ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

Διαδίκτυο των Πραγμάτων: Ευφυή Περιβάλλοντα σε Δίκτυα Νέας Γενιάς

Ανάπτυξη και διασύνδεση υλικού – λογισμικού σε κόμβους SoC FPGA υψηλής επεξεργαστικής ισχύος με στόχο την ενσωμάτωσή τους σε υπάρχον δίκτυο συσκευών IoT (Internet of Things)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Πέντε Γεώργιου

Επιβλέπων : Καλλίγερος Εμμανουήλ

Μέλη εξεταστικής επιτροπής: Σκιάνης Χαράλαμπος, Σκούτας Δημήτριος

Σάμος, Οκτώβριος 2022

Πρόλογος και ευχαριστίες

Η παρούσα Διπλωματική Εργασία εκπονήθηκε, στο πλαίσιο του Μεταπτυχιακού Προγράμματος Σπουδών "Διαδίκτυο των Πραγμάτων: Ευφυή περιβάλλοντα σε δίκτυα νέας γενιάς" του τμήματος Μηχανικών Πληροφοριακών και Επικοινωνιακών συστημάτων του Πανεπιστημίου Αιγαίου.

Η εργασία υλοποιήθηκε υπό την επίβλεψη του μόνιμου επίκουρου καθηγητή Κ. Καλλίγερου Εμμανουήλ και του επιστημονικού συνεργάτη του, Κ. Λογαρά Ευάγγελου. Θα ήθελα να ευχαριστήσω ιδιαίτερα και τους δύο, για την δυνατότητα που μου δώσανε να ασχοληθώ με το συγκεκριμένο θέμα, πράγμα το οποίο το βρήκα πολύ ενδιαφέρον και χρήσιμο, για την καθοδήγηση που μου έδωσαν, την άμεση βοήθεια τους σε ότι χρειάστηκα, την ψυχολογική υποστήριξη που μου παρείχαν και την υπομονή τους, καθ' όλη την διάρκεια της συγγραφής της παρούσας εργασίας.

Θα ήθελα να ευχαριστήσω επίσης τους καθηγητές του προγράμματος για την γνώση που μου μετέφεραν, τους συμφοιτητές και πλέον πολύ καλούς μου φίλους Δημήτρη Τ., Θάνο Α. και Κωνσταντίνα Κ. για την βοήθεια τους, αλλά και την σύντροφο μου και την οικογένεια μου για την στήριξη και την υπομονή που μου παρείχαν.

Πίνακας περιεχομένων

Λίστα Σχημάτων

Λίστα Πινάκων

Ακρωνύμια

ΔτΠ	Διαδίκτυο των Πραγμάτων
IoT	Internet of Things
SoC	System on Chip
FPGA	Field Programmable Gate Array (Επαναδιαμορφούμενη ψηφιακή λογική)
IC	Integrated Circuit (Ολοκληρωμένο κύκλωμα)
RFID	Radio Frequency Identification (Ταυτοποίηση μέσω ραδιοσυχνοτήτων)
PS	Processor Subsystem (Υποσύστημα επεξεργαστή)
PL	Programmable Logic (Επαναπρογραμματιζόμενη λογική)
MQTT	Message Queue Telemetry Transport (Μεταφορά μηνυμάτων τηλεμετρίας σε σειρά)
IP	Intellectual Property (Πνευματική ιδιοκτησία)
AMBA	Advanced Microcontroller Bus Architecture (Εξελιγμένη αρχιτεκτονική διαύλου μικοεπεξεργαστών)
AXI	Advanced eXtensible Interface (Εξελιγμένη διεπαφή με δυνατότητα επέκτασης)
TCP/IP	Transmission Control Protocol (Πρωτόκολλο ελέγχου μετάδοσης) / Internet Protocol (Πρωτόκολλο διαδικτύου)
BSP	Board Support Package (Αρχείο υποστήριξης υλικού)
FSM	Functional State Machine (Μηχανή πεπερασμένων καταστάσεων)
SDK	Software Development Kit (Πακέτο ανάπτυξης λογισμικού)
SSH	Secure Shell (Ασφαλής επικοινωνία)
SFTP	Secure File Transfer Service (Υπηρεσία ασφαλής μεταφοράς δεδομένων)
DDR	Double Data Rate (Διπλού ρυθμού)
DLL	Dynamic Link Library (Δυναμικός σύνδεσμος βιβλιοθήκης)

Περίληψη

Το διαδίκτυο των πραγμάτων (ΔτΠ) αποτελεί σημαντικό τεχνολογικό παράγοντα της καθημερινότητας του ανθρώπου. Ολοένα και περισσότερες συσκευές χρειάζεται να διασυνδεθούν μεταξύ τους αλλά και με το διαδίκτυο, ώστε να συμβάλλουν στην εξυπηρέτηση των ανθρώπων σε σημαντικές ή και όχι στιγμές της καθημερινότητας τους.

Σε κάποιες περιπτώσεις, οι συσκευές που χρειάζεται να διασυνδεθούν στο δίκτυο αναλαμβάνουν να επιτελούν εξειδικευμένες λειτουργίες. Αυτό έχει ως αποτέλεσμα την ανάγκη ύπαρξης υψηλής υπολογιστικής ισχύος στις παρυφές του δικτύου για την άμεση επεξεργασία των περισυλλεγμένων δεδομένων. Στην ανάγκη αυτή έρχεται να δώσει λύση η παρούσα διπλωματική εργασία, με την ανάπτυξη ενός τέτοιου συστήματος.

Για την ανάπτυξη του συστήματος, επιλέχθηκε η χρήση υλικού τύπου System on Chip (SoC), το οποίο ενσωματώνει επεξεργαστή τύπου ARM και αναδιαμορφούμενη ψηφιακή λογική (FPGA), ενώ ταυτόχρονα αποτελεί υλικό χαμηλής ενεργειακής κατανάλωσης, πράγμα που είναι πολύ σημαντικό σε ένα δίκτυο τύπου ΔτΠ. Στο υλικό αυτό εγκαταστάθηκε ειδική υποστηριζόμενη διανομή λειτουργικού συστήματος τύπου Linux για την λειτουργία του επεξεργαστή. Αναπτύχθηκε ειδικό λογισμικό ώστε να μπορεί να δέχεται δεδομένα από το διαδίκτυο, ενώ αναπτύχθηκε λογισμικό χαμηλού επιπέδου (low level driver) για την μεταφορά των δεδομένων από το επίπεδο του επεξεργαστή προς το επίπεδο της FPGA. Στο επίπεδο της FPGA σχεδιάστηκε ειδικό υλικό για την ταχύτατη επεξεργασία των εισερχόμενων δεδομένων και την εξαγωγή αποτελέσματος σύμφωνα με τις ανάγκες της εργασίας. Τελικό μέρος της λειτουργίας του λογισμικού που αναπτύχθηκε, αποτελεί η επιστροφή του εξαχθέντος αποτελέσματος στο επίπεδο του επεξεργαστή, μέσω του λογισμικού χαμηλού επιπέδου και στην συνέχεια η μετάδοση της χρήσιμης πληροφορίας προς τις υπόλοιπες συσκευές του δικτύου.

Η συγκεκριμένη εργασία δύναται να αποτελέσει μια έτοιμη λύση, εύκολη στη χρήση, για όποιον έχει ανάγκη να ενσωματώσει έναν κόμβο υψηλότερης υπολογιστικής ισχύος σε μια τοπολογία ΔτΠ. Η μόνη αλλαγή που θα χρειαστεί να γίνει, είναι κυρίως στην σχεδίαση της εξειδικευμένης λειτουργίας στην αναδιαμορφούμενη ψηφιακή λογική, σύμφωνα με τις ανάγκες της εργασίας που πρόκειται να επιτελέσει. Σε άλλη περίπτωση, μπορεί απλά να χρησιμοποιηθεί ως ένα βοηθητικό εργαλείο για το πώς αναπτύσσεται και ενσωματώνεται ένας κόμβος υψηλής υπολογιστικής ισχύος σε τοπολογία ΔτΠ.

Λέξεις Κλειδιά: *Διαδίκτυο των πραγμάτων, κόμβος υψηλής επεξεργαστικής ισχύος, SoC, FPGA, AMBA BUS, AXI4-Lite, C, Python, MQTT*

Abstract

Internet of Things (IoT) is an important technological feature of the majority of people's lives on daily basis. The need for interconnection between digital devices and the connection of said devices to the internet, keeps rising with each passing year, utilized as means for both important and trivial daily needs. There are cases where devices connected to the internet are designed and are capable of certain more sophisticated and dedicated functions. Due to this fact, the need for high computing power of the said network and the capacity to process the collected data, is frequently in high demand. This Thesis is dedicated to solve the former issue, establishing a solution through a system created for this specific reason.

The system's development and production were made by using a System on Chip (SoC). An ARM Processor is embedded on this chip accompanied by a Field Programmable Gate Array (FPGA) unit. The low energy consumption of this system, is one of its fundamental aspects, as it is an important factor when Internet of Things systems are developed. This device is running on a Linux type operating system which is installed so the processor functions properly. A special program was developed, dedicated to receive data from the Internet. In conclusion, a low-level software was developed so the data could be transferred from the processor towards the FPGA platform. On the other hand, a special device was designed in the FPGA system, so the incoming data could be processed as quickly as possible resulting to the appropriate outcome depending on the current demands of the user. The final part of the developed software is dedicated to the diversion of the results back to the processor via the low-level software mentioned before. The extracted information is delivered to the rest of the devices connected to the network.

This Thesis is meant to provide a fully functional and complete solution, easy to use and relatively user-friendly, to those that want to include a node of high processing power on an Internet of Things topology. Depending on the needs of each user, some minor changes should be made on the FPGA platform, according to the needs of their respecting projects. In general, this system can be utilized as a side tool assisting the development of a high processing power node in an Internet of Things environment.

Keywords: *Internet of things, high performance edge node, SoC, FPGA, AMBA BUS, AXI4-Lite, C, Python, MQTT*

1

Εισαγωγή

Η εξέλιξη της τεχνολογίας και συγκεκριμένα του Διαδικτύου των πραγμάτων έχει επηρεάσει αρκετά την καθημερινότητα μας, μέσω των συσκευών που χρησιμοποιούμε. Επίσης η πολυπλοκότητα των συσκευών αυτών αυξάνεται όσο αυξάνονται οι απαιτήσεις που έχουμε από τις διάφορες συσκευές και τα διάφορα συστήματα. Στην ανάγκη αυτή για πιο εξελιγμένα συστήματα, έρχεται να προτείνει μια λύση η παρούσα διπλωματική εργασία.

1.1 Το Διαδίκτυο των Πραγμάτων

Η διασύνδεση των συσκευών μεταξύ τους αλλά και με το διαδίκτυο αποτελεί σημαντικό πεδίο της καινοτομίας των συσκευών των τελευταίων χρόνων. Οι διασυνδεδεμένες συσκευές μπορούν να αποτελούν στοιχεία ενός ευρύτερου δικτύου, του διαδικτύου των πραγμάτων (ΔτΠ). Ο όρος διαδίκτυο των πραγμάτων σύμφωνα με τον Dorsemaine κ.ά [1], περιγράφει ένα σύνολο υποδομών που διασυνδέουν αντικείμενα, επιτρέποντας την διαχείριση τους, την εξόρυξη γνώσης και την πρόσβαση στα δεδομένα που παράγονται.

Η ιδέα του διαδικτύου των πραγμάτων ξεκινάει αρκετά χρόνια πριν. Μερικές, από τις πιο σημαντικές στιγμές αναφέρονται στη συνέχεια, το 1982, όπου παρουσιάζεται η έννοια του διαδικτύου (Internet) και του πρωτοκόλλου επικοινωνίας TCP/IP. Έναν χρόνο αργότερα, το 1983, αναπτύσσεται η επικοινωνία μηχανής με μηχανή (Machine to Machine) από φοιτητές στο πανεπιστήμιο Carnegie Mellon. Το 1995, η Siemens, αναπτύσσει το 1ο ολοκληρωμένο κύκλωμα το οποίο επικοινωνεί μέσω δικτύου Global System for Mobile Communications (GSM) για τον έλεγχο βιομηχανικών συστημάτων. Το 1999, αποτελεί χρονιά σταθμό για τα πρώτα βήματα του ΔτΠ, καθώς ερευνητές από το πανεπιστήμιο MIT, χρησιμοποιούν τεχνολογίες Radio Frequency Identification (RFID) και Bluetooth για την ασύρματη εγγραφή και ανάγνωση δεδομένων από αντικείμενα. Το 2000, εδραιώνεται η επικοινωνία μηχανής με μηχανή με αποτέλεσμα μηχανές που

βρίσκονται στο ίδιο δίκτυο να επικοινωνούν και να αλληλεπιδρούν μεταξύ τους. Οι χρονιές 2006 – 2008 αποτελούν ορόσημο για το ΔτΠ, καθώς αναγνωρίζεται για πρώτη φορά από την Ευρωπαϊκή Ένωση, ο όρος «Διαδίκτυο των Πραγμάτων» - Internet of Things (IoT). Έκτοτε, πολλές αναβαθμίσεις της υπάρχουσας υποδομής αλλά και καινοτομίες εμπλουτίζουν συνέχεια το ΔτΠ.

Σήμερα, ολοένα και περισσότερες συσκευές που χρησιμοποιούμε στην καθημερινότητα μας, αποτελούν συσκευές που μπορούν να συνδεθούν στο διαδίκτυο. Τέτοιες συσκευές μπορεί απλά να μας εξυπηρετούν στην καθημερινότητα μας και να την καθιστούν πιο εύκολη, όπως για παράδειγμα οι έξυπνοι λαμπτήρες στο σπίτι, ή μπορεί να είναι πολύ σημαντικές για τον άνθρωπο, όπως συσκευές που εξυπηρετούν ηλικιωμένους και καλούν βοήθεια όταν συμβεί κάτι έκτακτο. Στο ΔτΠ χρησιμοποιείται αρκετά ο όρος «Έξυπνος», ο όρος αυτός υποδηλώνει την δυνατότητα των συσκευών να λειτουργούν υπό συνθήκη, χωρίς την παρέμβαση του ανθρώπου. Η υπό συνθήκη λειτουργία μιας συσκευής μπορεί να οφείλεται σε παράγοντες, όπως διάφοροι αισθητήρες ή ακόμα και έλεγχος από άλλες συσκευές – συστήματα. Στο ΔτΠ μπορεί να συναντήσουμε μικρές τοπολογίες ΔτΠ, όπως ένα έξυπνο σπίτι, όπου για παράδειγμα μπορεί να είναι διασυνδεδεμένες μεταξύ τους αρκετές συσκευές του σπιτιού, όπως το σύστημα θέρμανσης και ψύξης, συστήματα σκίασης, συστήματα συλλογής δεδομένων από αισθητήρες, με απώτερο σκοπό τη λήψη αποφάσεων για την εξοικονόμηση περιβαλλοντικών και οικονομικών πόρων, αλλά και μικρότερες συσκευές όπως η καφετιέρα, η ηλεκτρική σκούπα, η τηλεόραση κυρίως για βελτίωση της λειτουργικότητάς τους (π.χ. απομακρυσμένη χρήση). Τοπολογίες ΔτΠ όμως συναντιούνται και σε μεγαλύτερη κλίμακα, όπως η τοπολογία ΔτΠ μιας έξυπνης πόλης. Σε μια έξυπνη πόλη μπορεί να έχουμε πάρα πολλά στοιχεία – αντικείμενα διασυνδεδεμένα μεταξύ τους. Μπορεί για παράδειγμα το κάθε σπίτι να έχει έξυπνους μετρητές κατανάλωσης ενέργειας και νερού, ώστε να γίνεται αποδοτική χρήση των υποδομών και να μπορεί να δίνεται η δυνατότητα στους πολίτες να λειτουργούν ως καταναλωτές αλλά και ως προμηθευτές. Επίσης, χρησιμοποιώντας διάφορες καινοτομίες σε μια έξυπνη πόλη, μπορεί να επιτευχθεί μείωση των καταναλισκόμενων πόρων, όπως για παράδειγμα εγκαθιστώντας έξυπνους λαμπτήρες φωτισμού, έξυπνους σηματοδότες, έξυπνους κάδους απορριμμάτων. Ένας ακόμη χώρος όπου επίσης συναντάται μεγάλης κλίμακας χρήση του ΔτΠ είναι η βιομηχανία. Η εξέλιξη της βιομηχανίας και η ένταξη του ΔτΠ σε αυτή, περιγράφεται ως Βιομηχανία 4.0. Το ΔτΠ δίνει δυνατότητες, όπως είναι, η μείωση του κόστους και του χρόνου παραγωγής, η παραγωγή ποιοτικότερων προϊόντων, αλλά και η μείωση εκπομπών CO_2, βελτιώνοντας έτσι το περιβαλλοντικό αποτύπωμα της βιομηχανίας. Η γεωργία είναι και αυτός ένας τομέας όπου το ΔτΠ έχει εδραιωθεί παίζοντας σημαντικό ρόλο. Ολοένα και περισσότερες καλλιέργειες μετατρέπονται σε έξυπνες, με αποτέλεσμα να επιτηρείται όλος ο κύκλος παραγωγής των προϊόντων και να γίνεται απλούστερη και πιο οικονομική η διαδικασία της παραγωγής σε σχέση με την παραγωγή σε μια συμβατική καλλιέργεια.

Λόγω της εξέλιξης της τεχνολογίας αλλά της αύξησης των απαιτήσεων καθημερινά, ολοένα και περισσότερες ανάγκες δημιουργούνται για σχεδίαση και ανάπτυξη συσκευών που να μπορούν να διασυνδεθούν στο δίκτυο και να κάνουν ακόμα πιο πολύπλοκες εργασίες.

1.2 Η ανάγκη για πιο εξελιγμένους κόμβους παρυφής στο δίκτυο IoT

Η ραγδαία εξέλιξη του ΔτΠ μέχρι σήμερα έχει επιφέρει την ανάγκη για πιο εξελιγμένους κόμβους παρυφής (edge nodes). Ολοένα και μεγαλύτερη επεξεργαστική ισχύ χρειάζεται στις παρυφές του δικτύου καθώς συλλέγεται και επεξεργάζεται μεγαλύτερος όγκος δεδομένων. Για παράδειγμα, μπορεί κόμβοι που βρίσκονται στις παρυφές του δικτύου να συλλέγουν εικόνες με χρήση κάμερας. Έπειτα, τα δεδομένα αυτά χρειάζεται να υποβληθούν σε επεξεργασία, επειδή όμως ο όγκος τους είναι μεγάλος, δεν είναι επιθυμητό όλη αυτή η πληροφορία να μεταδοθεί και να επεξεργαστεί κεντρικά στο νέφος (cloud). Επομένως, θα ήταν προτιμότερο να επεξεργαστεί τοπικά. Για να επιτευχθεί η επεξεργασία αυτή, απαιτείται η αναβάθμιση της επεξεργαστικής ισχύος των κόμβων. Ένα άλλο παράδειγμα κόμβων με υψηλή επεξεργαστική ισχύ είναι αυτό της αυτόνομης οδήγησης. Στην αυτόνομη οδήγηση, ένα σύνολο αισθητήρων και καμερών, συλλέγουν και επεξεργάζονται δεδομένα, με αποτέλεσμα να μπορούν να αναλύουν την θέση και την κατάσταση του οχήματος και να επικοινωνούν με γειτονικά οχήματα, σταθμούς βάσης και άλλα αντικείμενα όπως για παράδειγμα τους φωτεινούς σηματοδότες και τους έξυπνους φορτιστές. Για να γίνει όλο αυτό, είναι εμφανές ότι απαιτείται κάθε κόμβος του δικτύου του οχήματος να διαθέτει σημαντική υπολογιστική ισχύ, καθώς μεγάλος αριθμός δεδομένων θα πρέπει να τύχουν επεξεργασίας σε ελάχιστο χρόνο καθώς η κάθε χρονική στιγμή είναι σημαντική. Ένα τρίτο παράδειγμα στο οποίο υπάρχει ανάγκη χρήσης κόμβων υψηλής επεξεργαστικής ισχύος αποτελεί το απτικό διαδίκτυο (tactile internet). Το απτικό διαδίκτυο περιλαμβάνει μια σειρά εφαρμογών, με την βοήθεια των οποίων μπορεί να αλληλεπιδρά ένας άνθρωπος με έναν άλλο άνθρωπο ή με ένα μηχάνημα, ενώ μπορεί να βρίσκονται σε πάρα πολύ μεγάλη απόσταση μεταξύ τους. Παραδείγματα σχετικών εφαρμογών είναι η επαυξημένη πραγματικότητα, η εξ' αποστάσεως εκπαίδευση, η εξ' αποστάσεως ιατρική. Από τη χρησιμότητα και την περιπλοκότητα των εφαρμογών αυτών, γίνεται κατανοητό ότι χρειάζεται μεγάλη επεξεργαστική ισχύς στα υπολογιστικά συστήματα που θα χρησιμοποιηθούν στις παρυφές του δικτύου καθώς και υψηλή ταχύτητα των τηλεπικοινωνιακών υποδομών αυτού.

Η παρούσα διπλωματική εργασία κινείται ακριβώς προς την κατεύθυνση αυτή, δηλαδή την ανάπτυξη κόμβων παρυφής IoT υψηλής υπολογιστικής ισχύος. Συγκεκριμένα, αναπτύσσεται ένα σύστημα στην αναπτυξιακή πλακέτα Minized[1] της εταιρείας AVNET, με σκοπό την ταχύτατη επεξεργασία δεδομένων. Η πλακέτα αυτή παρέχει ολοκληρωμένο κύκλωμα τύπου Field Programmable Gate Array (FPGA) στο οποίο μπορεί να αναπτυχθεί υλικό που να επεξεργάζεται μεγάλο όγκο δεδομένων σε ελάχιστο χρόνο μαζί με επεξεργαστή τύπου ARM[2] όπου μπορεί να γραφεί εφαρμογή για την διαχείριση των δεδομένων αυτών, δημιουργώντας έτσι έναν κόμβο υψηλής επεξεργαστικής ισχύος.

1.3 Η διάρθρωση της εργασίας

Η διάρθρωση της παρούσας εργασίας έχει ως εξής: Το 1ο κεφάλαιο αποτελεί μια εισαγωγή στο ΔτΠ και στην ανάγκη για κόμβους υψηλής επεξεργαστικής ισχύος. Στο 2ο κεφάλαιο, παρουσιάζεται ο σκοπός της διπλωματικής εργασίας, αναλύονται τα δομικά στοιχεία που χρησιμοποιήθηκαν κατά

[1] https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/minized/

[2] https://www.arm.com/

την ανάπτυξη του εξειδικευμένου συστήματος που υλοποιήθηκε και γίνεται μια λεπτομερής αναφορά στην αναπτυξιακή πλακέτα Minized. Στο 3ο κεφάλαιο, γίνεται εκτενής αναφορά στη διαδικασία ανάπτυξης του συστήματος που υλοποιήθηκε στο Minized. Περιγράφεται η δημιουργία του προσαρμοσμένου υλικού, η ρύθμιση του λειτουργικού συστήματος της πλακέτας και η εφαρμογή που αναπτύχθηκε, για την λειτουργία του όλου συστήματος και την διασύνδεση αυτού με το δίκτυο. Στο 4ο κεφάλαιο, παρουσιάζονται τα αποτελέσματα από την ανάπτυξη του συστήματος ως προς την ταχύτητα επεξεργασίας δεδομένων και στο τελευταίο κεφάλαιο, γίνεται αναφορά στην συνεισφορά της παρούσας εργασίας.

2

Σκοπός διπλωματικής

Στο παρόν κεφάλαιο αναλύεται ο σκοπός της παρούσας διπλωματικής εργασίας. Στην αρχή γίνεται αναφορά στο σκοπό αυτό και στη συνέχεια αναλύονται τα μέρη από τα οποία αποτελείται η εργασία. Τέλος, παρουσιάζεται η αναπτυξιακή πλακέτα που χρησιμοποιήθηκε για την ανάπτυξη της συγκεκριμένης εφαρμογής.

2.1 Σκοπός διπλωματικής εργασίας

Σκοπός της εργασίας αυτής, είναι να προτείνει μια λύση στο ζήτημα της ανάπτυξης κόμβων παρυφής υψηλής επεξεργαστικής ισχύος σε ένα δίκτυο IoT. Για τον σκοπό αυτόν, αναπτύχθηκε εφαρμογή στην αναπτυξιακή πλακέτα “Minized”, με στόχο την επιτάχυνση της επεξεργασίας των δεδομένων. Η λήψη των δεδομένων γίνεται μέσω διαδικτύου, χρησιμοποιώντας τη σύνδεση WiFi του board και το πρωτόκολλο Message Queue Telemetry Transport (MQTT). Το πρωτόκολλο MQTT είναι εγκατεστημένο στο λειτουργικό σύστημα που εκτελείται στον επεξεργαστή τύπου ARM του board. Τα δεδομένα έπειτα μεταφέρονται στο υλικό FPGA, στο οποίο έχει διαμορφωθεί ειδικό περιφερειακό για την εκτέλεση μιας συγκεκριμένης εργασίας. Το υλικό αυτό, λόγω του ότι έχει σχεδιαστεί ώστε να εκτελεί μόνο μια συγκεκριμένη εργασία, πραγματοποιεί τη σχετική επεξεργασία των δεδομένων σε πάρα πολύ μικρό χρονικό διάστημα. Επίσης, η διασύνδεση του υλικού με τον επεξεργαστή γίνεται μέσω του γρήγορου διαύλου επικοινωνίας AXI Lite, πράγμα που συνεισφέρει και αυτό στην ταχύτατη επεξεργασία των δεδομένων. Τέλος, χρησιμοποιήθηκε η γλώσσα Python, για τη διαχείριση των δεδομένων και η γλώσσα C για την ανάπτυξη οδηγού χαμηλού επιπέδου (low level driver). Συγκεκριμένα, η εφαρμογή Python αναλαμβάνει τη λήψη των δεδομένων μέσω MQTT και την αποστολή αυτών, με χρήση του low level driver, στο ειδικό περιφερειακό στο FPGA. Όταν η επεξεργασία των δεδομένων στο υλικό ολοκληρωθεί, η εφαρμογή Python αναλαμβάνει την ανάγνωση του αποτελέσματος, με χρήση του low level driver, από το περιφερειακό, την επιστροφή του αποτελέσματος στον επεξεργαστή και την αποστολή αυτού πίσω στο υπόλοιπο δίκτυο IoT.

2.2 Τα δομικά μέρη του συστήματος

Τα δομικά μέρη στα οποία βασίζεται το σύστημα που αναπτύχθηκε με στόχο τη δημιουργία ενός κόμβου παρυφής υψηλής επεξεργαστικής ισχύος και αναφέρονται παρακάτω είναι:

- Το υλικό System on Chip
- Το πρωτόκολλο διασύνδεσης AXI4-Lite
- Το λειτουργικό σύστημα Petalinux
- Το πρωτόκολλο επικοινωνίας MQTT

2.2.1 Το υλικό System on Chip

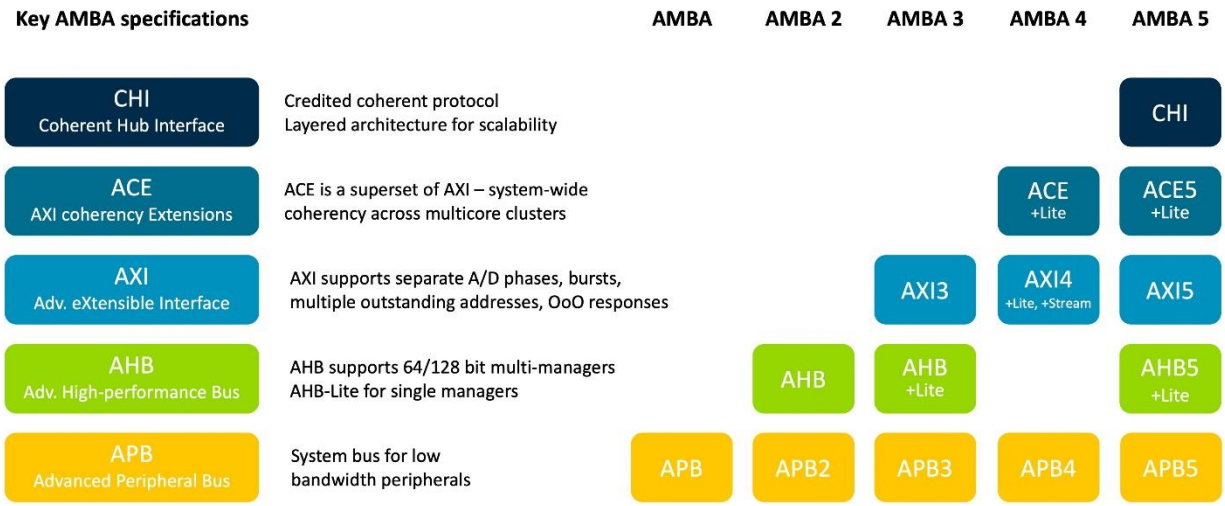
Ξεκινώντας από το υλικό, χρησιμοποιήθηκε ένα ολοκληρωμένο τύπου System on Chip (SoC). Με τον όρο SoC περιγράφεται υλικό που είναι τοποθετημένο σε ένα ολοκληρωμένο κύκλωμα και εσωτερικά αποτελείται από έτοιμες δομικές μονάδες, intellectual property (IP) blocks, οι οποίες έχουν σχεδιαστεί και δοκιμαστεί σε προηγούμενο χρόνο και μπορούν να χρησιμοποιηθούν σε διάφορα συστήματα. Τέτοιες δομικές μονάδες μπορεί να είναι ενσωματωμένοι επεξεργαστές, μνήμες, κυκλώματα διεπαφής, αναλογικά στοιχεία και άλλα κυκλωματικά στοιχεία τα οποία εξυπηρετούν την εκάστοτε εφαρμογή [2]. Ένα ολοκληρωμένο τύπου SoC περιλαμβάνει επίσης και λογική διασύνδεσης των μονάδων του, ενώ μπορεί να περιέχει και επαναδιαμορφούμενη λογική. Σε αυτή, ο σχεδιαστής του συστήματος μπορεί να υλοποιήσει όποια δομικά στοιχεία χρειάζεται από το σύνολο που παρέχει ο κατασκευαστής του ολοκληρωμένου, από τρίτους κατασκευαστές αλλά και μονάδες που ο ίδιος θα υλοποιήσει.

Η δυνατότητα της επαναδιαμορφούμενης λογικής παρέχεται χρησιμοποιώντας υλικό τύπου FPGA, όπως έγινε και στην παρούσα εργασία. Στην ουσία πρόκειται για υλικό όπου τα δομικά του στοιχεία διαμορφώνονται και συνδέονται αναλόγως της εκάστοτε σχεδίασης με αποτέλεσμα η λειτουργία του συστήματος να είναι κάθε φορά διαφορετική. Σε περίπτωση που ο σχεδιαστής αποφασίσει να αλλάξει κάτι στη σχεδίαση του συστήματος μπορεί με εύκολο τρόπο να μεταφέρει στο υλικό τη νέα σχεδίαση. Σημαντικά χαρακτηριστικά των FPGA, είναι επίσης η δυνατότητα ανάπτυξης περίπλοκων ψηφιακών κυκλωμάτων με μεγάλη επεξεργαστική ισχύ, πράγμα που τα κάνει ιδανικά σε περιπτώσεις συστημάτων με υψηλή πολυπλοκότητα και απαιτήσεις μικρού χρόνου απόκρισης.

2.2.2 Το πρωτόκολλο διασύνδεσης AXI4-Lite

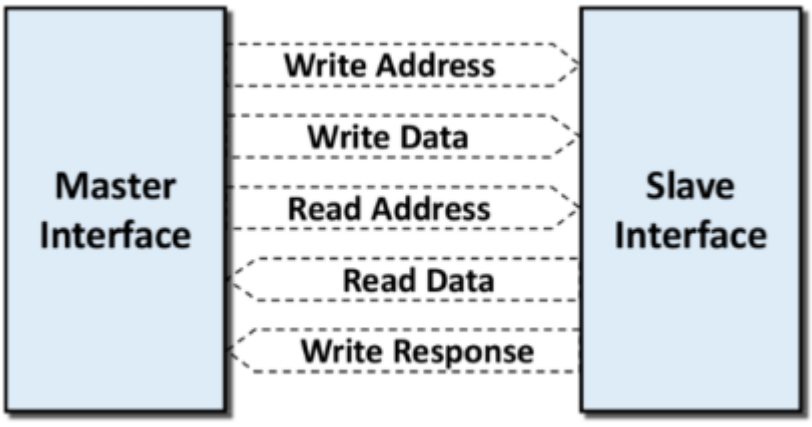
Για την επίτευξη υψηλής επεξεργαστικής ισχύος στο σύστημα που αναπτύχθηκε, ένα ακόμη σημαντικό στοιχείο που χρησιμοποιήθηκε στην σχεδίαση του υλικού, ήταν αυτό της διασύνδεσης των δομικών μονάδων μέσω του πρωτοκόλλου Advanced eXtensible Interface (AXI) και συγκεκριμένα του AXI4-Lite. Πρόκειται για ένα πρωτόκολλο επικοινωνίας που χρησιμοποιείται σε συστήματα SoC με επεξεργαστή της εταιρείας ARM για τη διασύνδεση των δομικών μονάδων μεταξύ τους, ώστε να επιτυγχάνεται υψηλή ταχύτητα μετάδοσης δεδομένων. Το πρωτόκολλο αυτό ανήκει, σε ένα ανοιχτό πρότυπο που ονομάζεται Advanced Microcontroller Bus Architecture (AMBA), το οποίο αναπτύχθηκε από την ARM. Στο πρότυπο αυτό ανήκουν επίσης και τα πρωτόκολλα επικοινωνίας Advanced Peripheral Bus (APB) και AMBA High-performance Bus (AHB) που αναπτύχθηκαν σε προηγούμενες εκδόσεις του AMBA (AMBA, AMBA2) αλλά

νεότερες εκδόσεις τους χρησιμοποιούνται μέχρι σήμερα (APB5 και AHB5). Στην Εικόνα 2.1 παρουσιάζεται το σύνολο των πρωτοκόλλων της αρχιτεκτονικής AMBA και οι εκδόσεις αυτών, μέχρι σήμερα.



Εικόνα 2.1: Οι εκδόσεις της αρχιτεκτονικής AMBA και τα αντίστοιχα πρωτόκολλα επικοινωνίας
Πηγή: https://developer.arm.com/

Παρότι η τελευταία έκδοση του πρότυπου AMBA είναι το AMBA5, στο πλαίσιο αυτής της εργασίας χρησιμοποιήθηκε η έκδοση AMBA4, η οποία είναι η μέγιστη δυνατή που υποστηρίζεται από την έκδοση του περιβάλλοντος σχεδίασης υλικού, Vivado 2019.1[3] που χρησιμοποιήθηκε. Στην έκδοση AMBA4 έχουν αναπτυχθεί τα ακόλουθα πρωτόκολλα: το AXI4, το οποίο χρησιμοποιείται όταν υπάρχει ανάγκη για μέγιστη απόδοση, το AXI4-Lite που ενσωματώθηκε στο σύστημα της παρούσας εργασίας και χρησιμοποιείται όταν υπάρχουν χαμηλότερες ανάγκες ως προς την μεταφορά δεδομένων και το AXI4-Stream, που χρησιμοποιείται όταν υπάρχουν ανάγκες υψηλής ταχύτητας και συνεχούς ροής δεδομένων, περιορίζοντας ταυτόχρονα τα σήματα ελέγχου του πρωτοκόλλου επικοινωνίας.



Εικόνα 2.2: Τα σήματα επικοινωνίας μεταξύ συσκευών Master και Slave στο πρότυπο AMBA
Πηγή: [3]

[3] https://www.xilinx.com/products/design-tools/vivado.html

Η απόδοση των παραπάνω εκδοχών του πρωτοκόλλου εξαρτάται από διάφορους παράγοντες όπως το αν το σύστημα θα σχεδιαστεί για συνεχή ροή δεδομένων (burst), αν θα ενσωματωθεί λειτουργία Direct Memory Access (DMA) για την άμεση προσπέλαση της μνήμης του συστήματος από το περιφερειακό κ.α. Στον Πίνακα 1 παρουσιάζονται τα αποτελέσματα απόδοσης των τριών εκδοχών του AXI4 σε μια εφαρμογή με και χωρίς την χρήση των παραπάνω παραμέτρων. Παρατηρείται ότι η εκδοχή AXI FULL για συνεχή ροή δεδομένων είναι αυτή που δίνει την μεγαλύτερη απόδοση, η οποία αυξάνεται ακόμα περισσότερο όταν χρησιμοποιείται η τεχνική DMA. Τα αποτελέσματα από την απόδοση του πρωτοκόλλου AXI4-Lite που χρησιμοποιήθηκε στην παρούσα εργασία παρουσιάζονται στο Κεφ. 4.

Πίνακας 2-1: Πίνακας απόδοσης των διαφόρων εκδοχών του πρωτοκόλλου AXI του προτύπου AMBA
Πηγή: Advanced Workshop on FPGA - based Systems - On - Chip for Scientific Instrumentation and Reconfigurable Computing | smr3249

Interface	Test Case Variant	Burst	Between Data min	typ	max	Per Frame PS (MB/s)	PL (MB/s)	PS/PL
EMIO	GPIO (XGpioPs_Read)	No	20	21	29	96954 (27.46)	22358 (27.48)	4.33
EMIO	GPIO (*Xil_In32*)	No	20	20	31	92502 (28.78)	21330 (28.80)	4.33
M_AXI_GP	AXI Lite (*Xil_In32*)	No	28	28	33	124386 (21.40)	28689 (21.41)	4.33
M_AXI_GP	AXI Full (*Xil_In32*)	No	24	24	26	106588 (24.97)	24581 (24.99)	4.33
M_AXI_GP	AXI Lite (*memcpy*)	No	19	20	31	90973 (29.26)	20974 (29.29)	4.33
M_AXI_GP	AXI Full (*memcpy*)	No	15	16	25	73336 (36.30)	16910 (36.33)	4.33
S_AXI_GP	AXI Lite	No	44	44	45	200229 (13.29)	46075 (13.33)	4.34
S_AXI_HP	AXI Lite	No	36	36	37	160386 (16.59)	36865 (16.66)	4.35
S_AXI_ACP	AXI Lite	No	36	36	36	160389 (16.59)	36864 (16.66)	4.35
S_AXI_GP	AXI Full	Yes	1	4	59	21962 (121.22)	4868 (126.21)	4.51
S_AXI_HP	AXI Full	Yes	1	3	40	16669 (159.72)	3675 (167.18)	4.53
S_AXI_ACP	AXI Full	Yes	1	3	37	15506 (171.70)	3409 (180.22)	4.54
M_AXI_GP	AXI Full with PS DMA	Yes	1	1	4	11425 (233.3)	1213 (506.51)	9.41
S_AXI_GP	AXI Full with AXI DMA	Yes	1	1	571	7245 (367.48)	1654 (371.46)	4.38
S_AXI_HP	AXI Full with AXI DMA	Yes	1	1	381	6048 (440.21)	1397 (439.79)	4.32
S_AXI_ACP	AXI Full with AXI DMA	Yes	1	1	422	6154 (432.62)	1418 (433.28)	4.33

2.2.3 Το λειτουργικό σύστημα Petalinux

Από την πλευρά του λογισμικού, στην παρούσα εργασία, χρησιμοποιήθηκε το λειτουργικό σύστημα Petalinux το οποίο αναπτύχθηκε χρησιμοποιώντας το πακέτο Yocto project. Το Yocto project αποτελεί ένα εργαλείο το οποίο βοηθά τους προγραμματιστές να δημιουργήσουν, σύμφωνα με τις ανάγκες του συστήματος που αναπτύσσουν, μια προσαρμοσμένη έκδοση του λειτουργικού συστήματος Linux για τη φόρτωση αυτού σε ενσωματωμένα συστήματα. Αποτελεί ένα ανοιχτού κώδικα εργαλείο με το οποίο μπορεί να δημιουργήσει κανείς λειτουργικό σύστημα για οποιαδήποτε αρχιτεκτονική υλικού. Επίσης, έχει την δυνατότητα να παραμετροποιείται εύκολα έτσι ώστε να καλύπτει διάφορες ανάγκες των σχεδιαστών συστημάτων, όπως μια τροποποίηση, ώστε μια υπάρχουσα υλοποίηση να μπορεί να εφαρμοστεί σε υλικό διαφορετικής αρχιτεκτονικής. Ένα ακόμη σημαντικό πλεονέκτημα είναι ότι χρησιμοποιείται από ένα μεγάλο σύνολο σημαντικών εταιρειών στο χώρο της τεχνολογίας. Αυτό παρέχει πλεονεκτήματα όπως, η βιωσιμότητα και η εξέλιξη, καθώς θα συνεχίσει να υπάρχει και να υποστηρίζεται από τις εταιρείες από τις οποίες χρησιμοποιείται, αλλά και η παροχή έτοιμων και ελεγμένων λύσεων, όπως βιβλιοθήκες και παλαιότερες υλοποιήσεις, οι οποίες βοηθούν σημαντικά ώστε να αναπτυχθεί μια νέα έκδοση λειτουργικού συστήματος πιο εύκολα και σύντομα.

Στην παρούσα εργασία, χρησιμοποιήθηκε η έκδοση Petalinux του λειτουργικού συστήματος Linux, η οποία αναπτύχθηκε με τη χρήση του εργαλείου Yocto. Πρόκειται για μια έκδοση που αναπτύσσεται από την εταιρεία Xilinx[4] για χρήση στα συστήματα επεξεργαστών Versal, Zynq UltraScale MPSoC, Zynq-7000 SoC και MicroBlaze, που παράγει η εταιρεία. Το Petalinux προσφέρεται ως μια πλήρη βάση, στην οποία υπάρχει ήδη ενσωματωμένο το μέρος που αφορά τη ρύθμιση του υλικού, δίνοντας τη δυνατότητα στους σχεδιαστές να την παραμετροποιήσουν αλλά και να αναπτύξουν πάνω σε αυτή τις δικές τους εφαρμογές. Οι εφαρμογές αυτές μπορεί να αφορούν είτε το επίπεδο υλικού (οδηγοί περιφερειακών) είτε το επίπεδο λειτουργίας του συστήματος. Κάποια από τα εργαλεία που προσφέρονται μέσω της διανομής Petalinux, αφορούν τη βελτιστοποίηση της λειτουργίας του επεξεργαστή, την ανάπτυξη εφαρμογών με χρήση γλωσσών προγραμματισμού όπως C, C++, Python, την αποσφαλμάτωση, τη χρήση νημάτων (threads) και μονάδων κινητής υποδιαστολής, και τη δικτύωση για εύκολη πρόσβαση στο σύστημα και παραμετροποίηση αυτού. Επίσης παρέχονται βιβλιοθήκες και εφαρμογές για περιβάλλον Linux.

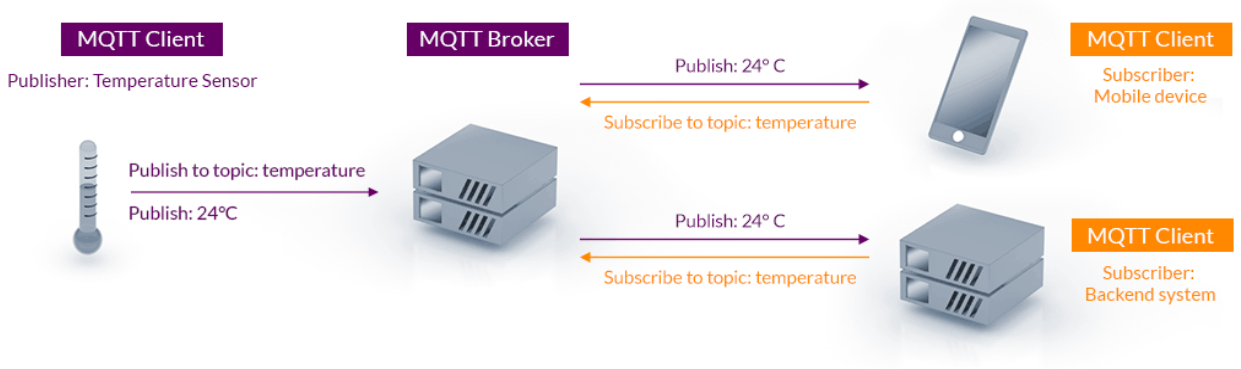
2.2.4 Το πρωτόκολλο επικοινωνίας MQTT

Για την επικοινωνία του συστήματος, μέσω του διαδικτύου, χρησιμοποιήθηκε το πρωτόκολλο MQTT. Η πρώτη υλοποίηση του συγκεκριμένου πρωτοκόλλου έγινε από τους μηχανικούς Andy Stanford-Clark (IBM) και Arlen Nipper (Eurotech, Inc.) [4,5] το 1999. Οι συγκεκριμένοι ανέπτυξαν το πρωτόκολλο ώστε να το ενσωματώσουν στο βιομηχανικό σύστημα ελέγχου "SCADA" για την επιτήρηση αγωγών πετρελαίου. Ο στόχος τους ήταν να αναπτύξουν ένα πρωτόκολλο το οποίο να είναι πολύ αποδοτικό ως προς την χρήση του εύρους ζώνης (bandwidth), εύκολο ως προς την παραμετροποίηση, «ελαφρύ» ως προς την ενσωμάτωση του στα διάφορα συστήματα και μη ενεργοβόρο καθώς τα συστήματα στα οποία θα χρησιμοποιούταν θα ήταν τροφοδοτούμενα από μπαταρία. Ο λόγος που τέθηκαν οι παραπάνω στόχοι είναι ότι η επικοινωνία θα γινόταν μέσω δορυφόρου, πράγμα το οποίο ήταν πολύ κοστοβόρο, επομένως θα έπρεπε να περιορίσουν την μεταφορά δεδομένων. Το 2010 το πρωτόκολλο υιοθετείται από τον οργανισμό OASIS[5]. Πρόκειται για έναν διεθνή οργανισμό, στον οποίο συμπράττουν δημόσιοι και ιδιωτικοί φορείς για να ορίσουν ανοιχτά πρωτόκολλα και πρότυπα που αφορούν την τεχνολογία. Σήμερα το MQTT συνεχίζει να αποτελεί ένα ανοιχτό πρωτόκολλο, το οποίο εκτελείται πάνω σε ένα πρωτόκολλο μεταφοράς που παρέχει αμφίδρομη και αξιόπιστη επικοινωνία (συνήθως τύπου TCP/IP) και χρησιμοποιείται κυρίως σε συστήματα με περιορισμένους υπολογιστικούς και τηλεπικοινωνιακούς πόρους. Στο MQTT μπορούν να οριστούν και να αλληλεπιδράσουν 2 τύποι συσκευών, η συσκευή τύπου «πελάτης» (client) και η συσκευή τύπου «δρομολογητής» (broker). Συνήθως, ένα δίκτυο IoT αποτελείται από πολλές συσκευές τύπου client, ο αριθμός των οποίων μπορεί να είναι πολύ μεγάλος, της τάξης των χιλιάδων και από μια συσκευή τύπου broker. Οι συσκευές τύπου client, ως επί το πλείστον, αποτελούν τους κόμβους παρυφής (edge) του δικτύου IoT, συνήθως είναι χαμηλής επεξεργαστικής ισχύος και ενεργειακής κατανάλωσης και στέλνουν δεδομένα σε μορφή μηνύματος προς το υπόλοιπο δίκτυο (publish). Το κάθε μήνυμα ανήκει σε μια συγκεκριμένη ομάδα μηνυμάτων. Η ομάδα αυτή περιγράφεται στο πρωτόκολλο ως «θέμα» (topic). Η κάθε συσκευή τύπου client εγγράφεται (subscribe) σε ένα θέμα, στέλνοντας ένα συγκεκριμένο μήνυμα προς την συσκευή

[4] https://www.xilinx.com

[5] https://www.oasis-open.org/

broker και έπειτα μπορεί να δημοσιεύει (publish) δηλαδή να στέλνει μηνύματα στο συγκεκριμένο θέμα. Κατά τη δημοσίευση ενός μηνύματος σε ένα θέμα, η συσκευή τύπου client στέλνει το μήνυμα προς τη συσκευή broker. Στη συνέχεια η συσκευή broker αναλαμβάνει να δρομολογήσει το μήνυμα που έλαβε από τη συσκευή client προς τις υπόλοιπες client συσκευές, οι οποίες έχουν εγγραφεί (subscribers) στο συγκεκριμένο θέμα (Εικόνα 2.3).



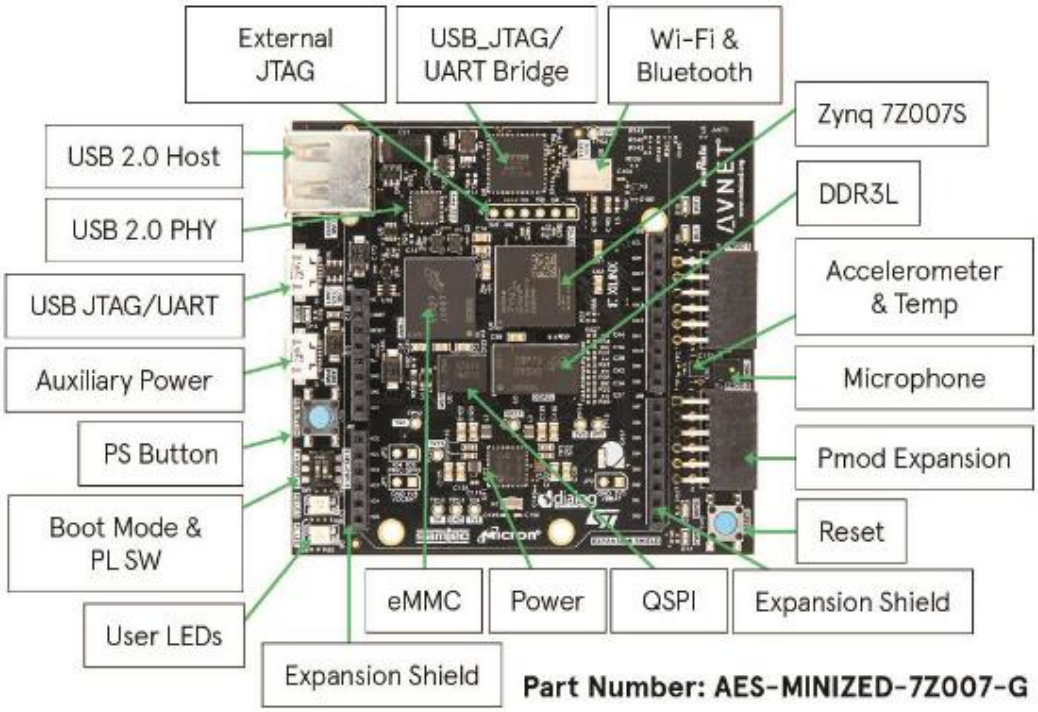
Εικόνα 2.3:Διασύνδεση συσκευών μέσω του πρωτοκόλλου επικοινωνίας MQTT
Πηγή: mqtt.org

Κάποιες επιπλέον λειτουργίες της συσκευής τύπου broker είναι η δυνατότητα αποθήκευσης ενός μηνύματος, τελευταίου ή μη, ώστε οι νέες συσκευές που εγγράφονται σε συγκεκριμένο θέμα να λαμβάνουν αυτόματα αυτό το μήνυμα μέχρι να αποσταλεί κάποιο νεότερο, η δυνατότητα ενημέρωσης των συσκευών client όταν κάποια άλλη συσκευή client αποσυνδέεται από το δίκτυο, αρκεί αυτό να έχει ήδη ρυθμιστεί από την τελευταία. Άλλα χαρακτηριστικά του πρωτοκόλλου είναι ότι τα μηνύματα που μπορούν να μεταφερθούν μπορεί να έχουν μέγεθος από 2 byte έως 256 Mbyte, οι συσκευές client που δημοσιεύουν κάποιο μήνυμα δεν χρειάζεται να γνωρίζουν κάτι για τις συσκευές που έχουν κάνει εγγραφή στο συγκεκριμένο θέμα, όπως και το αντίστροφο, καθώς και ότι το πρωτόκολλο μπορεί να συνεχίσει να είναι αξιόπιστο σε δυσλειτουργίες του δικτύου, όπως για παράδειγμα όταν γίνεται αποσύνδεση κάποιου κόμβου παρυφής. Όπως προαναφέρθηκε, το πρωτόκολλο MQTT εκτελείται πάνω από το πρωτόκολλο μεταφοράς TCP/IP. Άλλη υλοποίηση του πρωτοκόλλου είναι το MQTT-SN, το οποίο αποτελεί μια πιο ελαφριά έκδοση του πρωτοκόλλου MQTT και χρησιμοποιείται κυρίως σε δίκτυα αισθητήρων όπου μεταφέρεται χαμηλός όγκος δεδομένων. Αξίζει να αναφερθεί ότι το ίδιο το πρωτόκολλο δεν περιλαμβάνει κάποια τεχνική που να προσφέρει ασφάλεια κατά τη μετάδοση των δεδομένων, αλλά μπορούν να ενσωματωθούν σε αυτό υπάρχουσες μέθοδοι ώστε να προσφέρουν την απαιτούμενη ασφάλεια, όπως τα Secure Sockets Layer (SSL) ή Transport Layer Security (TLS).

2.3 Η αναπτυξιακή πλακέτα "Minized"

Η αναπτυξιακή πλακέτα Minized[6] της εταιρείας AVNET Εικόνα 2.4 χρησιμοποιήθηκε για τις ανάγκες της παρούσας διπλωματικής εργασίας. Αποτελεί υλικό το οποίο μπορεί να υποστηρίξει την ανάπτυξη ενός κόμβου παρυφής υψηλής επεξεργαστικής ισχύος με χαμηλή κατανάλωση ισχύος.

[6] https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/minized/



Εικόνα 2.4: Η πλακέτα Minized και τα περιφερειακά της
Πηγή: https://www.avnet.com

Οι επεξεργαστικές δυνατότητες του συστήματος οφείλονται στο ολοκληρωμένο κύκλωμα (IC) Zynq XC7Z007S, τύπου SoC της εταιρείας Xilinx. Πρόκειται για ένα chip, το οποίο ενσωματώνει επεξεργαστή μονού πυρήνα τύπου ARM® Cortex™-A9 και επαναδιαμορφούμενη ψηφιακή λογική FPGA τύπου Artix-7. Η πλακέτα Minized, διαθέτει επίσης 512MB μνήμη RAM DDR3, 128Mb μνήμη flash τύπου QSPI και 8GB μνήμη flash τύπου eMMC. Επίσης διαθέτει μια σειρά από περιφερειακά όπως WiFi 802.11b/g/n και Bluetooth 4.1 για επικοινωνία, αισθητήρα θερμοκρασίας, επιταχυνσιόμετρο, μικρόφωνο, 2 Leds, 1 κουμπί πίεσης (push button), JTAG για προγραμματισμό, 1 θύρα USB, 2 θύρες συνδεσιμότητας τύπου Pmod[7] και 1 διεπαφή τύπου Arduino[8].

Από την πλευρά του λογισμικού, ο επεξεργαστής του συστήματος έχει την δυνατότητα να υποστηρίξει «ελαφρές» εκδόσεις του λειτουργικού συστήματος Linux, όπως το Petalinux που αναφέρθηκε προηγουμένως. Στο λειτουργικό σύστημα μπορούν να προστεθούν δυνατότητες όπως η υποστήριξη της γλώσσας Python, καθώς και η μεταγλώττιση προγραμμάτων γραμμένων σε γλώσσα C. Η δημιουργία και η παραμετροποίηση του λειτουργικού συστήματος, γίνεται μέσω του Yocto project.

Για την υλοποίηση του υλικού στην επαναδιαμορφούμενη λογική χρησιμοποιείται το περιβάλλον σχεδίασης Vivado.

[7] https://digilent.com/shop/boards-and-components/system-board-expansion-modules/pmods/

[8] https://docs.arduino.cc/static/0ce63b05bd4614f09350f1dd947a0a20/A000066-full-pinout.pdf

Η κατασκευάστρια εταιρεία της πλακέτας Minized παρέχει μια σειρά από εργαλεία[9], τα οποία μπορούν να χρησιμοποιηθούν από τους σχεδιαστές κατά την υλοποίηση των δικών τους συστημάτων. Ως τέτοια μπορούμε να αναφέρουμε τα αρχεία τύπου bsp, τα οποία χρησιμοποιούνται στο Yocto project για την παραμετροποίηση και την υλοποίηση του προσαρμοσμένου λειτουργικού συστήματος Petalinux, το έτοιμο περιβάλλον ανάπτυξης τύπου Virtual Machine το οποίο περιέχει όλα τα πακέτα προγραμματισμού που χρειάζονται για να υλοποιηθεί ένα σύστημα στην πλακέτα Minized συμπεριλαμβανομένου και του πακέτου Yocto, το πρόγραμμα Vivado το οποίο χρησιμοποιείται για το σχεδιασμό υλικού στο FPGA, μαζί με τις βιβλιοθήκες των αντίστοιχων μονάδων υλικού (hardware components), τα οποία αφορούν τη συγκεκριμένη πλακέτα καθώς και τα περιφερειακά αυτής. Τέλος παρέχεται μια σειρά από οδηγούς εκμάθησης, σε μορφή pdf, και βίντεο, τα οποία είναι πολύ αναλυτικά και βοηθητικά κατά την υλοποίηση ενός συστήματος στο Minized.

[9] https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/minized/

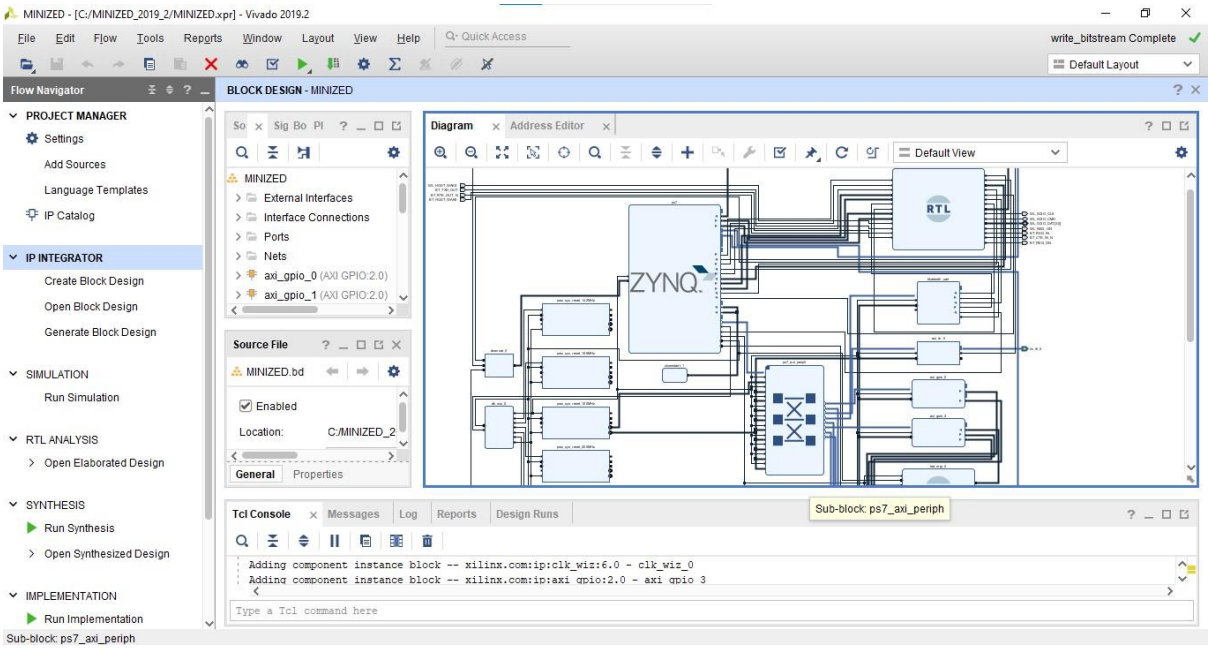
3

Η διαδικασία ανάπτυξης του συστήματος

Στο τρέχον κεφάλαιο περιγράφεται αναλυτικά η διαδικασία με την οποία αναπτύχθηκε το σύστημα της παρούσας διπλωματικής εργασίας. Η διαδικασία ξεκινάει από τη σχεδίαση του υλικού στο FPGA ώστε να εξαχθεί το αρχείο περιγραφής υλικού "xsa". Χρησιμοποιώντας το περιβάλλον Vivado σχεδιάστηκε μία μονάδα υλικού – περιφερειακό που αναλαμβάνει τον υπολογισμό μέσου όρου. Στην συνέχεια, περιγράφεται ο τρόπος με τον οποίο έγινε η δημιουργία του λειτουργικού συστήματος, ο τρόπος που το λειτουργικό σύστημα φορτώθηκε στην αναπτυξιακή πλακέτα Minized και η διαδικασία με την οποία η αναπτυξιακή πλακέτα τέθηκε σε λειτουργία. Τέλος, περιγράφεται η εφαρμογή που αναπτύχθηκε για τον επεξεργαστή του Minized. Η εν λόγω εφαρμογή αποτελείται από 2 μέρη. Το 1ο μέρος υλοποιήθηκε σε γλώσσα C και αφορά έναν οδηγό χαμηλού επιπέδου (low level driver). Ο οδηγός αυτός, αναλαμβάνει την επικοινωνία του επεξεργαστή με το περιφερειακό στο FPGA. Επίσης, μετρήθηκε η ταχύτητα της επικοινωνίας μεταξύ επεξεργαστή και περιφερειακού. Το 2ο μέρος της εφαρμογής υλοποιήθηκε σε γλώσσα Python και αφορά τη λήψη και την αποστολή δεδομένων μέσω του πρωτοκόλλου MQTT καθώς και την επικοινωνία με τον οδηγό χαμηλού επιπέδου για τη μεταφορά των δεδομένων από τον επεξεργαστή προς το FPGA και αντίστροφα.

3.1 Σχεδίαση υλικού

Για τη σχεδίαση του υλικού στην αναδιομορφούμενη λογική του FPGA της αναπτυξιακής πλακέτας, η Xilinx ως κατασκευάστρια εταιρεία του SoC, παρέχει το εργαλείο Vivado για τη διαδικασία αυτή (Εικόνα 3.1).



Εικόνα 3.1: Το περιβάλλον εργασίας του λογισμικού Vivado

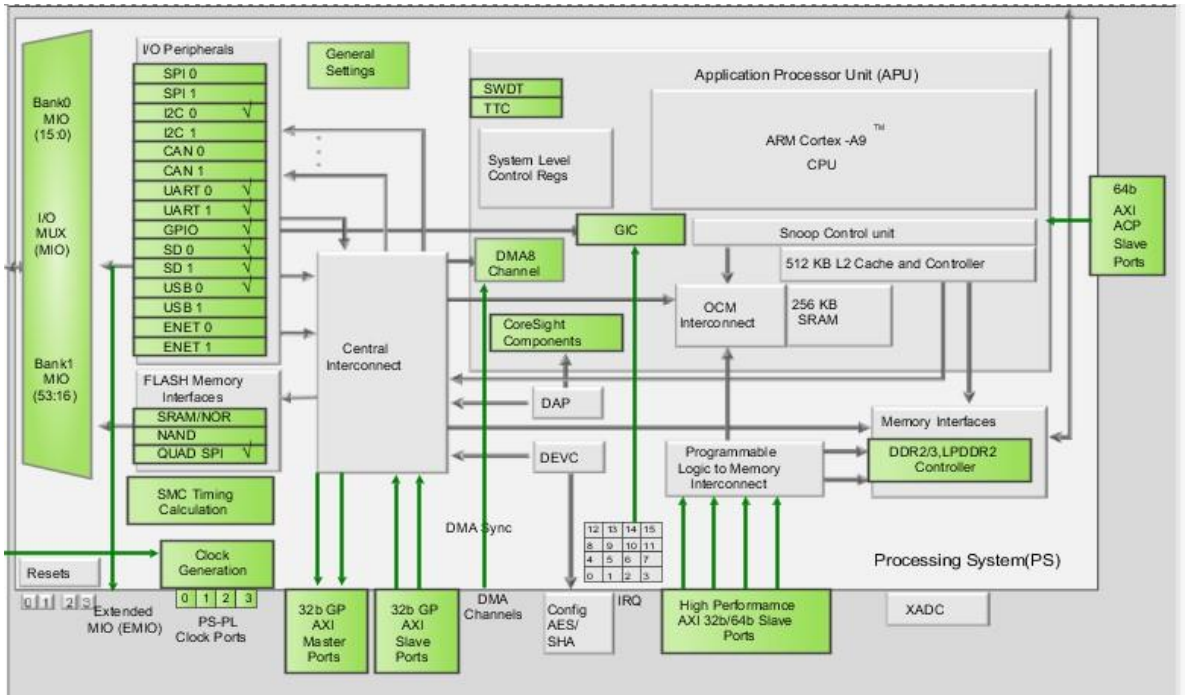
Στο χρήσιμο τεχνικό υλικό που παρέχεται από την AVNET, την κατασκευάστρια εταιρεία της αναπτυξιακής πλακέτας, περιλαμβάνονται οδηγοί χρήσης της πλακέτας, οδηγοί ανάπτυξης διαφόρων εφαρμογών σε αυτή, αρχεία για την εγκατάσταση εικονικού συστήματος (virtual machine) στο οποίο εγκαθίστανται εργαλεία για την υλοποίηση του λειτουργικού συστήματος Petalinux, όπως το Yocto project, και για τον χειρισμό και τον προγραμματισμό της πλακέτας. Επίσης, παρέχονται αρχεία τύπου board support package (bsp) που αφορούν την υποστήριξη του υλικού, στα οποία περιλαμβάνονται αρχεία που αφορούν την ανάπτυξη του λειτουργικού συστήματος καθώς και αρχεία που αφορούν το υλικό που θα σχεδιαστεί στο FPGA.

Για την υλοποίηση της παρούσας εργασίας, χρησιμοποιήθηκε η έκδοση Vivado 2019_2[10], το εικονικό σύστημα virtual machine v2019.2[11] και το αρχείο υποστήριξης υλικού minized_emmc_enhanced_2019_2.bsp[11]. Το πρώτο βήμα για τη δημιουργία του υλικού στο FPGA, ήταν να γίνει χρήση του υπάρχοντος αρχείου περιγραφής υλικού που περιλαμβάνεται στο παραπάνω bsp αρχείο. Στο αρχείο περιγραφής υλικού, υπάρχει μια βασική σχεδίαση, η οποία αποτελείται από τη μονάδα επεξεργασίας (processing system IP) τύπου ARM Cortex-A9 (Εικόνα 3.2) μαζί με τις υπόλοιπες μονάδες που συνδέονται σε αυτή, όπως το βασικό AXI περιφερειακό που αναλαμβάνει την επικοινωνία με όλες τις υπόλοιπες μονάδες που ανταλλάσσουν δεδομένα μέσω διαύλου AXI, τη μονάδα που αναλαμβάνει την διαχείριση του υλικού δικτύωσης WiFi και Bluetooth, και τις μονάδες χρονισμού και επανεκκίνησης. Επίσης περιλαμβάνεται μια σειρά από μονάδες που επικοινωνούν με τη μονάδα επεξεργασίας μέσω του κύριου AXI περιφερειακού, οι οποίες είναι: η μονάδα που αναλαμβάνει την επικοινωνία τύπου I2C, μονάδες εισόδου / εξόδου (General Purpose Input Output – GPIO) που χρησιμοποιούνται για τον έλεγχο ενός LED τριών χρωμάτων (Red Green Blue – RGB), η μονάδα για τη διαχείριση του σήματος του μικροφώνου που

[10]https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html

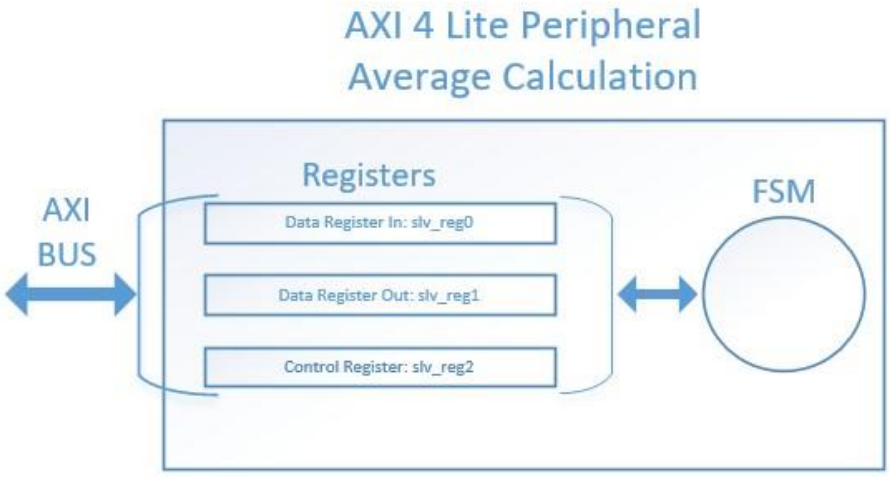
[11] https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/minized/

βρίσκεται ενσωματωμένο στην πλακέτα, η οποία συνδέεται με την μονάδα διαχείρισης του LED και ανάλογα με το σήμα που λαμβάνει το μικρόφωνο αλλάζει αντίστοιχα το χρώμα στο LED, η μονάδα που αναλαμβάνει την διαχείριση ενός διακόπτη τύπου push button που υπάρχει στην πλακέτα και η μονάδα που διαχειρίζεται τα σήματα εισόδου / εξόδου στη θύρα διασύνδεσης τύπου Arduino.



Εικόνα 3.2: Η μονάδα επεξεργασίας τύπου ARM Cortex-A9 όπως εμφανίζεται στο περιβάλλον Vivado
Πηγή: Vivado 2019.2

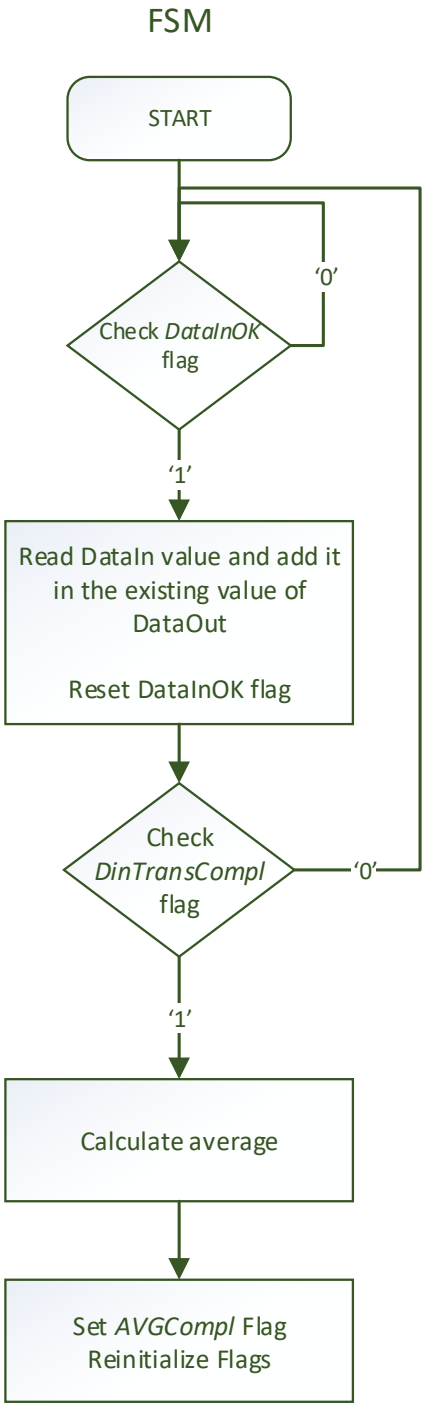
Το επόμενο βήμα στην υπάρχουσα σχεδίαση ήταν να προστεθεί το νέο περιφερειακό που σχεδιάστηκε για τη συγκεκριμένη εργασία. Πρόκειται για ένα περιφερειακό το οποίο αναλαμβάνει τον υπολογισμό μέσου όρου (Μ.Ο.) 16 θετικών ακεραίων αριθμών. Η λειτουργία αυτή επιλέχθηκε επειδή το να σχεδιαστεί κάτι πιο σύνθετο ήταν πέρα από τους σκοπούς της παρούσας εργασίας. Για τη σχεδίαση του περιφερειακού χρησιμοποιήθηκαν τρεις καταχωρητές, τους οποίους το περιβάλλον Vivado αντιστοίχησε σε συγκεκριμένες θέσεις μνήμης. Ο πρώτος χρησιμοποιείται για την καταχώρηση δεδομένων εισόδου (από τη μονάδα επεξεργασίας προς το περιφερειακό), ο δεύτερος για την καταχώρηση δεδομένων εξόδου (από το περιφερειακό προς τη μονάδα επεξεργασίας) και ο τρίτος χρησιμοποιείται ως καταχωρητής ελέγχου (control register). Οι καταχωρητές αυτοί συνδέθηκαν με το δίαυλο επικοινωνίας AXI Lite του περιφερειακού ώστε να υπάρχει η δυνατότητα ανάγνωσης ή εγγραφής αυτών και από τις υπόλοιπες μονάδες του συστήματος. Επίσης, σχεδιάστηκε μια μηχανή πεπερασμένων καταστάσεων (Finite State Machine - FSM) σε γλώσσα VHDL, για τον έλεγχο της λειτουργίας του περιφερειακού (διαχείριση των δεδομένων, υπολογισμός αποτελέσματος). Ένα σχήμα του περιφερειακού σε μορφή block παρουσιάζεται στην Εικόνα 3.3.



Εικόνα 3.3: Το περιφερειακό υπολογισμού μέσου όρου σε μορφή block διαγράμματος

Στη μηχανή πεπερασμένων καταστάσεων αναπτύχθηκε η εξής λειτουργία, όπως παρουσιάζεται και στο διάγραμμα ροής της Εικόνας 3.4. Αρχικά, ελέγχεται το bit(0) (DataInOK flag) του register slv_reg2, το οποίο αφορά την εγγραφή νέων δεδομένων από τον επεξεργαστή. Όταν ο επεξεργαστής γράφει νέα δεδομένα γράφει και την τιμή ‘1’ στο bit(0) του register slv_reg2. Το περιφερειακό ελέγχει την τιμή του συγκεκριμένου bit. Αν είναι ‘0’ σημαίνει ότι δεν έχουν εγγραφεί νέα δεδομένα στον καταχωρητή εισόδου slv_reg0 (DataIn), ενώ αν είναι ‘1’ τότε νέα δεδομένα έχουν εγγραφεί. Στην περίπτωση που υπάρχουν νέα δεδομένα η διαδικασία συνεχίζει, τα δεδομένα διαβάζονται και προστίθενται στην τιμή που υπάρχει στον καταχωρητή slv_reg1 (DataOut). Το αποτέλεσμα εγγράφεται εκ νέου στον καταχωρητή slv_reg1 και τίθεται η τιμή ‘0’ στο bit(0) του slv_reg2, ώστε να μπορεί να γίνει αντιληπτό πότε θα λάβει το περιφερειακό τα επόμενα δεδομένα. Έπειτα ελέγχεται αν έχει ολοκληρωθεί η διαδικασία μεταφοράς όλων των δεδομένων από τον επεξεργαστή προς το περιφερειακό. Ο επεξεργαστής όταν ολοκληρώσει την μεταφορά όλων των δεδομένων προς το περιφερειακό τότε γράφει την τιμή ‘1’ στο bit(1) (DinTransCompl flag) του register slv_reg2. Το περιφερειακό ελέγχει το bit(1) του register slv_reg2 και αν έχει τιμή ‘0’ σημαίνει ότι δεν έχουν μεταφερθεί όλες οι τιμές από τον επεξεργαστή προς το περιφερειακό, οπότε η προαναφερθείσα διαδικασία εκτελείται από την αρχή. Στην περίπτωση που το bit(1) του register slv_reg2 αποκτήσει την τιμή ‘1’, τότε όλες οι τιμές έχουν μεταφερθεί προς το περιφερειακό. Αφού ολοκληρωθεί η διαδικασία μεταφοράς όλων των τιμών, το περιφερειακό υπολογίζει τον Μ.Ο. Για τον υπολογισμό του Μ.Ο. χρησιμοποιείται η μέθοδος των δεξιών ολισθήσεων. Η μέθοδος αυτή εισάγει τους περιορισμούς ότι μπορεί να υπολογίσει τον Μ.Ο. μόνο για θετικούς ακεραίους, αφού οι κενές θέσεις στην αριστερή πλευρά του καταχωρητή γεμίζουν με την τιμή 0 και όχι με το bit προσήμου. Επίσης, το πλήθος των αριθμών για τους οποίους θα υπολογισθεί ο Μ.Ο. πρέπει να ισούται με δύναμη του 2. Για λόγους ευκολίας επιλέχθηκε να γίνουν 4 ολισθήσεις, επομένως το περιφερειακό μπορεί να υπολογίσει τον Μ.Ο. για 16 τιμές δεδομένων. Θα μπορούσε ο τρόπος υπολογισμού του Μ.Ο. να γίνεται για μεγαλύτερο αριθμό δεδομένων, οποίος δεν θα χρειάζεται να είναι ίσος με δύναμη του 2, και χωρίς τον περιορισμό των θετικών ακεραίων αλλά σε αυτή την περίπτωση το περιφερειακό θα γινόταν πιο πολύπλοκο, πράγμα το οποίο δεν μας ενδιέφερε στη συγκεκριμένη εργασία. Έπειτα από τον υπολογισμό του Μ.Ο., το περιφερειακό καταχωρεί το αποτέλεσμα στον καταχωρητή εξόδου slv_reg1 και θέτει το bit(2) (AVGCompl flag) του register slv_reg2 στην τιμή ‘1’ ώστε να

γνωστοποιήσει στον επεξεργαστή ότι η διαδικασία υπολογισμού του Μ.Ο. έχει ολοκληρωθεί. Τέλος αρχικοποιεί εκ νέου στην τιμή '0' το bit(1) του register slv_reg2 ώστε να δοθεί η δυνατότητα να ξεκινήσει η διαδικασία και πάλι από την αρχή. Το μέρος της FSM σε γλώσσα VHDL, παρατίθεται στο Παράρτημα Ι.



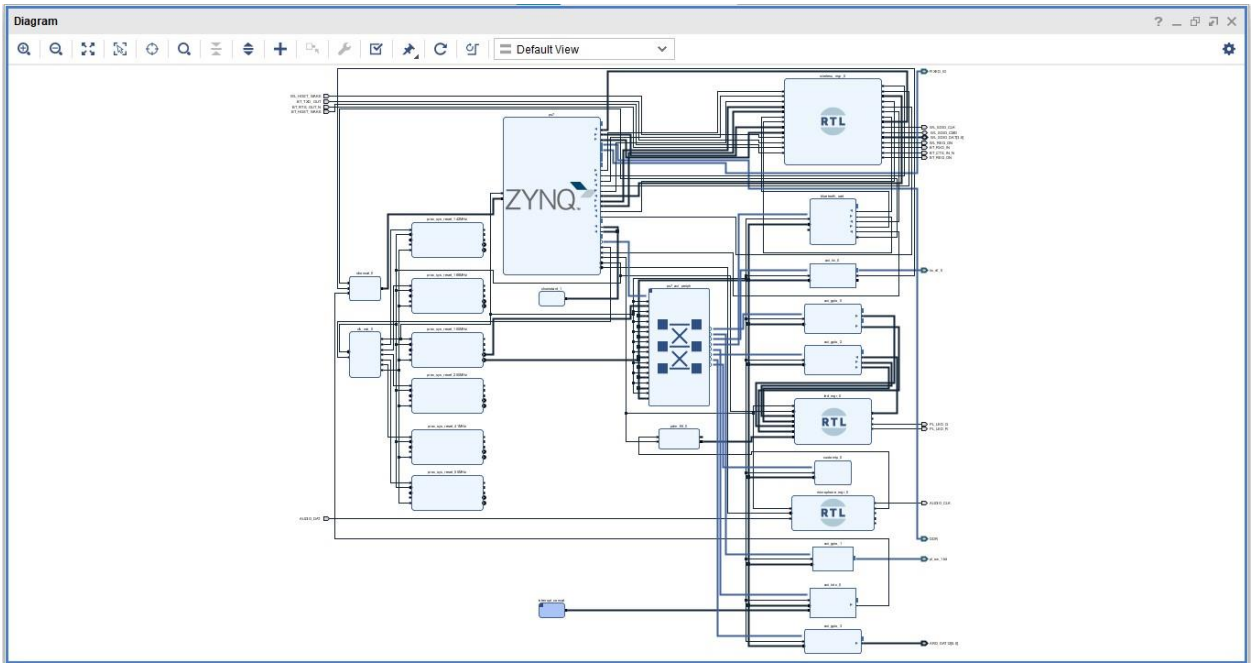
Εικόνα 3.4: Το διάγραμμα ροής της διαδικασίας που εκτελείται στη μηχανή πεπερασμένων καταστάσεων του περιφερειακού που υλοποιήθηκε

Το σημαντικότερο στοιχείο σε αυτό το ειδικό περιφερειακό, που σχεδιάστηκε για τις ανάγκες αυτής της εργασίας, είναι ο τρόπος διασύνδεσής του με την μονάδα επεξεργασίας. Η διασύνδεση αυτή επιτρέπει στο περιφερειακό να επικοινωνεί με τον επεξεργαστή για την ανταλλαγή δεδομένων, μέσω του πρωτοκόλλου AXI4-Lite. Η χρήση του συγκεκριμένου τρόπου διασύνδεσης αποτελεί έναν από τους σημαντικούς παράγοντες για την επίτευξη της υψηλής επεξεργαστικής ισχύος του συστήματος. Ο μέγιστος ρυθμός μεταφοράς δεδομένων (transfer rate) που μπορεί να επιτευχθεί μέσω του πρωτοκόλλου AXI4-Lite, εξαρτάται από τον χρονισμό του συστήματος (ρολόι) και από το μέγεθος του διαύλου επικοινωνίας και υπολογίζεται από τον πολλαπλασιασμό των εν λόγω τιμών (Εξίσωση 1).

$$Transfer\ rate_{max} = System\ Clock\ x\ Bus\ width \qquad (Εξίσωση\ 1)$$

Στη συγκεκριμένη σχεδίαση χρησιμοποιήθηκε ρολόι στα 100MHz και το εύρος του διαύλου ήταν 32bit, επομένως επιτεύχθηκε μέγιστος ρυθμός μεταφοράς δεδομένων ίσος με 3200Mb/sec.

Έπειτα από την ολοκλήρωση του σχεδιασμού του ειδικού περιφερειακού υπολογισμού Μ.Ο., με την βοήθεια του περιβάλλοντος Vivado έγιναν οι κατάλληλες διασυνδέσεις του νέου περιφερειακού με την υπάρχουσα σχεδίαση (Εικόνα 3.5). Στη συνέχεια, δημιουργήθηκε εκ νέου το αρχείο wrapper το οποίο περικλείει όλη τη σχεδίαση ώστε να συμπεριληφθεί και το περιφερειακό αυτό. Το τελικό βήμα που χρειαζόταν για την ολοκλήρωση της σχεδίασης του υλικού στο FPGA ήταν η σύνθεση του υλικού και η εξαγωγή αυτού σε αρχείο τύπου xsa. Το αρχείο αυτό χρησιμοποιείται στη συνέχεια τόσο για τον προγραμματισμό της πλακέτας όσο και για την ανάπτυξη του λειτουργικού της συστήματος.



Εικόνα 3.5: Το τελικό σύστημα σε επίπεδο υλικού

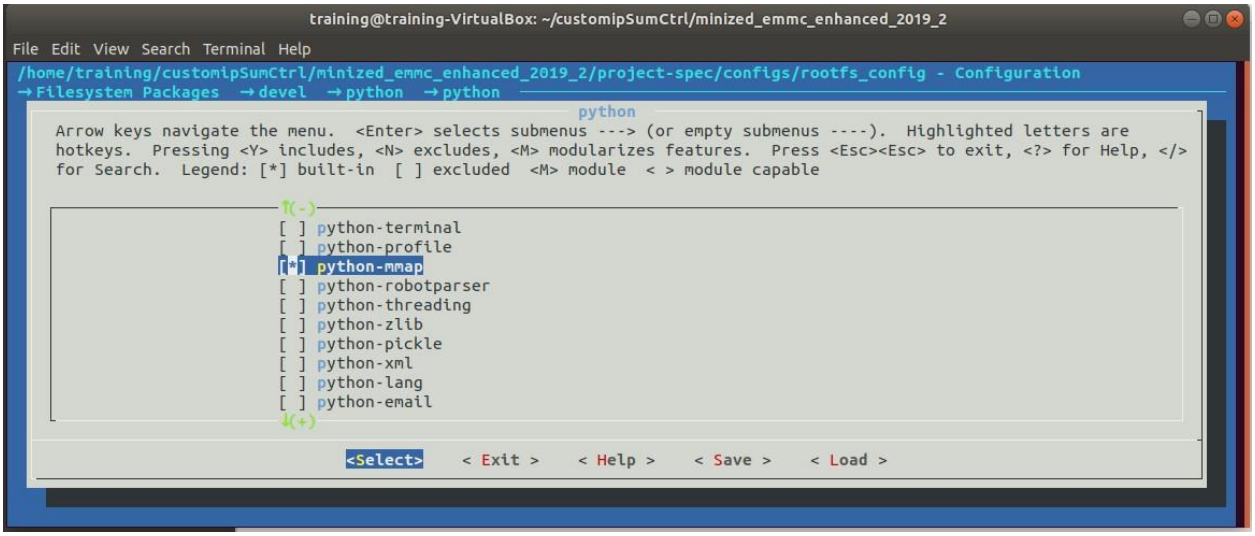
3.2 Δημιουργία λειτουργικού συστήματος και πρώτη λειτουργία της αναπτυξιακής πλακέτας

Το επόμενο βήμα ώστε να τεθεί σε λειτουργία η αναπτυξιακή πλακέτα είναι η δημιουργία μιας νέας έκδοσης του λειτουργικού συστήματος Petalinux, ώστε να καλύπτει τις ανάγκες της παρούσας εργασίας. Συγκεκριμένα η παραμετροποίηση του τρέχοντος λειτουργικού συστήματος χρειάζεται για δύο λόγους. Ο πρώτος είναι για να συμπεριληφθεί το νέο υλικό που περιγράφηκε στην προηγούμενη υποενότητα μαζί με τους drivers διαχείρισης αυτού, στο πυρήνα Linux και ο δεύτερος, για να συμπεριληφθεί σε αυτό η γλώσσα Python. Το λειτουργικό σύστημα Petalinux παρέχεται από την κατασκευάστρια εταιρεία του SoC, Xilinx. Για τη δημιουργία της νέας έκδοσης του Petalinux, χρησιμοποιήθηκε το εργαλείο δημιουργίας λειτουργικού συστήματος Yocto project. Η όλη διαδικασία έγινε στο virtual machine που παρέχει η κατασκευάστρια εταιρεία AVNET. Η εγκατάσταση και η παραμετροποίηση του virtual machine έγινε σύμφωνα με τον οδηγό εγκατάστασης[12] που παρέχεται από την AVNET. Το λειτουργικό σύστημα που χρησιμοποιήθηκε ήταν το Ubuntu, έκδοση 18.04. Επίσης, στο virtual machine εγκαταστάθηκε το εργαλείο Yocto και μια σειρά εργαλείων της Xilinx (software development kit - SDK) για τον προγραμματισμό της αναπτυξιακής πλακέτας.

Το πρώτο βήμα ήταν η δημιουργία ενός νέου project τύπου: λειτουργικό σύστημα, το οποίο βασιζόταν στο προϋπάρχον αρχείο περιγραφής υλικού τύπου bsp. Το αρχείο bsp περιλαμβάνει αρχεία υλικού και βιβλιοθήκες λογισμικού, οι οποίες χρησιμοποιούνται κατά τη δημιουργία του νέου project. Για το βήμα αυτό επιλέχθηκε η έκδοση του bsp αρχείου η οποία ενσωματώνει την χρήση της μη πτητικής μνήμης flash της αναπτυξιακής πλακέτας. Με αυτό τον τρόπο μπορεί το λειτουργικό σύστημα να χρησιμοποιεί τη μνήμη flash ως χώρο αποθήκευσης. Το επόμενο βήμα της διαδικασίας ήταν να συμπεριληφθεί το αρχείο περιγραφής υλικού τύπου xsa, από την προηγούμενη υποενότητα. Στη συνέχεια γίνεται η ρύθμιση του πυρήνα (kernel) του λειτουργικού συστήματος. Κατά το βήμα αυτό δεν χρειάζεται να αλλάξουν οι προκαθορισμένες τιμές που έχουν ορισθεί από το εργαλείο Yocto. Το επόμενο στάδιο της διαδικασίας αφορά την παραμετροποίηση του λειτουργικού συστήματος σε επίπεδο εφαρμογών (Εικόνα 3.6). Το εργαλείο παρέχει μια πληθώρα εφαρμογών οι οποίες μπορούν να ενσωματωθούν στο λειτουργικό σύστημα όπως ο μεταγγλωτιστής gcc, οδηγοί για την υποστήριξη ήχου, υποστήριξη επικοινωνίας τύπου Secure Shell (SSH), υποστήριξη της γλώσσας Python, υποστήριξη πακέτων του περιβάλλοντος προγραμματισμού qt και άλλα. Για τις ανάγκες αυτής της εργασίας προστέθηκε το πακέτο της γλώσσας Python, η οποία χρησιμοποιήθηκε στην συνέχεια για την ανάπτυξη της εφαρμογής υπολογισμού Μ.Ο. Επίσης προστέθηκε και το εργαλείο Python-setuptools που είναι απαραίτητο για την εγκατάσταση βιβλιοθηκών που χρησιμοποιούνται σε ένα πρόγραμμα Python. Αξίζει να αναφερθεί ότι η προσθήκη εφαρμογών κατά το βήμα αυτό χρειάζεται να γίνει με πολύ προσοχή καθώς η επιλογή προς εγκατάσταση πολλών εφαρμογών μπορεί να κάνει την εικόνα (image) του λειτουργικού συστήματος αρκετά μεγάλη με αποτέλεσμα τελικά να μην μπορεί να φορτωθεί στη μνήμη το λειτουργικό σύστημα κατά την εκκίνηση.

[12] Virtual machine installation guide

Τελευταίο βήμα της διαδικασίας δημιουργίας του λειτουργικού συστήματος είναι να γίνει η παραγωγή του (build).



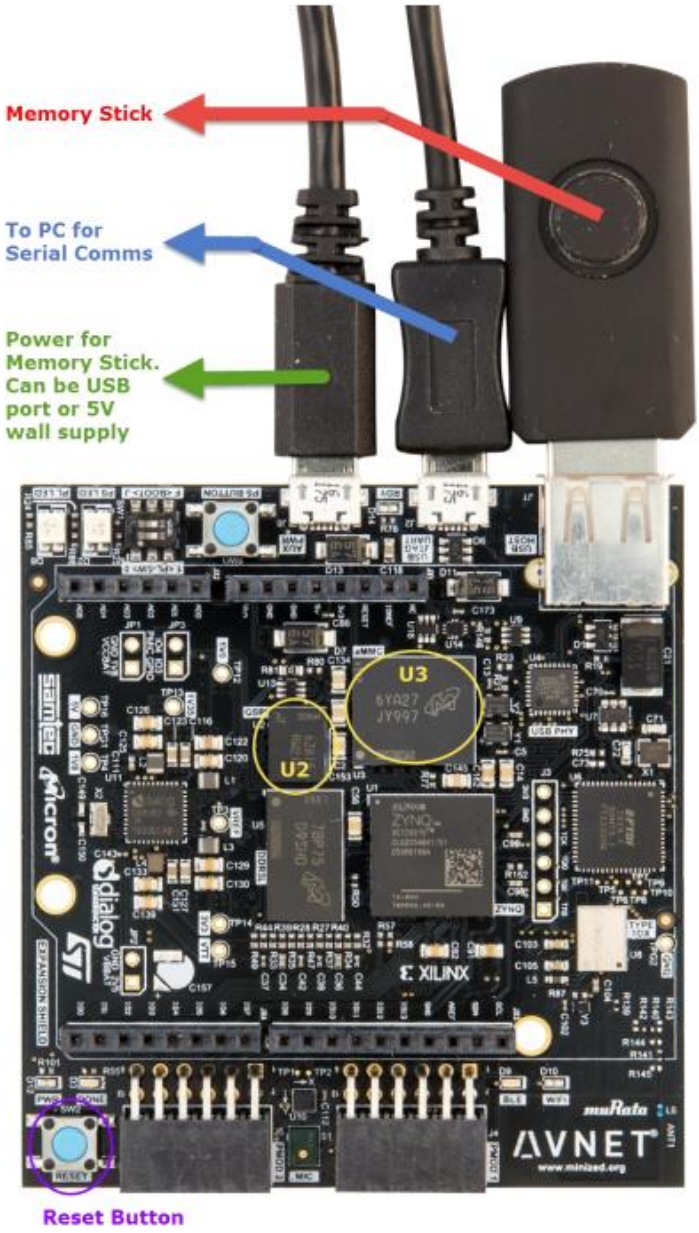
Εικόνα 3.6: Προσθήκη εφαρμογών στο λειτουργικό σύστημα της πλακέτας

Αφού ολοκληρώθηκε η παραγωγή του προσαρμοσμένου στις απαιτήσεις της εργασίας λειτουργικού συστήματος, έπρεπε να δημιουργηθεί το αρχείο εκκίνησης boot.bin με την βοήθεια των εργαλείων της Xilinx. Έπειτα, το αρχείο αυτό μαζί με την εικόνα του λειτουργικού συστήματος image.ub αποθηκεύτηκαν στην μνήμη flash της αναπτυξιακής πλακέτας.

Για την αποθήκευση των αρχείων στην αναπτυξιακή πλακέτα, είναι απαραίτητο η τελευταία να βρίσκεται σε λειτουργία. Χρησιμοποιώντας τον οδηγό χρήσης[13] πρώτης λειτουργίας της πλακέτας που παρέχεται από την AVNET, μπορεί η πλακέτα να τεθεί σε λειτουργία χρησιμοποιώντας τη βασική έκδοση του λειτουργικού συστήματος που δίνεται για αυτήν. Συνοπτικά η διαδικασία έχει ως εξής: αρχικά, χωρίς να βρίσκεται σε λειτουργία, ρυθμίζεται ο κατάλληλος διακόπτης στην πλακέτα, ώστε αυτή να μπει σε διαδικασία “Flash” κατά την εκκίνηση. Κατά τη διαδικασία αυτή, η αναπτυξιακή πλακέτα χρησιμοποιεί ένα προεγκατεστημένο, πολύ ελαφρύ λειτουργικό σύστημα. Το λειτουργικό σύστημα αυτό βρίσκεται στη μνήμη τύπου QSPI της αναπτυξιακής πλακέτας. Έπειτα η πλακέτα συνδέεται με τον υπολογιστή μέσω καλωδίου USB και εκκινεί χρησιμοποιώντας την ελαφριά έκδοση του λειτουργικού συστήματος. Η επικοινωνία του χρήστη με την αναπτυξιακή πλακέτα μπορεί να γίνει με οποιοδήποτε πρόγραμμα τύπου τερματικό (terminal) χρησιμοποιώντας τη σειριακή θύρα που δημιουργήθηκε στον προσωπικό υπολογιστή συνδέοντας το καλώδιο USB. Για να μεταφερθούν τα αρχεία που παράχθηκαν κατά την προηγούμενη διαδικασία στην αναπτυξιακή πλακέτα, χρησιμοποιείται μια συσκευή αποθήκευσης τύπου USB (USB stick). Τα αρχεία φορτώνονται από τον υπολογιστή στο USB stick, έπειτα το USB stick συνδέεται στο Minized (Εικόνα 3.7), αναγνωρίζεται από αυτό και μπορεί να γίνει η αντιγραφή των αρχείων στην εσωτερική μνήμη. Επίσης, στην μονάδα αποθήκευσης USB προστέθηκε και το αρχείο “wpa_supplicant.conf”. Το συγκεκριμένο αρχείο είναι απαραίτητο ώστε

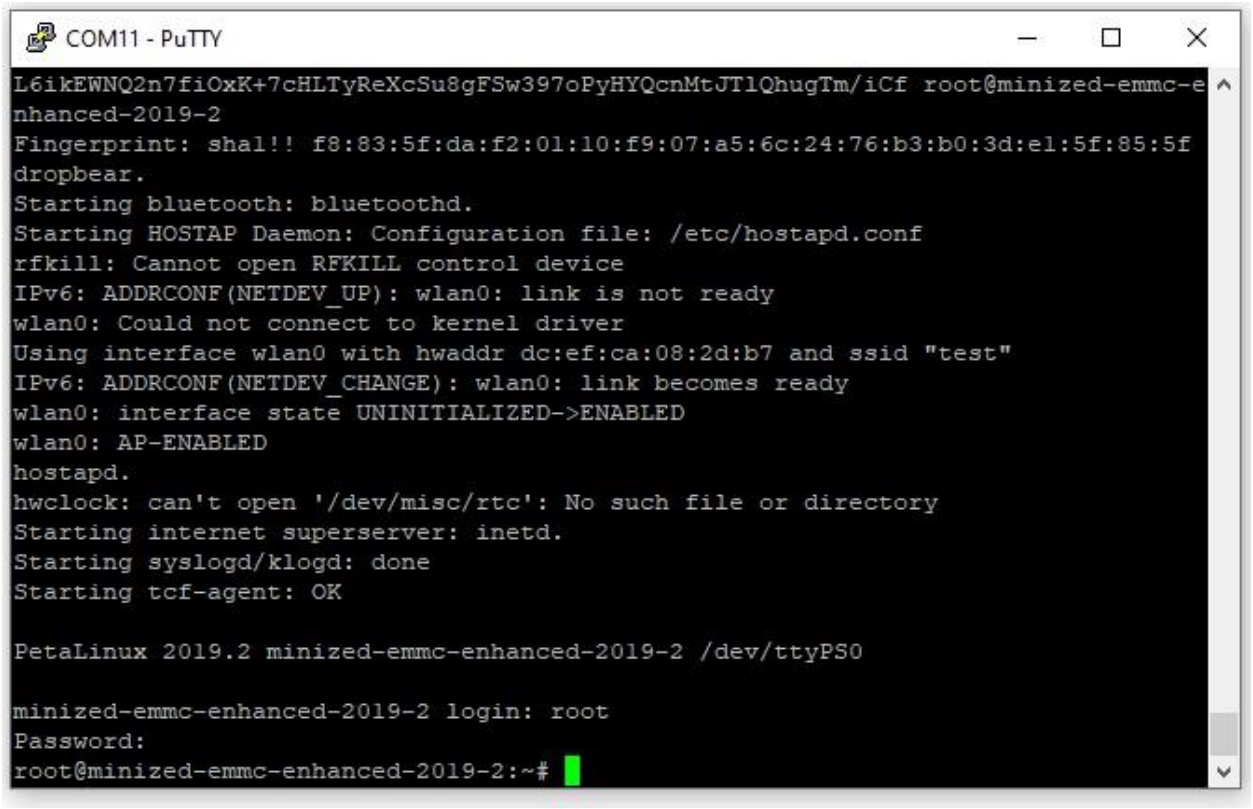
[13] MiniZed-GSG-v1-2 - Getting Started Guide - Link

να μπορεί να συνδεθεί η αναπτυξιακή πλακέτα σε κάποιο δίκτυο WiFi και, συγκεκριμένα, περιέχει τα στοιχεία του δικτύου στο οποίο θα συνδεθεί.



Εικόνα 3.7: Συνδέσεις της πλακέτας για την μεταφορά σε αυτή νέου λειτουργικού συστήματος
Πηγή: avnet.com

Τελευταίο βήμα είναι να γραφεί το boot.bin αρχείο που παράχθηκε κατά την προηγούμενη διαδικασία δημιουργίας του λειτουργικού συστήματος και είναι υπεύθυνο για την λειτουργία της FPGA. Το αρχείο αυτό γράφεται στη μνήμη QSPI, όπως περιγράφεται από τον οδηγό πρώτης λειτουργίας της πλακέτας, και είναι απαραίτητο ώστε κατά την εκκίνησή της, η πλακέτα να λειτουργήσει με το νέο λειτουργικό σύστημα. Αφού γραφεί το αρχείο, χρειάζεται να γίνει επανεκκίνηση ώστε το Minized να ξεκινήσει χρησιμοποιώντας το νέο λειτουργικό σύστημα (Εικόνα 3.8).



Εικόνα 3.8: Το περιβάλλον εργασίας του λειτουργικού συστήματος Petalinux 2019.2

Αφού ολοκληρωθεί η διαδικασία της εκκίνησης μπορεί να χρησιμοποιηθεί το παρεχόμενο από την AVNET, αρχείο τύπου script, με το όνομα wifi.sh, για να συνδεθεί η αναπτυξιακή πλακέτα στο δίκτυο WiFi. Κατά τη διαδικασία σύνδεσης λαμβάνονται υπόψη τα στοιχεία από το αρχείο wpa_supplicant.conf που αντιγράφηκε στην εσωτερική μνήμη flash σε προηγούμενο βήμα. Συνδέοντας το Minized στο ασύρματο δίκτυο, ο χρήστης έχει εύκολα πρόσβαση στο σύστημα αρχείων (file system) της πλακέτας, μέσω οποιασδήποτε εφαρμογής τύπου Secure File Transfer Service (SFTP). Με αυτόν τον τρόπο μπορεί να διαχειριστεί εύκολα οποιοδήποτε αρχείο αλλά και να προσθέσει νέα χωρίς να χρειάζεται να χρησιμοποιήσει USB stick. Επίσης, μέσω της σύνδεσης Wi-Fi, ο χρήστης μπορεί να συνδεθεί και να διαχειριστεί ασύρματα την αναπτυξιακή πλακέτα μέσω τερματικού συνδεδεμένο με SSH. Αξίζει να σημειωθεί ότι και οι δύο τρόποι επικοινωνίας υπάρχουν ήδη ενεργοποιημένοι στις ρυθμίσεις του λειτουργικού συστήματος κατά τη δημιουργία του.

3.3 Η εφαρμογή για τον υπολογισμό του μέσου όρου

Το τελικό στάδιο της εργασίας, μετά από την διαδικασία της πρώτης λειτουργίας και την ρύθμιση της αναπτυξιακής πλακέτας για χρήση του παραμετροποιημένου λειτουργικού συστήματος, ήταν η ανάπτυξη της εφαρμογής για τον υπολογισμό του Μ.Ο. μέσω του σχετικού περιφερειακού που υλοποιήθηκε στο FPGA.

Η εφαρμογή που αναπτύχθηκε αποτελείται από δύο βασικά μέρη. Το πρώτο μέρος αναλαμβάνει την επικοινωνία μεταξύ του επεξεργαστή και του περιφερειακού (low level driver). Το δεύτερο μέρος αναλαμβάνει τη διαχείριση των δεδομένων από και προς τον επεξεργαστή καθώς και τη λήψη και την αποστολή δεδομένων από και προς το διαδίκτυο.

3.3.1 Η ανάπτυξη του οδηγού χαμηλού επιπέδου για την επικοινωνία επεξεργαστή με FPGA

Για την ανάπτυξη του πρώτου μέρους της εφαρμογής, έγινε διεξοδική μελέτη ώστε να βρεθεί ο πιο αποδοτικός και γρήγορος τρόπος επικοινωνίας μεταξύ του περιφερειακού υπολογισμού Μ.Ο. και του επεξεργαστή. Λόγω της διασύνδεσης του περιφερειακού με τον επεξεργαστή σε φυσικό επίπεδο μέσω του AXI διαύλου, η μέγιστη ταχύτητα που μπορεί να επιτευχθεί μεταξύ τους, για μετάδοση δεδομένων, όπως υπολογίστηκε στην ενότητα 3.1 είναι 3200Mb/sec. Αυτό που δημιουργεί περιορισμό στην ταχύτητα εγγραφής και ανάγνωσης δεδομένων από τον επεξεργαστή προς το περιφερειακό και αντίστροφα, είναι η προσπέλαση του υλικού μέσω λογισμικού. Όσο ταχύτερη είναι τόσο αυξάνεται και ο ρυθμός μεταφοράς δεδομένων από και προς το περιφερειακό. Συγκεκριμένα μελετήθηκαν τρείς τρόποι για την ανάπτυξη του οδηγού χαμηλού επιπέδου.

Πριν προχωρήσουμε στη περιγραφή τους θα πρέπει να αναφέρουμε ότι κατά την σχεδίαση του περιφερειακού στο περιβάλλον Vivado, οι τρείς καταχωρητές που σχεδιάστηκαν στο περιφερειακό αντιστοιχήθηκαν, από το περιβάλλον σχεδίασης, σε συγκεκριμένες διευθύνσεις για την προσπέλαση τους. Γράφοντας και διαβάζοντας δεδομένα στις συγκεκριμένες θέσεις, στην ουσία γράφονται και διαβάζονται δεδομένα από τους καταχωρητές του περιφερειακού.

Ο πρώτος τρόπος προσπέλασης των θέσεων μνήμης που μελετήθηκε για τις ανάγκες του low level driver, ήταν μέσω της εντολής του linux "*devmem"*. Πρόκειται για ένα εργαλείο το οποίο αντιστοιχεί τις φυσικές διευθύνσεις μνήμης του συστήματος σε εικονικές ώστε να μπορεί να γίνει προσπέλαση των φυσικών διευθύνσεων. Συγκεκριμένα, δημιουργήθηκαν δύο εκτελέσιμα αρχεία τύπου bash script, ένα για εγγραφή και ένα για ανάγνωση. Εκτελέστηκε η αντίστοιχη εντολή εγγραφής ή ανάγνωσης δεδομένων σε συγκεκριμένη διεύθυνση, επαναλαμβανόμενα για 100.000 φορές, ώστε να μετρηθεί η ταχύτητα εγγραφής και ανάγνωσης (throughput). Στην Εικόνα 3.9 παρουσιάζεται ο κώδικας για τη διαδικασία εγγραφής. Στη γραμμή 7 είναι η δομή επανάληψης και στη γραμμή 11 η διαδικασία εγγραφής στη διεύθυνση *0x4123 0000* της τιμής *0x0000 0001* μεγέθους 32bit. Η διεύθυνση *0x4123 0000*, αντιστοιχεί σε καταχωρητή του περιφερειακού στο FPGA που υλοποιήθηκε για εγγραφή και ανάγνωση δεδομένων ώστε να χρησιμοποιηθεί για την διαδικασία των δοκιμών αυτών. Στη γραμμή 5 υπάρχει μια εξωτερική επανάληψη, η οποία τελικά δεν χρησιμοποιήθηκε καθώς η εκτέλεση της δομής επανάληψης της γραμμής 7 ήταν ήδη αρκετά χρονοβόρα. Ο χρόνος εκτέλεσης του προγράμματος μετρήθηκε με την βοήθεια της εντολής time. Για την ανάγνωση δεδομένων υλοποιήθηκε αντίστοιχο script. Τα αποτελέσματα της απόδοσης του τρόπου αυτού δεν ήταν αποδεκτά καθώς τόσο η εγγραφή όσο και η ανάγνωση πραγματοποιούνταν πάρα πολύ αργά. Στο Κεφάλαιο 4 παρουσιάζονται αναλυτικά τα δεδομένα απόδοσης και των τριών διαφορετικών μεθόδων ανάπτυξης του οδηγού χαμηλού επιπέδου.

```
1  #!/bin/bash
2  # Write value to register
3  echo "Bash version ${BASH_VERSION}... Write to register"
4  #k=0
5  # for ((l=0x40000000;l<=0x40000008;l+=4))
6  # do
7      for ((i=0;i<=100000;i+=1))
8      do
9          #Write to address 0x41230000 the 32 bit
10         #value 0x00000001
11         devmem 0x41230000 32 0x00000001
12         #k=$((k+1))
13         #printf "%x\n" $i
14     done
15 # done
16 #echo $k
```

Εικόνα 3.9: Bash script για εγγραφή δεδομένων με χρήση της εντολής devmem

Ο δεύτερος τρόπος που μελετήθηκε για την ανάπτυξη του οδηγού χαμηλού επιπέδου ήταν μέσω της γλώσσας Python, χρησιμοποιώντας την βιβλιοθήκη mmap. Η συγκεκριμένη βιβλιοθήκη αναλαμβάνει την προσπέλαση δεδομένων που βρίσκονται σε γνωστές θέσεις μνήμης και έχει παρόμοια λειτουργία με την εντολή devmem που χρησιμοποιήθηκε κατά την προηγούμενη προσέγγιση. Και σε αυτή την περίπτωση, μελετήθηκε ο ρυθμός μεταφοράς για την εγγραφή και την ανάγνωση δεδομένων ξεχωριστά. Το αρχείο με τον κώδικα Python (Εικόνα 3.10) που υλοποιήθηκε για τη συγκεκριμένη δοκιμή καλεί 100.000 φορές (Εικόνα 3.10, γραμμή 52) την υπορουτίνα εγγραφής δεδομένων (γραμμή 53).

Η υπορουτίνα εγγραφής δεδομένων *write* (γραμμή 28) κάνει εγγραφή δεδομένων στον καταχωρητή του περιφερειακού στο FPGA ακολουθώντας την εξής διαδικασία: αρχικά αποκτά πρόσβαση στην εικονική μνήμη του συστήματος για ανάγνωση και εγγραφή σε αυτή (γραμμή 32). Στη συνέχεια προσθέτει σε αυτή, τη διεύθυνση του καταχωρητή του περιφερειακού (γραμμή 34) και θέτει τη διεύθυνση εγγραφής στη διεύθυνση του καταχωρητή *0x4123 0000* (γραμμή 38). Έπειτα μετατρέπει τα δεδομένα "value" 32bit προς εγγραφή, από μορφή ακέραιου αριθμού σε μορφή χαρακτήρων (γραμμή 41 και 42), ώστε να είναι σε κατάλληλη μορφή για εγγραφή στο καταχωρητή (γραμμή 44). Τέλος ολοκληρώνει την διαδικασία κλείνοντας τα αρχεία προσπέλασης μνήμης (γραμμή 46 και 47). Για να υπολογιστεί η απόδοση του συγκεκριμένου τρόπου εγγραφής δεδομένων, ο κώδικας εκτελέστηκε χρησιμοποιώντας την εντολή time. Αντίστοιχα μελετήθηκε και η ανάγνωση δεδομένων. Η ίδια διαδικασία της ανάγνωσης ολοκληρώθηκε σε μικρότερο χρόνο σε σχέση με αυτόν της εγγραφής, αλλά και πάλι όχι επαρκή. Τα αποτελέσματα ήταν καλύτερα από την προηγούμενη μέθοδο αλλά όχι ικανοποιητικά για την επίτευξη μιας γρήγορης μετάδοσης δεδομένων για το συγκεκριμένο σύστημα. Όπως προαναφέρθηκε τα αποτελέσματα και αυτής της μεθόδου παρουσιάζονται αναλυτικά στο επόμενο κεφάλαιο.

```
27  #Subroutine for writing to FPGA register
28  def write(addr, value, length=4, mask=0xffffffff):
29      global MAP_MASK
30      #Open memory map file
31      #for read and write
32      f = os.open("/dev/mem", os.O_RDWR)
33      #Map register address of FPGA Component into memory map file
34      mem = mmap.mmap(f, mmap.PAGESIZE, mmap.MAP_SHARED,
35                      mmap.PROT_READ | mmap.PROT_WRITE,
36                      offset=addr & ~MAP_MASK)
37      #Go to address (addr) position
38      mem.seek(addr & MAP_MASK)
39      valueString = ""
40      #Prepare value for writing to register
41      for i in range(length):
42          valueString += chr((value >> (i*8)) & 0xff)
43      #Write new value
44      mem.write(valueString)
45      #Close map file
46      mem.close()
47      os.close(f)
48
49  #Test: write 32bit value to register 0x41230000
50  #repeat for  100.000 times
51  def write_test():
52      for x in range(0, 100000):
53          write(0x41230000, x)
54              #time.sleep(1)
55
56  #Main function
57  if __name__ ==  '__main__':
58      sys.exit(write_test())
```

Εικόνα 3.10: Κώδικας Python για εγγραφή δεδομένων με χρήση της βιβλιοθήκης mmap

Ο τρίτος τρόπος που μελετήθηκε για την ανάπτυξη του οδηγού χαμηλού επιπέδου, ήταν μέσω της γλώσσας C, κάνοντας χρήση της βιβλιοθήκης "mman.h". Ο τρόπος λειτουργίας της συγκεκριμένης βιβλιοθήκης είναι αντίστοιχος με αυτών της πρώτης μεθόδου, δηλαδή κάνει προσπέλαση της μνήμης του συστήματος ώστε να μπορεί αυτή να είναι διαχειρίσιμη από το πρόγραμμα C. Για τη μελέτη της απόδοσης της μεθόδου, υλοποιήθηκαν και πάλι ξεχωριστά προγράμματα για εγγραφή και ανάγνωση δεδομένων. Το πρόγραμμα της εγγραφής δεδομένων (Εικόνα 3.11) μέσω της συνάρτησης *map_register* (γραμμή 35) κάνει προσπέλαση της μνήμης (γραμμή 19) και αντιστοίχηση της θέσης μνήμης *reg_addr* στον δείκτη (pointer) *reg_ptr* (γραμμή 25). Έπειτα, γίνεται μια σειρά 100 εκατομμύριων επαναλήψεων (γραμμές 39,40) της εντολής καταχώρησης της τιμής *val* στην θέση μνήμης *reg_ptr* όπου στην ουσία καταχωρείται η τιμή στην θέση μνήμης *0x4123 0000* που ορίστηκε στην αρχή του προγράμματος (γραμμή 13). Ο χρόνος εκτέλεσης του προγράμματος μετρήθηκε και σε αυτή την περίπτωση, με την βοήθεια της εντολής time. Επίσης, όπως και για αυτόν τον τρόπο προσπέλασης, μελετήθηκε ξεχωριστά η διαδικασία ανάγνωσης. Τα αποτελέσματα και αυτής της μεθόδου παρουσιάζονται μαζί με των υπολοίπων στο επόμενο κεφάλαιο. Κρίνοντας από τον χρόνο εκτέλεσης των προγραμμάτων εγγραφής και ανάγνωσης, πετυχαίνοντας ρυθμό ανάγνωσης 16.67 MB/sec και εγγραφής 15.38 MB/sec, η

συγκεκριμένη μέθοδος με χρήση της γλώσσας C είναι πολύ πιο αποδοτική και καλύπτει τις ανάγκες της εργασίας, γι' αυτό και χρησιμοποιήθηκε στην ανάπτυξη του οδηγού χαμηλού επιπέδου.

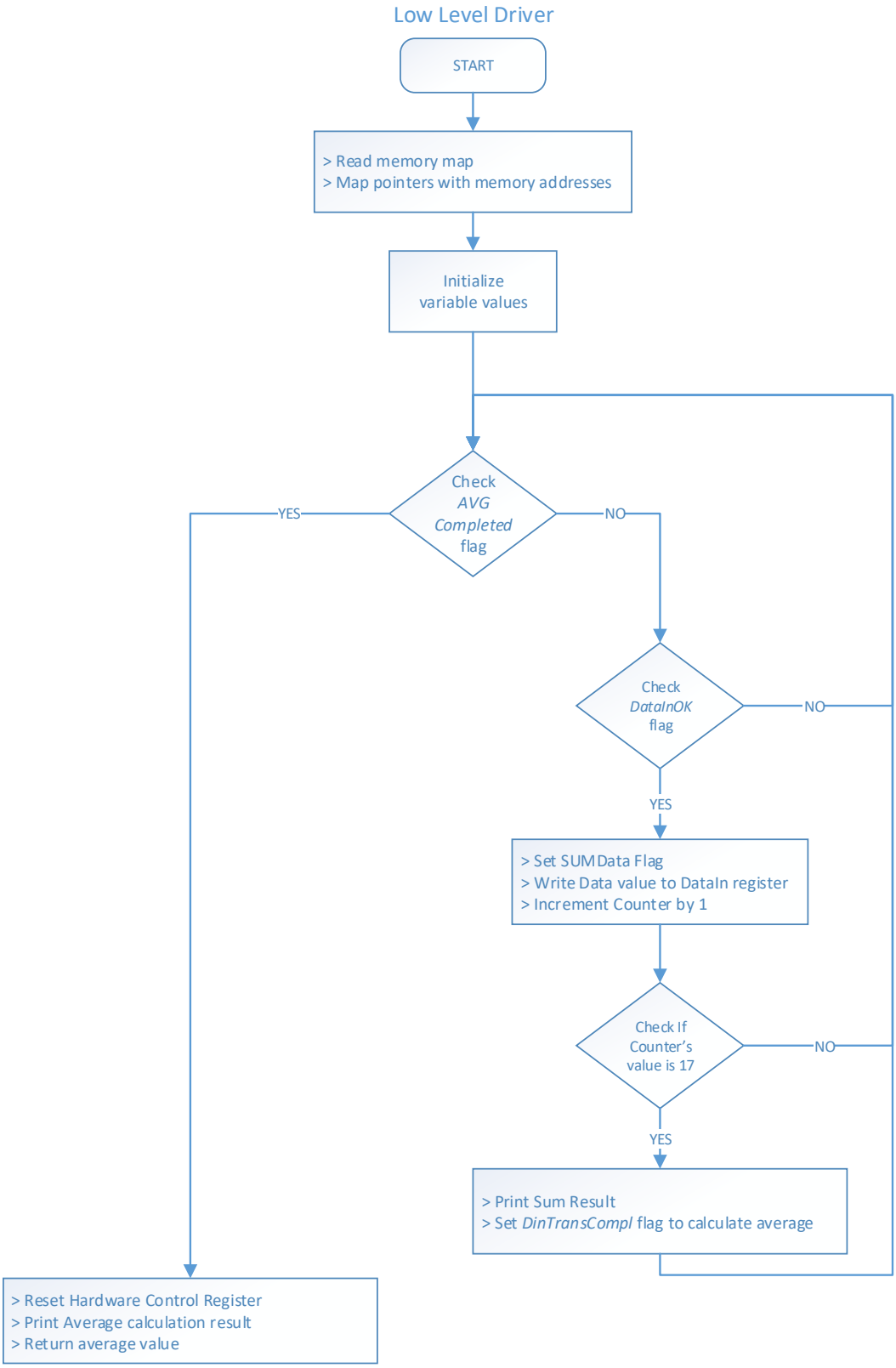
```
12  static volatile uint32_t *reg_ptr;
13  static const uint32_t reg_addr = 0x41230000;
14
15  void map_register(void)
16  {
17    int fd;
18    //Open memory map file for Read and Write
19    if ((fd = open("/dev/mem", O_RDWR|O_SYNC)) < 0) {
20      perror("/dev/mem");
21      exit(-1);
22    }
23    //Map register address of FPGA Component into memory map file
24    //and save the position to reg_ptr
25    if ((reg_ptr = mmap(0, getpagesize(), PROT_READ|PROT_WRITE,
26                        MAP_SHARED, fd, reg_addr)) == MAP_FAILED) {
27      perror("reg_ptr");
28      exit(-1);
29    }
30  }
31
32  /* Main Function */
33  int main()
34  {
35    map_register();  //Map FPGA register to memory
36    int val = 1;     //Value to be written in FPGA register
37    *reg_ptr = val;  //Store value in FPGA register through pointer
38    printf("Starting loop:\r\n");
39    for (int k=0; k<1000000; k++){    //Repeat write process for
40      for (int j=0; j<100; j++){      //100.000.000 times
41        *reg_ptr = val;
42      }
43    }
44    *reg_ptr = 0;
45    printf("Loop finished! \r\n");
46    return 0;
47  }
```

Εικόνα 3.11: Πρόγραμμα C για εγγραφή δεδομένων με χρήση της βιβλιοθήκης mman.h

Ο οδηγός χαμηλού επιπέδου που αναπτύχθηκε για τις ανάγκες της εργασίας περιγράφεται από το διάγραμμα ροής της Εικόνας 3.12. Ο κώδικας του οδηγού παρατίθεται στο Παράρτημα ΙΙ. Κατά την κλήση του οδηγού, η διεύθυνση του πίνακα που περιέχει τις τιμές, για τις οποίες θα υπολογιστεί ο Μ.Ο., δίνεται σαν όρισμα. Η λειτουργία του οδηγού έχει ως εξής: αρχικά διαβάζεται ο χάρτης μνήμης του συστήματος και αντιστοιχίζονται οι δείκτες με τις προκαθορισμένες τιμές διευθύνσεων που έχει το περιφερειακό υπολογισμού Μ.Ο. Στη συνέχεια αρχικοποιούνται οι τιμές των μεταβλητών που θα χρησιμοποιηθούν στο πρόγραμμα. Έπειτα ελέγχεται το Average Completed Flag. Πρόκειται για το bit που ενημερώνει αν έχει ολοκληρωθεί η διαδικασία υπολογισμού του Μ.Ο. Αν δεν έχει ολοκληρωθεί, ελέγχεται το DataInOK flag, που ενημερώνει σχετικά με το αν το περιφερειακό είναι έτοιμο να λάβει νέο δεδομένο. Αν δεν είναι έτοιμο, η

διαδικασία επιστρέφει στην αρχή της επανάληψης. Σε περίπτωση που είναι έτοιμο, τότε ενεργοποιείται το SUMData flag, εγγράφεται η τιμή δεδομένου από τον πίνακα που περάστηκε σαν όρισμα κατά την κλήση του οδηγού χαμηλού επιπέδου και αυξάνεται ο μετρητής τιμών που έχουν μεταδοθεί στο περιφερειακό κατά 1. Το επόμενο βήμα της διαδικασίας είναι να γίνει έλεγχος για το πόσες τιμές έχουν μεταδοθεί στο περιφερειακό διαβάζοντας την τιμή του μετρητή Counter. Αν η τιμή του μετρητή είναι μικρότερη του 17, τότε η διαδικασία επανεκτελείται για το επόμενο στοιχείο του πίνακα. Ο λόγος που η διαδικασία επιστρέφει στην αρχή της, είναι για να συνδυαστούν οι περιπτώσεις αναμονής αποτελέσματος και εισαγωγής νέων δεδομένων. Αν η τιμή του μετρητή είναι 17, τότε τυπώνεται η τιμή του αθροίσματος, την οποία έχει υπολογίσει το περιφερειακό και ορίζεται η τιμή 1 στο DinTransCompl flag. Ορίζοντας την τιμή 1 στο flag, το περιφερειακό υπολογίζει την τιμή του Μ.Ο. Ο λόγος που ο μετρητής ελέγχεται για την τιμή 17, είναι ότι το περιφερειακό έχει προκαθοριστεί να υπολογίζει τον Μ.Ο. για 16 τιμές δεδομένων. Όταν το περιφερειακό υπολογίσει τον Μ.Ο., τότε το AVG Completed flag ορίζεται στην τιμή 1 και η διαδικασία στον οδηγό χαμηλού επιπέδου συνεχίζει, επαναφέροντας το AVG Completed flag στην τιμή 0, τυπώνοντας το αποτέλεσμα του υπολογισμού του Μ.Ο. και τερματίζει επιστρέφοντας την τιμή του αποτελέσματος. Ο οδηγός χαμηλού επιπέδου θα εκτελεστεί εκ νέου για την επόμενη σειρά 16 δεδομένων.

Ο κώδικας του οδηγού χαμηλού επιπέδου μεταγλωττίστηκε με την κατάλληλη εντολή (Παράρτημα IV) ώστε να μπορέσει να εξαχθεί και να χρησιμοποιηθεί ως βιβλιοθήκη (dynamic link library - dll) από την γλώσσα Python.



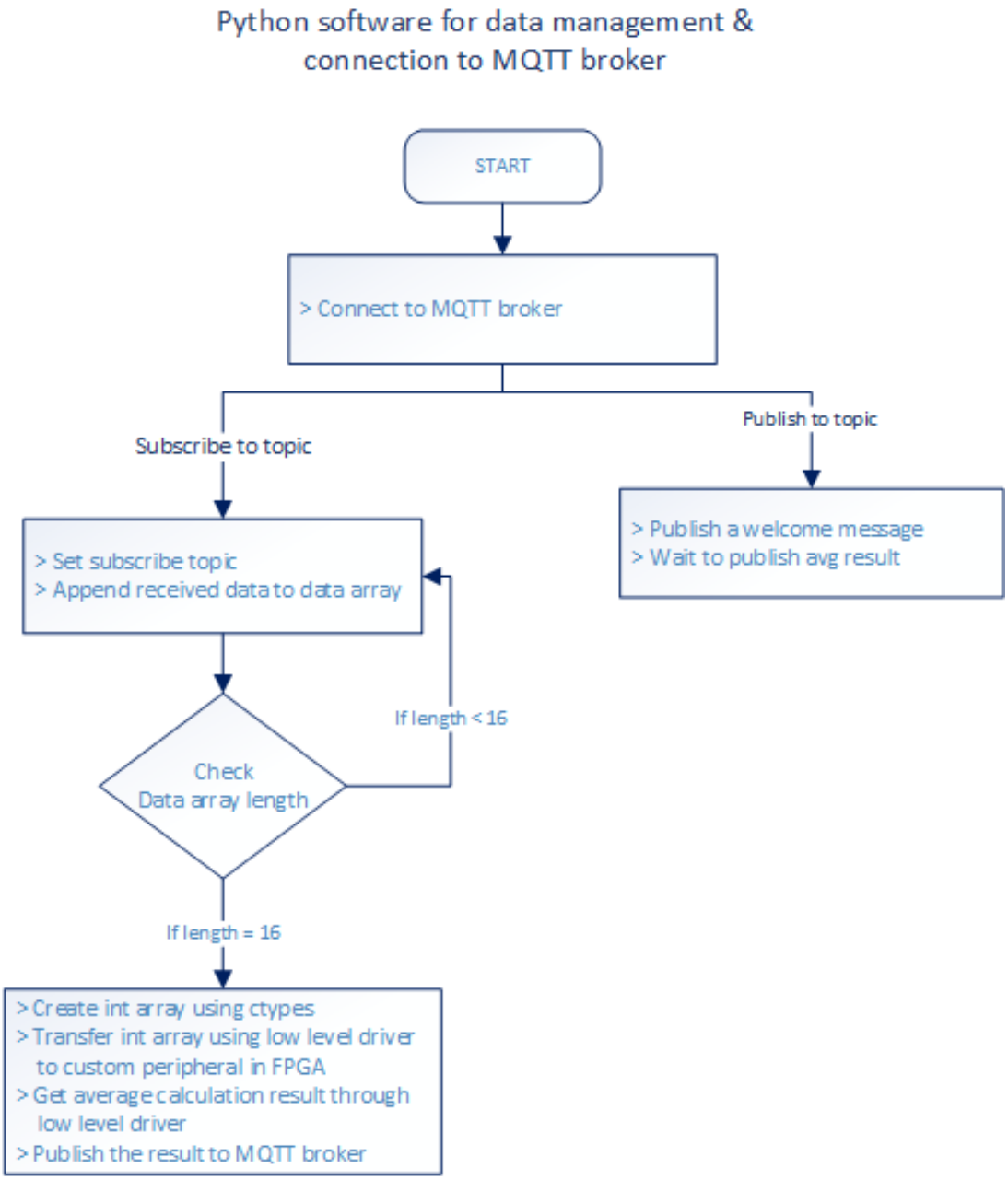
Εικόνα 3.12: Το διάγραμμα ροής του αλγορίθμου του οδηγού χαμηλού επιπέδου

3.3.2 Διαχείριση των δεδομένων και επικοινωνία με τον server

Το δεύτερο μέρος της εφαρμογής που αναπτύχθηκε αναλαμβάνει τη διαχείριση των δεδομένων και την επικοινωνία με τον server (MQTT broker) για την ανταλλαγή δεδομένων. Για τον σκοπό αυτόν

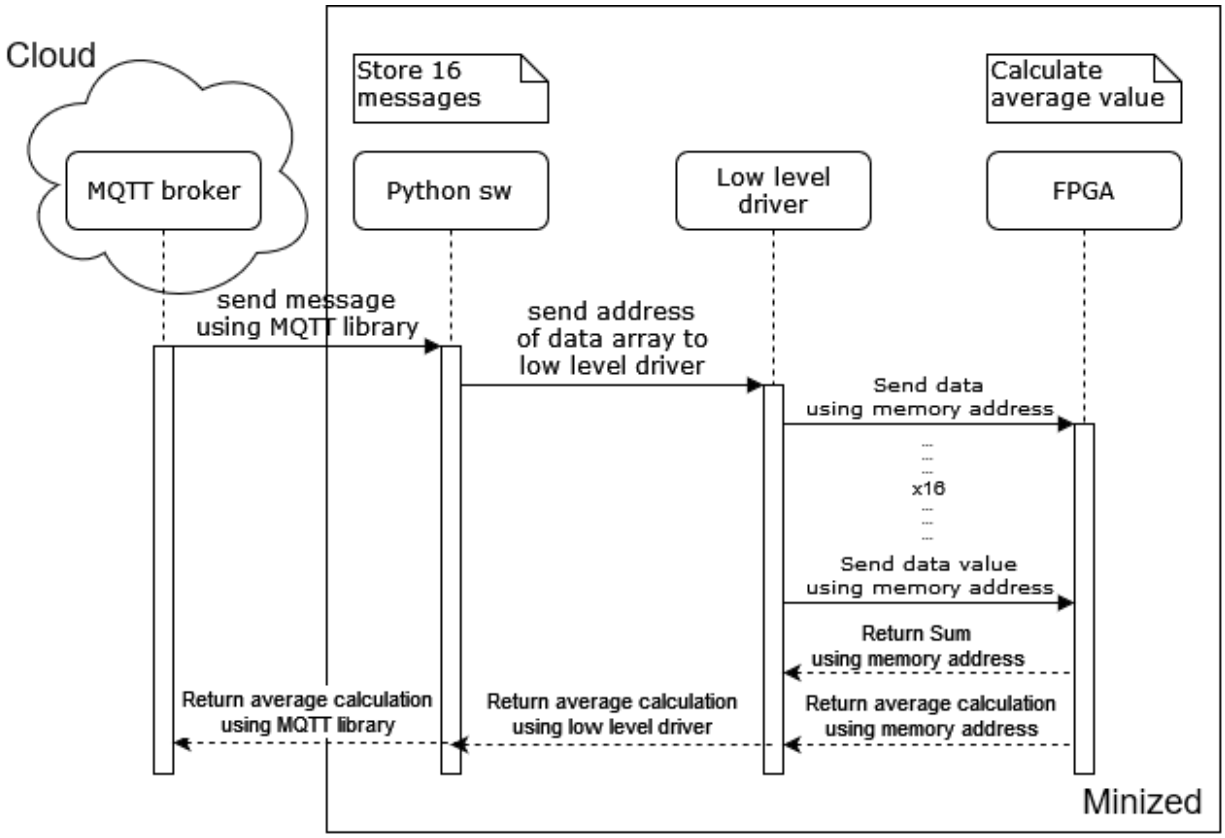
χρησιμοποιήθηκε η γλώσσα Python. Ο λόγος που επιλέχθηκε η συγκεκριμένη γλώσσα προγραμματισμού είναι ότι η υπάρχουσα υλοποίηση του δικτύου IoT έχει γίνει με χρήση της Python. Επίσης σε περίπτωση που κάποιος θα ήθελε να εξελίξει το υπάρχον σύστημα και να το χρησιμοποιήσει σε μια τελική συσκευή μπορεί εύκολα, μέσω βιβλιοθηκών Python, να επεξεργαστεί τα δεδομένα του συστήματος και να εξάγει συμπεράσματα.

Η διαδικασία που υλοποιείται στην εφαρμογή Python περιγράφεται στο διάγραμμα ροής της Εικόνας 3.13 και ο κώδικας της εφαρμογής παρατίθεται στο Παράρτημα ΙΙΙ. Απαραίτητη προϋπόθεση για την επικοινωνία της εφαρμογής μέσω του πρωτοκόλλου MQTT με το διαδίκτυο, είναι η εγκατάσταση των αντίστοιχων βιβλιοθηκών. Η διαδικασία ξεκινάει με την σύνδεση της εφαρμογής στον MQTT broker. Έπειτα καλείται η κλάση loop start η οποία εκτελεί τις συναρτήσεις που αφορούν την αποστολή και τη λήψη μηνυμάτων από τον MQTT broker. Στη συνέχεια καλείται η υπορουτίνα subscribe, μέσω της οποίας ορίζεται σε ποιο θέμα (topic) μηνυμάτων κάνει εγγραφή η εφαρμογή. Επίσης ορίζεται ότι σε κάθε λήψη μηνύματος θα γίνεται η εξής διαδικασία: αρχικά θα ελέγχεται αν ο αριθμός των ληφθέντων δεδομένων που έχουν αποθηκευθεί στον πίνακα εισερχομένων δεδομένων είναι ίσος με 16. Αν όχι, η εφαρμογή θα συνεχίσει να λαμβάνει νέα μηνύματα μέχρι να συμπληρωθούν 16. Όταν συμπληρωθεί το πλήθος των 16 μηνυμάτων, ο πίνακας με τα δεδομένα μετατρέπεται σε πίνακα ακεραίων χρησιμοποιώντας τη μέθοδο ctypes. Η διεύθυνση μνήμης του πίνακα περνιέται ως παράμετρος στον οδηγό χαμηλού επιπέδου, ο οποίος με τη σειρά του μεταφέρει τα δεδομένα του πίνακα στο περιφερειακό υπολογισμού Μ.Ο. Το περιφερειακό ολοκληρώνει την διαδικασία υπολογισμού Μ.Ο. και επιστρέφεται το αποτέλεσμα στην εφαρμογή μέσω του οδηγού χαμηλού επιπέδου. Στη συνέχεια το αποτέλεσμα αποστέλλεται μέσω της υπορουτίνας publish στον MQTT broker ώστε να μεταφερθεί στις υπόλοιπες συσκευές του δικτύου IoT.



Εικόνα 3.13: Το διάγραμμα ροής της εφαρμογής λήψης, διαχείρισης και αποστολής δεδομένων

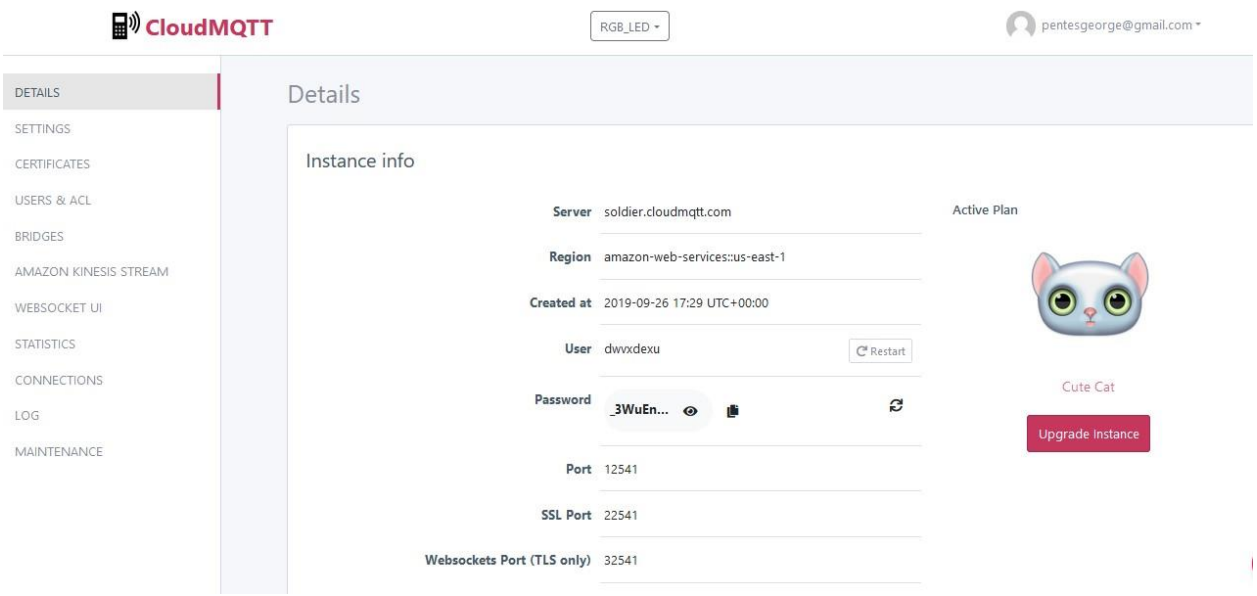
Στην Εικόνα 3.14 παρουσιάζεται ολόκληρη η διαδρομή των δεδομένων. Η διαδικασία ξεκινάει από το cloud όπου αποστέλλονται τα δεδομένα προς το Minized. Η εφαρμογή Python λαμβάνει τα δεδομένα και τα προωθεί στο FPGA μέσω του low level driver χρησιμοποιώντας τον σχετικό πίνακα. Το περιφερειακό υπολογίζει το άθροισμα των δεδομένων του πίνακα και στη συνέχεια υπολογίζει τον Μ.Ο. Το αποτέλεσμα επιστρέφεται μέσω του οδηγού χαμηλού επιπέδου στην εφαρμογή Python και τελικά αποστέλλεται στο cloud μέσω αυτής.



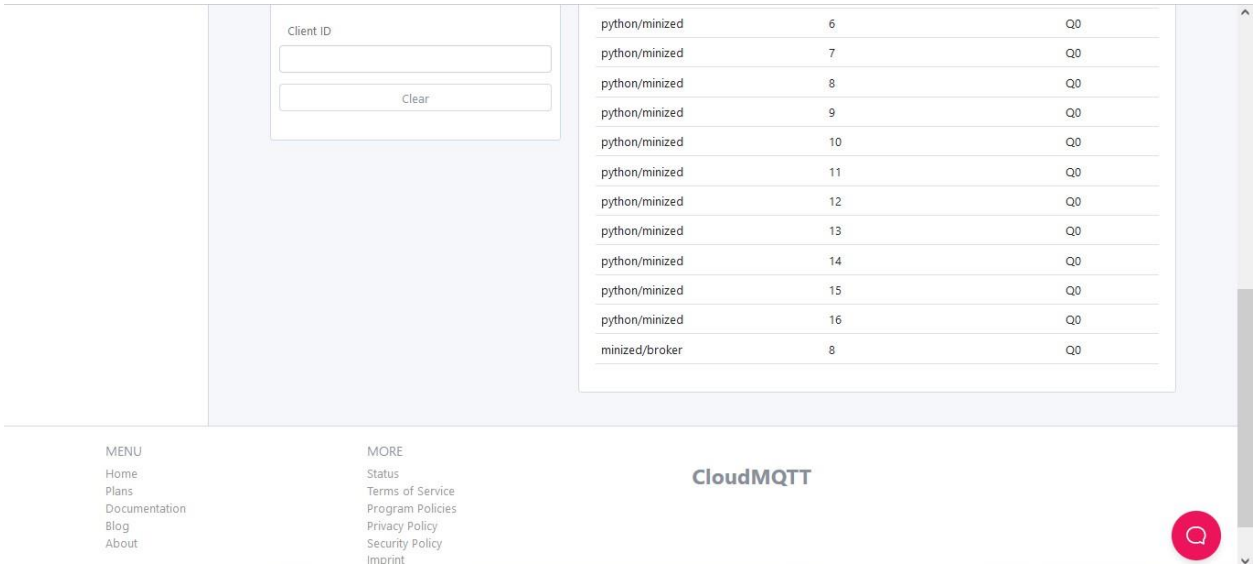
Εικόνα 3.14: Η συνολική διαδρομή των δεδομένων από το Cloud στο FPGA και αντίστροφα

3.3.3 Δοκιμή ολόκληρου του συστήματος

Για τη δοκιμή ολόκληρου του συστήματος χρησιμοποιήθηκε ο broker Cloud MQTT (Εικόνα 3.15). Η εφαρμογή Python ενημερώθηκε με τα στοιχεία επικοινωνίας με τον broker και έγινε η εκτέλεση του κώδικα μέσω τερματικού στο Minized. Από το περιβάλλον διαχείρισης του broker στάλθηκαν 16 μηνύματα (τιμές ακεραίων) στο κατάλληλο θέμα (topic), με το οποίο είχε επίσης ενημερωθεί ο κώδικας της εφαρμογής Python. Μετά το δέκατο έκτο εισερχόμενο μήνυμα, το Minized υπολόγισε και επέστρεψε στο αντίστοιχο θέμα στον broker (Εικόνα 3.16) την τιμή του αποτελέσματος υπολογισμού του Μ.Ο.



Εικόνα 3.15: Το περιβάλλον διαχείρισης του broker Cloud Mqtt



Εικόνα 3.16: Αποστολή δεδομένων προς το Minized και επιστροφή αποτελέσματος στον Broker

4

Αποτελέσματα αξιολόγησης μεθόδων ανάπτυξης του οδηγού χαμηλού επιπέδου

Στο παρόν κεφάλαιο παρουσιάζονται τα αποτελέσματα που ελήφθησαν από τη χρήση των τριών διαφορετικών μεθόδων προσπέλασης του υλικού που υλοποιήθηκαν για την ανάπτυξη του οδηγού χαμηλού επιπέδου. Η κάθε μέθοδος αξιολογήθηκε ως προς τον επειτευχθέντα ρυθμό μεταφοράς δεδομένων (throughput) κατά την διαδικασία ανάγνωσης και εγγραφής δεδομένων σε συγκεκριμένη θέση μνήμης του συστήματος η οποία αντιστοιχεί στον καταχωρητή *0x4123 0000* του περιφερειακού δοκιμών στο FPGA που υλοποιήθηκε για εγγραφή και ανάγνωση δεδομένων. Για να υπολογιστεί ο εκάστοτε ρυθμός μεταφοράς μετρήθηκε ο χρόνος εκτέλεσης του αντίστοιχου script ή προγράμματος μέσω της εντολής time και στη συνέχεια χρησιμοποιήθηκε για τη διαίρεση του πλήθους των δεδομένων που διαβάστηκαν ή γράφτηκαν.

Η πρώτη μέθοδος που υλοποιήθηκε ήταν μέσω της χρήσης αρχείου τύπου bash script και, πιο συγκεκριμένα, μέσω της εντολής devmem. Έγιναν 1.000 αναγνώσεις και 1.000 εγγραφές, του ίδιου δεδομένου μεγέθους 4B. Ο χρόνος εκτέλεσης για τη διαδικασία ανάγνωσης ήταν 5,953 δευτερόλεπτα, ενώ για την διαδικασία εγγραφής 6,303 δευτερόλεπτα.

Στη δεύτερη μέθοδο, έγινε χρήση της βιβλιοθήκης Python mmap. και ακολουθήθηκε η ίδια διαδικασία με την πρώτη μέθοδο. Ο χρόνος που απαιτήθηκε για την ανάγνωση του ίδιου όγκου δεδομένων ήταν 0,142 δευτερόλεπτα και για την εγγραφή 0,239 δευτερόλεπτα.

Τέλος, η τρίτη μέθοδος αναπτύχθηκε με χρήση της γλώσσας C και της βιβλιοθήκης mman.h. Η διαδικασία υπολογισμού της απόδοσης της μεθόδου ήταν όπως και των δύο προηγούμενων, με τη μόνη διαφορά ότι πραγματοποιήθηκαν 100.000.000 εγγραφές και αναγνώσεις ενός δεδομένου μεγέθους 4B. Η αλλαγή αυτή έγινε επειδή ο χρόνος εκτέλεσης και των δύο προγραμμάτων (ανάγνωσης και εγγραφής) για 1.000 επαναλήψεις ήταν πάρα πολύ μικρός. Το αποτέλεσμα από την μέτρηση χρόνου για ανάγνωση και εγγραφή $4*10^8$ B δεδομένων ήταν 24 και 26 δευτερόλεπτα αντίστοιχα.

Στον Πίνακα 1 παρατίθενται οι χρόνοι εκτέλεσης των προγραμμάτων και ο ρυθμός μεταφοράς δεδομένων των τριών μεθόδων, ξεχωριστά για τις διαδικασίες ανάγνωσης και εγγραφής.

Πίνακας 4-1: Ρυθμός μετάδοσης δεδομένων για τις τρείς μεθόδους υλοποίησης του οδηγού χαμηλού επιπέδου που εξετάστηκαν

	Read		**Write**	
Method	**Time**	**Throughput**	**Time**	**Throughput**
Bash script ($4*10^3$ B)	**6,303 sec**	**634,6 B/sec**	**5,953**	**672 B/sec**
Python ($4*10^3$ B)	**0,142 sec**	**28.169 B/sec**	**0,239 sec**	**16.736,4 B/sec**
C ($4*10^8$ B)	**24 sec**	**16,67 MB/sec**	**26 sec**	**15,38 MB/sec**

Συγκρίνοντας και τις τρείς μεθόδους παρατηρούμε ότι οι ρυθμοί μεταφοράς της μεθόδου με χρήση της γλώσσας C, είναι αρκετά μεγάλοι και καλύπτουν τις ανάγκες της παρούσας εργασίας αλλά και αντίστοιχων κόμβων παρυφής δικτύων IoT που ανταλλάσσουν δεδομένα μέσω του πρωτοκόλλου MQTT. Για τον λόγο αυτόν, επιλέχθηκε τελικά η συγκεκριμένη υλοποίηση.

5

Βελτιώσεις και συμπεράσματα

Ολοκληρώνοντας την παρούσα εργασία είναι χρήσιμο να επισημάνουμε μια σειρά βελτιώσεων και επεκτάσεων / τροποποιήσεων του συστήματος που αναπτύχθηκε. Ξεκινώντας από το περιφερειακό υπολογισμού Μ.Ο. που αναπτύχθηκε στο FPGA, αν και η υλοποίησή του έγινε καθαρά με σκοπό να διαπιστωθεί η δυνατότητα διασύνδεσης μεταξύ λογισμικού και επαναδιαμορφώσιμου υλικού, θα μπορούσε να γίνει μια αναβάθμιση ώστε να υπολογίζεται ο Μ.Ο. για μεταβαλλόμενο πλήθος δεδομένων, ίσο όμως με μια δύναμη του 2. Αυτός ο περιορισμός οφείλεται στην τεχνική υπολογισμού της διαίρεσης, η οποία κάνει χρήση της μεθόδου των δεξιών ολισθήσεων. Για να ξεπεραστεί αυτός ο περιορισμός, μπορεί να γίνει μια ακόμη βελτίωση, και να υλοποιηθεί στο FPGA μονάδα διαίρεσης η οποία να υποστηρίζει και αρνητικούς αριθμούς, είτε ακέραιους είτε αριθμούς κινητής υποδιαστολής. Με αυτόν τον τρόπο θα μπορεί να γίνεται υπολογισμός Μ.Ο. για οποιοδήποτε πλήθος αριθμών. Μια άλλη βελτίωση του συστήματος, η οποία όμως ξεφεύγει αρκετά από την λογική του υπολογισμού του Μ.Ο., είναι να σχεδιαστεί ένα περιφερειακό το οποίο θα δέχεται μια ροή δεδομένων υψηλής ταχύτητας, όπως δεδομένα από μια κάμερα, θα τα επεξεργάζεται και θα επιστρέφει το αποτέλεσμα τόσο τοπικά όσο και στο υπόλοιπο δίκτυο IoT. Επίσης θα μπορούσε να σχεδιαστεί ένα περιφερειακό, το οποίο να υλοποιεί κάποιο μοντέλο τεχνητής νοημοσύνης ώστε να συμβάλει σε μία διαδικασία μηχανικής μάθησης. Βελτιώσεις μπορούν να γίνουν και στην πλευρά του λογισμικού, και συγκεκριμένα στην εφαρμογή Python. Μια τέτοια βελτίωση θα μπορούσε να αφορά τον έλεγχο των εισερχομένων δεδομένων, ώστε να μεταφέρονται προς το περιφερειακό συγκεκριμένα δεδομένα προς επεξεργασία.

Χρησιμοποιώντας το Minized στην παρούσα εργασία αναπτύχθηκε ένας κόμβος υψηλής επεξεργαστικής ισχύος, λόγω της υλοποίησης εξειδικευμένων μονάδων υλικού στο FPGA. Στόχος ήταν να δειχθεί ότι είναι δυνατόν τα δεδομένα να μετακινηθούν γρήγορα προς και από την επαναδιαμορφούμενη λογική, η οποία έχει υψηλές επεξεργαστικές δυνατότητες. Από τη χρήση της αναπτυξιακής πλακέτας Minized, συμπεραίνεται ότι πρόκειται για μία πολύ εύχρηστη, αξιόπιστη, υψηλής απόδοσης και χαμηλού κόστους πλατφόρμα. Χρησιμοποιώντας το Minized ένας μηχανικός μπορεί να σχεδιάσει ένα πολύπλοκο σύστημα υψηλής επεξεργαστικής ισχύος το οποίο λόγω της χαμηλής κατανάλωσης και της δικτύωσης που προσφέρεται από το Minized μπορεί να εγκατασταθεί στις παρυφές ενός δικτύου IoT. Τέλος, η μεγάλη πληθώρα πληροφοριών και η υποστήριξη που παρέχεται από την κατασκευάστρια εταιρεία της αναπτυξιακής πλακέτας βοηθούν

στη δημιουργία του συστήματος, εξοικονομώντας σημαντικό χρόνο για τον μηχανικό. Συνοψίζοντας, σύμφωνα με όλα τα παραπάνω η αναπτυξιακή πλακέτα Minized αποτελεί πολύ καλή λύση για την ανάπτυξη κόμβων χαμηλού κόστους και υψηλής επεξεργαστικής ισχύος.

Βιβλιογραφία

[1] Dorsemaine, B., Gaulier, J. P., Wary, J. P., Kheir, N., & Urien, P. (2015, September). Internet of things: a definition & taxonomy. In 2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies (pp. 72-77). IEEE.

[2] Saleh, R., Wilton, S., Mirabbasi, S., Hu, A., Greenstreet, M., Lemieux, G. & Ivanov, A. (2006). System-on-chip: Reuse and integration. *Proceedings of the IEEE*, *94*(6), 1050-1069.

[3] Benhani, E. M., Bossuet, L., & Aubert, A. (2019). The Security of ARM TrustZone in a FPGA-based SoC. *IEEE Transactions on Computers*, *68*(8), 1238-1248.

[4] "MQ Telemetry Transport," http://mqtt.org.

[5] "Beginners Guide To The MQTT Protocol", http://www.steves-internet-guide.com/mqtt/

Παράρτημα Ι – Μέρος κώδικα VHDL από την σχεδίαση του περιφερειακού υπολογισμού Μ.Ο.

```
if slv_reg2(1) = '1' then                -- if data_transfer_completed  -> calculate avg
   slv_reg1 <= std_logic_vector(shift_right(unsigned(slv_reg1),4));     -- Calculate average value
   slv_reg2(1) <= '0';                   -- Reset data_transfer_completed flag
   slv_reg2(2) <= '1';                   -- Set avg_completed_flag
else if slv_reg2(0) = '1' then           -- Write data flag
    slv_reg1 <= slv_reg1 + slv_reg0;     -- sum value
    slv_reg2(0) <= '0';                  -- Reset write_data flag, for new data reception
end if;
end if;
```

Παράρτημα ΙΙ – Οδηγός χαμηλού επιπέδου

```
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/mman.h>

static volatile uint32_t *dataIn;   //Data input pointer
static volatile uint32_t *dataOut;  //Data output pointer
static volatile uint32_t *ctrlReg;  //Control Register pointer

static const uint32_t data_in_addr = 0x43C10000;  //Data input register
static const uint32_t data_out_addr = 0x43C10004; //Data output register
static const uint32_t ctrl_addr = 0x43C10008;     //Ctrl register

void map_gpios(void)
{
  int fd;

  //open memory file for Read and Write
  if ((fd = open("/dev/mem", O_RDWR|O_SYNC)) < 0){
    perror("/dev/mem");
    exit(-1);
  }

  //map data_in_addr address in memory
  if ((dataIn = mmap(0, getpagesize(), PROT_READ|PROT_WRITE, MAP_SHARED, fd,
data_in_addr))
      == MAP_FAILED) {
    perror("dataIn");
    exit(-1);
  }

  dataOut = dataIn+1;
  ctrlReg = dataIn+2;
}


int main(int *data)
{
  map_gpios();
  int i=0;
  /* ------- Initiallize Registers -------*/
  *dataIn = 0;
  *dataOut = 0;
  *ctrlReg = 0;

  /* ------- Starting Loop -------*/
  printf("Reading registers...\r\n");

  while (*ctrlReg != 4){   //ctrlReg=4 (AVG_Compl Flag): Average calculation
                           //completed if AVG_Compl Flag == '1'
```

```
    if (*ctrlReg == 0){     //ctrlReg=0 (DataIn_OK Flag): HW ready to receive
                                //new data if DataIn_OK Flag == '0'
      *ctrlReg = 1;         //ctrlReg=1 (Set DataIn_OK Flag): Set HW to sum data
      *dataIn = data[i];    //Pass data to dataIn register
      i++;                  //go to next data
    }
    if (i==17){             //When all data (16) passed
      printf("Sum Result: 0x%08x \n", *dataOut); //Print Sum
      *ctrlReg = 2;        //ctrlReg=2 (Set Din Trans Compl Flag): set HW
                                //to calculate average
    }
  }
  *ctrlReg = 0;            //ctrlReg=0 : Reset HW control register

  printf("dataOut reg: 0x%08x \n",*dataOut); // Print Average

  return *dataOut;         //return Average
}
```

Παράρτημα ΙΙΙ – Εφαρμογή μεταφοράς δεδομένων

```
# python 3.6

import random
import time
from paho.mqtt import client as mqtt_client
import os
import subprocess
import sys
import ctypes

#Low level driver
lib = ctypes.CDLL("./avgDriver.so")

broker = 'xxxxx.xxxxx.com'
port = xxxxx
topic_rx = "python/minized"
topic_tx = "minized/broker"
client_id = "minized"
username = 'xxxxx'
password = 'xxxxx'

#List to save data
dataList = []

#Connect to MQTT Broker subroutine
def connect_mqtt():
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", rc)
    client = mqtt_client.Client(client_id)
    client.username_pw_set(username, password)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client

#Publish data subroutine
def publish(client):
    time.sleep(1)
    msg = "Hi I am minized!"
    result = client.publish(topic_tx, msg)
    status = result[0]
    if status == 0:
        print("OK")
    else:
        print("FAILED")

#Subscribe to topic subroutine
def subscribe(client: mqtt_client):
    #On message reception subroutine
    def on_message(client, userdata, msg):
        print("Received:")
        print(msg.payload.decode("utf-8"))
```

```
            dataArray.append(int(msg.payload))  #Append data in data array
            if len(dataList)==16:               #When 16 values have received
                #Convert values to integer format
                arr1 = (ctypes.c_int * len(dataArray))(*dataArray)
                #Call the low level driver
                avgRes = lib.main(ctypes.byref(arr1))
                #Print average result
                print(avgRes)
                #Publish average value
                sentResult = client.publish(topic_tx, avgRes)
                status = sentResult[0]
                if status == 0:
                    print("OK")
                else:
                    print("FAILED")
        #Subscribe to topic
        client.subscribe(topic_rx)
        #Call on_message subroutine on message reception
        client.on_message = on_message

    #Main subroutine
    def run():
        #Connect to MQTT
        client = connect_mqtt()
        #Start client
        client.loop_start()
        #Register subscribe subroutine
        subscribe(client)
        #Register publish subroutine
        publish(client)
        #Client loop forever
        client.loop_forever()

    if __name__ == '__main__':
        run()
```

Παράρτημα IV – Εντολές Linux

Εντολή για την μεταγλώττιση του κώδικα σε αρχείο τύπου βιβλιοθήκης (dll):

arm-linux-gnueabihf-gcc avgDriver.c –o avgDriver.so –shared -fpic