπτυξη επιπλέον κώδικα που θα διαχειρίζεται την επικοινωνία με τον ethernet controller αλλά και κώδικα για δυναμική διάθεση διεύθυνσης IP (DHCP) ώστε να γίνεται λιγότερο περίπλοκο για τον κάτοχο του ενσωματωμένου συστήματος.

Ένα δεύτερο παράδειγμα βελτιστοποίησης θα μπορούσε να αφορά το πρωτόκολλο I^2C το οποίο χρησιμοποιείται από το ενσωματωμένο σύστημα για επικοινωνία με εξωτερικές συσκευές όπως αισθητήρες και μικροελεγκτές. Στην παρούσα φάση ο απομακρυσμένος χρήστης πρέπει να γνωρίζει την διεύθυνση της κάθε συσκευής που βρίσκεται συνδεδεμένη στον δίαυλο I^2C ώστε να επικοινωνήσει μαζί της (δείτε κεφάλαιο 4). Μελλοντικά το ενσωματωμένο σύστημα θα μπορούσε να διαθέτει έναν μηχανισμό που ανακτά αυτόματα τις διευθύνσεις όλων των συσκευών που βρίσκονται στο δίαυλο I^2C.

Μια ενδιαφέρουσα βελτίωση θα ήταν η δυνατότητα δυναμικής τροποποίησης του υλικού ώστε το ενσωματωμένο σύστημα να μπορεί αυτόματα να προσαρμόζεται στις εκάστοτε ανάγκες. Το σύστημα σε μια προσπάθεια προσαρμογής στις απαιτήσεις που προκύπτουν θα μπορούσε να κάνει προσθήκη ή αφαίρεση περιφερειακών, αυξομείωση του μεγέθους της μνήμης αλλά και να συνδέει τα προαναφερθέντα περιφερειακά στις κατάλληλες εξόδους (ακροδέκτες κλπ). Σε μια τέτοια υλοποίηση πιθανότατα ένα τμήμα του υλικού θα ήταν πάντα σταθερό με μόνο στόχο να προγραμματίζει και να προσαρμόζει το υπόλοιπο υλικό το οποίο θα διαχειρίζεται τις ανάγκες που υφίστανται.

6.1.2 Επεκτάσεις και βελτιστοποιήσεις στην εφαρμογή του χρήστη

Σχετικά με την εφαρμογή στο τερματικό του χρήστη ένα σημαντικό θέμα, που θα μπορούσε μελλοντικά να βελτιωθεί, αφορά την ασφαλή, γρήγορη και έγκυρη παράδοση των δεδομένων στο απομακρυσμένο ενσωματωμένο σύστημα.

Το πρωτόκολλο TCP χρησιμοποιήθηκε σκόπιμα καθώς εξασφαλίζει την έγκυρη παράδοση των δεδομένων. Παρόλα αυτά, απρόσμενα διαδικτυακά προβλήματα θα έκαναν χρήσιμη την ύπαρξη ενός επιπλέον μηχανισμού που επιβεβαιώνει την παράδοση των δεδομένων.

Όσον αφορά την ασφάλεια θα ήταν σημαντικό να χρησιμοποιηθεί κωδικοποίηση των δεδομένων ώστε το ενσωματωμένο σύστημα να μην επηρεάζεται από κακόβουλη χρήση. Θα μπορούσε επίσης να επαληθεύεται ο αποστολέας των δεδομένων ώστε να αποφεύγεται και η μη εξουσιοδοτημένη χρήση.

Σε περιπτώσεις που η ταχύτητα παίζει σημαντικό ρόλο θα έδινε λύση η παράλληλη χρήση πρωτοκόλλου επικοινωνίας UDP. Ο χρήστης της εφαρμογής σε περιπτώσεις που αναζητά την ταχύτερη δυνατή παράδοση των δεδομένων θα μπορούσε, σύμφωνα με την κρίση του , να επιλέγει το πρωτόκολλο UDP για μετάδοση με μειονέκτημα την πιθανότητα απώλειας δεδομένων. Κάτι τέτοιο θα είχε εφαρμογή σε περιπτώσεις χρήσης του ενσωματωμένου συστήματος για τηλεκατεύθυνση ή σε περιπτώσεις συνεχούς μετάδοσης εικόνας (streaming).

Μια ενδιαφέρουσα μελλοντική προσθήκη θα ήταν η δυνατότητα προβολής εισερχόμενης εικόνας μέσα στο περιβάλλον της εφαρμογής. Το ενσωματωμένο σύστημα μπορεί να μεταδώσει προς τον χρήστη δεδομένα από κάμερα εφόσον αυτή επικοινωνεί με κάποια από τα πρωτόκολλα που αυτό υποστηρίζει. Συνεπώς η μετάδοση εικόνας (streaming) από την κάμερα θα μπορούσε να προβάλλεται απευθείας στο περιβάλλον της εφαρμογής.

Ένα τελευταίο από τα πολλά παραδείγματα βελτίωσης της εφαρμογής του χρήστη θα ήταν η χρήση συντομεύσεων πληκτρολογίου. Ο χρήστης θα ήταν σε θέση να αναθέτει σε κάθε πλήκτρο διαφορετική ενέργεια. Θα μπορούσε, για παράδειγμα, ένα σύνολο πλήκτρων να ελέγχει τους ακροδέκτες του ενσωματωμένου συστήματος και το πάτημα ή η απελευθέρωση των πλήκτρων να εναλλάσσει την κατάσταση των ακροδεκτών μεταξύ λογικού μηδέν και ένα. Μια τέτοια βελτίωση θα ήταν χρήσιμη σε περιπτώσεις όπως της τηλεκατεύθυνσης, που προαναφέρθηκε, όπου ο χρήστης θα πρέπει να ανταποκρίνεται άμεσα σε αλλαγές της συμπεριφοράς του οχήματος.

6.1.3 Επεκτάσεις και βελτιστοποιήσεις στην εφαρμογή του server

Η εφαρμογή του server είναι στην ουσία ένας απλός μηχανισμός μεταβίβασης δεδομένων. Παρόλο που πιθανότατα η εφαρμογή αυτή θα συνεχίσει να έχει αυτόν τον βασικό ρόλο υπάρχει και εδώ χώρος για βελτίωση.

Η εφαρμογή αυτή θα μπορούσε να λειτουργεί ως ένας κεντρικός μηχανισμός μεταβίβασης δεδομένων ώστε να μπορεί ταυτόχρονα να υποστηρίξει και να εξυπηρετήσει πολλά ενσωματωμένα συστήματα και πολλούς απομακρυσμένους χρήστες για κάθε ένα σύστημα αντίστοιχα.

Ένα δεύτερο παράδειγμα θα ήταν η δυνατότητα πρόσβασης στο ενσωματωμένο σύστημα από διαφορετικές πηγές. Ένας απομακρυσμένος χρήστης θα μπορούσε να αλληλεπιδράει με το σύστημα, ακόμα και ταυτόχρονα, από το κινητό του (i-Phone ή Android), από τον υπολογιστή του ή από έναν άλλο υπολογιστή. Δεδομένου ότι υπάρχει έλεγχος εξουσιοδότησης μια τέτοια βελτίωση θα έδινε μεγάλη ευελιξία.

6.2 Συμπεράσματα

Σκοπός της εργασίας αυτής ήταν η υλοποίηση ενός συστήματος που δίνει την δυνατότητα σε έναν χρήστη να επικοινωνήσει αλλά και να αλληλεπιδράσει διαδικτυακά και ασύρματα με απομακρυσμένες συσκευές και εξαρτήματα. Παραδείγματα τέτοιων είναι μεταξύ άλλων μικροελεγκτές, αισθητήρες, διακόπτες και leds.

Προκειμένου να επιτευχθεί ο ζητούμενος στόχος και να ικανοποιηθούν οι απαιτούμενες προσδοκίες χρειάστηκε σε κάθε στάδιο πολύ έρευνα που ως σύνολο κάλυψε μεγάλο εύρος γνώσεων στο πεδίο της τεχνολογίας υπολογιστών. Η επιτυχία αυτής της εργασίας είναι αποτέλεσμα μελέτης που περιλαμβάνει, μεταξύ άλλων, αρχιτεκτονική ενσωματωμένων συστημάτων, σχεδιασμό υλικού και ανάπτυξη λογισμικού, διαδικτυακές γνώσεις, πρωτόκολλα επικοινωνίας (TCP/UDP, I^2C, RS-232, GPIO), γλώσσες προγραμματισμού (java και c), γνώσεις λειτουργίας modem και εντολών modem (AT commands), εξειδίκευση σε λειτουργία αισθητήρων και κατασκευή PCB.

Στα πλαίσια υλοποίησης αυτής της εργασίας, στα διάφορα στάδια έρευνας, σχεδιασμού, πειραματισμού, υλοποίησης αλλά και διόρθωσης/τροποποίησης επιλέχτηκαν τα καταλληλότερα μέσα (υλικό και λογισμικό) που θα έδιναν το βέλτιστο αποτέλεσμα μέσα στα περιθώρια των γνωστικών και οικονομικών δυνατοτήτων.

Αποτέλεσμα ήταν η επιτυχής ανάπτυξη ενός συστήματος που ξεπερνάει το πλάνο που είχε αρχικά τεθεί. Η εργασία αυτή, που απαρτίζεται από το ενσωματωμένο σύστημα και τις εφαρμογές στο τερματικό του χρήστη και στον server, παραδίδει υπηρεσίες διαδικτυακής επικοινωνίας με απομακρυσμένες συσκευές και εξαρτήματα σε πλαίσια αξιοπιστίας και ευελιξίας.

Βιβλιογραφία

[1] Embedded System, http://en.wikipedia.org/wiki/Embedded_system.

[2] Field-programmable gate array, http://en.wikipedia.org/wiki/Field-programmable_gate_array.

[3] FPGA, http://el.wikipedia.org/wiki/FPGA.

[4] Universal asynchronous receiver/transmitter, http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter.

[5] I^2C, http://en.wikipedia.org/wiki/I%C2%B2C.

[6] Using the I2C Bus, http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html.

[7] I2C-Bus: What's that?, http://www.i2c-bus.org.