ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Design and Acceleration of Omnitrap Ion Storage Device Instrument Control

Διπλωματική Εργασία

Σταύρος Π. Κούβαρης

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2020

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Design and Acceleration of Omnitrap Ion Storage Device Instrument Control

Διπλωματική Εργασία

Σταύρος Π. Κούβαρης

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 10η Ιουλίου 2020

……………..	……………..	……………..
Δημήτριος Σούντρης	*Παναγιώτης Τσανάκας*	*Παύλος Σωτηριάδης*
Καθηγητής Ε.Μ.Π.	*Καθηγητής Ε.Μ.Π.*	*Καθηγητής Ε.Μ.Π.*

Αθήνα, Ιούλιος 2020

....

Σταύρος Κούβαρης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η παγίδα ιόντων Omnitrap, της εταιρείας Fasmatech, είναι μία διάταξη που επιτρέπει την παγίδευση και επεξεργασία ιόντων σε φασματομετρικές εφαρμογές. Στην παρούσα πειραματική διάταξη το Omnitrap χρησιμοποιείται ως πρόσθετο εξάρτημα του φασματομέτρου μάζας Q-Exactive, της εταιρείας Thermo Fisher Scientific. Η συγκεκριμένη διπλωματική εργασία αφορά την ανάπτυξη συστήματος FPGA που επιτρέπει τον έλεγχο της λειτουργίας του Omnitrap μέσω προσωπικού υπολογιστή. Ιδιαίτερη έμφαση δίνεται σε μία από τις λειτουργίες του Omnitrap, που είναι η απομόνωση ιόντων συγκεκριμένης μάζας, στο εσωτερικό της παγίδας, με τη χρήση κυματομορφών απομόνωσης Sweep και Filtered-Noise-Field. Για του σκοπούς του πειράματος, υλοποιείται κατάλληλος δίαυλος επικοινωνίας ανάμεσα σε PC και FPGA. Επιπλέον, μέθοδοι για τη γέννηση και παραμετροποίηση των κυματομορφών απομόνωσης υλοποιούνται τόσο στο PC όσο και στο FPGA. Στη συνέχεια συγκρίνονται μεταξύ τους και αξιολογούνται ως προς την καταλληλότητα τους για το συνολικό σύστημα. Τελικά, προκύπτουν συμπεράσματα για την τελική μορφή του συστήματος ελέγχου του οργάνου, με οριστικό στόχο την προϊοντοποίηση.

Λέξεις Κλειδία

FPGA, ψηφιακή σύνθεση κυμαρομορφών, κυματομορφές Sweep, κυμαρομορφές Filtered-Noise Field, προϊοντοποίηση, φασματομετρία μάζας, παγίδα ιόντων, omnitrap

Abstract

The Omnitrap linear segmented ion trap, developed in Fasmatech company, allows enhanced ion activation and storage in mass spectrometry applications. In the current experimental setup, Omnitrap is connected in series with Q-Exactive mass spectrometer of Thermo Fisher Scientific company. This diploma thesis concerns the development of an FPGA system that allows control of Omnitrap operation via a personal computer. Special emphasis is given on one of the Omnitrap functionalities, which is the isolation of ions of specific mass to charge ratio by means of Sweep and Filtered-Noise-Field isolation waveforms. For the experimental purposes, an effective communication port is developed between PC and FPGA. In addition to that, several methods for waveform generation, both by PC and FPGA, are implemented, compared and evaluated in regard to their efficiency for the overall system. Finally, conclusions are drawn concerning the final instrument design, towards productization.

Keywords

FPGA, direct digital synthesis, Sweep waveform, Filtered-Noise Field waveform, productization, mass spectrometry, ion trap, omnitrap

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων (Microlab) στο ΕΜΠ, σε συνεργασία με την εταιρεία Fasmatech Science & Technology SA που στεγάζεται στο Τεχνολογικό Πάρκο Λεύκιππος στο Εθνικό Κέντρο Έρευνας Φυσικών Επιστημών «Δημόκριτος».

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέποντά μου, καθηγητή του ΕΜΠ Δημήτριο Σούντρη και τον CEO της εταιρείας Fasmatech, Δημήτριο Παπαναστασίου, που συνεργάστηκαν με επιτυχία τόσο μαζί μου όσο και μεταξύ τους για την ολοκλήρωση της συγκεκριμένης διπλωματικής εργασίας. Επιπλέον, θα ήθελα να ευχαριστήσω τον μεταδιδακτορικό ερευνητή του Microlab Γεώργιο Λεντάρη καθώς και τους μηχανικούς της Fasmatech, Ιωάννη Ορφανόπουλο και Αντρέα Μποζατζίδη, για τη βοήθεια που μου προσέφεραν τόσο σε επιστημονικό όσο και σε προσωπικό επίπεδο.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου, Πέτρο και Αργυρώ και την αδερφή μου Ματίνα για την διαρκή στήριξη και κατανόηση τους σε στιγμές αγχώδεις και πιεστικές.

Contents

1 Εκτεταμένη Περίληψη

1.1 Εισαγωγή

Η **φασματομετρία μάζας** είναι μια ευαίσθητη τεχνική για τον ποιοτικό και ποσοτικό προσδιορισμό χημικών ενώσεων και βρίσκει εφαρμογή σε μια μεγάλη γκάμα επιστημονικών πεδίων όπως βιολογία, χημεία, φυσική, φαρμακευτική, ακόμα και στην εξερεύνηση του διαστήματος. Σημαντικές εφαρμογές της φασματομετρίας μάζας στον σύγχρονο κόσμο, όπως αναδεικνύονται από την μεγαλύτερη εταιρεία ανάπτυξης και πώλησης φασματομετρικών οργάνων, τη Thermo Fisher Scientific, ανήκουν στα ερευνητικά πεδία της πρωτεϊνωματικής (proteomics), της μεταβολισμικής (metabolomics), της περιβαλλοντικής ανάλυσης, της εγκληματολογίας και των κλινικών αναλύσεων και ενδεικτικά κάποιες από αυτές είναι ο προσδιορισμός πρωτεϊνικών δομών, η αναγνώριση ακολουθιών πεπτιδίων, η διάγνωση καρκίνου, ο ποιοτικός έλεγχος του πόσιμου νερού, η ανάλυση υπολειμμάτων εκρηκτικού εξοπλισμού, η ανάπτυξη κλινικών φαρμάκων και η ανάλυση ασθενειών.

Μία φασματομετρική διαδικασία περιλαμβάνει κατά κανόνα τον ιονισμό των μορίων της αναλυτέας ουσίας, το διαχωρισμό τους σύμφωνα με τον λόγο της μάζας προς το φορτίο τους, τον ποσοτικό προσδιορισμό της έντασης της κάθε μάζας στο δείγμα και τελικά χρήση των παραπάνω δεδομένων για τον καθορισμό της δομής και των συστατικών του αναλύτη.

Η παρούσα διπλωματική εργασία βασίζεται στην παγίδα ιόντων **Omnitrap** που αναπτύχθηκε και τελικά κατοχυρώθηκε ως πατέντα από την εταιρεία Fasmatech. Η παγίδα ιόντων είναι μια διάταξη τετραπόλου η οποία επιτρέπει την παγίδευση φορτισμένων σωματιδίων στο κέντρο συμμετρίας της, με την εφαρμογή δυναμικών ηλεκτρικών πεδίων στους τέσσερις πόλους της. Η παγίδα ιόντων μπορεί επίσης να χρησιμοποιηθεί και ως φίλτρο μαζών, δηλαδή ως διάταξη που επιτρέπει τη διατήρηση ιόντων συγκεκριμένου εύρους μαζών στο εσωτερικό της και επομένως την απομάκρυνση των υπολοίπων μαζών. Η συσκευή Omnitrap περιλαμβάνει οκτώ παγίδες ιόντων, Q1 έως Q8, με ελλειψοειδείς πόλους, σε σειριακή διάταξη μεταξύ τους, και παρέχει ένα ευρύ φάσμα δυνατοτήτων ως προς την επεξεργασία των ιόντων πριν την έναρξη της φασματομετρικής διαδικασίας.

Στην πειραματική διάταξη της συγκεκριμένης εργασίας, το Omnitrap αποτελεί μία επέκταση του φασματόμετρου μάζας Q Exactive της Thermo Fisher Scientific. Συγκεκριμένα, παρεμβάλλεται στον κύκλο λειτουργίας του και προσθέτει ένα ακόμα επίπεδο επεξεργασίας των ιόντων πριν φτάσουν στο Orbitrap, δηλαδή στο τμήμα του Q Exactive στο οποίο συντελείται η φασματική μέτρηση. Ένας επιτραπέζιος υπολογιστής χρησιμοποιείται για τον προσδιορισμό της πειραματικής διαδικασίας και για τα δύο όργανα καθώς και για τη λήψη των αποτελεσμάτων. Το είδος αυτό της φασματομετρικής διάταξης που περιλαμβάνει την σειριακή σύνδεση και λειτουργία δύο ή περισσοτέρων συσκευών ανάλυσης μάζας, με σκοπό την βελτίωση των δυνατοτήτων τους στην ανάλυση χημικών δειγμάτων, είναι γνωστό με τον αγγλικό όρο **Tandem Mass Spectrometry.**

Ως προς τις λειτουργίες του Omnitrap, κύρια έμφαση δίνεται στις παγίδες Q2 και Q5 στις οποίες παρέχεται η δυνατότητα εφαρμογής κυματομορφών απομόνωσης. Οι κυματομορφές αυτές είναι σήματα τάσης τα οποία αναπαράγονται στους πόλους της παγίδας και ανάλογα με το συχνοτικό τους περιεχόμενο, απομακρύνουν ένα εύρος μαζών από το εσωτερικό της παγίδας. Πιο συγκεκριμένα, οι μάζες των οποίων η ιδιοσυχνότητα ταλάντωσης δεν περιλαμβάνεται στο συχνοτικό περιεχόμενο της κυματομορφής απομόνωσης, διατηρούνται στην παγίδα. Αντίθετα, οι υπόλοιπες μάζες ταλαντώνονται, οδηγούνται σε συντονισμό, και τελικά απομακρύνονται, προσκρούοντας στους πόλους της παγίδας. Η διατήρηση μόνο των ιόντων ενδιαφέροντος στην παγίδα συμβάλλει σε μεγάλο βαθμό στη βελτίωση της διακριτικής ικανότητας του φασματόμετρου μάζας.

Συσκευή FPGA

Για τον έλεγχο των εξαρτημάτων του Omnitrap χρησιμοποιείται μία συσκευή FPGA. Η συσκευή FPGA (Field Programmable Gate Array ή συστοιχία επιτόπια προγραμματιζόμενων πυλών) αποτελείται εσωτερικά από ένα δίκτυο προγραμματιζόμενων ψηφιακών “κυττάρων” (cells) που μπορούν να υλοποιήσουν ψηφιακές συναρτήσεις. Πρακτικά, μια συσκευή FPGA επιτρέπει τη σχεδίαση και υλοποίηση ενός ψηφιακού κυκλώματος στον πραγματικό κόσμο, με τη χρήση λογισμικού. Επιπλέον διαθέτει πόρτες εισόδου – εξόδου για την αλληλεπίδραση του κυκλώματος με εξωτερικά εξαρτήματα. Μία συσκευή FPGA χρησιμοποιείται για τους εξής λόγους:

- Επιτρέπει την επιτάχυνση και βελτιστοποίηση βαρέων υπολογισμών, όπως η ψηφιακή σύνθεση κυματομορφών.
- Περιλαμβάνει πόρτες εισόδου – εξόδου, επιτρέποντας άμεσο έλεγχο των περιφερειακών εξαρτημάτων του οργάνου και επομένως σχεδόν μηδενικό χρόνο αντίδρασης σε μεταβολές.
- Συμβάλλει στην απομόνωση του συνολικού συστήματος από εξωτερικές επιρροές και επιτρέπει ντετερμινιστική συμπεριφορά υψηλής χρονικής ακρίβειας, καθώς οι περισσότερες διαδικασίες που συντελούνται στο Omnitrap απαιτούν αξιόπιστο και ακριβή έλεγχο, σε επίπεδο παλμών ρολογιού.

Στόχοι διπλωματικής εργασίας

Οι στόχοι της παρούσας διπλωματικής εργασίας είναι:

- Η ανάπτυξη ενός πλήρους συστήματος που επιτρέπει τον χαμηλού επιπέδου έλεγχο της συσκευής Omnitrap και παρέχει ευελιξία στον προσδιορισμό της πειραματικής διαδικασίας καθώς το όργανο βρίσκεται ακόμα σε στάδιο ανάπτυξης και νέες τεχνικές για χρήση της παγίδας εξετάζονται συνεχώς.
- Η ανάπτυξη ενός αποτελεσματικού και αξιόπιστου διαύλου επικοινωνίας μεταξύ PC και FPGA , καθώς η συσκευή FPGA αποτελεί το πυρήνα επεξεργασίας των λειτουργιών του Omnitrap.

- Η εξερεύνηση των δυνατοτήτων του FPGA σχετικά με την ψηφιακή σύνθεση κυματομορφών.
- Η παραχώρηση προτάσεων για την τελική μορφή του οργάνου.

Κατά την εκπόνηση της διπλωματικής εργασίας, έγιναν εκτεταμένες προσπάθειες για την επιτάχυνση της μεταφοράς δεδομένων από το PC στο FPGA. Επιπλέον, πραγματοποιήθηκε σύγκριση μεταξύ της σύνθεσης κυματομορφών από το PC έναντι της άμεσης σύνθεσης και αναπαραγωγής τους από το FPGA. Πλεονεκτήματα και μειονεκτήματα και των δύο τεχνικών αναφέρονται και συμπεράσματα προκύπτουν τα οποία βρίσκουν εφαρμογή σε όλα τα σύγχρονα συστήματα που περιλαμβάνουν συσκευές FPGA, και ειδικότερα που αφορούν εργαστηριακό εξοπλισμό και όργανα πειραματικών μετρήσεων.

1.2 Ανάπτυξη Συστήματος

Διεπαφή χρήστη για σύνθεση κυματομορφών

Για τον πειραματισμό και την εφαρμογή κυματομορφών απομόνωσης στην παγίδα Omnitrap, αναπτύχθηκε ο απαραίτητος αλγόριθμος για τη δημιουργία των κυματομορφών καθώς επίσης και το αντίστοιχο περιβάλλον στο PC μέσω του οποίου ο χρήστης μπορεί να παραμετροποιήσει το εκάστοτε σήμα. Οι κυματομορφές αυτές συντίθενται στο PC και στη συνέχεια αποστέλλονται στο FPGA για αναπαραγωγή στους πόλους του αντίστοιχου τετραπόλου. Υπάρχουν δύο είδη σημάτων απομόνωσης, η κυματομορφή Sweep και η κυματομορφή Filtered Noise Field.

<u>Κυματομορφή Sweep:</u> Η κυματομορφή Sweep είναι μία αρμονική συνάρτηση με γραμμικά αυξανόμενη συχνότητα. Το συχνοτικό της περιεχόμενο περιλαμβάνει όλες τις συχνότητες από μία αρχική έως μία τελική τιμή. Ενδεικτικά οι εξισώσεις υπολογισμού της κυματομορφής φαίνονται παρακάτω:

$\boldsymbol{Sweep(n) = \sin\,[Phase(n)],}$

όπου η φάση *<u>Phase(n)</u>* υπολογίζεται ως εξής:
$\boldsymbol{Phase(n) = Phase(n-1) + 2pi * Ts * frequency(n),}$
με *Ts:* η περίοδος δειγματοληψίας.

Η συχνότητα *<u>Frequency(n)</u>* αυξάνεται γραμμικά:
$\boldsymbol{frequency(n) = frequency(n-1) + frequency\_step.}$

Ο χρήστης, μέσω του λογισμικού δύναται να παραμετροποιήσει την κυματομορφή Sweep ως προς το πλάτος, την αρχική και την τελική συχνότητα.

Για το σκοπό της διατήρησης συγκεκριμένων μαζών στην παγίδα, δίνεται η δυνατότητα στον χρήστη εξαίρεσης από το συχνοτικό περιεχόμενο της κυματομορφής Sweep, ενός εύρους συχνοτήτων, ανάλογα με την εφαρμογή. Όπως έχει ήδη αναφερθεί, οι μάζες των οποίων η συχνότητα ιδιοταλάντωσης δεν περιλαμβάνεται στο συχνοτικό περιεχόμενο του σήματος, παραμένουν στην παγίδα μετά την αναπαραγωγή του Sweep, ενώ οι υπόλοιπες διεγείρονται και απομακρύνονται.

Επιπλέον δυνατότητες ως προς την παραμετροποίηση του Sweep εμφανίζονται στη διεπαφή χρήστη που αναπτύχθηκε, όπως η εξομάλυνση του πλάτους εισόδου και εξόδου καθώς και ο καθορισμός της συνολικής διάρκειας του Sweep.

<u>Κυματομορφή FNF:</u> Η κυματομορφή Filtered Noise Field (FNF) συντίθεται από το άθροισμα ημιτονοειδών συναρτήσεων διαφορετικής συχνότητας. Ομοίως και εδώ, η κυματομορφή χαρακτηρίζεται από μία αρχική και μία τελική συχνότητα, στις οποίες προστίθεται και η παράμετρος του βήματος

συχνότητας. Το βήμα συχνότητας καθορίζει τη διακριτική ικανότητα του συχνοτικού περιεχομένου του FNF σήματος που προκύπτει.

Όπως και στην περίπτωση της κυματομορφής Sweep, έτσι και εδώ ο χρήστης έχει τη δυνατότητα να προσθέσει συχνοτικά κενά στην FNF κυματομορφή. Επιπλέον, υπάρχουν δυνατότητες τροποποίησης του πλάτους καθώς και εφαρμογής τεχνικών διαμόρφωσης φάσης για καλύτερη κατανομή της ισχύος κατά μήκος του σήματος.

Βασική διαφορά του σήματος Sweep με το σήμα FNF είναι το γεγονός ότι στο πρώτο οι συχνότητες αναπαράγονται σειριακά η μία μετά την άλλη, ενώ στο δεύτερο οι συχνότητες αναπαράγονται ταυτόχρονα καθ’όλη τη διάρκεια της κυματομορφής. Πειραματικά κάθε μία χρησιμοποιείται για διαφορετικές εφαρμογές και παρουσιάζει τόσο πλεονεκτήματα όσο και μειονεκτήματα.

Έλεγχος Συστήματος

Για τον έλεγχο της πειραματικής διαδικασίας του Omnitrap, ο χρήστης καλείται μέσω λογισμικού να συνθέσει μία ακολουθία εντολών. Κάθε εντολή σχετίζεται μία ξεχωριστή ενέργεια του οργάνου ενώ των σύνολο των εντολών προσφέρει δυνατότητες τροποποιήσεων του πειράματος σε χαμηλό επίπεδο. Ενδεικτικά ορισμένες από τις σημαντικότερες εντολές αφορούν δράσεις όπως:

- **Διακοπή της πειραματικής διαδικασίας μέχρι τα ιόντα να περάσουν από το όργανο Q Exactive στην παγίδα ιόντων Omnitrap**. Το γεγονός αυτό σηματοδοτείται από το θετικό παλμό ενός σήματος trigger που προέρχεται από το Q Exactive και παραλαμβάνεται από το FPGA που ελέγχει το Omnitrap.
- **Μετακίνηση του δείγματος ιόντων από μία παγίδα Qx σε μία άλλη παγίδα Qy** (υπάρχουν οκτώ παγίδες ιόντων διαθέσιμες, Q1 έως Q8)
- **Αναπαραγωγή κυματομορφής απομόνωσης (στα τετράπολα Q2 και Q5).** Οι κυματομορφές απομόνωσης διακρίνονται σε δύο είδη, τα σήματα Sweep και τα σήματα Filtered Noise Field (FNF) το κάθε ένα με τα δικά του χαρακτηριστικά.

Για την εκτέλεση του πειράματος, τα δεδομένα της ακολουθίας εντολών κωδικοποιούνται κατάλληλα από το PC και στέλνονται στο FPGA, όπου και αποθηκεύονται σε εσωτερική μνήμη block RAM. Στη συνέχεια, μία custom IP είναι υπεύθυνη για την εκτέλεση των εντολών, όταν ο χρήστης σηματοδοτήσει, από το λογισμικό στο PC, την έναρξη της διαδικασίας.

Βελτιώσεις Συστήματος

Επικοινωνία με το FPGA

Για τη συγκεκριμένη διάταξη, τα δεδομένα που χρειάζεται να μεταφέρονται από το PC στο FPGA είναι τα εξής:

- Η κωδικοποιημένη ακολουθία εντολών που καθορίζει την πορεία του πειράματος
- Τα δείγματα των ψηφιακών κυματομορφών απομόνωσης Sweep και FNF

Τα δεδομένα που χρειάζεται να μεταφέρονται από το FPGA στο PC είναι¨

- Μετρήσεις τάσεων και θερμοκρασίας που πραγματοποιούνται σε διάφορα σημεία του οργάνου

Για την δημιουργία διαύλου επικοινωνίας μεταξύ PC και FPGA υλοποιήθηκαν και δοκιμάστηκαν τέσσερις διαφορετικές μέθοδοι οι οποίες περιγράφονται περιληπτικά παρακάτω.

USB 2.0 σε πρωτόκολλο UART: Το πρωτόκολλο UART (Universal Asynchronous Receiver/Transmitter) είναι ένα σειριακό ασύγχρονο πρωτόκολλο που επιτρέπει τη μεταφορά δεδομένων μεταξύ δύο ή περισσότερων συσκευών οι οποίες είναι σύγχρονες σε διαφορετικά ρολόγια. Απαιτεί δύο άκρα εισόδου εξόδου για την εγκατάσταση του μεταξύ δύο συσκευών. Κάθε συσκευή οφείλει να διαθέτει ένα άκρο λήψης και ένα άκρο αποστολής δεδομένων.

Για την υλοποίηση της συγκεκριμένης επικοινωνιακής μεθόδου χρησιμοποιήθηκε το ολοκληρωμένο FT2232H της FTDI σε λειτουργία γέφυρας USB 2.0 σε UART. Η εταιρεία παρέχει το απαραίτητο API για την ανάπτυξη desktop εφαρμογών που χρησιμοποιούν το συγκεκριμένο δίαυλο επικοινωνίας. Τα δεδομένα μεταφράζονται από το USB πρωτόκολλο του PC στο απλοποιημένο πρωτόκολλο UART, ώστε να ληφθούν από το FPGA.

Για τη λήψη και διαχείριση των δεδομένων από το FPGA χρησιμοποιήθηκαν κατά κύριο λόγο οι παρακάτω IPs:

- **Microblaze soft-core** : ο επεξεργαστής που παραχωρείται από την Xilinx για χρήση εσωτερικά του FPGA. Υλοποιείται με τη χρήση πόρων του FPGA.
- **AXI UARTlite**: ο ελεγκτής πρωτοκόλλου UART που παρέχεται από τη Xilinx.
- **Memory Interface Generator:** ο ελεγκτής εξωτερικής μνήμης τυχαίας προσπέλασης (DDR) που παρέχεται από τη Xilinx.

Οι IPs εσωτερικά του FPGA επικοινωνούν μεταξύ τους μέσω του AXI bus. Τα δεδομένα από τον δίαυλο UART παραλαμβάνονται από τον Microblaze, ο οποίος είναι υπεύθυνος για την προώθηση τους στον εκάστοτε προορισμό. Επιπλέον, επιστρέφει στο PC ένα byte επιβεβαίωσης το οποίο σηματοδοτεί την επιτυχή ολοκλήρωση της μεταφοράς των δεδομένων. Σε περίπτωση που τα δεδομένα αντιστοιχούν σε σημεία κυματομορφών, ο Microblaze τα προωθεί σε εξωτερική μνήμη DDR3.

USB 2.0 σε σύγχρονο πρωτόκολλο FIFO 245: Για τη συγκεκριμένη υλοποίηση χρησιμοποιήθηκε ξανά το ολοκληρωμένο FT2232H της FTDI, αυτή τη φορά σε λειτουργία γέφυρας USB 2.0 σε FIFO245. Το πρωτόκολλο FIFO 245 είναι ένα σύγχρονο παράλληλο πρωτόκολλο με μέγεθος λέξης 8 bits. Για τη λήψη των δεδομένων από το FPGA απαιτείται δημιουργία IP καθώς δεν παρέχεται από τη Xilinx. Η επικοινωνία του FPGA με το ολοκληρωμένο πρέπει να είναι σύγχρονη με ρολόι συχνότητας 60 MHz το οποίο παρέχεται από το chip. Σε αυτή την υλοποίηση δεν χρησιμοποιείται ο Microblaze αλλά τα δεδομένα κατευθύνονται στον προορισμό τους μέσω custom IP που δημιουργήθηκε. Με αυτό τον τρόπο αποφεύγονται καθυστερήσεις που οφείλονται στην αρχιτεκτονική του soft-core επεξεργαστή, και οι ανταλλαγές δεδομένων βελτιστοποιούνται.

USB 2.0 με streaming δεδομένων: Η υλοποίηση αυτή επιχειρεί να εξαλείψει τις καθυστερήσεις που προκύπτουν από τη μεταφορά μεγάλων πακέτων δεδομένων από το PC στο FPGA. Για την παρούσα εφαρμογή τα δεδομένα κυματομορφών αποτελούν τέτοιου είδους πακέτα. Χρησιμοποιεί ξανά το ολοκληρωμένο FT2232H σε λειτουργία γέφυρας USB 2.0 σε FIFO 245. Τα δεδομένα κυματομορφών πλέον δεν αποθηκεύονται σε εξωτερική μνήμη. Αντιθέτως, χρησιμοποιούνται ταυτόχρονα με την αποστολή τους από το PC, ενώ η αναπαραγωγή της κυματομορφής στους πόλους της παγίδας ξεκινά μόλις το πρώτο σημείο έχει ληφθεί από το FPGA. Μία FIFO (First In / First Out) εσωτερική μνήμη χρησιμοποιείται για το συγχρονισμό του ρυθμού δειγματοληψίας και της ταχύτητας μεταφοράς των δεδομένων. Η υλοποίηση αυτή είναι εφικτή καθώς η συχνότητα δειγματοληψίας είναι μικρότερη από τη συχνότητα του ρολογιού των 60MHz στο οποίο είναι σύγχρονη η μεταφορά δεδομένων.

USB 3.0 σε σύγχρονο πρωτόκολλο FIFO 245: Τέλος, στη συγκεκριμένη υλοποίηση γίνεται αναβάθμιση του πρωτοκόλλου από USB 2.0 σε USB 3.0 με σκοπό υψηλότερες ταχύτητες μεταφοράς δεδομένων. Χρησιμοποιείται το ολοκληρωμένο FT601Q της FTDI σε λειτουργία γέφυρας USB 3.0 σε FIFO 245. Η σχετική ψηφιακή λογική για τη λήψη των δεδομένων από το FPGA αναπτύσσεται ενώ τα δεδομένα κυματομορφών αποθηκεύονται σε εξωτερική DDR μνήμη. Η μεταφορά των δεδομένων στην DDR μνήμη πραγματοποιείται με τη χρήση AXI Stream πρωτοκόλλου με στόχο την επίτευξη υψηλότερων ταχυτήτων εγγραφής.

Επιτάχυνση της παραγωγής κυματομορφών με τη χρήση FPGA

Το γεγονός ότι οι βασικές λειτουργίες του οργάνου Omnitrap ελέγχονται από το FPGA, καθιστά αξιόλογη την ιδέα της υλοποίησης των κυματομορφών απομόνωσης από το ίδιο το FPGA. Παράλληλα, οι διάφορες παραμετροποιήσεις των Sweep και FNF κυματομορφών που παρέχονται στο λογισμικό πρέπει να παραμείνουν διαθέσιμες και στην περίπτωση παραγωγής τους από το FPGA.

Ψηφιακή σύνθεση κυματομορφής Sweep στο FPGA: Για την παραγωγή της κυματομορφής Sweep από το FPGA χρησιμοποιήθηκε ο ίδιος αλγόριθμος υπολογισμού που υλοποιεί το Sweep στο PC και αναφέρεται σε προηγούμενη ενότητα. Όπως φαίνεται, για τον υπολογισμό του σήματος Sweep απαιτείται ο υπολογισμός της συνάρτησης του ημιτόνου. Για το σκοπό αυτό χρησιμοποιήθηκε η τεχνική του Look-Up Table κατά την οποία το τεταρτημόριο μίας ημιτονικής περιόδου δειγματοληπτείται επαρκώς και τα σημεία αποθηκεύονται σε εσωτερική μνήμη (distributed RAM) του FPGA. Ο αριθμός των δειγματοληπτημένων σημείων καθορίζει το εύρος των ημιτονικών συχνοτήτων που μπορούν να αναπαραχθούν, σύμφωνα με τον θεώρημα δειγματοληψίας Nyquist.

Η πράξη του πολλαπλασιασμού που εμφανίζεται στις εξισώσεις του Sweep εξαλείφεται με κατάλληλη κανονικοποίηση των δεδομένων αρχικοποίησης. Έτσι ο αλγόριθμος υλοποιείται εσωτερικά του FPGA με τη χρήση μερικών αθροιστών, του ημιτονικού Look-Up Table και ενός συγκριτή ο οποίος είναι υπεύθυνος για τον καθορισμό του προσήμου του ημιτόνου κάθε φορά. Το κύκλωμα που πραγματοποιεί τον υπολογισμό Sweep αποτελείται από στάδια, σε pipeline μορφή, για επιτάχυνση των πράξεων. Με την προσθήκη επιπλέον σταδίων pipeline, πριν ή μετά τον βασικό αλγόριθμο, μπορούν να υλοποιηθούν πιο εξειδικευμένες παραμετροποιήσεις του Sweep, όπως τροποποίηση του πλάτους και εφαρμογή συχνοτικών κενών.

Ψηφιακή σύνθεση κυματομορφής FNF στο FPGA: Μία κυματομορφή FNF μπορεί να υπολογιστεί από το άθροισμα ημιτόνων διαφορετικής συχνότητας, οι οποίες καθορίζονται από το επιθυμητό εύρος συχνοτήτων του FNF σήματος, καθώς και από την επιθυμητή συχνοτική ανάλυση. Για τον υπολογισμό της κυματομορφής FNF χρησιμοποιήθηκε αντίστροφος μετασχηματισμός Fourier. Το σήμα κατασκευάζεται αρχικά στο πεδίο των συχνοτήτων, ορίζοντας ως μηδέν το πλάτος και τη φάση των συχνοτικών τόνων που δε θέλουμε να συμπεριλαμβάνονται στο σήμα. Επίσης για εξοικονόμηση χρόνου και πόρων του FPGA, ο αντίστροφος μετασχηματισμός πραγματοποιείται για μία μόνο περίοδο του FNF σήματος ενώ το τελικό σήμα δημιουργείται από αλληλουχία FNF περιόδων, μέχρι να επιτευχτεί η επιθυμητή διάρκεια.

Για συγκεκριμένη υλοποίηση χρησιμοποιήθηκε η IP Fast Fourier Transform v9.1 που παρέχει η Xilinx. Η συγκεκριμένη IP εκτελεί τον αλγόριθμο Cooley-Turkey για τον υπολογισμό του μετασχηματισμού. Το μέγεθος του IFFT (Inverse Fourier Transform) καθορίζεται από τη μέγιστη περίοδο ενός FNF σήματος η οποία δεν ξεπερνά τα 5 milliseconds. Αφού ο ρυθμός δειγματοληψίας της παρούσας διάταξης ισούται με 12.5 MHz, το μέγιστο απαιτούμενο μέγεθος του μετασχηματισμού ανέρχεται στα 2^{16} σημεία.

Προσθέτοντας επιπλέον στάδια στην παρούσα υλοποίηση, σε μορφή pipeline, μπορούν να υλοποιηθούν περεταίρω παραμετροποιήσεις του σήματος, σε αντιστοιχία με αυτές του σήματος Sweep.

2 Introduction

Mass spectrometry is a powerful analytical technique that finds application in a huge variety of different fields like biology, chemistry and physics, but also in clinical medicine and even space exploration. It constitutes one of the most powerful modern physical and chemical methods for identifying compounds and for studying their structure and reactivity. Namely, some of the most common applications of mass spectrometry in the modern world, as pointed out by Thermo Fisher Scientific company, belong to the fields of proteomics (characterization of proteins, sequencing of peptides), metabolomics (cancer screening and diagnosis, biofuels generation and use), environmental analysis (drinking water testing, carbon dioxide and pollution monitoring), forensic analysis (analysis of trace evidence, identification of explosive residues) and clinical purposes (clinical drug development, clinical tests, disease screening) (1).

2.1 Mass Spectrometry and experimental set-up

A mass spectrometry experimental procedure includes ionization of the analyte's molecules, separation according to their mass-to-charge ratio, measurement of the detected ions' intensity in the sample and finally use of these data to decide the structure and contents of the analyte of interest. The mass-to-charge ratio of a cation (ion of negative charge) is defined as the mass of the cation divided by its charge. The results of a mass spectrometry experimental process are typically presented in a mass spectrum, a plot of intensity as a function of mass-to-charge ratio. Intensity in general refers to the quantity of different ions inside a chemical substance and is usually expressed in arbitrary values as its nature defers according to the mass spectrometry method that is followed.

Fasmatech's recent accomplishment is the invention and development of Omnitrap (Figure 2). Omnitrap is a segmented linear quadrupole ion trap which is capable of enhanced ion activation (ionization) and storage (2). A quadrupole ion trap (Figure 1), in general, is a type of ion trap that uses dynamic electric fields to trap charged particles. The linear ion trap uses a set of quadrupole rods to confine ions radially and a static electrical potential on-end electrodes to confine the ions axially. The linear form of the trap can be used as a selective mass filter, or as an actual trap by creating a potential well for the ions along the axis of the electrodes.

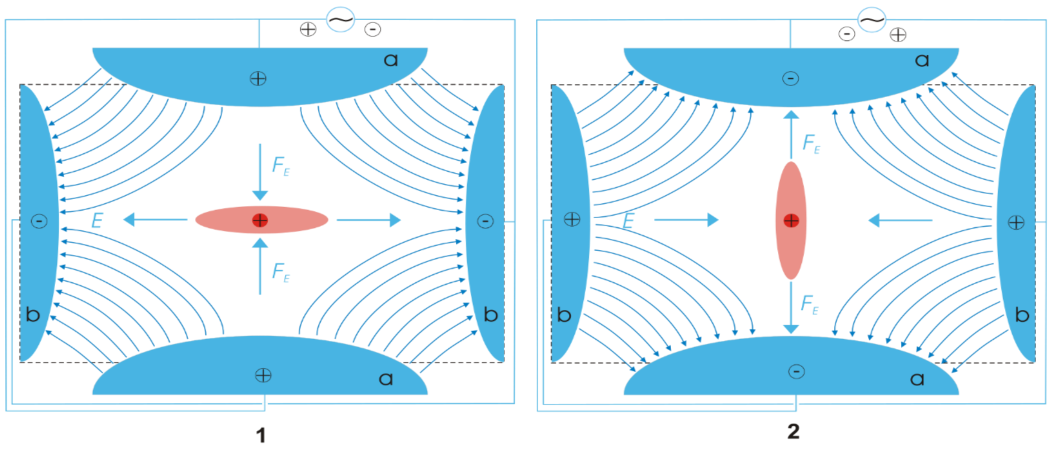


Figure 1: Scheme of a Quadrupole ion trap of classical setup with a particle of positive charge (dark red), surrounded by a cloud of similarly charged particles (light red). The electric field E (blue) is generated by a quadrupole of endcaps (a, positive) and a quadrupole of endcaps (a, positive) and a ring electrode (b). Picture 1 and 2 show two states during an AC cycle (3).

Linear ion traps are extremely powerful analytical devices, either deployed as stand-alone mass spectrometers or integrated in hybrid systems. Linear ion traps are also ideal platforms for developing and testing new techniques for manipulating gas phase ions in radio frequency (RF) trapping fields. Omnitrap provides multiple methods for sequential manipulation of ions in multiple trapping regions afforded by fast switching DC electrical potentials for high level control of ion potential energy and transfer between segments.

Omnitrap is designed with eight segments, Q1 to Q8, and hyperbolic surface electrodes supported on a stainless-steel structure cell. Differential pumping is provided through gaps between the bottom set of electrode-poles. A two-layer printed circuit board configuration is connected at the top of the Omnitrap and DC, RF and other AC signals are distributed to the electrode poles using spring contacts. A second pulse valve is used to admit fast gas pulses and an additional needle valve is employed to control background pressure.

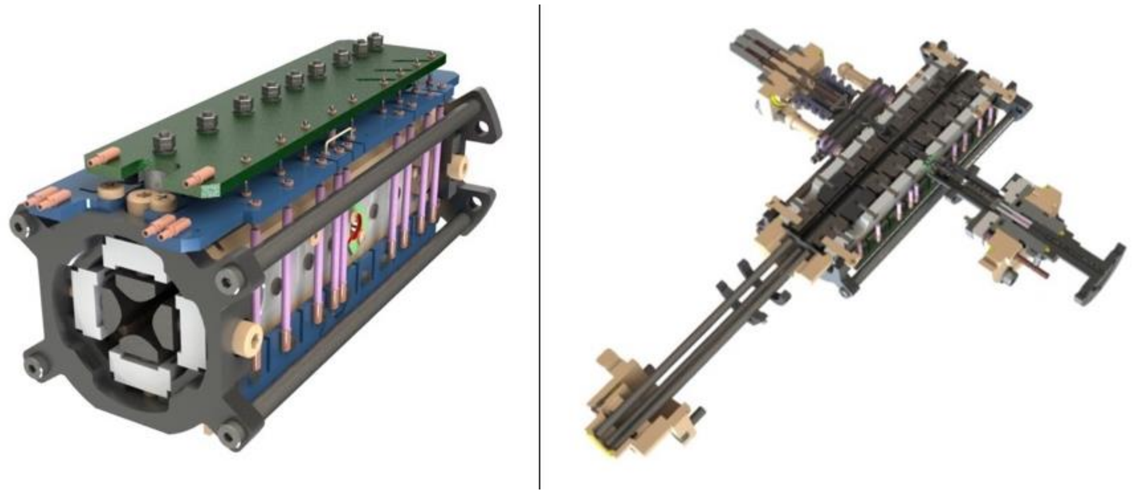


Figure 2: Omnitrap (4)

Different regions of Omnitrap are configured to support a diverse set of substantially different functions and the length of each segment is optimized accordingly. Each of the segments is connected to an independent switching module capable of switching the DC potential between 8 different levels during the course of an experiment.

Slow heating CID (collisionally activated dissociation, a technique to induce fragmentation of ions in the gas phase) and ion isolation using the Filtered-Noise-Field (FNF) and Sweep method are performed on one of the segments, Q2. Another interesting segment is Q5, which is designed with two apertures on opposite electrode-poles to allow injection of charged particle beams and photons.

For the application that is developed in the specific diploma thesis, Omnitrap constitutes and external instrument connected to Orbitrap Q Exactive mass analyzer by Thermo Fisher Scientific. Its components and a brief description of their functionality is given, for the sake of completeness, in Figure 3. The Q Exactive mass analyzer includes six main subcomponents: an ion source (1), an ion guide (2), a mass filter (3), an ion trap (5), an HCD cell (4) and the Orbitrap analyzer (6).

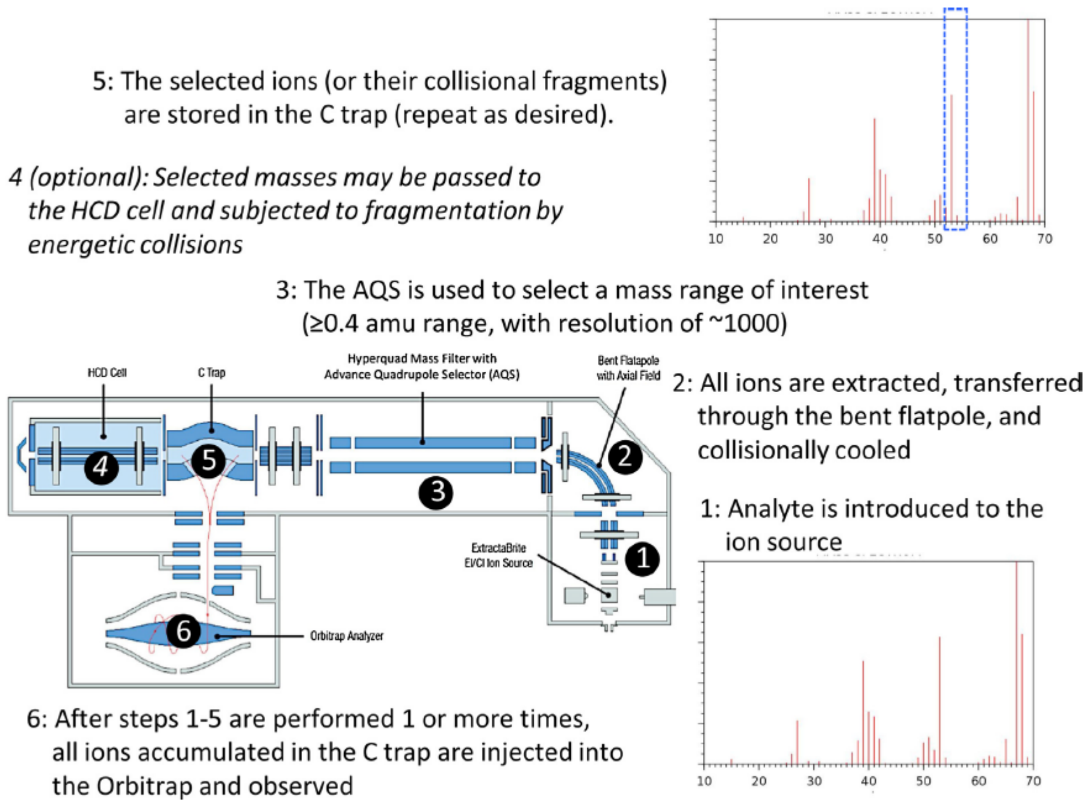

5: The selected ions (or their collisional fragments) are stored in the C trap (repeat as desired).
4 (optional): Selected masses may be passed to the HCD cell and subjected to fragmentation by energetic collisions
3: The AQS is used to select a mass range of interest (≥0.4 amu range, with resolution of ~1000)
2: All ions are extracted, transferred through the bent flatpole, and collisionally cooled
1: Analyte is introduced to the ion source
6: After steps 1-5 are performed 1 or more times, all ions accumulated in the C trap are injected into the Orbitrap and observed
HCD Cell
C Trap
Hyperquad Mass Filter with Advance Quadrupole Selector (AQS)
Bent Flatapole with Axial Field
ExtractaBrite EI/CI Ion Source
Orbitrap Analyzer
10
20
30
40
50
60
70

Figure 3: Schematic illustration of the major components of Q Exactive Orbitrap (5)

The Omnitrap platform is connected to the Q Exactive instrument in series with the HCD cell. Ions are processed in the Omnitrap platform and products are redirected back to the Q Exactive instrument for measuring mass-to-charge using the Orbitrap mass analyzer, as shown in Figure 4. Both instruments are controlled by the use of a PC which is responsible for both sending configuration data to them and receiving the corresponding experimental measurements, via USB interface. The technique of instrumental analysis where two or more mass analyzers are coupled together using an additional reaction step to increase their abilities to analyse chemical samples, is knows as **Tandem Mass Spectrometry**.

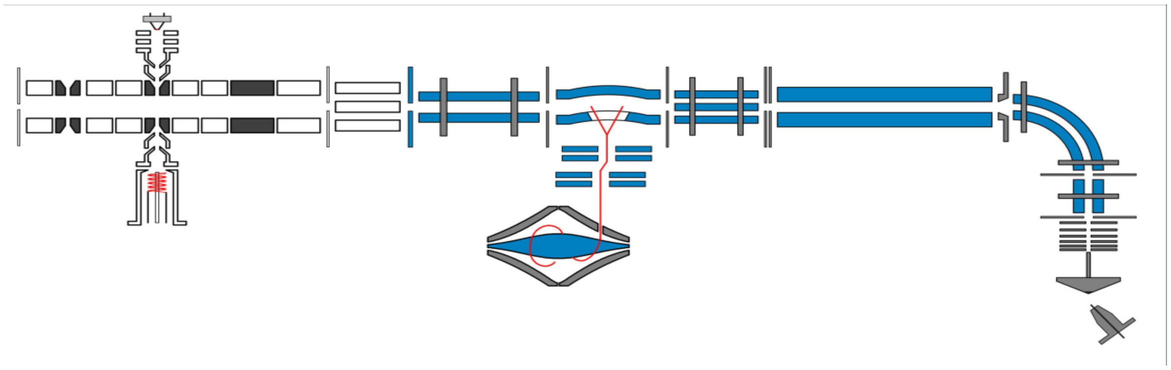


Figure 4: Omnitrap in series with Q Exactive (4)

Strong emphasis is given on the ion isolation methods by the use of Sweep and FNF signals waveforms. Ion isolation is the procedure, which aims to exclude every ion of an ionized analyte inside the ion trap, with the exception of a selected ion type with specified mass-to-charge ratio. Such a task is accomplished by applying voltage waveforms of specific frequency content in the poles of the trap. The frequency content of these waveforms defines the ions that remain in the ion trap and those that are excluded. To be more specific, ions whose resonance frequency is included in the isolation waveform's frequency spectrum, resonate and are consequently excluded from the ion trap. On the other hand, ions that do not resonate remain inside the trap in a stable state. Under this perspective, the main flow of experimental operation is predefined: Omnitrap is responsible for receiving an analyte, applying an ion excitation technique (Sweep or FNF) and afterwards passing the resulting ion substance to Q Exactive for mass spectrometry analysis and evaluation of results.

2.2 FPGA device

FPGAs (Field Programmable Gate Arrays) are devices that allow the design and real-world generation of digital circuits, by software. In contrast with microcontrollers, which include an already implemented processing architecture, FPGAs' circuitry has to be designed by scratch, in order to implement a desirable functionality. This fact makes FPGAs extremely customizable in comparison to microcontrollers. In order to program an FPGA and generate a digital logic circuit, hardware description language has to be used, which as implied by its name, is a programming language that is used to describe a circuit by code. In the current thesis, the hardware description language that is used is VHDL and the integrated design environment that is used is the version 18.1 of Vivado Design Suite.

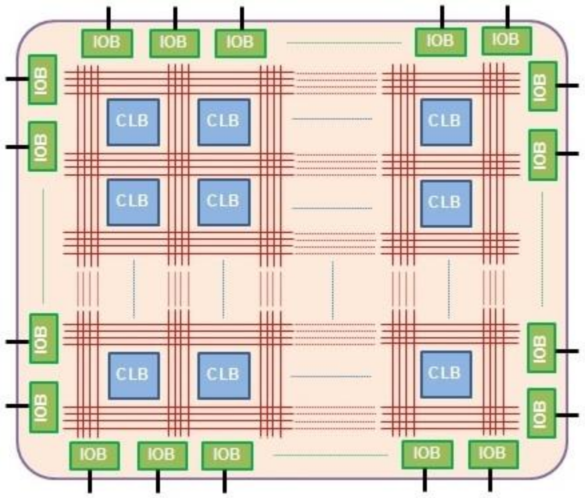


Figure 5: FPGA Configurable Logic Blocks (CLBs) network

An FPGA internally consists of a network of configurable logic blocks (CLBs, Figure 5), which can be programmed to implement specific digital logic functions while memory cells are also available for storage purposes. In some FPGAs, DSP slices are also included. Their name stands for Digital Signal Processing cells which are capable of implementing efficiently signal processing functions. Finally, each FPGA incorporates a prefixed number of Input-Output ports, so that it can interact with external circuitry.

For the purposes of the current project, an FPGA device is chosen on top of a microcontroller (6), because:

- It allows acceleration and optimization of heavy task computations, like digital waveform synthesis.
- It includes Input-Output ports that allow direct contact with hardware components and thus, hardly instant reaction capabilities.
- It contributes to overall system isolation from external interferences and allows deterministic behavior of high timing precision, as most of the Omnitrap functionality demands accurate, reliable and high speed timing control.

2.3 Thesis Scope

Omnitrap control by FPGA had been already implemented to some extent before the realization of this diploma thesis. Omnitrap is controlled by PC software and its actions are configurable by means of a series of instrument commands, which are summarily described in the first chapters. However, deficiencies have been present concerning PC – FPGA communication and overall FPGA processing speed. Overcoming this deficit and upgrading the overall instrument design to a more robust and efficient one has been a major need. Under this perspective, the main goals of this diploma thesis have been:

- The development of a complete system that achieves low-level Omnitrap instrument control and offers high versatility in the experimental process while new techniques of using the linear trap are still being examined.
- Developing an effective communication method among PC and FPGA, as the FPGA device constitutes the center module that controls the main hardware components of Omnitrap.
- FPGA exploration concerning methods of digital signal generation inside the FPGA.
- Ultimately, suggesting a final product design.

Throughout the realization of this thesis, extensive attempts took place concerning FPGA data transfer acceleration. In addition, a comparison took place between software generation of custom digital waveforms by PC against their direct digital synthesis by the FPGA device. Advantages and disadvantages of both techniques are examined and overall conclusions are drawn, which find application in every modern FPGA system, especially concerning applications in scientific instrumentation and equipment

3 System Development

The whole instrument functionality is controlled via PC application. Through software that is already developed, Omnitrap user is given a wide variety of possibilities, concerning experiment configuration, depending on the application needs. At the stage of instrument development, before commercial production and sale, low-level instrument control by PC is necessary so that validation testing is efficiently performed by specialized scientists and instrument functionalities are verified. In addition, the invention and trial of new mass spectrometry techniques, using the equipment available, is encouraged.

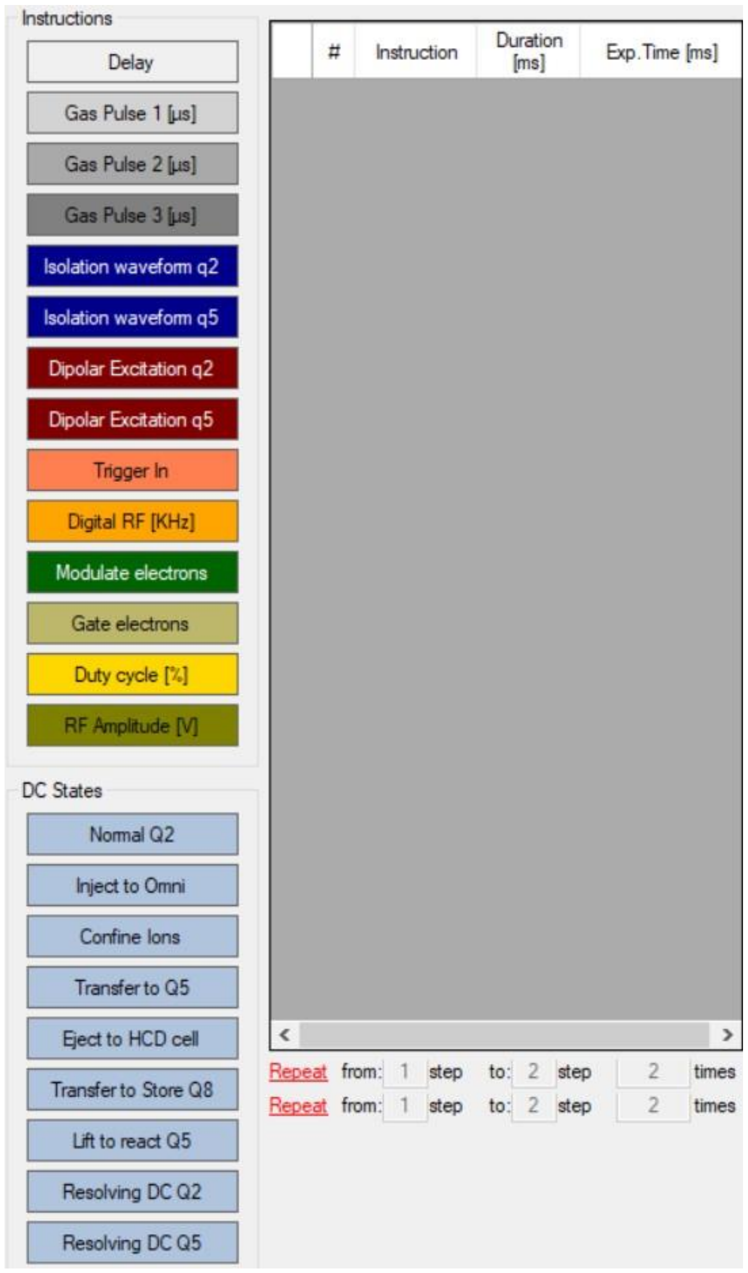


Figure 6: Omnitrap sequence of commands

The main idea for creating a customizable while comprehensive and efficient user interface for Omnitrap control has been based on the fact that the whole experimental process can be expressed by a sequence of commands. These commands are linked to pre-specified instrument actions and, therefore, a sequence of these commands represents a series of actions that the instrument is ordered to deliver. A different sequence of commands can be generated each time, depending on the application purposes. By splitting the experiment in smaller sub-parts more precise instrument control is accomplished, malfunctions are more easily detected and modifications in the experimental flow are simpler to implement.

As already mentioned, the processing core of Omnitrap instrument is an FPGA. The FPGA is intended to receive the sequence of commands from PC and execute them accordingly. The FPGA can interact with external instrument circuitry via Input / Output ports. Thus, every command corresponds to appropriately controlling the equivalent FPGA IO ports, each time, which are connected to a specific part of the instrument.

3.1 Instrument control

For the purpose of providing low-level instrument control and allow Omnitrap user to create a highly customizable experimental flow, following a different purpose each time, a bank of instrument actions is available through the User Interface. These actions are referred to as commands or instructions and each one of them represents a different experimental step. In Figure 6, a screenshot of the main user interface is presented. There are twenty four commands in total, with the first fourteen representing instrument functionalities and the remaining ten corresponding to ion transition actions, along the segmented ion trap, by the use of DC states manipulation. By clicking on a specific instruction, the corresponding command is added to the instruction list, where its arguments and real time experimental duration (in milliseconds) are shown. The arguments are also editable by the user in most of the commands. A brief description of the available instrument commands is given below:

- **Delay**: stalls the execution of the sequence by N milliseconds. As every action is executed by independent modules inside the FPGA, adding delays between commands is sometimes necessary, especially in cases where moving to the next action demands the completion of the previous command.
- **Gas Pulse 1 to 3:** Generates a helium gas pulse of configurable flux and duration inside the empty space of the ion trap. It is used for pressure management (mainly a gas pulse increases internal pressure) and speed manipulation of ions. There are three different gas pulse modules along the segmented ion trap. The main reason for this, is the fact that every gas pulse module needs a specific amount of time to be available for use again, while there are cases where multiple gas pulses are necessary to occur in close timing intervals.
- **Digital RF (KHz):** Sets the frequency of the rectangular RF signal that is being reproduced on the trap’s electrodes. Different frequency values allow trapping of ions with different mass-to-charge ratio (m/z).
- **RF Amplitude (V):** Sets the amplitude of the rectangular RF.
- **Duty Cycle [%]**: Sets the duty cycle of the rectangular RF. Modifications on duty cycle have a determinant impact on the trapping efficiency, depending on the m/z range of the ions of interest.
- **Gate/Modulate electrons:** Actions that relate to substance ionization process.
- **Trigger IN:** Stall the sequence until a trigger pulse is generated by Q Exactive. This signal plays an important role on synchronizing the two instruments together. A rising edge of the trigger signal, notifies that ions are located inside the HCD cell of Q Exactive, and that they can be received by Omnitrap, if the appropriate DC state is applied.
- **Dipolar Excitation (Q2 or Q5):** Initializes the application of dipolar excitation isolation waveform on the corresponding segment of Omnitrap. This signal is mainly a sine waveform with configurable amplitude and frequency. Depending on the frequency of the dipolar sine wave, ion masses that have matching resonance frequencies are excluded

from the trap. The term excitation derives from the fact that all masses, except only a small mass range, remain inside the trap.

- **Isolation Waveform (Q2 or Q5):** Initialized the application of isolation waveforms. These are broadband signals which are distinguished in two kinds, the Filtered-Noise-Field and the Sweep signal, each one with different properties. Depending on the frequency content of the isolation waveform, all ion masses except the ones that their resonance frequency is not included in the signal's frequency spectrum are excluded from the trap. The term isolation derives from the fact that only a small range of masses remains inside the trap, while the rest are excluded.
- **DC states:** These commands are used to transfer ions between trap segments. Ions are moving from high to low voltage. Therefore, by applying high voltage to source segment and low voltage to destination segment, ion transfer inside the trap can be accomplished.

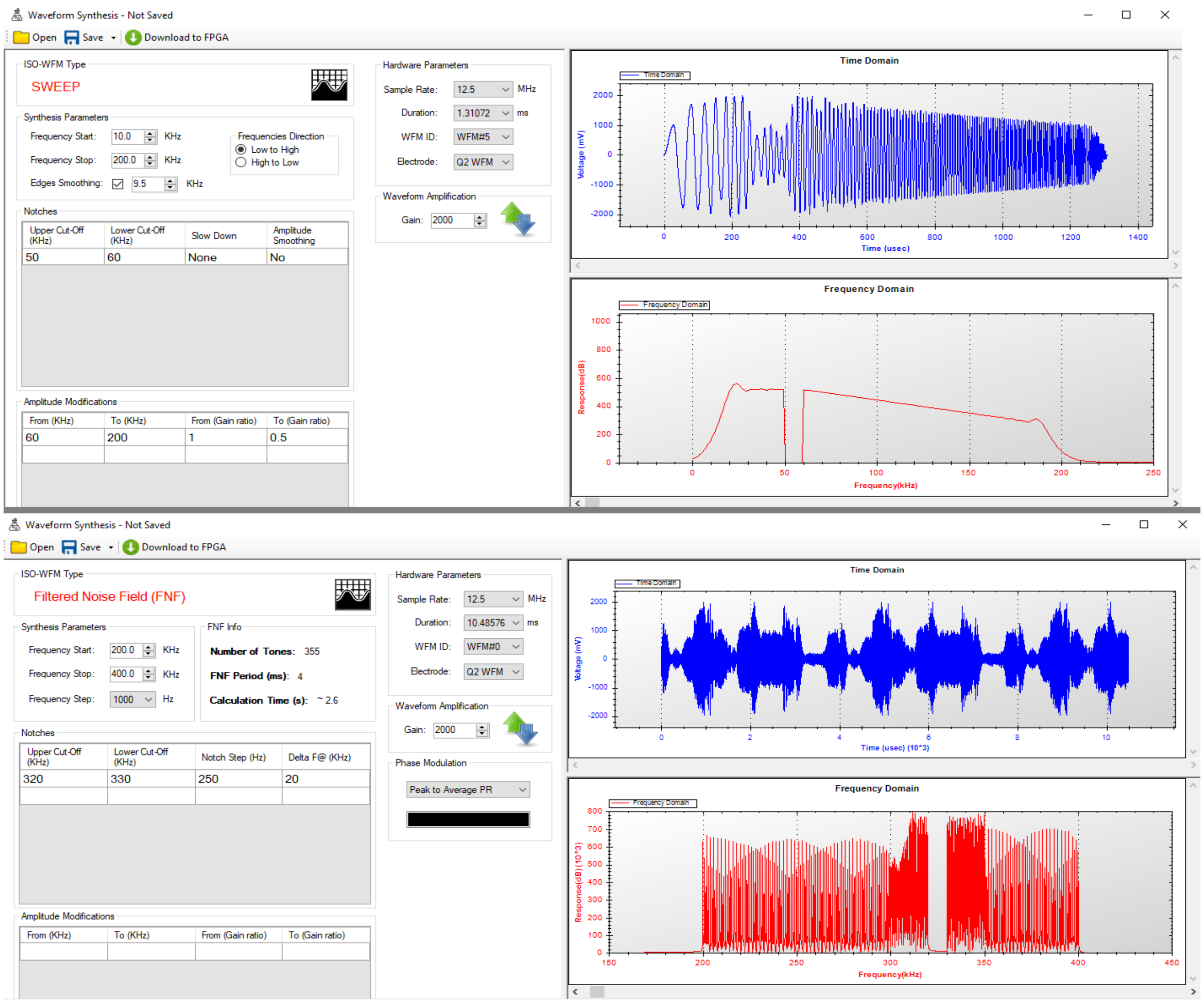
Sequence loop capabilities are also provided, in case an experimental process needs to be repetitively executed. An external and an internal loop and their repetition number are customizable through the user interface.

After sequence initialization, commands and their arguments are appropriately encoded to binary form and sent to FPGA, through communication methods that will be extensively examined in the upcoming chapters. These data are stored inside FPGA block RAM and a custom IP takes over their execution. Each command triggers an independent module which is responsible for the corresponding action. Multiple modules can run simultaneously without interfering each other. In this manner, high timing precision is accomplished, which is plays a major role issue in the success of the experiment. Finally, Custom digital logic is also responsible for controlling the loops inside the sequence.

3.2 User Interface for waveform generation

Purpose

In order to efficiently experiment with isolation waveforms, like Sweep and FNF, convenient software User Interface is necessary. It is essential that software allows user to highly customize excitation waveforms, while UI remains comprehensive and easy to use. In addition to that, code must be robust, efficient and as simple as possible so that, potential future modifications will be easier to implement. Under this perspective, two Windows Forms (see Figure 7) where created to allow Sweep and FNF waveform synthesis. Their implementation and functionality are analytically described below. All code is written in C# programming language and user interface is developed using the Visual Studio environment.



ISO-WFM Type: SWEEP

Synthesis Parameters

Frequency Start: 10.0 KHz

Frequency Stop: 200.0 KHz

Edges Smoothing: ✓ 9.5 KHz

Frequencies Direction: [✓ Low to High] High to Low

Notches

Upper Cut-Off (KHz)	Lower Cut-Off (KHz)	Slow Down	Amplitude Smoothing
50	60	None	No

Amplitude Modifications

From (KHz)	To (KHz)	From (Gain ratio)	To (Gain ratio)
60	200	1	0.5

Hardware Parameters

Sample Rate: 12.5 MHz

Duration: 1.31072 ms

WFM ID: WFM#5

Electrode: Q2 WFM

Waveform Amplification

Gain: 2000

ISO-WFM Type: Filtered Noise Field (FNF)

Synthesis Parameters

Frequency Start: 200.0 KHz

Frequency Stop: 400.0 KHz

Frequency Step: 1000 Hz

FNF Info

Number of Tones: 355

FNF Period (ms): 4

Calculation Time (s): ~2.6

Notches

Upper Cut-Off (KHz)	Lower Cut-Off (KHz)	Notch Step (Hz)	Delta F@ (KHz)
320	330	250	20

Amplitude Modifications

From (KHz)	To (KHz)	From (Gain ratio)	To (Gain ratio)

Hardware Parameters

Sample Rate: 12.5 MHz

Duration: 10.48576 ms

WFM ID: WFM#0

Electrode: Q2 WFM

Waveform Amplification

Gain: 2000

Phase Modulation

Peak to Average PR

Figure 7: Sweep and FNF user interface

3.2.1 Sweep

A sweep waveform is a harmonic function with linearly increasing frequency from an initial frequency to some final frequency (7). The foundation of the algorithm that generates such a broadband waveform is described in the following steps and visual representation of the results is also provided. The **blue** color represents the time domain while the **red** color represents the frequency domain:

- Calculation of the sweep signal in the time domain, according to the following mathematical expressions (where parameter n refers to discrete time):

$$\boldsymbol{Sweep(n) = \sin\,[Phase(n)],}$$

where *Phase(n)* is calculated by the formula:

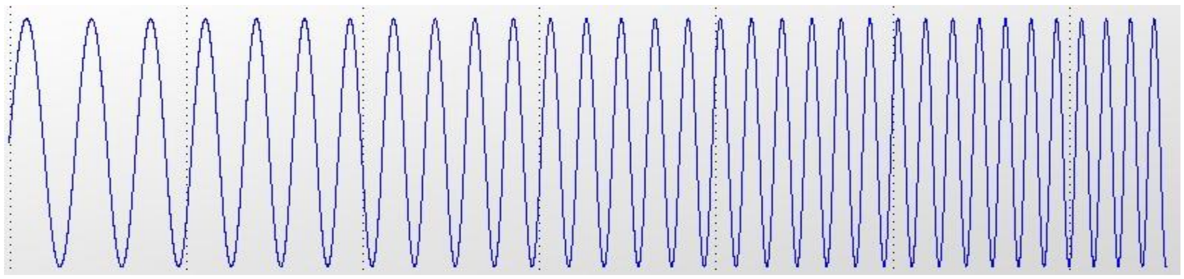
$$\boldsymbol{Phase(n) = Phase(n-1) + 2pi * Ts * frequency(n)},$$

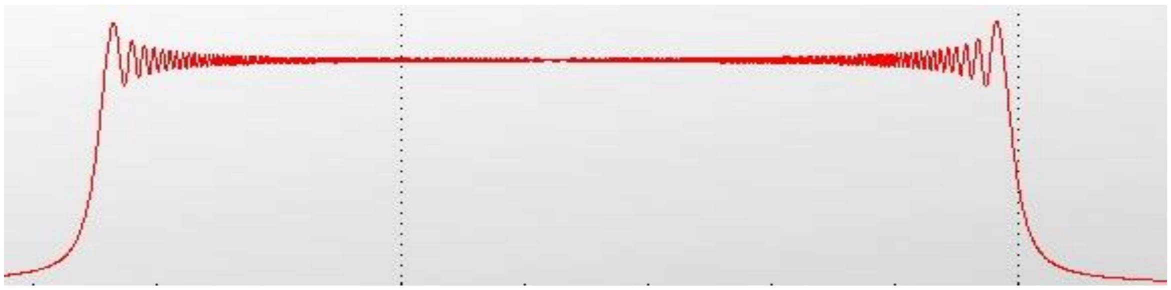
where *Ts* is the sampling period.

Frequency(n) is increasing linearly:

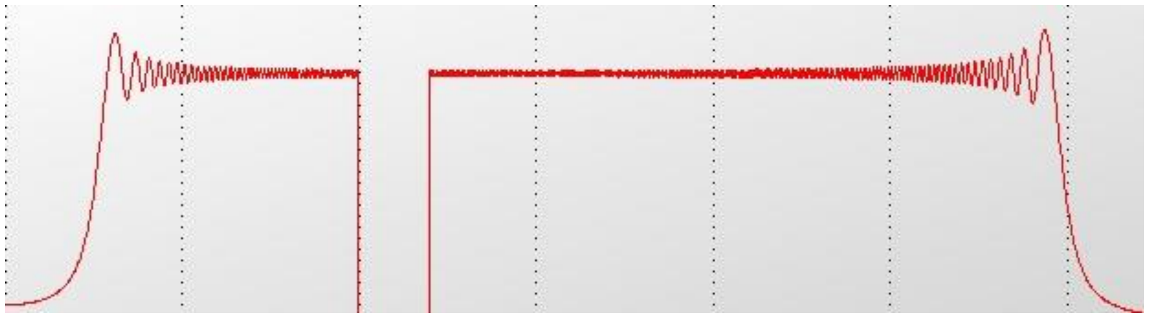
$$\boldsymbol{frequency(n) = frequency(n-1) + frequency\_step}.$$

At its initial state, the shape of a sweep waveform is shown below:

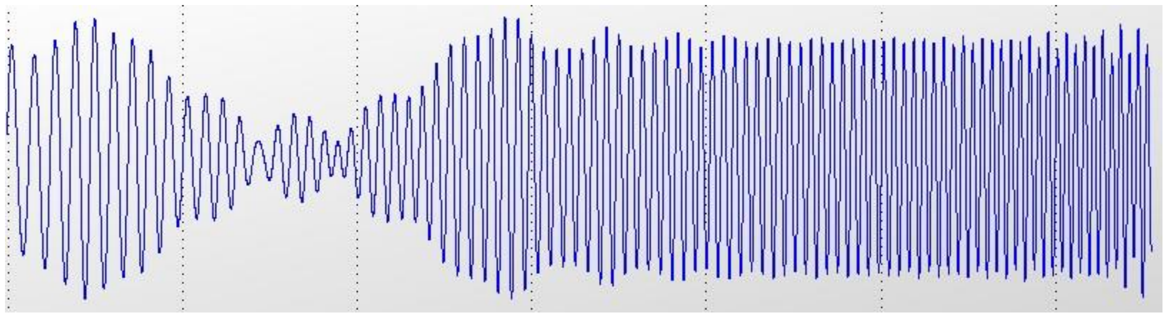




- Sweep function is then transformed into Fourier frequency domain, by means of Fast Fourier Transform. This way, amplitude and phase data are obtained for every discrete harmonic of the Fourier spectra. Objective of this procedure is to remove unwanted frequency components off the original sweep function and to obtain a new signal that does not contain unwanted components, but in other respect looks very similar to the original sweep. The main goal is to remove frequency components that cause resonance to the target isolation masses. Such frequency gaps are referred to the UI as frequency **notches**. In order to create a frequency notch that ranges from a frequency f_1 to a frequency f_2, the amplitude of the equivalent harmonics in the frequency domain, is set to zero. A notch in the frequency domain, should look like this:



- Then, the signal is transformed back to time domain, by means of Inverse Fourier Transform (IFFT) :

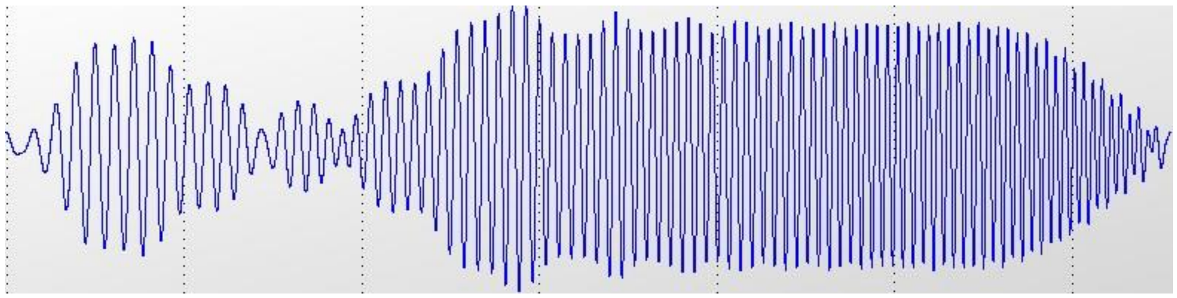


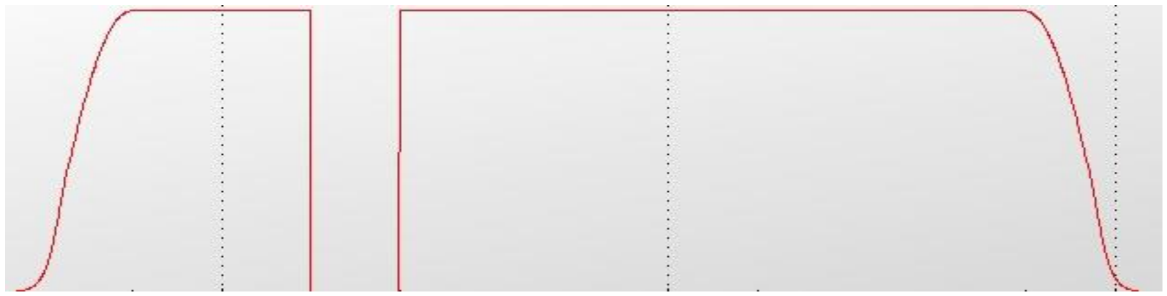
Extra parameterizations of the sweep waveform are available in UI for the purpose of experimenting and optimizing the ion isolation process:

- **Edge smoothing:**
  Some amplitude “spikes” near the starting and ending frequencies of the sweep waveform are obvious by examining the frequency domain plot. Such amplitude spikes are considered to potentially cause unwanted resonances. Thus, smoothing should be applied in the sweep edges by multiplying the sweep waveform with a reducing factor, which is given by the following mathematical expression:

$$factor(n) = \sin\left[\pm 1.57 * \left(\frac{n}{smoothing_{width}}\right)\right],$$

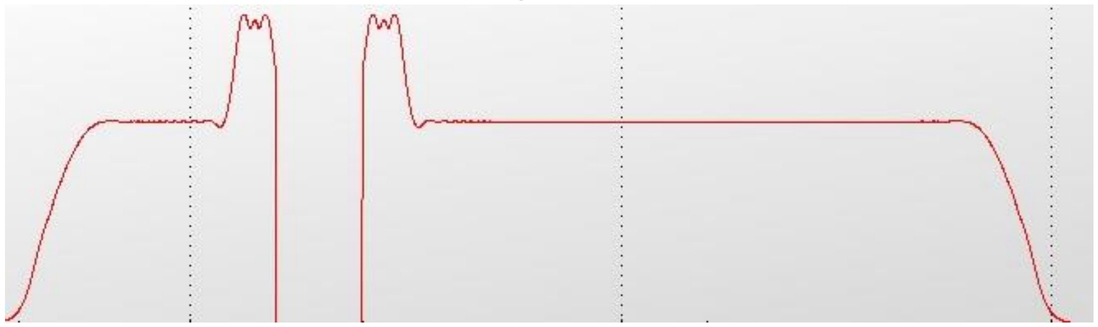
where positive sign is selected for smoothing the starting part of the sweep and negative sign is selected for the ending part. The edge smoothing results are shown below for both time and frequency domain:





Through UI, the length and rate of edge smoothing are both configurable.

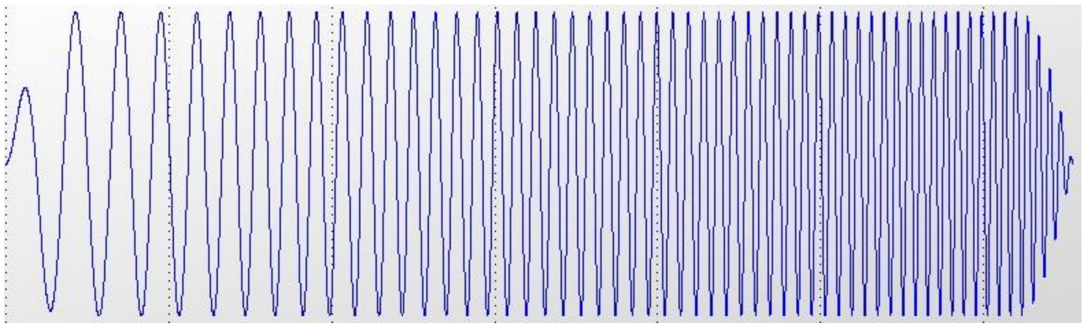
- **Slow down around notch areas:**
  Intense and fast changes in frequency content near notch areas are experimentally shown to have negative impact in the isolation process. Therefore, reduction of the frequency change step around notch areas needs to occur. As the sweep duration and frequency range is initially specified, slowing down the frequency sweep in some regions leads to inevitable speed up of frequency sweep in every other waveform part. Through the implemented UI, user can configure the range and rate of the notch slow down as well as whether it occurs before or after entering the notch (or both).

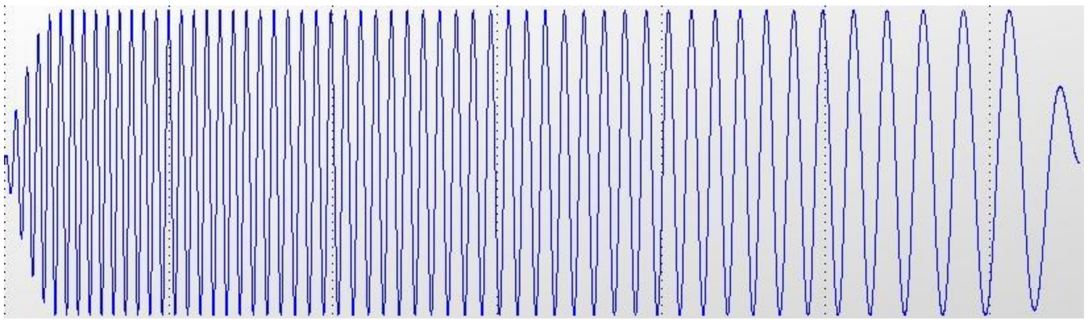


The slowing down, as it occurs by the frequency domain plot above, leads to an increase in the amplitude of the harmonics where the slowdown occurred, and a decrease in the amplitude of the rest harmonics.

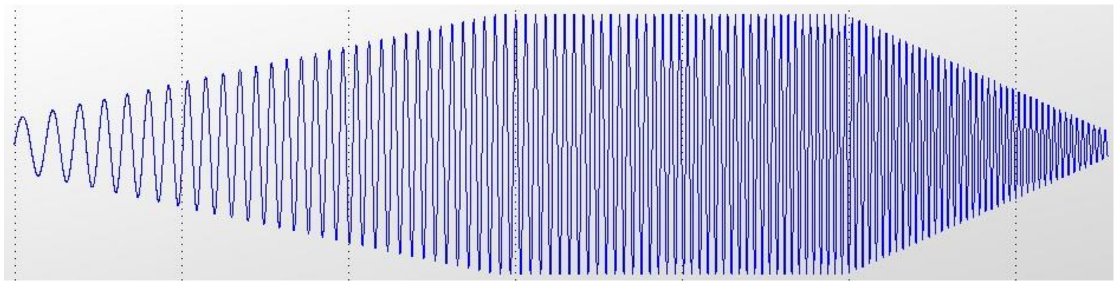
- **Frequency sweep to opposite direction:**
  In some occasions, scanning of the required frequency range needs to occur in declining

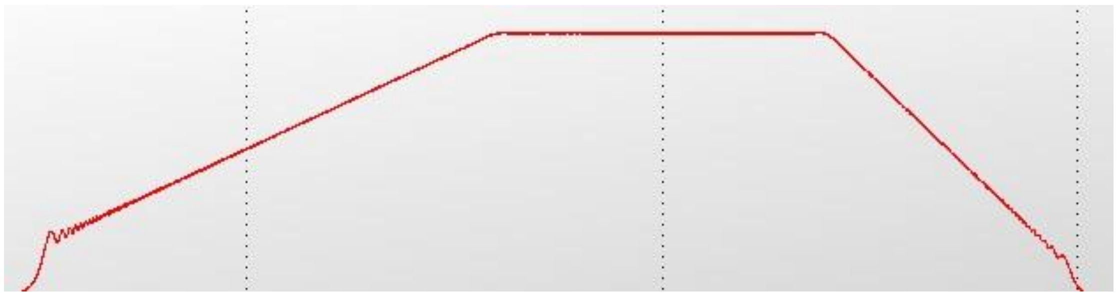
order, from high to low frequencies. Therefore, this capability is also implemented and is configurable by the user.





- **Amplitude modifications:**
  Through the implemented UI, user is given the possibility of generating his own custom amplitude modifications. Starting frequency, ending frequency, starting amplitude and ending amplitude can be configured. Applying an amplitude modification leads to a linear amplitude increase (if starting amplitude > ending amplitude) or decrease (if ending amplitude > starting amplitude) in the specified frequency interval. In the following example, two amplitude modifications are applied and their effect is visualized.





- **Other configurations:**
  Sweep waveform generation is also configurable regarding sampling rate, total duration and target quadrupole (Q2 or Q5). Two sampling rates are available, 12.5 and 25 MHz.

3.2.2 FNF

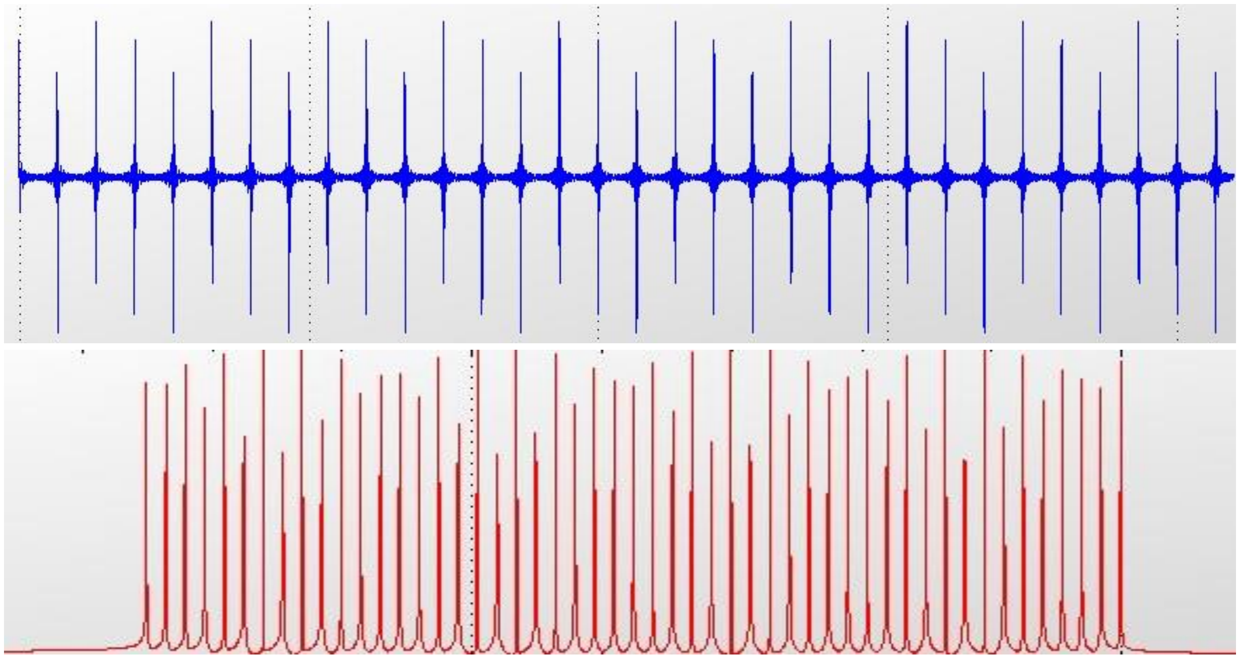
On the other hand, filtered noise field (FNF) waveforms constitute a completely different kind of broadband signals. FNF‘s basic feature is the fact that it reproduces all frequencies, inside a configurable interval, simultaneously, in comparison to sweep waveform which sweeps from smaller to higher frequencies over time (8). Consequently, its different properties make FNF a quite interesting field of study and experimentation. The steps for calculating an FNF waveform through software are the following:

- Configuring the number of frequency harmonics that the FNF waveform will incorporate. In this stage, harmonics inside desirable frequency notches are removed and thus the notch is created.
- Summing up all remaining harmonics in the time domain.

As it seems, the construction of an FNF waveform is fairly simple and straightforward. However, summing up a relatively large number of sinusoids is not computationally efficient and takes an infeasible amount of time. For this reason, a different approach to this computation is necessary. Finding the fundamental frequency of the FNF and subsequently summing up the frequency tones for only one waveform period is a quite efficient way of saving time and computational resources. The complete FNF waveform then can be calculated by concatenating multiple FNF period waveforms until desirable duration is achieved. Furthermore, the fact that there is no calculating dependence among FNF waveform points makes the calculation easy to be parallelized. By applying such contrivances, FNF calculation time is significantly reduced.

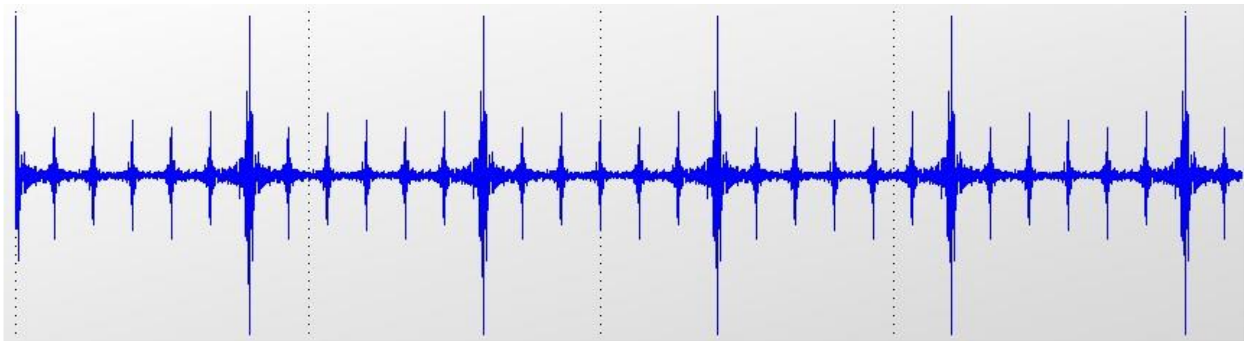
The most important FNF parameters that are configurable in the FNF user interface is the frequency content of the FNF (starting and ending frequency) as well as the frequency step. The division of frequency width with the frequency step provides the number of frequency tones which are incorporated by the FNF waveform. FNF duration, sample rate and target quadrupole are also configurable, similarly to sweep UI.

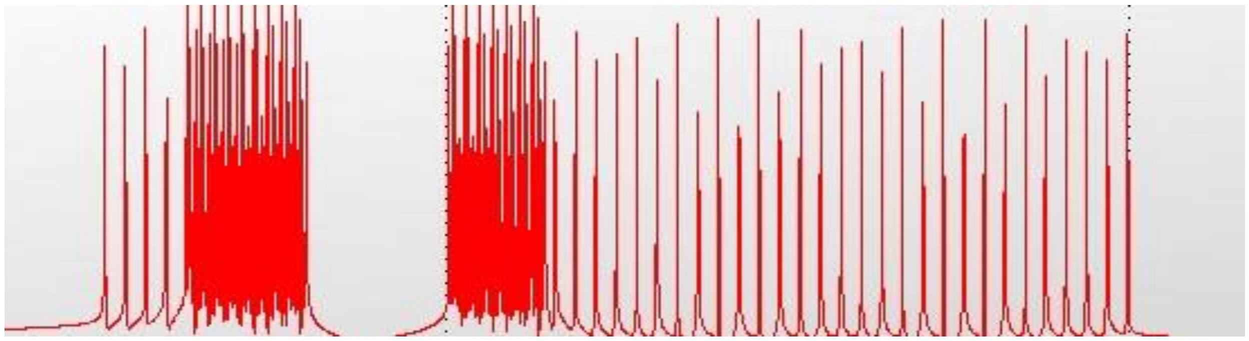
An example FNF waveform is shown below:



In comparison to sweep waveforms, this time the frequency domain is not continuous as it only includes some predefined frequency tones. As the configured frequency step decreases, the frequency spectrum of the calculated waveform gets denser and the frequency resolution is increased.

As already mentioned, the notch generation occurs by excluding from the summation, the frequency tones that belong to a desired notch. Apart from that, in most cases higher frequency resolution around notches is necessary, in order to allow more effective ion excitation in the corresponding frequencies. This feature is implemented in the FNF calculation code by modifying the frequency step around these areas. The range of effect as well as the frequency step before and after a notch area is configurable by the user. Visual representation of the incorporation of a notch in the FNF calculation is presented below:

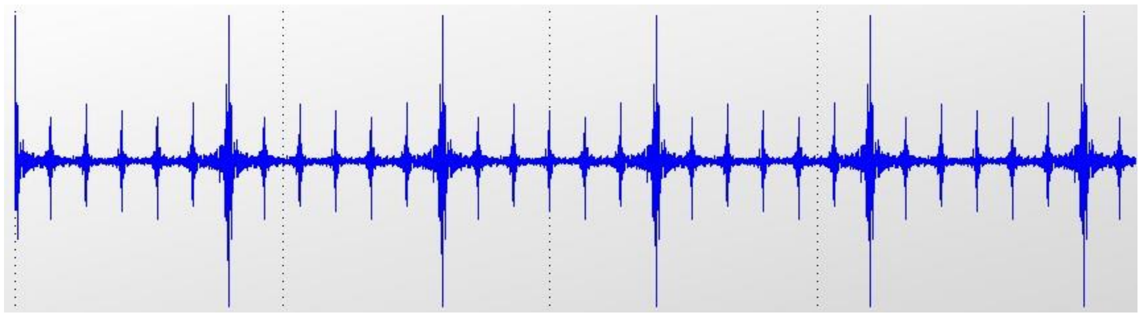


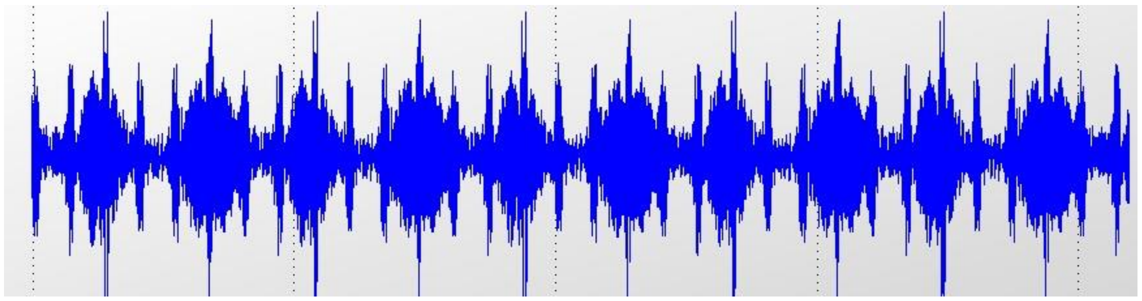


As it seems in the frequency domain above, the frequency resolution around notches is increased while the amplitude of all frequency tones inside the notch is zero. Other available FNF configurations are the following:

- **Phase modulation:**
  Adding frequency tones of equal phase results in uneven distribution of voltage levels in the waveform. In order to achieve even power distribution throughout the FNF signal, a phase modulation is necessary. There are unlimited choices regarding the mathematical expression that gives us an effective phase modulation, but the most widely used are the following:
  - *Newman Peak To Average Power Ratio:*
    $$phase(n) = mod(n^2 * \frac{pi}{number_{of_{tones}}} - 2pi,\ pi)$$
  - *Narahashi Peak To Average Power Ratio:*
    $$phase(n) = \mathrm{mod}(\mathrm{n} * (\mathrm{n} - 1) * \frac{pi}{number_{of_{tones}}} - 2pi,\ pi)$$

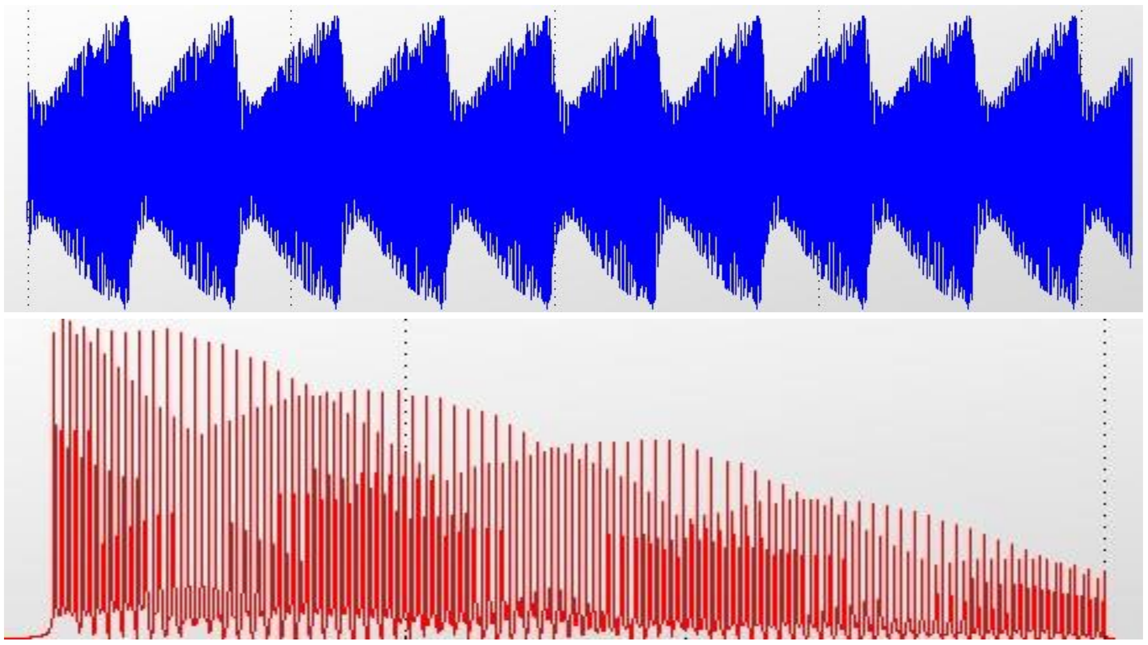
  Besides these two phase modulation techniques, user can specify its own phase modulation expression, which better suits the application. An example before and after phase modulation is presented below:





Both the above FNF waveforms have similar frequency contents but the second one makes a superior power distribution and is experimentally feasible.

- **Amplitude modifications:**
  Identical amplitude modification capabilities, as the sweep generation UI, are also available for FNF. A visual example is presented below, where the amplitude of frequency tones is linearly reduced from 100% to 20% of the initial value:



3.3 System level improvements

3.3.1 FPGA Communication

While setting up robust digital logic and circuitry inside the FPGA seems as the most essential aspect of a digital design, securing solid communication between software (PC) and hardware (FPGA) is equally important. Data corruption and communication loss with hardware can lead to a series of negative consequences, from misleading experimental results to even instrument damage. Such events, can cause huge delays to the instrument development process and decrease the overall instrument reliability. In addition, the fact that mass spectrometry analyzers, like Omnitrap, can be used for long periods of time (weeks or months) without shutting down, creates the extra requirement of constant data transaction verification and operation interrupt in case of communication failure. Therefore, the development of a stable communication channel with FPGA and techniques for real time evaluation of the data transfers is an urgent need.

For the current project, the data that need to be transmitted to the FPGA device are:

- **The sequence of commands**, which define Omnitrap's operational flow.
- **Samples of digital voltage waveforms**, which correspond to 12.5 MHz sample rate. These waveform points are generated and configured by developed PC user interface. These waveforms are intended to be used for ion excitation purposes.

Data that need to be received by the FPGA device are:

- **ADC read-back values**, which correspond to voltage and temperature measurements.

Nowadays, most products that require an interface to a host computer consider United Serial Bus (USB) as a primary option. USB is an industry standard that establishes specifications for cables and connectors, and protocols for connection, communication and power supply between computers and peripherals. It was firstly released in 1996 and its main purpose was to allow peripheral devices to connect with computers using a standard and common type of connectors. These connectors would make the use of peripherals more immediate while no host restart would be necessary for connecting or disconnecting a slave device.

Until today there have been released the following versions of the USB protocol, from oldest to latest:

- **USB 1.0** (*1.5 Mbit/s Low Speed, 12 Mbit/s Full Speed*)
- **USB 2.0** (*1.5 Mbit/s Low Speed, 12 Mbit/s Full Speed, 480 Mbit/s High Speed*)
- **USB 3.0** *(5 Gbit/s SuperSpeed)*
- **USB 3.1** *(10 Gbit/s SuperSpeed+)*
- **USB 3.2** *(20 Gbit/s SuperSpeed+)*
- **USB4** *(40 Gbit/s SuperSpeed+ and Thunderbolt 3)*

A USB interface can be added to an FPGA through USB protocol converter ICs. A USB protocol converter is an external device that incudes USB controller logic and can interact with FPGA, in a higher level than USB signaling, through a match simpler communication protocol. There are many choices available in the market regarding the methods that can be followed, with a variety in performance, ease of configuration, flexibility and time to market length. In the current thesis, some of the well-known FTDI USB bridge ICs are used for USB 2.0 and USB 3.0 connectivity in different approaches and the results are evaluated.

3.3.1.1 USB 2.0 to UART

The FPGA development board that is used for the current thesis incorporates the FT2232H chip by FTDI (Figure 8). This chip is a USB 2.0 to UART/FIFO bridge. It has two independent communication channels which can be configured in a variety of industry standard serial or parallel interfaces.

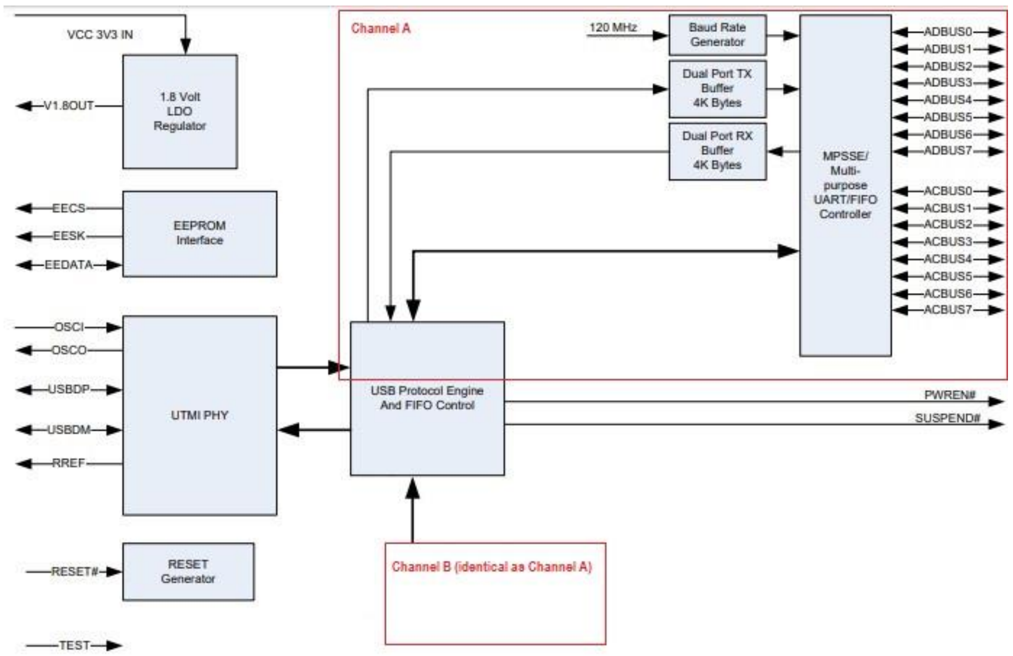


Figure 8: FTDI FT2232H block diagram

The chip can be easily configured through software. The configuration data are saved into the EEPROM interface and the configuration occurs on chip boot up. The functionality of each block is summarily described:

- Multi-Purpose UART/FIFO controller: There are two instances of them inside the chip, one for each channel. These control the UART or FIFO data.
- USB Protocol Engine and FIFO control: It controls and manages the interface between UTMI PHY and the FIFO memories. It is also responsible for power management and USB protocol specification
- Dual Port FIFO TX Buffer: Data coming from the Host PC are stored here until the channel master is available to receive them (maximum size: 4kBytes per channel)
- Dual Port FIFO RX Buffer: Data coming from the FPGA master are stored here until USB interface of PC is ready to receive them (maximum size: 4kBytes per channel)

- RESET Generator: Provides a reliable reset circuitry for the chip. External reset is also available if needed.
- Baud rate Generators: There are two of them, one for each channel. They are independent so that each channel can be configured in a different baud rate.
- UTMI PHY: Its name stands for Universal Transceiver Macrocell Interface. This block handles the USB SERDES (serialise – deserialise) circuitry, which is compatible to USB 2.0 and backwards compatible to all other USB versions. It also provides the clocks for the rest of the chip.

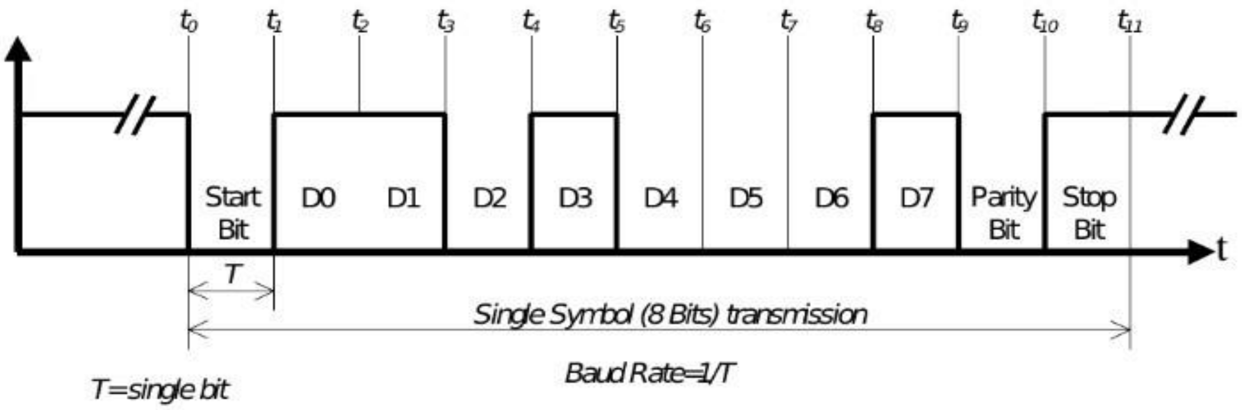
Among the different available modes of the chip, the **USB 2.0 to RS232 UART** is chosen for the first implementation of the current project. In this mode of operation the chip functions as a protocol converter, which converts USB to dual wire UART (Rx and Tx wires) interface, to be handled by the FPGA. The term RS232 refers to the configuration circuitry of the chip and the appropriate voltage levels to operate.

UART protocol

UART (Universal Asynchronous Receiver/Transmitter, Figure 9) is a serial asynchronous communication protocol and a circuit that allows data transfer between two (or more) devices that can be synchronous to different clocks. UART is widely used, due to its simplicity of implementation, practical ease of use and its general application capabilities.

UART’s asynchronous communication takes place via a wired single bit connection, between the receiver (RxD), which drives the signal, and the transmitter (TxD) which samples and examines it (9). The data to be sent are usually whole bytes (8 bits) and are transmitted serially, bit by bit, from the least significant to the most significant bit. As there is no clock to synchronize the communication between the two devices, in asynchronous communication the data is preceded by a recognizable **start bit**. This signifies the receiver for the beginning of the communication. In a similar way, the transmission of a byte and the end of the communication is signified by the **stop bit**.

If needed, the UART bus can be customized so that it also incorporates a **parity bit**. The parity bit precedes the stop bit and includes information relative to the number of logic ones in the byte word that was sent. It is set to logic one by the transmitter, if the number of logic ones in the transmitting byte is even, and to logic zero if the number is odd.



t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11}
Start Bit
D0
D1
D2
D3
D4
D5
D6
D7
Parity Bit
Stop Bit
t
T
Single Symbol (8 Bits) transmission
T=single bit
Baud Rate=1/T

Figure 9: UART protocol

In order to ensure that no data is lost or multiplied in the UART protocol, the transmitter and the receiver have to agree in advance regarding the sampling rate of the communication channel. This sampling rate is expressed in bauds, where each baud is equal to 1 bit per second. The configuration of baud rate is not part of the protocol and is done at a higher level. The standard and most commonly used baud rates are the following: 110, 150, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800 and 921600.

For a bidirectional communication system between two devices, which use UART protocol, two wires are needed, one for each data direction.

For the FPGA to interact with the UART bus, two I/O pins are necessary, a receiver (Rx) and a transmitter (Tx). Rx pin is used for data reception from USB to FPGA, while Tx pin is used for data transmission from FPGA to USB. Internal FPGA logic is also necessary. For this purpose, **AXI UARTlite** is used. It is a Xilinx Intellectual Property core which is capable of interacting with UART interfaces while communicating with FPGA's Programmable Logic (PL) via an **AXI4 Lite** slave interface.

FPGA Design

AXI BUS

AXI, which stands for Advanced eXtensible Interface, is an interface protocol defined by ARM as part of the AMBA standard (10). AMBA is an open standard for SoC (System-on-Chip) design created by ARM to allow for high performance, modular, and reusable designs that are reliable while minimizing both power and silicon. In the current project the AXI4 version of AXI is used (which responds to AMBA 4.0 version). There are three types of AXI4-interfaces:

- **AXI4 (Full AXI4)**: It is used for memory- mapped data transfer. In this communication protocol, for every data transfer, an address is required, which is followed by the transaction data. Data size can vary from 1 to 256 words while data word size ranges from 32 to 128 bits. The AXI4 Full can interconnect multiple master to multiple slaves that correspond to the same bus protocol.

- **AXI4-Lite**: This protocol is a simplified version of AXI4 Full. It is also a memory-mapped protocol but it does not allow transactions of multiple data words (burst transactions). Thus, for each data word transfer, a data address is required. The data word size ranges from 32 to 128 bits. The implementation of this bus needs less resources, which is its major advantage in comparison to AXI4 Full

- **AXI4-Steam**: Is not a memory-mapped protocol. It is used for very fast data transfer from a single master to a single slave. It supports burst transactions of unlimited size and it is the fastest of the three AXI4 protocols.

IP cores

AXI UARTlite

This Xilinx Intellectual Property core is shown in Figure 10. It executes serial to parallel data conversion for data that come from UART interfaces and parallel to serial data conversion for data that come from AXI4 Lite interfaces. It can be configured to manage data words of 5, 6, 7 or 8 bits. Furthermore, a parity bit is available if needed. It incorporates a transmit and a receive FIFO, each one of 16 data words depth. In case receive FIFO is full, data from UART interface are not received. Respectively, if transmit FIFO is full, AXI data are rejected and an AXI bus error is generated so that it notifies PL that the transaction failed. Finally the core can accept and assert interrupt signals (11).

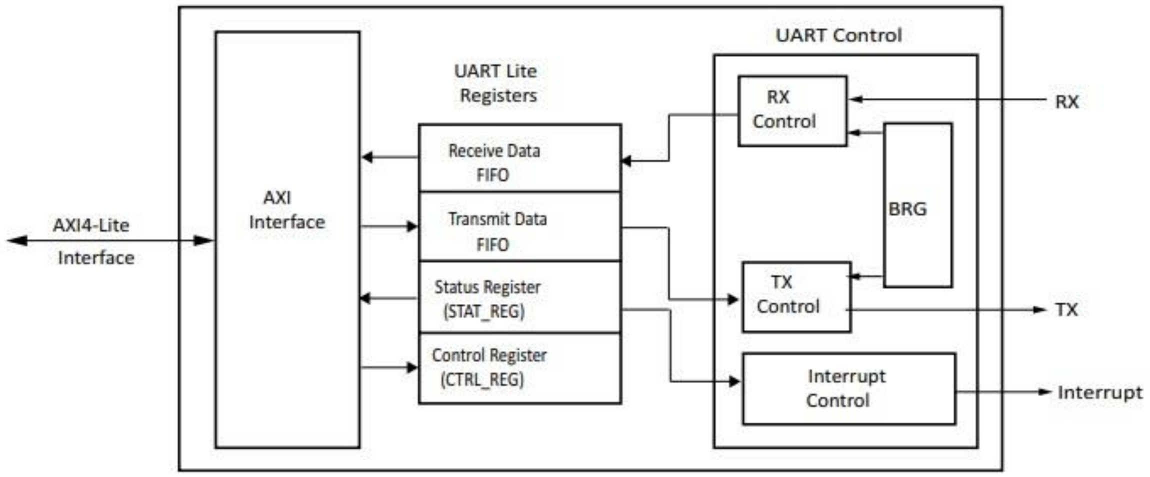


Figure 10: AXI UartLite block diagram

Two registers are available to give information about the status of the IP and offer IP control capabilities. More specifically the status register provides the status of the receive and transmit data

FIFOs and asserts error bits if an error occurs. On the other side, control register contains the enable interrupt bit and the reset bit for the receive and transmit FIFOs.
Using the appropriate input clock, the IP can be configured to support all standard UART protocol baud rates from 110 to 921600. Block diagram of UARTlite IP core is shown above.

AXI Interconnect

As mentioned above, AXI Interconnect is a Xilinx Intellectual Property core which is used for connecting one or multiple master to one or multiple slave interfaces (12). The block diagram of the IP core is shown in Figure 11.

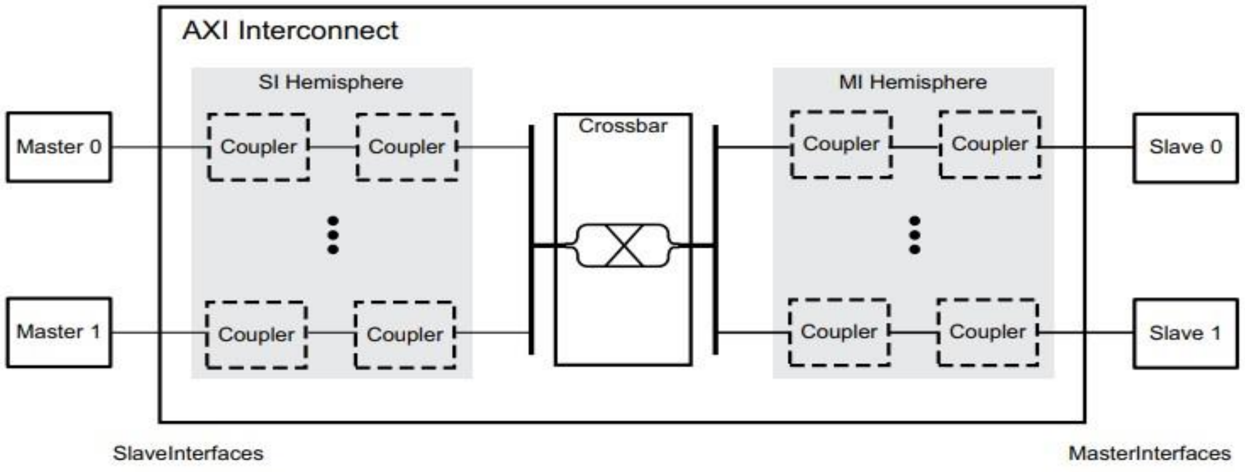


Figure 11: AXI Interconnect block diagram

The core constitutes of three sub-cores, the slave interface (SI), the master interface (MI) and the crossbar IP. AXI4 Master Interfaces are connected to SI while AXI4 Slave interfaces are connected to MI. Through SI masters can issue transaction read or write commands and wait for the related slaves to respond.

The AXI Interconnect implementation is automatically configured based on the number and kind of master and slave interfaces that are connected to it. The implementation kinds are the described below:

- **Pass through (1 to 1):**
  In case only one master and one slave of identical protocol are connected, the AXI Interconnect IP does not need to implement protocol conversion or pipelining functionalities. Subsequently, the two interfaces are connected directly and the core does not occupy hardly any resources.
- **Conversion only (1 to 1):**
  In case a data width conversion, a clock rate conversion, an AXI4-Lite slave adaption or

pipelining needs to occur, the implementation only omits the circuitry that is responsible for arbitration, decoding and command fetching.

- **N-to-1 Interconnect:**
  In case multiple masters are connected to a unique slave, arbitrary circuitry is implemented so that each time only one master has access to that slave. In addition, protocol conversion circuitry is included if needed.
- **1-to-N Interconnect:**
  In case one master has access to multiple slaves there is no need for arbitrary logic. However, circuitry which decodes and issues the commands each time to the corresponding slave, is necessary.
- **N-to-M Interconnect (Crossbar Mode):**
  In case multiple masters are connected to multiple slaves, the AXI Interconnect implements intermediate logic manages the transaction commands, even multiple commands at a time.
- **N-to-M Interconnect (Shared Access Mode):**
  Same as crossbar mode, except in this mode commands are issued one at a time. Subsequently, this implementation consumes fewer resources but the AXI transactions are slower.

Microblaze

Microblaze is the processing core that Xilinx offers to be used for 7-Series FPGA architectures. It is a soft core which means that it is implemented by the FPGA LUT (Logic Unit Table) cell resources (13). It incorporates 32bit RISC architecture with pipelining capabilities and it is highly customizable. Furthermore, it can be customized so that it responds to different kinds of events like resets, interrupts and exceptions.

Microblaze communicates with others IP cores through the AXI4 bus. For this purpose, AXI Interconnect IP core is a precious tool as it can connect multiple masters (one of them is obviously Microblaze) with multiple slaves that correspond to different kinds of AXI4 protocols.

Memory Interface Generator

The Memory Interface Generator constitutes a Xilinx Intellectual Property core. It is a combined pre-engineered controller for interfacing 7 series FPGA user designs and AXI4 slave interfaces, to DDR3 and DDR2 SDRAM devices (14). This core is used in our design as a memory controller which allows robust communication from our custom FPGA logic to the DDR3 memory, through AXI4 protocol bus. It also includes internal logic for monitoring the DDR3 temperature and keeping its functionality within safety limits.

Block Design

The purpose of this FPGA design is to set up a communication channel between PC software and FPGA hardware which will serve the waveform data download and some extra functionality like setting up ADC values and reading back DAC values. Creating the FPGA design for this purpose is fairly straightforward and does not demand custom IP generation. The Xilinx IP cores mentioned above are enough for our current demands. The FPGA design is shown in Figure 12.

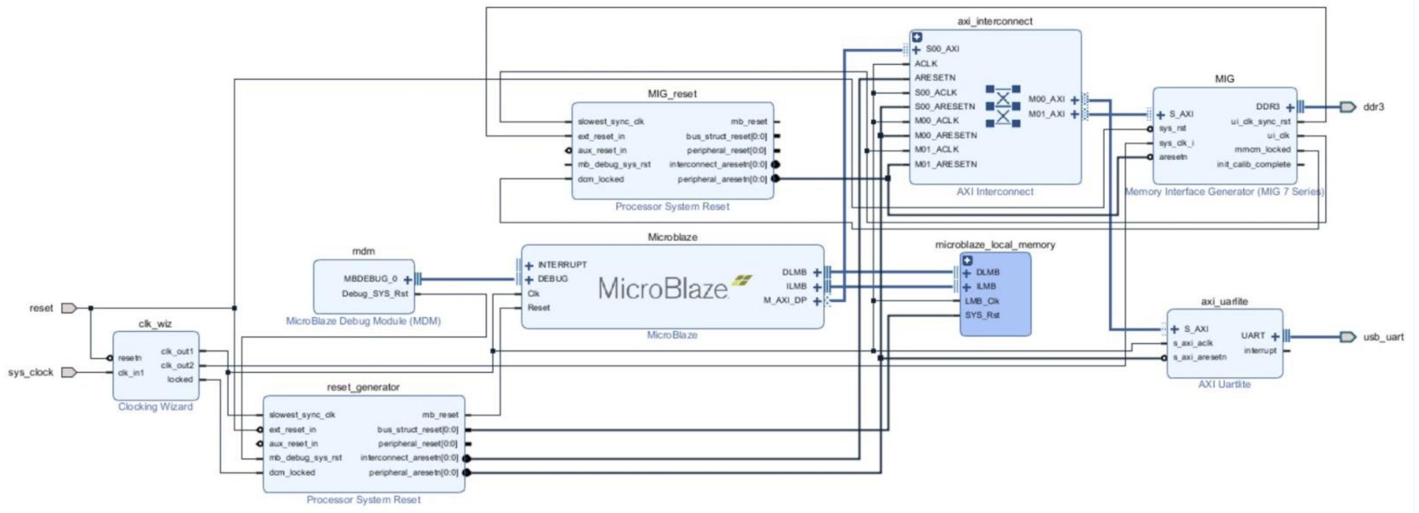


Figure 12: Implementation of PC-FPGA communication based on UART protocol

In this block design, a clock wizard is responsible for creating two clocks, clk_out1 of 75 MHz and the clk_out2 of 200 MHz, from the input system clock of 100 MHz. The clock of 75 MHz is the input clock for AXI UARTlite IP. Such a clock frequency is selected as it allows configuration of the UART channel in maximum baud rate of 921600. Microblaze is also clocked at 75MHz.

On the other hand, the 200MHz clock rate is used as the input clock of MIG controller. The MIG controller is responsible for executing every DDR3 transaction that an AXI4 master requests. Microblaze soft-core possesses the only AXI4 master interface in the design so it is the only IP that can initiate DDR3 transactions.

For the purpose of supporting multiple functionalities in the same data channel, it is obvious that some common rules between software and hardware need to be established. In our case, a convenient way of achieving this is by setting up a group of identifier bytes. The idea is that the PC through software will transmit bytes from an “identifier bytes bank” and the hardware will reply by executing the corresponding action every time. The identifier bytes that are used and a brief description of the action they are linked to are presented below:

- **CONNECT** (0xFD): Opens a connection port between the software (PC) and the hardware (FPGA).

- **DISCONNECT** (0xFE): Closes an already opened connection port between the PC and the FPGA.
- **EXECUTION_START** (0x1D): Starts the execution of the commands sequence.
- **EXECUTION_STOP** (0x1E): Stops the execution of the commands sequence.
- **READ_SEQUENCE_STATUS** (0x3A or 0xF6): Returns the status of the command sequence (Idle or Running)
- **READ_LOOP_COUNTER** (0x3B): Returns the number of times that the command sequence has been executed
- **DOWNLOAD_SEQ** (0x1C): Initializes the command sequence download. This means that the next bytes that are sent to the FPGA represent command sequence data.
- **READ_RTD_X** (0xF7 or 0x2F or 0xF9 or 0xFA): Returns the measurement of resistance thermometer X after averaging it in a window of size 16. Four resistance thermometers are available so integer X can range from values 1 to 4.
- **SET_RF_I** (0xF8): Set the current (I) value of the RF (radio frequency) Power Supply Unit.
- **READ_RF_V** (0x3D): Returns the voltage value of the RF Power Supply Unit.
- **READ_RF_I** (0x2E): Returns the current value of the RF Power Supply Unit.
- **DOWNLOAD_WFM** (0xFC): Initializes a waveform data download process. This means that the next bytes which are sent to the FPGA represent waveform data.
- **UPLOAD_WFM** (0xFB): Read the waveform data saved to DDR3 (mainly used for evaluation purposes)

This set of predefined identifier bytes are considered known for both the transmitter (PC) and the receiver (FPGA), while Microblaze is responsible for executing each functionality. Using Xilinx Software Development Kit (SDK) a loop process is written in C code, for the Microblaze to execute by the time the FPGA is powered on. This process is a polling function which waits for data to be available in the UART channel. When the first byte is available, Microblaze receives it and acts accordingly. By the time an action is complete, Microblaze is polling until the next request.

This process can be also implemented using interrupts. Each time data are available in UART interface, an interrupt occurs. Microblaze accepts the interrupt and realizes the equivalent operation, according to the identifier byte that is received. In this manner, Microblaze is not polling when UART data are missing and can be used for further operational load. As in our design, Microblaze functionality is limited to UART demands serve, interrupt logic is excluded so that minimum FPGA resources are occupied.

For the purpose of executing the actions mentioned above, like triggering execution start or saving data to DDR3, Microblaze (PS) (programmable Software - Microblaze) needs to communicate with PL (Programmable Logic - IPs). This communication occurs through AXI4 bus. In order for Microblaze master to set reading or writing requests to a target slave, functions Xil_In and Xil_Out are used accordingly. These functions are already implemented in the board support package, which can be automatically generated by Vivado SDK tool for every FPGA design. They can perform input or output operations of 8, 16, 32 and 64 bits for AXI4 memory mapped registers. More specifically, Xil_In function takes, as its unique parameter, the AXI4 slave address and returns the corresponding data word. On the other hand, Xil_Out takes as variables the AXI4 target slave address and the data word to be written.

3.3.1.2 USB 2.0 to FIFO245 Synchronous

For the purpose of creating a more efficient, in terms of speed, design and achieving faster download speeds, especially concerning waveform data download to FPGA, another communication approach is tested. The same FT2232H chip is used, but this time it is configured in a different mode of operation. In this configuration, data that come from (or are intended to be sent to) the USB interface, are stored internally in the FT2232H chip's FIFO memory blocks (15). Two separate FIFO blocks are necessary for this functionality, a receiving and a transmitting one. Data from USB interface (to FPGA) are stored in the receiving FIFO while data from the FPGA master (to PC) are stored in the transmitting FIFO block of the chip.

Furthermore, for this mode the chip provides an external 60MHz clock to be used by the FPGA master. The channel's control logic has to be synchronous to this clock domain in order to ensure robust communication. The pins used in a 245 synchronous FIFO mode are the following (16):

- **DATA[7…0]**: These are the bidirectional data pins. There used for both input (from PC host to FPGA master) and output (from FPGA master to PC host) data words.

- **RXF#**: Output pin, active low. When asserted, it notifies the FPGA master that there is data available in the receiving FIFO, coming from the PC host.

- **TXE#**: Output pin, active low. When asserted, it notifies the FPGA master that the transmitting FIFO is available to be written by the FPGA master, so that a transaction from FPGA to PC host occurs.

- **RD#**: Input pin, active low. It should be asserted when FPGA master is about to start a read operation.

- **WR#**: Input pin, active low. It should be asserted when FPGA master is about to start a write operation.

- **CLKOUT**: Output pin. It is the 60MHz chip output clock.

- **OE#**: Input pin, active low. When asserted it causes a bus turnaround, so that DATA[7…0] are driven by the FT2232H chip. It should be driven low at least one clock cycle before driving RD# low.

For a read transaction to occur, the RXF# signal has to be asserted. This assertion notifies the FPGA master that data are available in the receiving FIFO. For the purpose of receiving the data, OE# must be asserted by the FPGA master before the assertion of RD#. Thus, two clock cycles are at least necessary for a reading transaction to begin. After these events, ft2232h chip drives one data word on each clock cycle that RD# is set to logic zero. FPGA master can either burst read the available data or receive them in parts by repetitively asserting and deasserting the RD# pin. A read transaction is visualized in Figure 13.

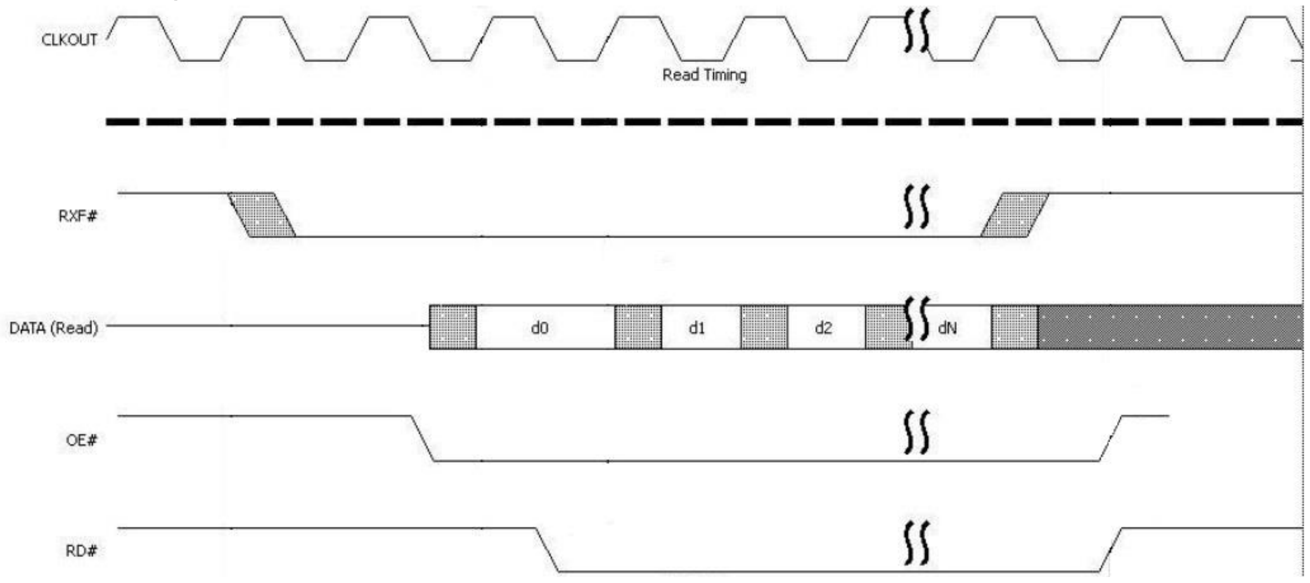


Figure 13: FT2232H FIFO 245 read transaction

For a write transaction to occur, TXE# signal has to be asserted. This signal notifies the FPGA master that the transmitting FIFO is available to receive data. By the time WR# is asserted, channel master should drive the data bus with the data to be sent. A write transaction ends when TXE# is deasserted (usually when transmitting FIFO is full) or when master sets WR# to logic one. A write transaction is visualized in Figure 14.

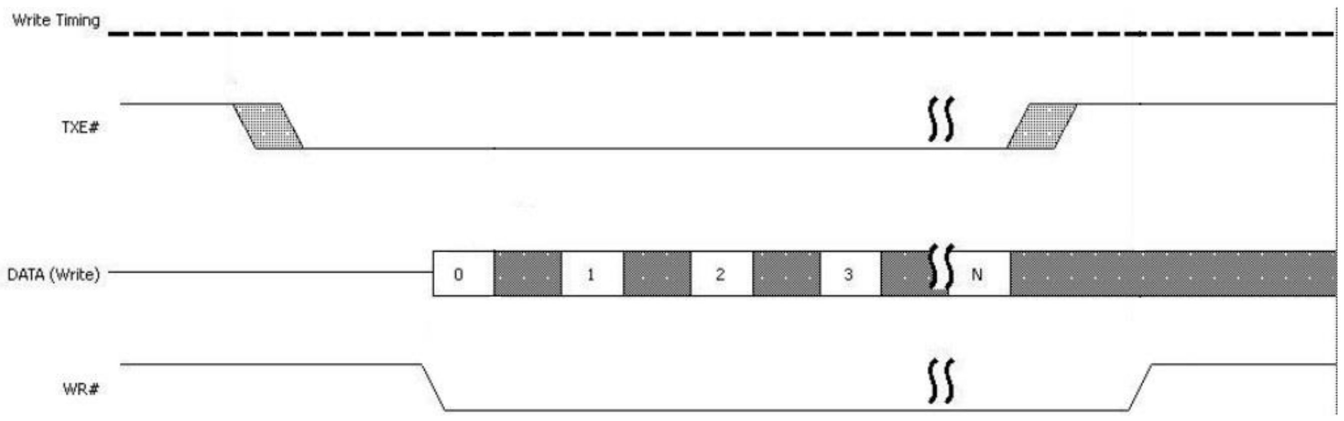


Figure 14: FT2232H FIFO 245 write transaction

<u>FPGA design</u>

The incorporation of the synchronous FIFO 245 communication mode in our FPGA design requires the creation of a custom IP module which will be responsible for the control of FT2232H signals and direction of receiving data, each time in the target AXI4 slave interface. This mode is intended to be used for waveform download to FPGA, thus only reading transaction interface, from PC to FPGA is implemented and tested here. Sequence data and instrument read-backs transactions will still occur through the already implemented UART channel. As a result, a new USB cable and extra FPGA resources are needed for this implementation.

The custom module that is designed for the above purpose should be able to issue writing transfers to DDR3. Towards that end, our IP is occupied with an AXI4 Lite master interface. In Vivado environment, AXI bus default logic can be automatically generated. Further modifications are required so that our custom logic is able to use the AXI interface accordingly. In our case, the AXI signals that are controlled by custom IP are the AXI target write address (**M_AXI_AWADDR**), write data (**M_AXI_WDATA**) and transaction start trigger pulse (**init_txn_pulse**). Signal **init_txn_pulse** is a single clock pulse which triggers the start of an AXI4 lite transaction of data **M_AXI_WDATA** to AXI bus address **M_AXI_WADDR**.

A block diagram of the Finite State Machine that serves the above purposes is presented in Figure 15.

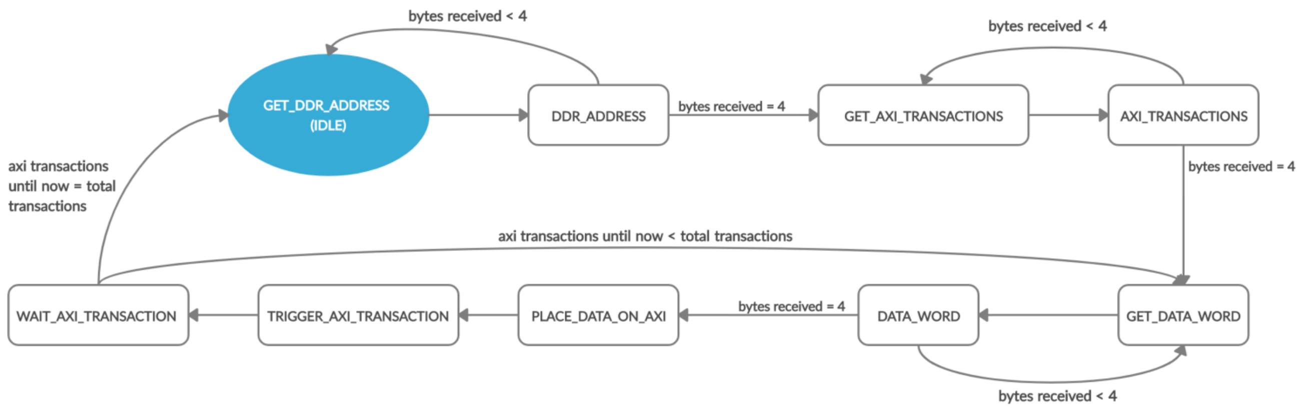


Figure 15: FIFO 245 Finite State Machine

Finite State Machine description in words:

The above FSM receives a whole waveform data packet from the USB 2.0 to FIFO interface of FT2232H chip and saves the received data to DDR3 memory. The size of the waveform as well as the writing address are both configurable.

To begin with, while data from the USB 2.0 interface are not available, FSM remains in IDLE condition. By the time data are available in the bus, FSM moves on to the necessary actions to receive them. The first bytes, by default, represent the DDR3 write address. Every AXI4 address is of size 32 (bits), so 4 bytes are necessary for its configuration. The write address can range from 0x80000000 (DDR3 AXI address) to 0x8FFFFFFF, as our DDR3 memory has 2Gbit of usable space and each AXI address is mapped to one single byte. After DDR3 write address is configured, the next 4 bytes should represent the number of AXI transactions. Since each waveform point is represented by 2 bytes (16 bits) and each AXI transaction has word length of 4 bytes (32 bits), the AXI transactions number should be equal to waveform points / 2, and therefore represent the waveform's total size. Subsequently, all following incoming bytes constitute waveform data, until waveform size is reached. For every 4 bytes, one AXI transaction is issued and no more data bytes are received before this AXI transaction is completed.

Finally, when waveform size is reached, waveform download is complete and FSM returns to its initial IDLE state, waiting for the next data byte to be available in the bus.

Clock domain crossing (CDC)

In the above circuitry, the input signal **single_write_done** notifies the FSM that the 32 bit write transaction which was last issued, has been completed, thus next data bytes can be received. This signal is controlled by the AXI4 bus default logic and it asserts for a unique clock cycle, each time a (reading or writing) transaction comes to an end. A single rising edge pulse, coming from its clock domain, would last for a 10ns timing width, as the AXI4 bus clock frequency is synchronous to 100MHz.

However, in our design this pulse has to be sampled by the 60 MHz clock of FT2232H chip, which FSM is synchronous to. It is self-evident that directly sampling a pulse of 10ns with a sample rate of 16.6ns (60 MHz) is insufficient. Indeed, after testing this implementation, the **single_write_done** pulse was detected by our 60MHz clock in a success rate of about 60%, which coincides to the difference between the two clock periods.

The above issue constitutes a cross clocking domain problem. More specifically, a clock domain is a part of a design that has a clock that operates asynchronous to another clock in the design. For example, in this design we have two clock domains, one at 100 MHz and one at 60 MHz. Furthermore, a clock domain crossing signal is a signal that is sampled by a register in another clock domain. Therefore, single_write_done signal is a clock domain crossing signal. Directly sampling signals of a clock domain synchronously to another clock, which operates asynchronous or has a variable phase relationship with the prior clock, has a high probability of failure due to flip-flop meta-stability. Flip flop meta-stability derives from flip-flop's setup and hold time. These timing properties represent the time in which the data input is not legally permitted to change before and after a sampling clock edge, accordingly. An example of a D flip-flop getting in meta-stability is shown in Figure 16.

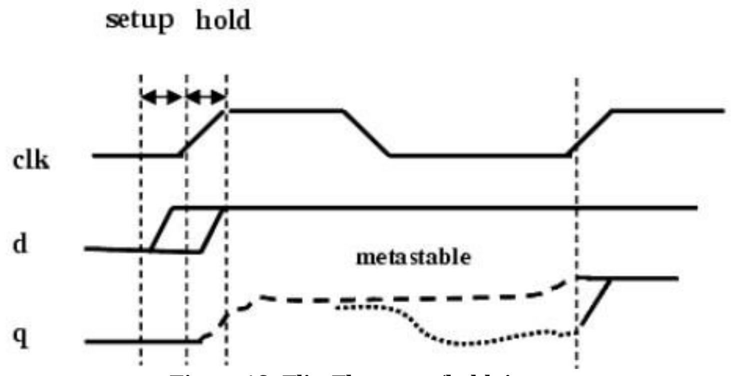


Figure 16: Flip-Flop setup/hold time

In case input d of the D flip-flop violates the setup and hold time, the output q of the flip-flop keeps oscillating for an indefinite amount of time. There is a possibility that this unstable value may not converge to a stable value before the next sampling clock edge arrives.

Entirely avoiding meta-stability issues in a multi-clock design is inevitable. However, there are certain design techniques that can help to reduce the probability of their appearance asymptotically to zero. In our design, the main method used is the 2-FF Synchronizer (17). The equivalent circuit is shown in Figure 17.

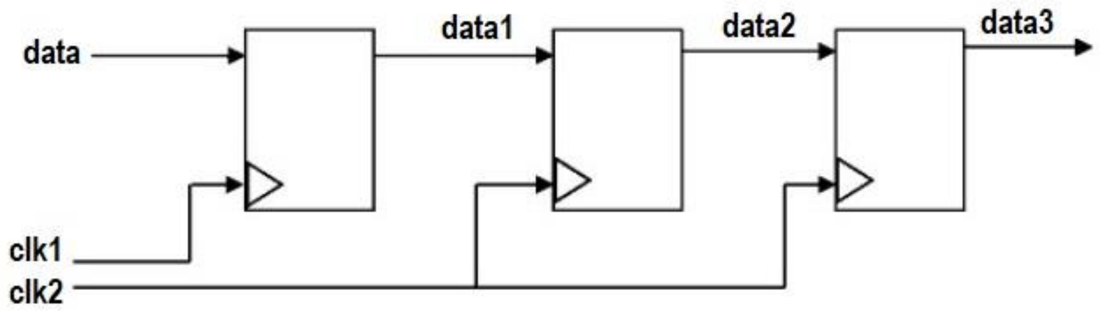


Figure 17: 2-FF synchronizer

In the 2-FF synchronizer, the first flip-flop samples the asynchronous input signal, data1, into the destination clock domain, clk2. A second flip-flop buffers the signal once again, synchronous to the destination clock. The second flip-flop has the intended goal of providing signal data2 sufficient time to get stable. Circuitry synchronous to clk2 can now safely sample data3. It is important to note that meta-stability is a probabilistic phenomenon, thus there is always a chance that data3 will still be unstable. However, this probability is practically minor and not to be taken into consideration.

ILA cores

The digital functionality described above is evaluated using Integrated Logic Analyzers (ILAs) in Vivado IDE (18). These Xilinx IP cores enable real-time debugging of FPGA designs while they are running in hardware. Such and evaluation method is highly efficient as it is applied under the actual system environment and speed conditions. Therefore, through this process, circuit functionality can be verified while potential timing issues are prevented.

The ILA cores function like digital oscilloscopes. The sampling frequency is defined by the core's clock domain. In this case, two clock domains exist in the design; therefore two ILA cores are necessary for effective design testing. Vivado interface allows setting up the trigger condition of every ILA core. When trigger condition is met, the corresponding ILA core is activated and signal samplings are depicted in a waveform viewer. Waveforms from different parts of the FSM logic are given below (the vertical red line represents the triggering moment):

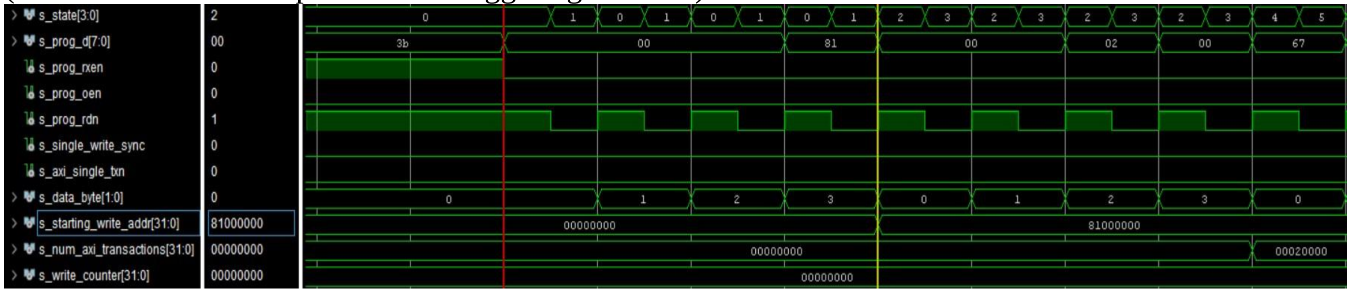


Figure 18: FIFO 245 FSM - Address configuration

In the digital waveforms of Figure 18 the most important signals of our logic are depicted. This capture is taken in the configuration phase, where write address and packet size is decided. The ILA triggering condition is falling edge of the signal **rxen** which is asserted when read data are available in the USB bus. The FSM remains between states GET_DDR_ADDRESS and DDR_ADDRESS (0 and 1, respectively) until 4 bytes are received. For one byte to be read, signal **rdn** has to be asserted.

Right after that, the DDR_ADDRESS is configured as equal to 0x81000000 and FSM moves on to states GET_AXI_TRANSACTIONS and AXI_TRANSACTIONS until the size of the waveform is also configured. In this example (Figure 19), the waveform to be downloaded is represented by the **s_num_axi_transactions** signal which is equal to 0x20000 after its configuration. As already mentioned, the total number of AXI transactions is equal to half the waveform size as with each AXI writing transfer, two waveform points are saved.

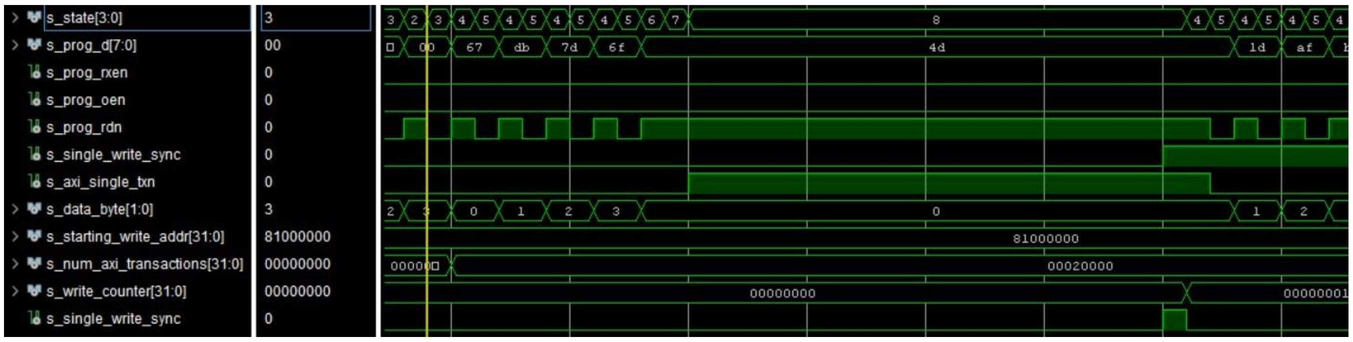


Figure 19: FIFO 245 FSM: Packet size configuration

The next phase of the packet download is the data transfer. The data points are transferred in words of four bytes through AXI interface. In the same manner as before, FSM remains between states GET_DATA_WORD and DATA_WORD (4 and 5 respectively) until a 32 bit AXI data word is created. Then an AXI transaction demand is issued by asserting the **s_single_axi_txn** signal. This signal's rising edge triggers AXI default logic to take action. When the transaction is done, AXI asserts the **s_single_write** signal and FSM moves on to receive the next data point.

Similarly, all waveform points are written to DDR3 memory and FSM returns in IDLE state after all data points are received.

3.3.1.3 USB 2.0 Streaming Approach

Up until now, storing the necessary data in an external DDR3 memory and latter reading them back to the FPGA, is the main approach for having large packets of customized data (in our case, signal waveforms customized by the instrument user) available for use in the hardware. The necessity for external memory device derives from the fact that the FPGA block ram cells cannot provide enough available storage space to support instrument functionalities. Inevitably, the instrument's overall performance is limited by the rate of data exchange between PC and FPGA.

A potential way of overcoming this limitation is by taking advantage of the fact that not the whole waveform packet needs to be available at hardware at a given time. Only one waveform point for every 12.5 MHz clock cycle (DAC sample rate) is needed to replicate our custom signal. Therefore, streaming each waveform point by PC to FPGA instead of downloading the whole data packet at once could eliminate the download time limitation.

Description

For implementing this idea the FT2232H chip is used once again in FIFO 245 mode. While the process of receiving data from chip remains similar, the internal digital logic that manages the incoming data words is changed. These are no longer directed to DDR3 memory, through AXI bus, but to FPGA internal block ram. This block ram is implemented as dual port First-In-First-Out (FIFO) memory of 16 bit data width and 2^{12} (4096 waveform points) data words depth. The following block diagram represents the overall idea (see Figure 20):

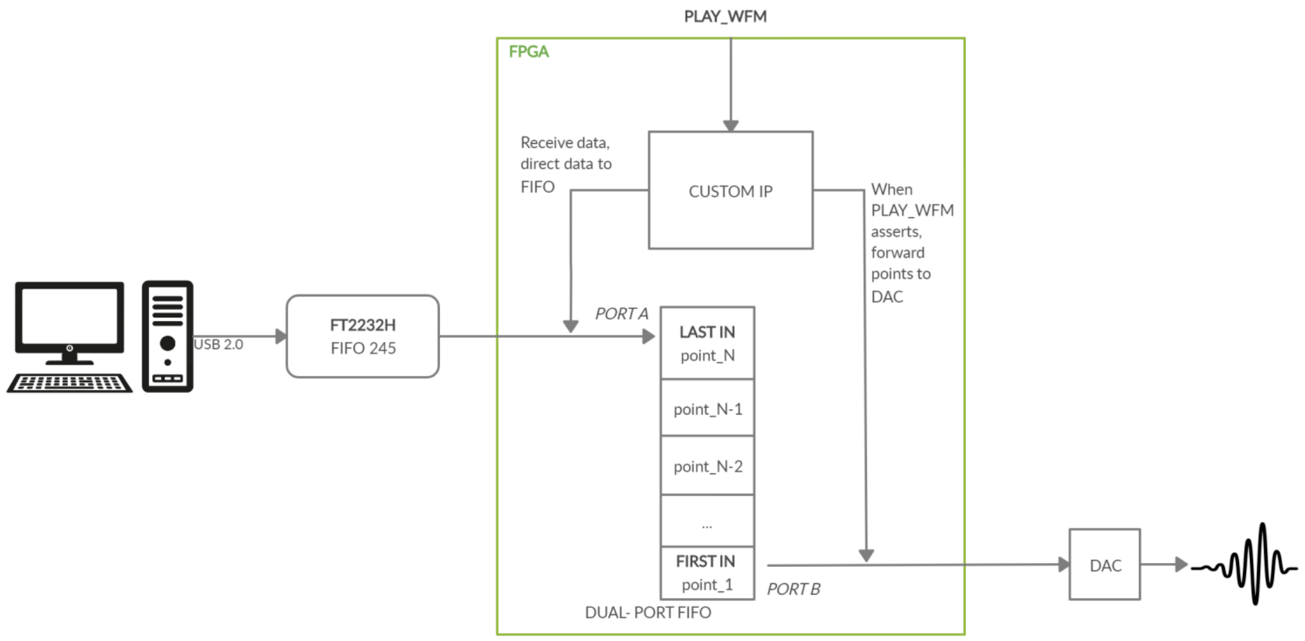


Figure 20: Waveform streaming using FT2232H FIFO 245 protocol

The waveform data path begins from PC software. After waveform customization and generation, data points are downloaded through USB 2.0 port to FT2232H chip. FPGA custom IP receives the data available and forwards them to internal FIFO block ram through writing port A. In any case, FIFO storage space is not enough to host a whole waveform signal. As data can be received until the point when FIFO is full, remaining data are stored inside FT2232H's internal FIFO block or inside PC memory. Waveform data remain static in this manner, until PLAY_WFM signal asserts. This signal signifies that a waveform play demand has been made and custom IP should initialize waveform point flow to the Digital-to-Analog (DAC) converter, at a rate of 12.5 MHz. As FPGA FIFO gets empty, custom IP is free to receive new incoming data from FT2232H interface. Considering the fact that the receiving rate (60 MHz) is faster than the transmitting rate (12.5 MHz) waveform points will always be available for DAC to reproduce.

There are two FSM processes, each one controlling one port of the FIFO block ram. **FTDI_TO_FIFO_PROC** process is responsible for reading the available data from the FT2232H and writing FIFO. In case FIFO is full, FSM maintains the byte that was received last and was never written in FIFO, until storage space is available. In this manner, data points are not lost even if FIFO runs out of memory available.

On the other side of the FIFO, **FIFO_TO_DAC_READ_PROC** process takes care of the waveform generation demands. It remains in idle state until **play_wfm** signal is asserted. Signal **wfm_Q2_Q5** determines the quadrupole where the waveform will be generated (so it is a DAC selector pin) and **wfm_points_num** determines the number of waveform points that will be generated. Cases where no data are available when a play-waveform demand is issued or FIFO runs out of data before **wfm_points_num** is reached, are both considered error occasions and **error** signal is activated.

3.3.1.4 USB 3.0 to FIFO245

Up to this point, the main goal is to achieve high download speeds which will not compromise in reliability and tolerance. This is the reason why an upgrade is examined, regarding the version of the communication protocol that is being used.

Towards this end, a comparison between USB 2.0 and USB 3.0 communication protocols was made. More precisely, USB 3.0 is the third major version of the Universal Serial Bus standard for interfacing computers and electronic devices. Its nominal speed is defined as 4.8 Gbps (Gigabits per second) while USB 2.0 gives a maximum throughput of 480 Mbps (Megabits per second), namely 10 times slower. In addition to that, a USB 3.0 channel is an asynchronous full-duplex communication port, which means that it can communicate on both directions at the same time (both send receive data simultaneously). On the other hand, USB 2.0 consists of a half-duplex polling mechanism which implies that, at some point, data can be either sent or received, but not both. The better performance of USB 3.0 is achieved against the number of wires within a communication cable (9 for USB 3.0 vs. 4 for USB 2.0) and power consumption (up to 500mA for USB 2.0 vs. up to 900mA for USB 3.0) (19).

Finding an appropriate chip

The FT60x Series - Superspeed USB 3.0 ICs chips, by FTDI, constitute a great solution for the incorporation of the USB 3.0 communication for our project. Those chips operate as USB3.0 to FIFO bridges and are being extensively used for demanding FPGA communication purposes. Two versions are available, the FT600Q and the FT601Q (20). The main difference between these two solutions is the size of their parallel FIFO data bus. To be more specific, FT600Q incorporates a 16bit parallel bus while FT601Q incorporates a parallel bus of 32 bits. The FT601Q version is selected as it can achieve twice the data bandwidth in comparison to the FT600Q chip.

Furthermore, FTDI provides customers with the associated DLL library files which can be used for the development of software applications. This way, bidirectional communication between FPGA and PC client can be accomplished.

FT601Q functionality includes two main modes of operation:

- 245 Synchronous FIFO mode: is a single-channel bidirectional communication protocol. The internal chip FIFO is divided in two parts, the Receive FIFO and the Transmit FIFO (4kBytes each). Thus the read and write operations occur independently.
- 600 Synchronous FIFO mode: is a multi-channel bidirectional communication protocol. The number of communication channels is configurable and can be either 2 or 4. Each one has its own set of Receive and Transmit FIFOs.

The main goal for the current project is the fast data transfer from a client PC to the DDR3 memory of our development board, through the Artix7 FPGA. A multi-channel configuration would be of no use for this purpose. Therefore, 245 synchronous mode is chosen as more appropriate for achieving the highest possible data throughput. The main pin signals of FT601Q that participate in 245 synchronous FIFO mode and that will be controlled by our Artix7 FPGA device, are listed below (21):

- **DATA0…31:** the I/O 32 bit parallel FIFO data bus
- **RXF_N:** is an output signal, Receive FIFO Full. It is active low and when active it indicates that the Receive FIFO has data available and it is ready to be read by the FIFO master (FPGA)
- **TXE_N:** is an output signal, Transmit FIFO Empty. It is active low and when active it indicates the Transmit FIFO has space available and it is ready to receive data from the FIFO master (FPGA)
- **OE_N:** is an input signal, Output Enable. It is active low and when it is driven low by the bus master (FPGA), the slave (chip) will drive the data bus.
- **WR_N:** is an input signal, Write Enable. It is active low and when it is driven low by the bus master, the master has write cycle access.
- **RD_N:** is an input signal, Read Enable. It is active low and when it is driven low by the bus master, the master has read cycle access.

As far as the waveform data download is concerned, the data will be transferred in one direction, from the PC client to DDR3 memory, by the FPGA communication master. For this reason, the signals **TXE_N** and **WR_N** are of no use, as their purpose is limited to the opposite data flow (from FPGA communication master to PC host). Signal **WR_N** (active low) is set to 1 by the FPGA master and is never asserted.

A 245 Synchronous FIFO mode bus master read transaction is represented in the diagram of Figure 21. It is significant to notify that the FT601Q chip provides its own digital clock of 100 MHz. The FPGA functionality is set to be synchronous to this clock domain.

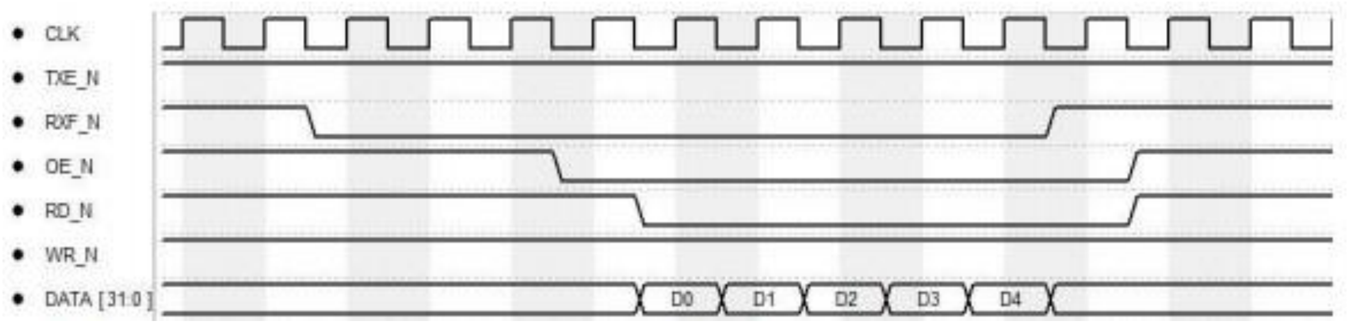


Figure 21: FT601Q read transaction

The block diagram of Figure 22 depicts an FSM (Finite State Machine) logic that could read data from the FT601Q Receive FIFO. It depicts the same information as above, but in a more comprehensive manner.

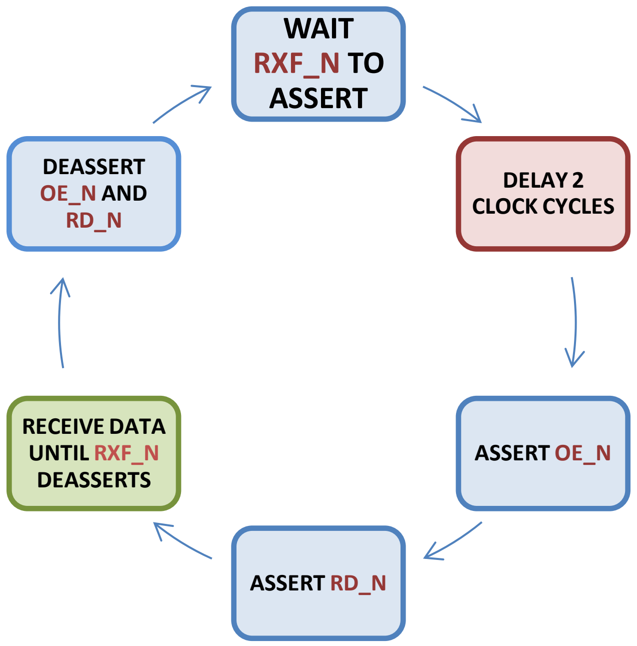


Figure 22: FT601Q Finite State Machine for a reading transaction

<u>USB107 Humandata</u>

The first revision of the USB 3.0 connectivity is implemented using the USB-107 FT601 Evaluation Board by Humandata. This board incorporates the FT601Q chip in a way that allows all possible configurations, such as different supply and I/O voltages for the chip (1.8, 2.5 or 3.3 Volts). In addition to that, it includes all the necessary ESD (electrostatic discharge) and surge protection components which ultimately eliminate the unwanted noise interference, a major issue in superspeed communications.

So that it is possible to execute the necessary tests, an intermediate board is designed. On this PCB, pin connectors for both the Numato Artix7 Neso and the Humandata USB107 are routed so that the FPGA can communicate with the FT601Q chip and control its functionality. The PCB is designed in such a way that impedance matching of 50 Ohms among both evaluation boards, is guaranteed. To achieve this goal several parameters are taken into consideration, but most importantly:

- A 4-layer design is chosen for the intermediate connectivity PCB. Using the impedance match calculator of Saturn PCB Toolkit Version 7.09, the thickness of each layer is chosen so that it satisfies the 50 Ohms impedance restriction.
- The rooting copper lines of the signal layer which are connecting the FPGA I/Os to the FT601Q pins, are being designed so that they have similar lengths and minimum travel distance on the board.

Block Design

AXI Stream

In the specific design, AXI4 Stream bus protocol is used to transfer data among the FPGA logic components. The AXI4-Stream protocol is used as a standard interface to connect components that wish to exchange data. The interface can be used to connect a single master that generates data, to a single slave, that receives data. The protocol can also be used when connecting larger numbers of master and slave components. In comparison to the other available AXI4 protocol versions, AXI4-FULL and AXI4-LITE, the AXI4 Stream is the most efficient for accessing consecutive memory addresses. This derives from the fact that AXI4-LITE demands master- slave signal handshaking and address reference for every data word transfer, while AXI4-FULL allows a maximum of 256 burst length (the term "burst transaction" defines a transaction of data that consists of only the data and the address of the first data word). On the other hand, AXI4 Stream allows unlimited burst length and is ideal for sending data to consecutive memory locations.

AXI4 Stream is not a bidirectional bus which means that data transfer can only happen in one direction from master to slave. The main AXI4 Stream signals are the following:

- Tready: a slave signal, which indicates that the master is ready to accept data.
- Tvalid: a master signal, which indicated that a valid data word is available.
- Tdata: the data that are streamed from master to slave
- Tlast: a master signal, which asserts every time the last data word of a packet is streamed

For an AXI4 Stream transaction to occur, write address is not necessary. The **tdata** word is streamed from master to slave every time both signals **tready** and **tvalid** are asserted. Finally, a streaming packet transaction should complete every time **tlast** signal is asserted.

The block diagram of the FPGA design is presented in Figure 23.

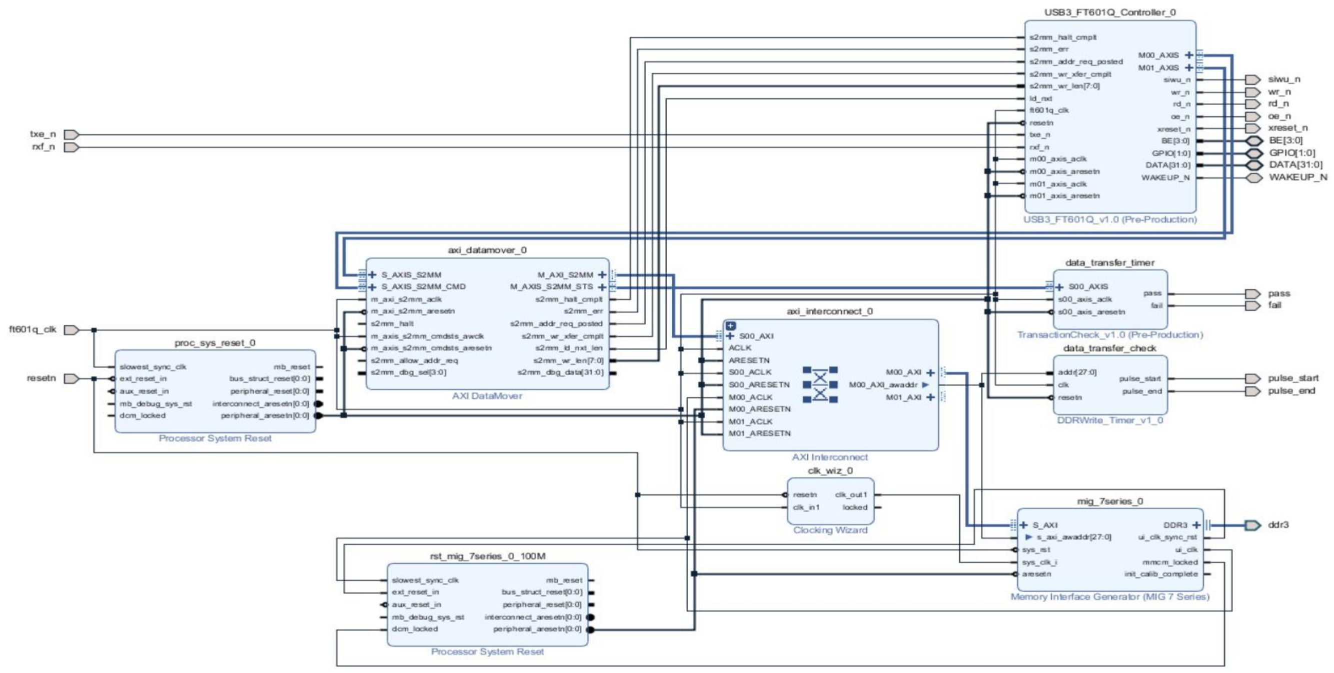


Figure 23: Implementation of PC-FPGA communication based on USB3.0

It consists of the following blocks:

- **AXI DataMover (version 5.1, Xilinx)**
- **AXI Interconnect (version 2.1, Xilinx)**
- **Memory Interface Generator (version 4.1, Xilinx)**
- **USB3 FT601Q Controller (custom IP)**
- **Data Transfer Check (custom IP)**
- **Data Transfer Timer (custom IP)**

AXI DataMover

AXI DataMover is a Xilinx IP core whose functionality converts AXI4 Stream to Memory Mapped AXI4 FULL communication protocol, and vice versa (22). As far is this FPGA design is concerned, this core is used to convert an AXI4 Stream Protocol , coming from the USB 3.0 Controller, into an AXI4-FULL protocol bus, so that communication with MIG memory controller can be accomplished.

AXI DataMover is configured via commands of specific format, which are issued through an AXI4 Stream slave interface. The most crucial information that every DataMover command has to include is the number of data bytes that are intended for streaming as well as the target write address.

After a command is issued, DataMover is ready to accept the streaming data. Data are transferred through another AXI4 stream slave interface. When the configured number of bytes is streamed, DataMover completes the transaction and moves on the execution of the next issued command.

In case incorrect number of bytes is provided, DataMover stops the execution of commands and remains in an error state. Therefore, proper logic that evaluates the DataMover transactions and monitors its status, is necessary.

Data Transfer Check

This custom IP is constantly sampling the AXI DataMover status AXI4 Stream interface. In case of transfer malfunction, error output is asserted. Otherwise, pass bit is asserted to notify that AXI DataMover's last transaction has been successfully completed.

Data Transfer Timer

This IP is used for data transfer speed measurement purposes. It samples the AXI4 interface of MIG controller. A pulse is generated on both transfer start and transfer end. Then, using an oscilloscope the data packet download and transfer to DDR3 memory can be precisely measured.

USB3 FT601Q Controller

The most essential IP core is the controller of the FT601Q chip. This logic core is responsible for both receiving the USB3 data and controlling the DataMover IP. Its design is being tested so that maximum data throughput is achieved, while VHDL code remains efficient and customizable in case future modifications are required.

Finite State Machine Description in words:

The USB3 FT601Q Controller FSM is specifically designed to receive a packet of bytes from the USB3 interface and replicate it to a DDR3 memory location. Both the size of the packet and the DDR3 write address are configurable. At the end of the transaction, the FSM returns in a stand-by state where it remains until new write data are available in the USB3 bus.

To begin with, when FPGA power-on occurs, FSM is set in **CONFIG_IDLE** state. In this state, the **rxf_n** bit of FT601Q chip is inspected every clock cycle. When this bit is asserted (set to 0, active low as already mentioned), FPGA is notified that USB3 data bus is not empty anymore and that FSM should proceed in receiving the data available. The first 32bit data word represents the transaction configuration data. More specifically, bits 24 downto 0 specify the packet size in 32 bit words, which is identical to the USB3 data bus width.

The remaining 7 bits (31 downto 25) represent the DDR3 write location. The available storage space inside the DDR3 memory is 2 Gb (Giga Bits) and is split into 128 parts of 2 MB (Mega Bytes) size each. Every memory part has its own ID number. Consequently, ID values range from 0 to 127, so 7 bits are just enough for their representation.

After the transaction is configured, FSM is set in **DATA_IDLE** state and awaits the **rxf_n** bit to be asserted once again. When the expected signal assertion occurs, FSM proceeds in receiving every data available in the USB3 bus, until packet size is reached.

In this phase, it is significant to notify that, FT601Q chip transfers data in small parts of 4kB (kilo Bytes). This derives from the fact that the specific device incorporates an internal FIFO of 4kB data depth.

Every FT601Q data packet that is received, is stored temporarily inside the FPGA, in bounded block ram cells (FIFO) of 4kB maximum capacity. Whenever this block ram space is full, FSM pauses data receive process and issues a DataMover transaction command. Right afterwards, block ram data are being streamed to DataMover AXI4 Stream data port. When all 4kB are finally streamed, block ram should be empty once again. Subsequently, FSM proceeds in receiving more data, until packet size is reached and eventually the DDR3 write transaction is complete.

A more detailed description of FSM states:

Our FSM can be found in one of the following states (the order is identical to the operation flow):

TRANSACTION CONFIGURATION(Figure 24)

- **CONFIG_IDLE**: The beginning state. Assertion of **rxf_n** is expected.
- **STALL1, STALL2:** Two consecutive stall clock cycles that are necessary to occur before FT601Q is able to export the data available.
- **SET_CONFIG_OEN:** Output enable pin (**oe_n**) is asserted.
- **SET_CONFIG_RD:** Read enable (**rd_n**) is asserted.
- **RECEIVE_CONFIG_DATA:** The transaction configuration data are received.
- **CONFIG_TRANSFER1:** Data type conversion occurs. From **ddr3_id_slv** (std_logic_vector) to **DDR3_ID** (integer) and from **data_size_slv** (std_logic_vector) to **DATA_SIZE** (unsigned).
- **CONFIG_TRANSFER2:** DDR3_ID number is matched to the equivalent DDR3 starting write address.

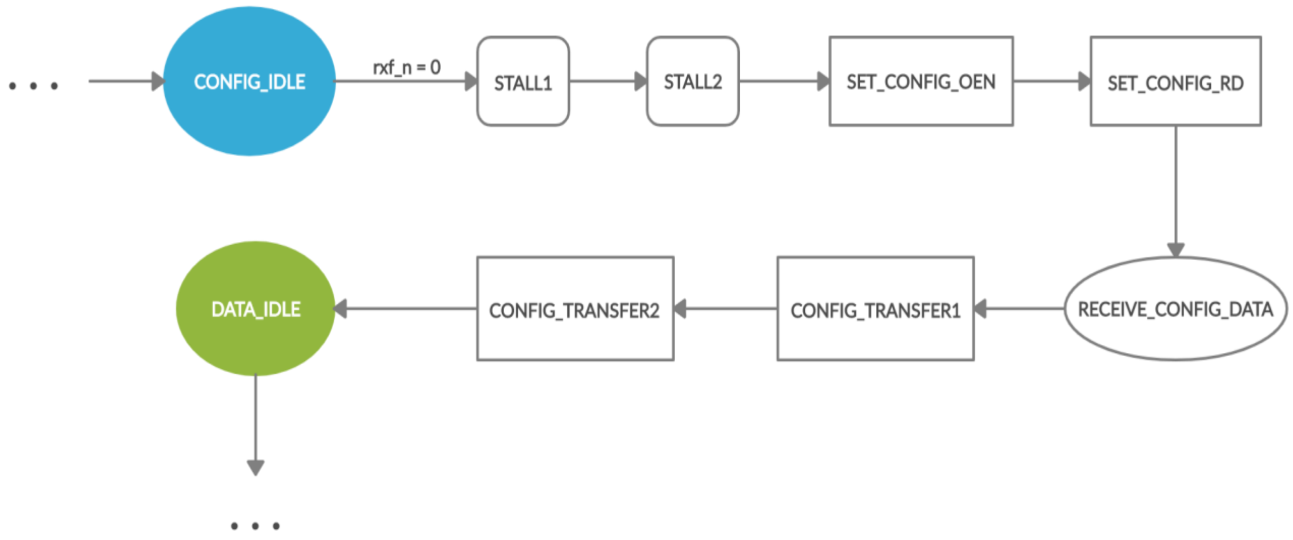


Figure 24: FT601Q FSM transaction configuration

<u>DATA PACKET RECEIVE(</u>Figure 25<u>)</u>

- **DATA_IDLE:** Wait for **rxf_n** to assert.
- **STALL3, STALL4:** same as STALL1, STALL2
- **SET_OEN:** Output enable pin (**oe_n**) is asserted.
- **SET_RD:** Read enable (**rd_n**) is asserted.
- **RECEIVE_DATA:** Receive all USB3 data until **rxf_n** is deasserted. Data are organized in 128bit data width which is identical to the DataMover AXI4 Stream port width.
- **CONFIG_DATAMOVER_CMD:** The DataMover transaction command is configured.
- **FETCH_DATAMOVER_CMD:** A trigger pulse is asserted so that the DataMover command is issued.
- **WAIT_DATAMOVER_LDNXT:** Wait until the DataMover command fetch phase is completed. The command is ready when **ld_nxt** signal of DataMover IP core is asserted.
- **CONFIG_DATAMOVER_STREAM:** The number of 128bit data words that are about to be streamed is configured.
- **TRIG_DATAMOVER_STREAM:** A trigger pulse is asserted so that the DataMover stream begins.
- **WAIT_UNTIL_DONE:** Assertion of stream_done, which signifies the end of the data stream, is expected/ If packet size is reached, FSM returns to CONFIG_IDLE state, else it is set to DATA_IDLE so that it receives the remaining data.

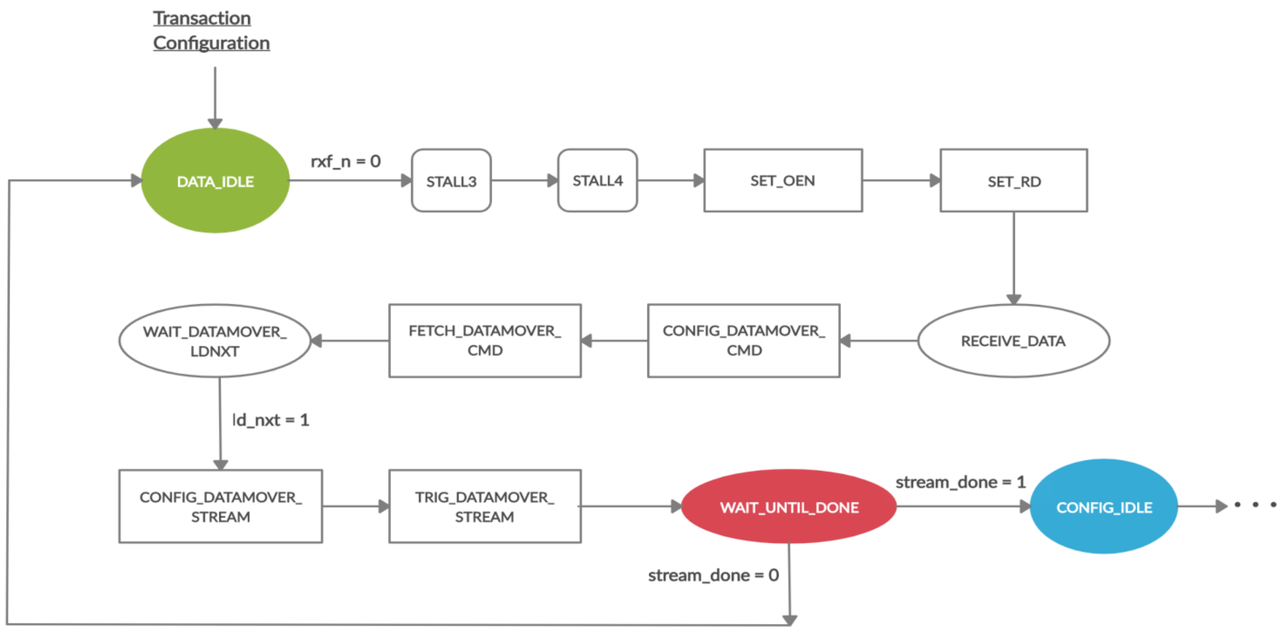


Figure 25: FT601Q FSM packet receive

ILA cores

The sampling frequency is defined by the core's clock domain, which in our case is chosen to be the FT601Q clock of 100MHz. This sampling rate is sufficient because our whole digital system is synchronous to it and no clock of higher frequency exists in the design.

Vivado interface allows setting up the trigger condition of every ILA core. When trigger condition is met, the corresponding ILA core is activated and signal samplings are depicted in a waveform viewer. Waveforms from different parts of the FSM logic are given below (the vertical red line represents the triggering moment):

Transaction Configuration(Figure 26):

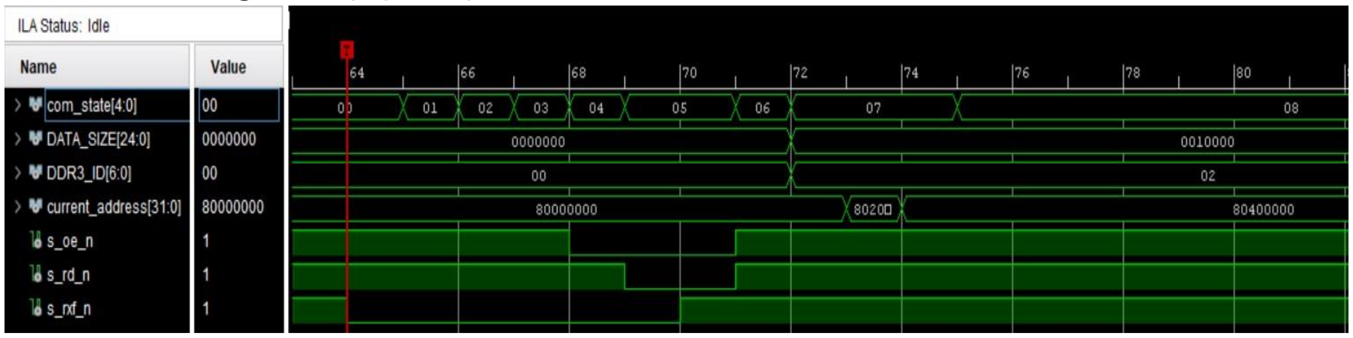
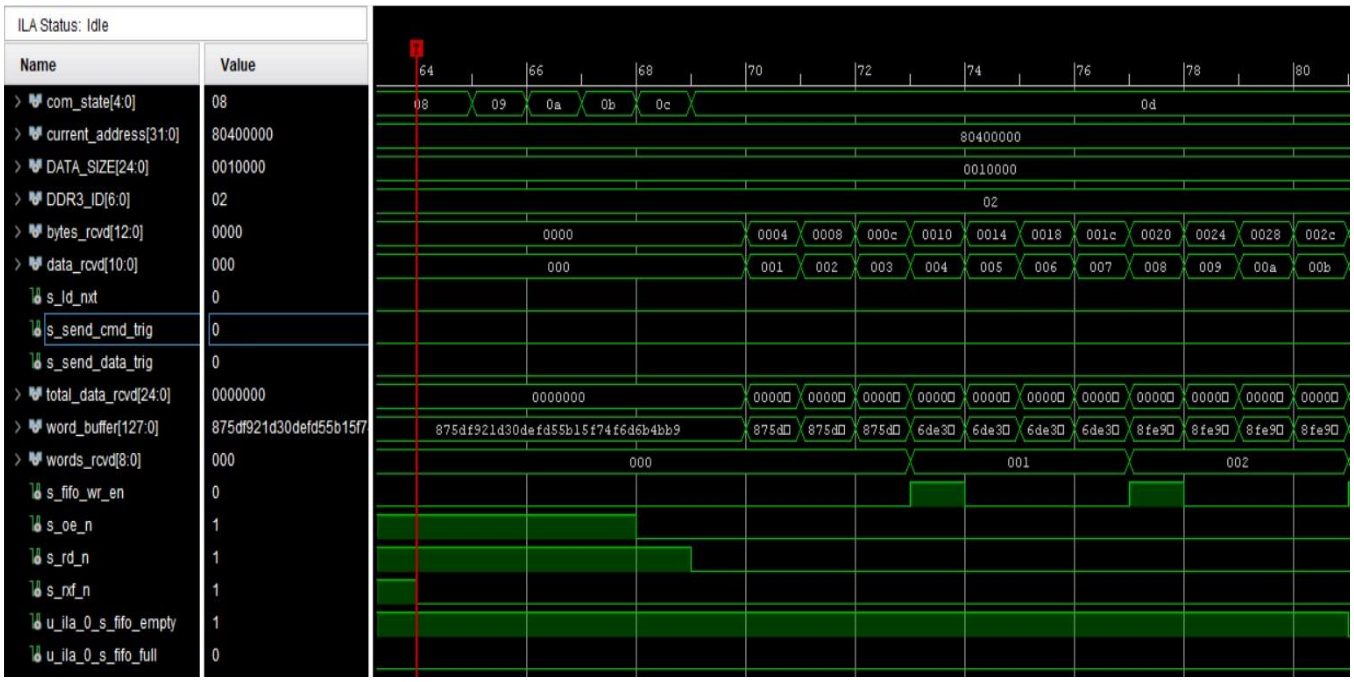


Figure 26: Transaction configuration, ILA cores

When signal **rxf_n** is asserted, ILA core is activated and results are plotted in the waveform viewer.

After two stall clock cycles, signals **oe_n** and **rd_n** are asserted. Subsequently, **DATA_SIZE** and **DDR3_ID** are both configured by the first 32 bit data word. In this example, data size is equal to 0x10000 (decimal value: 2^{16} = 65,536) while DDR3_ID is equal to 2. The write ID is matched to write address 0x80400000 (DDR3 base address = 0x8000000, ID segment size = 0x200000). Finally, the FSM is set to state 0x08 (hex equivalent of **DATA_IDLE** state) waiting for next data to be available in the USB3 bus.

<u>DATA PACKET RECEIVE (start, Figure 27)</u>



ILA Status: Idle

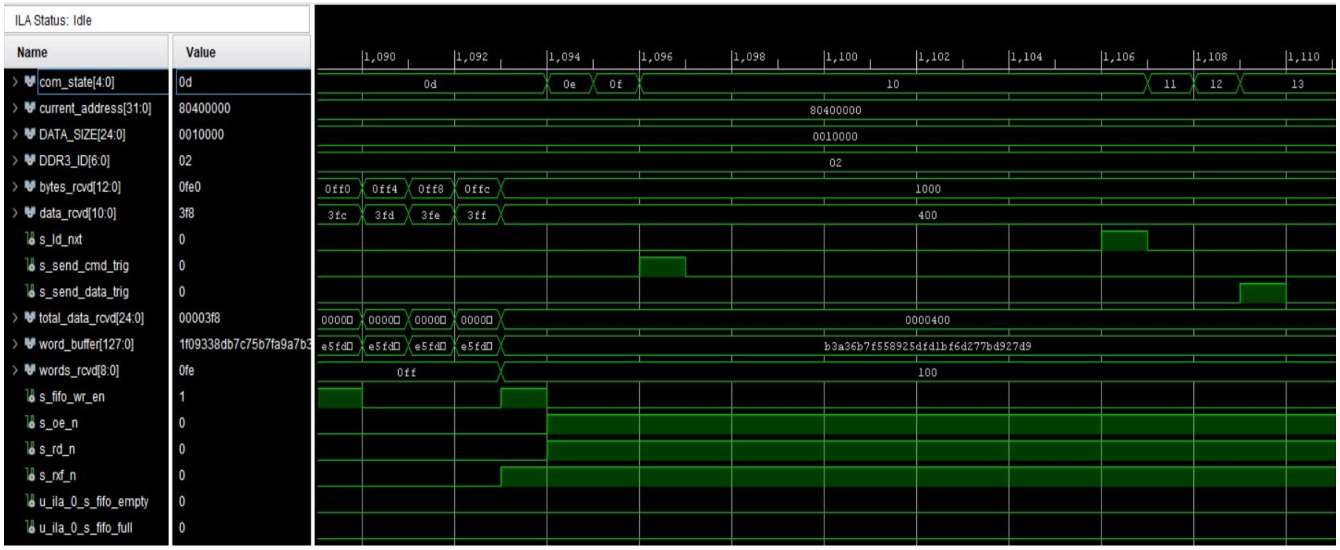
Name	Value
com_state[4:0]	08
current_address[31:0]	80400000
DATA_SIZE[24:0]	0010000
DDR3_ID[6:0]	02
bytes_rcvd[12:0]	0000
data_rcvd[10:0]	000
s_ld_nxt	0
s_send_cmd_trig	0
s_send_data_trig	0
total_data_rcvd[24:0]	0000000
word_buffer[127:0]	875df921d30defd55b15f7
words_rcvd[8:0]	000
s_fifo_wr_en	0
s_oe_n	1
s_rd_n	1
s_rxf_n	1
u_ila_0_s_fifo_empty	1
u_ila_0_s_fifo_full	0

64
66
68
70
72
74
76
78
80
08 09 0a 0b 0c 0d
80400000
0010000
02
0000 0004 0008 000c 0010 0014 0018 001c 0020 0024 0028 002c
000 001 002 003 004 005 006 007 008 009 00a 00b
0000000
875df921d30defd55b15f74f6d6b4bb9
000 001 002

Figure 27: USB 3.0 waveform signal receiving - configuration

After transaction configuration, FSM should stay in an idle state until **rxf_n** is asserted once again. Indeed, right after the necessary signal assertion, FSM sets **oe_n** and **rd_n** to logical zero so that USB3 chip moves on to generate the data available. Data received are stored in a local FIFO of 128 write width and 4 kB write depth. Thus, FIFO's write enable signal is asserted every four data words (of 32bit length), allowing **word_buffer** (of 128bit length) to be pushed in.

<u>DATA PACKET RECEIVE (end of a random packet, Figure 28)</u>

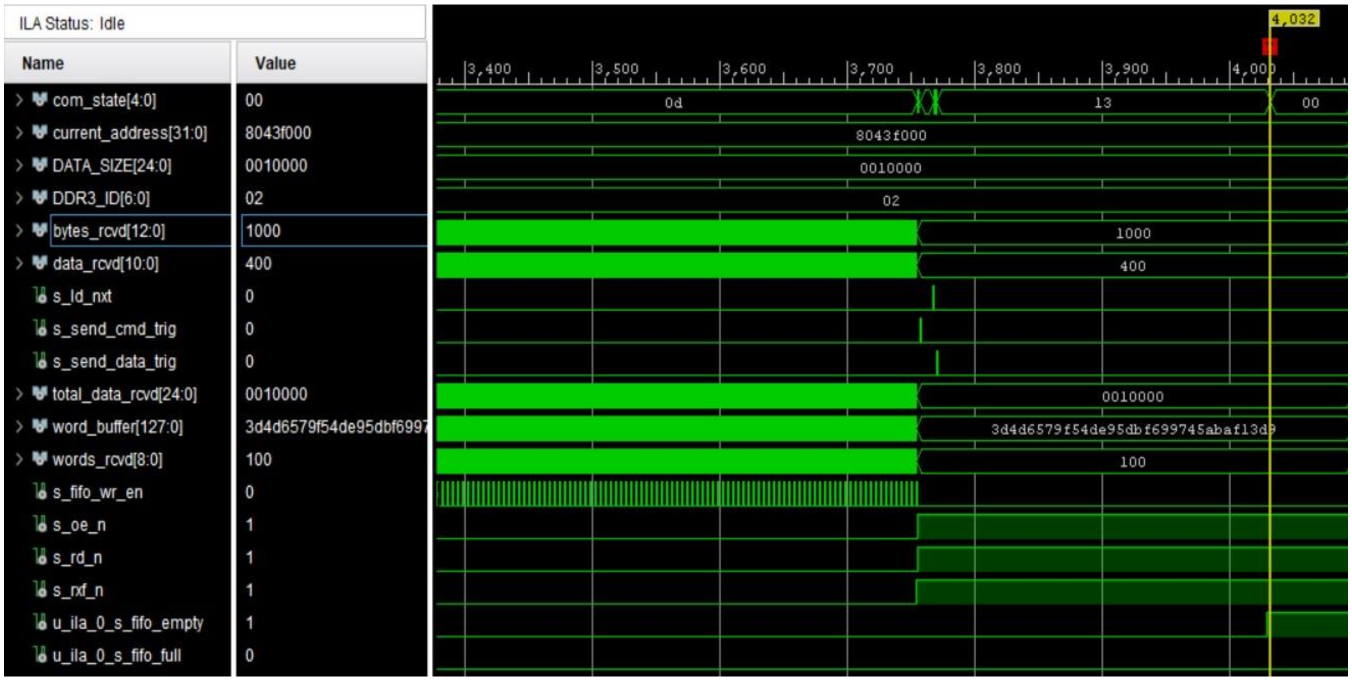

ILA Status: Idle

Name	Value
com_state[4:0]	0d
current_address[31:0]	80400000
DATA_SIZE[24:0]	0010000
DDR3_ID[6:0]	02
bytes_rcvd[12:0]	0fe0
data_rcvd[10:0]	3f8
s_ld_nxt	0
s_send_cmd_trig	0
s_send_data_trig	0
total_data_rcvd[24:0]	00003f8
word_buffer[127:0]	1f09338db7c75b7fa9a7b3
words_rcvd[8:0]	0fe
s_fifo_wr_en	1
s_oe_n	0
s_rd_n	0
s_rxf_n	0
u_ila_0_s_fifo_empty	0
u_ila_0_s_fifo_full	0

Figure 28: USB3.0 waveform data packet receiving - completion

Data are received by the FPGA master until **rxf_n** signal is deasserted. Right afterwards, a transfer command is issued to DataMover's AXI Stream command port by asserting the **send_cmd_trig** signal. The moment when signal **ld_nxt** is set to logical one by the DataMover, indicates that our transfer command has been successfully issued and that it is ready for execution. Therefore, FSM generates a **send_data_trig** pulse to initiate the stream of received data, to DataMover's AXI Stream data port. The maximum number of data words that the USB3 chip can generate in one **rxf_n** assertion – deassertion event, is 1024 which is equivalent to 4 kB.

DATA PACKET RECEIVE (end of transaction, Figure 29)


ILA Status: Idle

Name	Value
com_state[4:0]	00
current_address[31:0]	8043f000
DATA_SIZE[24:0]	0010000
DDR3_ID[6:0]	02
bytes_rcvd[12:0]	1000
data_rcvd[10:0]	400
s_ld_nxt	0
s_send_cmd_trig	0
s_send_data_trig	0
total_data_rcvd[24:0]	0010000
word_buffer[127:0]	3d4d6579f54de95dbf6997
words_rcvd[8:0]	100
s_fifo_wr_en	0
s_oe_n	1
s_rd_n	1
s_rxf_n	1
u_ila_0_s_fifo_empty	1
u_ila_0_s_fifo_full	0

Figure 29: USB3.0 waveform signal receiving - completion

The unsigned signal **total_data_received** serves as a counter which represents the total number of data words of 32bit length that the FPGA master has received. When **total_data_received** gets equal to **DATA_SIZE** the transaction is complete and the FSM should be set to **CONFIG_IDLE** state, waiting for the next transaction configuration. The waveform above depicts the moment when the last packet of 4 kB is written to the DDR3 and the FSM resets to its initial state.

4 FPGA acceleration of waveform generation

Purpose

Waveform generation and customization constitutes a vital part of the instrument's functionality. The fact that Omnitrap's sequence of actions during and experimental process is primarily controlled by the FPGA, makes the idea of direct digital waveform synthesis, by the FPGA itself, quite appealing. On the other side, the same signal configuration capabilities have to remain open to the instrument user, while speed and robustness cannot be neglected. For the purpose of comparing waveform generation by PC against direct digital synthesis by FPGA, the latter is also implemented and examined in terms of practicality, effectiveness and results, in a broader sense, that this modification would have in the overall instrument design. As already mentioned, there are two kinds of isolation waveforms that are used, Sweep and Filtered Noise Field (FNF). Each of these signals has its unique properties and thus requires different approach in the purpose of generating it inside the FPGA.

4.1 Sweep Direct Digital Synthesis

The calculations that are necessary for a sweep waveform generation are already mentioned in chapter 3.2.1 and are also given here for reader convenience:

$$\boldsymbol{Sweep(n) = \sin\,[Phase(n)]},$$

where *Phase(n)* is calculated by the formula:

$$\boldsymbol{Phase(n) = Phase(n-1) + 2pi * Ts * frequency(n)},$$

where *Ts* is the sampling period.

Frequency(n) is increasing linearly:

$$\boldsymbol{frequency(n) = frequency(n-1) + frequency\_step}.$$

These mathematical expressions constitute the base of Sweep generation by the FPGA, too. A new custom IP is implemented that executes the above calculation process in a pipelining manner so that high calculation speeds are accomplished. Sweep waveform is still configured by instrument user through user interface. Starting frequency, frequency step, sweep duration, notch placement and amplitude modifications are open to user, offering high signal customizability. The configuration data are sent to FPGA through USB to UART interface. This data size is minor, so speed is not an important factor here and thus the simple-to-implement UART interface is used.

The design and implementation of FPGA logic that effectively realizes the sweep calculation, while seeming quite straightforward, includes some points that require additional thinking and attention. The first issue that initially comes up, is finding a proper way to represent decimal numbers inside the FPGA. A one-by-one numeric representation is not possible to occur, as the FPGA can only manage integer values in the form of binary vectors. On top of that, additional difficulty occurs by the fact that sweep calculation includes the mathematical operation of multiplication which is a high cost process that requires multiple clock cycles to complete and increases the overall design complexity. Finally, an FPGA sine generator is necessary to be implemented for completing the sweep calculation.

The above issues are approached and solved in a way that minimal FPGA resources are used while maximum calculation speed is achieved. To begin with, in order to eliminate the need of multiplication operation, data that are sent to FPGA are normalized in advance by a factor of 2π*Ts. This way, the sweep calculation is reduced to a series of additions and sine calculation and no resources for multiplication digital logic are required.

Sine calculation by FPGA is achieved by the use of Look-Up-Table (LUT) (23). A LUT is an array that replaces runtime computation with a simple array indexing operation. More specifically, discrete sine values are permanently stored inside FPGA, using either distributed or block ram resources, and calculation occurs by means of these pre-stored values (24). The sampling resolution of the stored sine wave can vary depending on the sine wave frequency range that needs to be generated. Furthermore, not a whole sine wave period needs to be stored. Only one quarter of the total period is enough to serve the sine calculation for every phase value. This, however, comes with the cost of some minor digital logic which is responsible for deciding which sine period quarter corresponds to the specified phase value, every time. This obviously adds an extra pipeline stage to the calculation but requires little FPGA resources to be implemented. For our purposes, a sine wave quarter is stored in FPGA distributed memory, with a resolution of 512 points and sine values range from 0 to 8191 (13 bits). The length of the sine values stored is chosen to be 13 bits as an extra bit is needed for sign representation and the DAC which is used for digital to analog sweep conversion takes 14-bit size words as inputs.

The main sweep configuration data that are sent to FPGA are:

- **The total of sweep points**: it is dependent to our sample rate (12.5 MHz) and sweep duration that is chosen by the user.
- **The normalized starting frequency**: it is the sweep starting frequency normalized by a factor N.
- **A number K that defines the speed rate of frequency change**: this value is related to the ending frequency of the sweep waveform.

To begin with, an integer counter that ranges from 0 to 2^20 (reflects to 80ms which is the maximum waveform duration) represents the discrete timing axis of the signal. For every sweep point that is generated, timing counter is increased by one and by the time it reaches the **sweep points** value, IP stops generating sweep points.

As far as starting frequency is concerned, it is mentioned above that it is pre-normalized by a factor of N before sent to FPGA. This factor is equal to: $\mathrm{N} = \frac{1024}{2\pi} * 2\pi \mathrm{Ts} * 256$

where the first term reflects to normalizing data according to the stored sine wave values (256 sine values from 0 to π/2, first sine quarter), the second term (2π*Ts) is used to replace the multiplication operation, as already mentioned, and the third term (256) is used to cancel out some of the FPGA rounding error due to the integer phase representation .Multiplication by 256 is easily reversed inside FPGA by always cutting out the last 8 bits of the phase value (division by 256).

Finally, the last sweep configuration data word is a number that represents the number of sweep points that the frequency remains stable. This number is inversely proportional to the frequency width. Large frequency width for a specified sweep duration means that frequency needs to change in a faster rate so that the whole desirable frequency range is covered. This number is equal to:

$\mathrm{round}\left(\frac{1}{\mathrm{frequency}_{\mathrm{step}} * N}\right)$, where frequency step is calculated in Hz and is equal to frequency width divided by the total of sweep points. Every time a counter reaches this value, the frequency register is incremented by one.

The Sweep generator is implemented in a pipeline manner and its functionality, as well as its major digital components, is represented in the block diagram of **Error! Reference source not found.**:

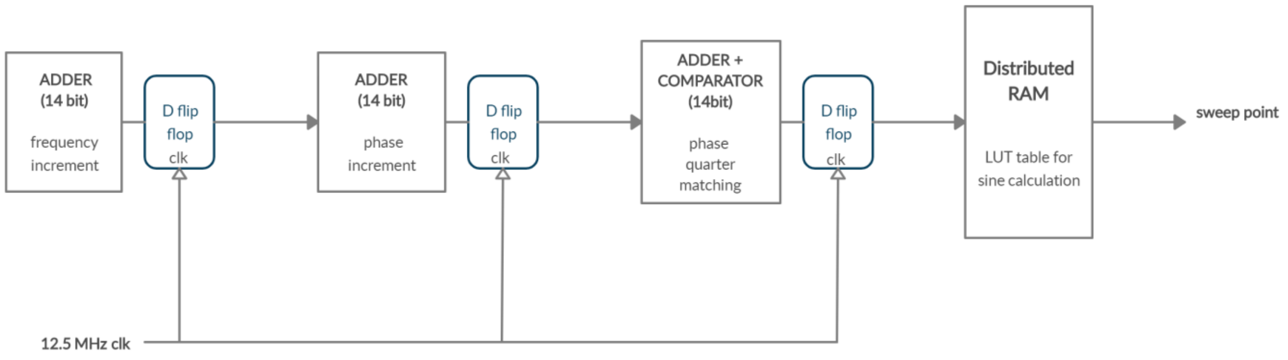
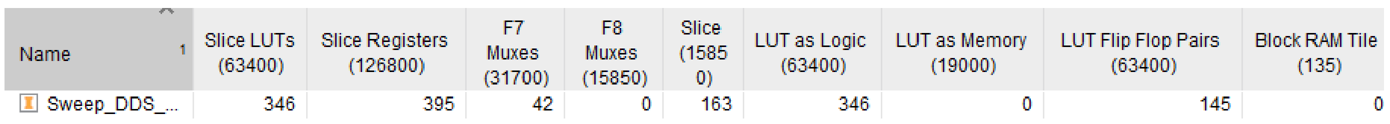


Figure 30: Sweep generation by FPGA

The pipeline stages are described below:

- **Frequency increment**: frequency is incremented by one if the corresponding counter reaches a specified value K.
- **Phase increment**: the phase of the next sweep point is calculated
- **Phase Quarter Matching**: digital logic figures out the sine quarter that corresponds to the current phase and calculates the sine LUT index of the corresponding sine value.
- **Sine calculation**: Read sine value from sine LUT and generate a sweep point.



Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	LUT Flip Flop Pairs (63400)	Block RAM Tile (135)
Sweep_DDS_...	346	395	42	0	163	346	0	145	0

Figure 31: FPGA resource utilization of Sweep generation circuitry

4.2 Filtered Noise Field (FNF) Direct Digital Synthesis

The Filtered Noise Field waveform can be generated by the sum of sine wave signals of different frequencies with a specified range and frequency step. An FNF waveform can include a large number of frequency tones from 500 to 1000. It is self-evident that summing up such a significant number of sine waves is not an operation that can be effectively implemented inside an FPGA. As a result, a different approach is needed. The main idea is that the FNF calculation can start from the frequency domain, where the amplitude and phase of all desirable frequency tones will be defined. After signal frequency content is specified, the resulting signal is transformed to time domain by means of Inverse Fourier transform, implemented inside the FPGA.

For the purpose of the IFFT implementation, the Fast Fourier Transform Xilinx IP core is used (25). This IP core implements the Cooley-Tukey FFT algorithm. It can be configured to execute both direct and inverse FFT (26). The transform size can take powers of two as its value, from 3 to 16 (8 to 65536 points). As far as the IFFT is concerned, which is used in our case, data samples are being fed to the IP core in complex form, through AXI4 Stream slave interface. The real and imaginary part length of the input is configurable and can range from 8 to 34 bit precision, each. After the necessary number of points is provided to the IP core, output data are available after a short period of time, when tvalid signal of AXI4 Stream master data channel is asserted. The output data format can be either fixed or floating point, with the latter offering better precision but demanding significantly more resources for the core implementation and cause a greater delay to the overall computation. In case overflow occurs during calculation, FFT IP core can be configured to apply a scaling method so that output data remain intact. The scaling rate is provided by another output master interface, so that precise calculation is attainable.

For the purpose of FNF digital synthesis by the FPGA, a custom IP is created. This IP is equipped with an AXI4 slave interface and two AXI4 master interfaces. The slave channel is used for FNF signal customization. Configuration data are specified through user interface and are sent to FPGA through USB to UART data channel. Microblaze is responsible for receiving the data and feed them to our custom IP. Among this information, the most important contents are the following:

- **Starting/Ending Frequency**: they represent the frequency width of the FNF signal. These values are not given in Hz but in the form of an index, instead. This index reflects to the first (or last) frequency domain point that has non-zero amplitude and should correspond to the desirable starting (or ending) frequency. In order to find the frequency domain point that links to a specific frequency the following formula is used, which derives from the basic principles of the Fast Fourier Transform:
  $\mathrm{index} = \mathrm{round}(\mathrm{frequency} * \frac{\mathrm{NFFT}}{\mathrm{Fs}})$, where frequency is given in Hz, NFFT corresponds to the FFT size and Fs is the signal sampling rate in Hz.
- **Frequency step**: it is also given in the form of frequency domain index.
- **IFFT size**: it must be a power of two. It represents the Inverse Fourier Transform size. An FNF signal size can range from 131,072 points (2^{17} , 10ms waveform) to 1,048,576 points (2^{20} , 80ms waveform). Such long IFFT transforms are not supported by the FFT

core used. Thus a technique is applied, similar to software FNF generation, so that IFF transform size is reduced. For this purpose, the FNF period is primarily calculated in software (it is the least common multiple of the frequency tones that are included in the waveform) and the final signal occurs by concatenation of multiple FNF periods. Consequently, the IFFT size is reduced to one FNF period. By choosing appropriate frequency step, it is quite simple to minimize FNF period size so that it does not exceed the maximum IFFT width available (65,536 transform size, corresponds to 5ms signal), without any consequences in the experimental process. However, for the signal concatenation purposes, storage space of at least one FNF period size, is necessary to be available, either internally (block ram or distributed ram) or externally (memory module, i.e. DDR3 chip) of the FPGA.

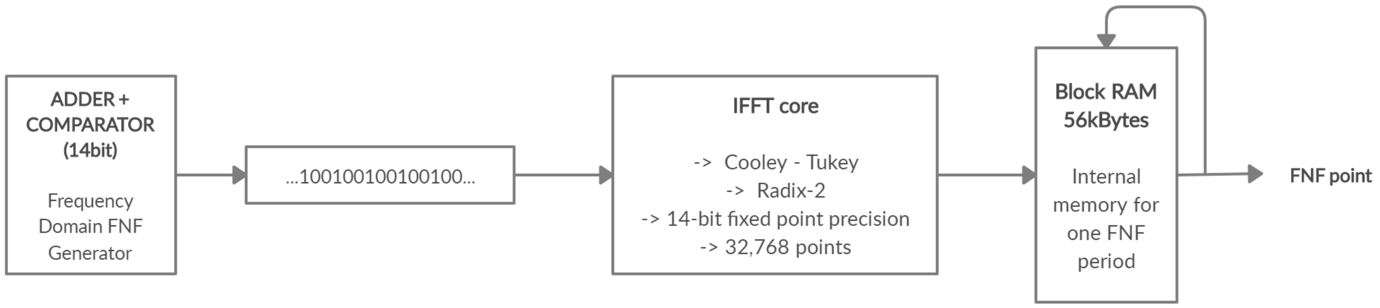
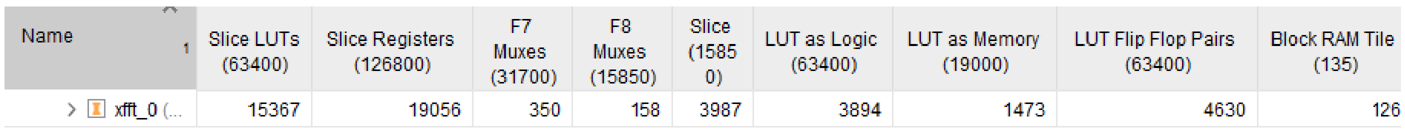

ADDER + COMPARATOR (14bit)
Frequency Domain FNF Generator
...100100100100100...
IFFT core
-> Cooley - Tukey
-> Radix-2
-> 14-bit fixed point precision
-> 32,768 points
Block RAM 56kBytes
Internal memory for one FNF period
FNF point

Figure 32: FNF generation by FPGA



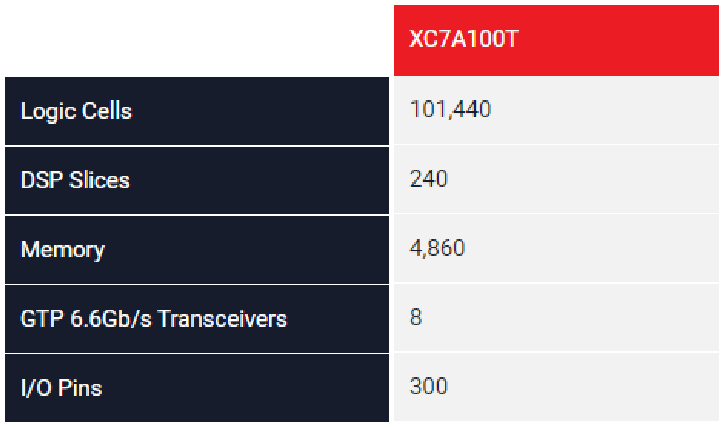
Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	LUT Flip Flop Pairs (63400)	Block RAM Tile (135)
xfft_0 (...	15367	19056	350	158	3987	3894	1473	4630	126

Figure 33: FPGA resource utilization of FNF generation circuitry

5 Evaluation

5.1 FPGA Development Board

The FPGA development board that is used is the version 2 of Neso – Artix7 by Numato (Figure 35). It incorporates the XC7A100T version of a Xilinx Artix7 FPGA device. The Artix 7 FPGA is chosen for the current project, as it incorporates a sufficient amount of programmable resources, namely logic cells, internal FPGA memory (block ram cells) and DSP slices. Digital signal processing exploration takes place in this diploma thesis, and therefore the use of an FPGA device which incorporates DSP slices is appropriate. Moreover, concerning Artix 7, it is considered to be the most efficient Xilinx FPGA device in terms of performance-per-watt, which can ultimately contribute to a low consumption overall implementation, as FPGAs in general are characterized as devices with relatively high power requirements. Subsequently, after making an estimation of the number of I/O pins that would be necessary for the final implementation and also taking into consideration some extra I/O pins for future design extensions, the XC7A100T version of the Artix7 is picked, whose specifications are shown in Figure 34**Error! Reference source not found.**.



	XC7A100T
Logic Cells	101,440
DSP Slices	240
Memory	4,860
GTP 6.6Gb/s Transceivers	8
I/O Pins	300

Figure 34: Artix 7 XC7A100T FPGA specifications

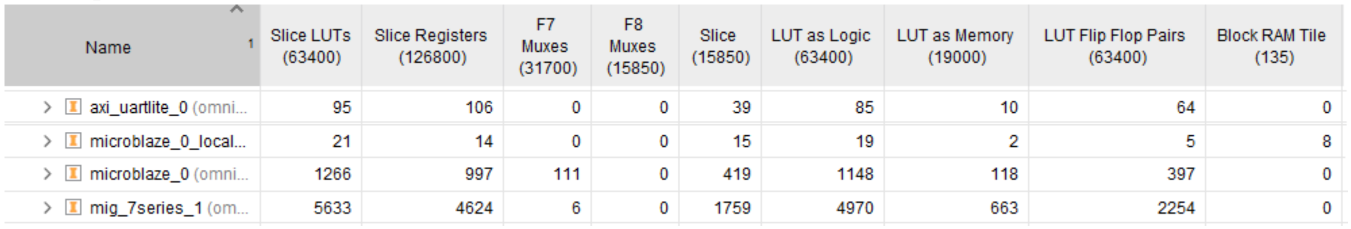
Finally, the Neso development board is chosen as it incorporates both a USB 2.0 interface and a 2Gbit DDR3 memory module. USB 2.0 interface is necessary for generating a communication port from PC to FPGA while external DDR3 memory offers extra storage space for the FPGA data and opens a wide variety of possibilities concerning the FPGA functionality. Using an FPGA development board instead of a single FPGA device, adds simplicity to the instrument development as the demanding and time consuming procedure of rooting appropriately the FPGA device, the DDR3 memory and the USB interface on a custom printed circuit board, is passed over. This practice is ideal in the early stages of instrument development, where a fast and effective primitive implementation is the main goal.

Figure 35: Neso developement board

5.2 PC - FPGA communication results

Attempts were made so that data from PC to FPGA are transferred in an efficient way that satisfies the timing restrictions and instrument specifications while effectively takes advantage of the available FPGA resources and contributes to the experimental process acceleration. These implementations find application in modern laboratory equipment and instrumentation in general that requires use of FPGA devices and data exchange between PC and FPGA. Details that were derived from every method's testing and evaluation are given below (see also Figure 40):

USB 2.0 to UART protocol, Microblaze instantiation for data receive and DDR3 save:
The Microblaze firmware code used while simple on its implementation provides a reliable way of communicating with the FPGA. For every action, the corresponding identifier byte is sent back to the PC master as confirmation. In this manner, PC master is always updated concerning the communication status and can take action in case of failure i.e. a potential case where Microblaze did not respond back after a specified amount of time or the confirmation byte was wrong. The FPGA resources occupied by this implementation are shown in Figure 36.



Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	LUT Flip Flop Pairs (63400)	Block RAM Tile (135)
axi_uartlite_0 (omni...	95	106	0	0	39	85	10	64	0
microblaze_0_local...	21	14	0	0	15	19	2	5	8
microblaze_0 (omni...	1266	997	111	0	419	1148	118	397	0
mig_7series_1 (om...	5633	4624	6	0	1759	4970	663	2254	0

Figure 36: USB 2.0 to UART protocol, Microblaze instantiation for data receive and DDR3 save resource utilization

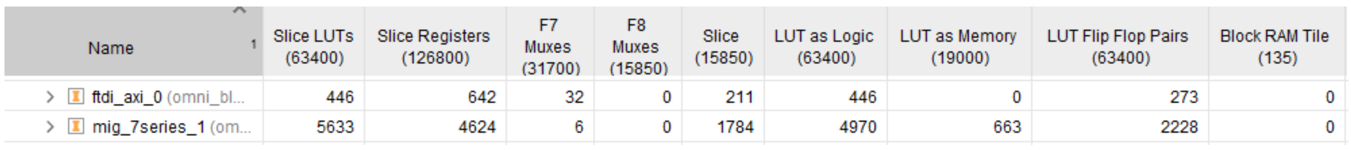
While this design is efficient enough for instrument measurement read-backs and small data transfer in general (like commands sequence download), it lacks high speed capabilities and can cause long delays in case large packets of data (i.e. a whole waveform) need to be transmitted to the FPGA.

Indicatively, downloading a data packet of 512kBytes (which is the average waveform size for the experimental purposes) and saving it to DDR3 has been counted (through PC software timers) to range around 8 to 10 seconds, where around 4 to 5 seconds belong to data download time to FPGA using UART protocol at 921600 baud rate, and the rest 4 to 5 seconds correspond to data transfer from Microblaze to DDR3 memory module. Nevertheless, the error rate of the data transfer is practically zero. In addition, after running the instrument under the specific design for a period of 1 month, no issues were reported. Therefore, the practical evaluation of this implementation adds an extra level of confidence concerning its reliability.

UART protocol incorporation constitutes the simplest and more straightforward solution for setting up a robust communication channel from PC to FPGA. Efficient UART controllers are already provided by Xilinx so no controller implementation is required. Microblaze can be used to receive the UART data and drive them accordingly. In our case, data are transmitted through AXI bus to DDR3 memory controller, which is also provided by Xilinx. The above facts, drastically reduce the time-to-market, as no major digital logic needs to be implemented. On the contrary, the largest portion of the communication development is limited to Microblaze C code. As a result, such a data transmission method can be potentially expanded to serve purposes other than waveform download, with minor interference in the overall design. In addition to that, the minimum of 2 FPGA IO pins, one receiving and one transmitting data, are needed. Therefore, it is offered for applications that require frequent modifications, when the FPGA design is still under development, and in cases where FPGA IO port saving is a major issue. However, the instantiation of Microblaze and DDR3 controller demand a large portion of FPGA resources which may not be available in some cases. Moreover, the highest achievable data transfer bandwidth is limited to approximately 0.11 MB/s which is fairly slow, especially for large data packet transfer.

USB2.0 to FIFO245 synchronous protocol, custom IP for data receive, AXI4 Lite for DDR3 save:

Using this implementation, significant increase in download speed is achieved. The download time of a 512 kBytes data packet ranges from 50 to 60ms which is of no comparison to the previous implementation through UART interface. It is important to note that this timing period includes both data download to FPGA and transfer to DDR3 memory module. The resource utilization is shown in Figure 37.



Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	LUT Flip Flop Pairs (63400)	Block RAM Tile (135)
ftdi_axi_0 (omni_bl...	446	642	32	0	211	446	0	273	0
mig_7series_1 (om...	5633	4624	6	0	1784	4970	663	2228	0

Figure 37: USB 2.0 TO FIFO245 synchronous protocol, custom IP for data receive, AXI4 Lite for DDR3 save resource utilization

The reasons that lead to such a great difference are the following**: 1) Upgrade of communication protocol:** The serial UART protocol, using AXI UARTlite IP can achieve a maximum data rate of 0.11 Mbytes / second (921600 Baud rate). On the other hand, the FIFO 245 parallel protocol can namely achieve data bandwidth of 57.2 MB/s (1 byte / clock cycle of 60MHz), which has been practically measured around 30 MB/s as the FT2232H chip that was used has not been able to deliver a byte on every clock cycle and stall time intervals were noticed.

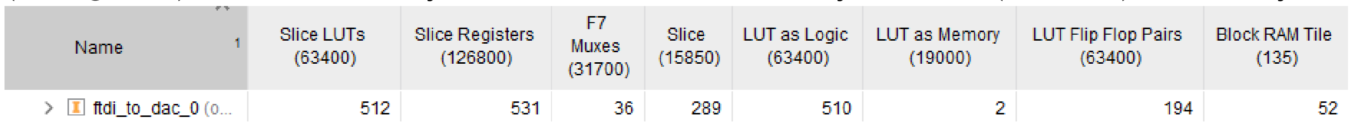
2) Microblaze replaced by custom IP: Soft-core utilization can reduce difficulty in FPGA design and increase time-to-market. On the other hand, an IP module can be precisely customized at a lower level so that less clock cycles are required for same logic functionality.

In order to verify the efficiency of the communication port, evaluation tests were applied. These tests included download of waveform data packets in various DDR3 memory locations. Reading back these data through a UART port (whose error rate is already measured as zero) showed that data are transferred with practically zero error rates which imply a successful implementation of the communication logic.

FT2232H FIFO 245 protocol is based on USB 2.0 and takes advantage of the USB 2.0 bandwidth more efficiently than UART protocol. It can achieve maximum data transfer rates of 30 MB/s. The implementation of an FT2232H chip controller is necessary for setting up such a communication method, as it is not provided by Xilinx. This digital logic can be customized to promote data directly to an external DDR3 memory, through Xilinx DDR3 controller. Therefore, the use of Microblaze softcore is unnecessary (although it can be included if needed). By excluding the instantiation of Microblaze significant FPGA resources are saved. The minimum number of FPGA IO ports that are needed is twelve, from which eight bits are intended for the 8-bit size FIFO 245 parallel bus and four pins constitute control signals. It is important to note that this communication method needs to be synchronous to an external clock generated by the FT2232H chip. This ultimately adds an extra clock domain to the overall design which is a negative consequence in many cases.

<u>USB2.0 to FIFO245 synchronous protocol, data stream:</u>

The above implementation has both advantages and disadvantages in comparison to the conventional use of external memory device for saving large data packets. In this design, download time is defined by the time duration from the moment PC initializes a packet download until the first two bytes are available in the internal FPGA block ram. This duration is approximated experimentally to range from 0.5 to 1 ms and is mostly dependent to PC operational load and FT2232H chip's reaction time to handling data transactions. Reducing download time to such negligible time lengths, is quite important for the overall instrument performance in terms of speed and practicality. In addition, FPGA resources (see Figure 38) used are drastically reduced as no external memory controller (like MIG) is necessary.



Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	LUT Flip Flop Pairs (63400)	Block RAM Tile (135)
ftdi_to_dac_0 (o...	512	531	36	289	510	2	194	52

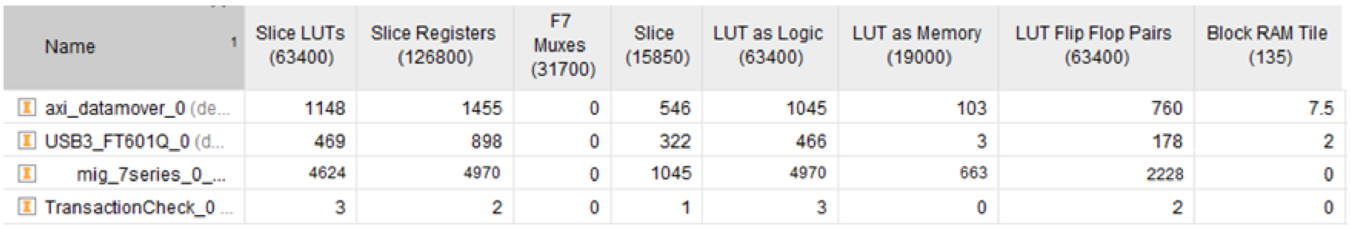
Figure 38: USB 2.0 to FIFO245 synchronous protocol, data stream resource utilization

On the other hand, some downsides and limitations occur. For example, no multiple waveforms can be available in the hardware at the same time, unless multiple instances of the same logic are generated inside the FPGA, an effect which burdens FPGA resources and design convenience and expandability. Furthermore, data processing capabilities inside the FPGA are quite limited by the fact that only one part of the data packet is available to the hardware, at one time. In addition to these issues, the fact that waveform reproduction gets highly dependent to PC operating system may be the cause of malfunctions and instabilities, in case a low-performance PC is used to control the instrument. Namely, this design while highly efficient in terms of speed and FPGA resources lacks customizability and adaptability to potential future modifications.

In this method FT2232H FIFO 245 protocol is also used. Therefore, the need for implementing an FT2232H communication controller is present here, too. This communication method represents the idea of direct data utilization, right after they are downloaded to the FPGA and is highly restricted to occasions where a big data packet can be used before the whole packet is available in hardware. No internal memory is needed, except a FIFO module of minor storage space, for the sake of synchronizing the FT2232H clock with the internal FPGA system clock. This way, minimum FPGA resources are occupied while download time is minor as data can be utilized by the time the first data word arrives to FPGA. It is important to note that each data packet can only be used once as all information are disposed after being processed. Overall, this method is appropriate for streaming interfaces where data are serially processed while no memory is needed and high download speed is a major factor. Additionally, it can find application in cases where FPGA resources are limited.

<u>USB 3.0 to FIFO 245 synchronous, custom IP for data receive, AXI4 Stream for DDR3 write</u>:

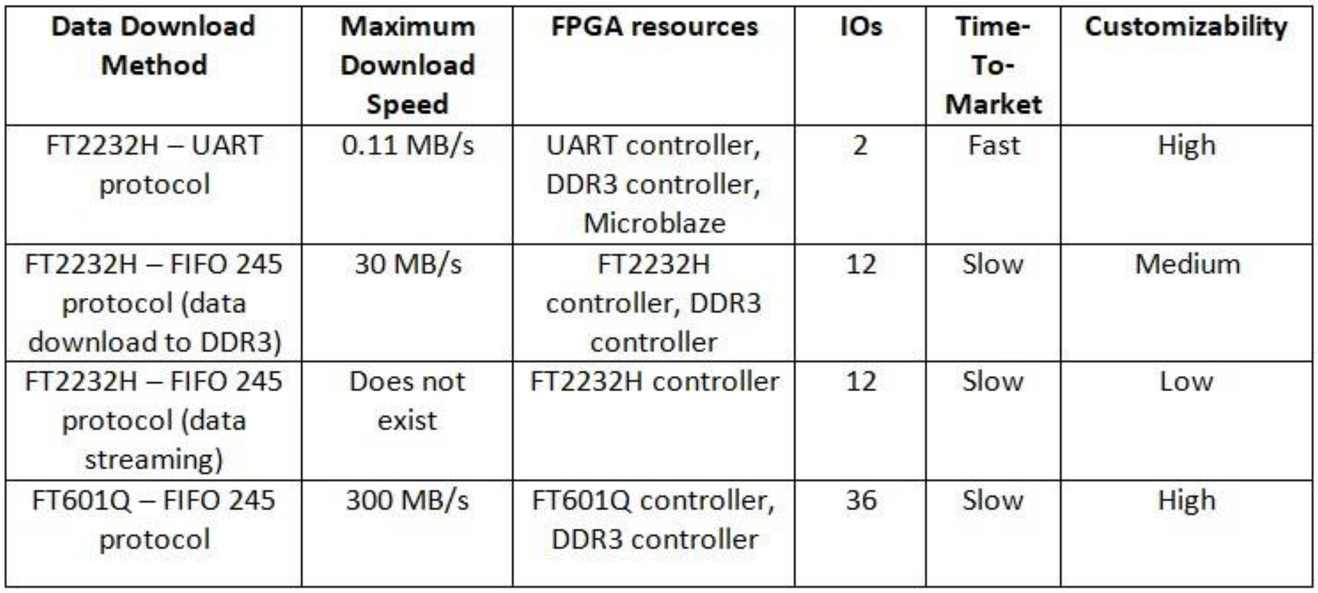
The use of FT601Q for incorporating USB 3.0 connectivity to FPGA design has quite rewarding results. The average data download speed is approximated at 300 MB/s, which sets the required download time of a 512kB waveform, to 1.5 ms. Data transfer to DDR3 has been significantly reduced by the use of AXI4 stream and is approximated around 0.5 ms for the same data packet. This data bandwidth was measured by means of oscilloscope and is defined by the duration between transaction's starting and ending pulse which are controlled by custom IP. PC software timers also confirmed these measurements. In order to verify the errorless data transfer similar evaluation methods as previous implementations are used. Data are sent to FPGA, saved to DDR3 and received back to PC through UART interface, in a loop process. No errors were detected, a fact that implies successful implementation of both FPGA logic and PCB design. This method both combines high speed data download and allows use of external DDR3 memory, however is not conservative in terms of FPGA resources (see Figure 39).



Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	LUT Flip Flop Pairs (63400)	Block RAM Tile (135)
axi_datamover_0 (de...	1148	1455	0	546	1045	103	760	7.5
USB3_FT601Q_0 (d...	469	898	0	322	466	3	178	2
mig_7series_0_...	4624	4970	0	1045	4970	663	2228	0
TransactionCheck_0 ...	3	2	0	1	3	0	2	0

Figure 39: USB 3.0 to FIFO 245 synchronous, custom IP for data receive, AXI4 Stream for DDR3 write, resource utilization

This communication channel is based on USB 3.0 so it can achieve substantially higher bandwidth, with download speed reaching 300 MB/s, and thus, constitutes the fastest data transfer method that is presented here. FT601Q controller development and instantiation are necessary. Data are stored in an external DDR3 memory, so resources for memory controller are also occupied. The FT601Q chip includes a 32-bit parallel data bus. Consequently, by including the four additional control signals coming from the chip, a total of thirty-six FPGA IO pins are occupied by this implementation. The chip also includes a multichannel mode, where up to four communication channels can act independently from each other. This adds an extra level of customizability to the overall design, under the cost of slow time-to-market, as higher VHDL complexity is added to the FT601Q controller. Additionally, an extra clock domain is added to the FPGA design, as data transfer must be synchronous to a clock generated by the FT601Q chip. Overall, this method fits projects where saving IO pins is not a primary issue, while there are high demands concerning data bandwidth, communication channel multiplexing and design complexity.



Data Download Method	Maximum Download Speed	FPGA resources	IOs	Time-To-Market	Customizability
FT2232H – UART protocol	0.11 MB/s	UART controller, DDR3 controller, Microblaze	2	Fast	High
FT2232H – FIFO 245 protocol (data download to DDR3)	30 MB/s	FT2232H controller, DDR3 controller	12	Slow	Medium
FT2232H – FIFO 245 protocol (data streaming)	Does not exist	FT2232H controller	12	Slow	Low
FT601Q – FIFO 245 protocol	300 MB/s	FT601Q controller, DDR3 controller	36	Slow	High

Figure 40: Communication methods comparison

5.3 Digital waveform generation using FPGA

Sweep

The whole pipeline process, implemented for sweep signal generation, is synchronous to the sample rate clock of 12.5 MHz. Therefore, the sweep waveform is reproduced by DAC while being calculated, and thus no calculation delay exists. By adding extra pipeline stages, additional sweep parameterizations become available (27). For instance, with the aim of applying amplitude modifications to our signal an extra pipeline stage can be added in the end of the pipeline chain (right after sine calculation) which is responsible for modifying the output value according to the specified amplitude modification. Likewise, notch generation can be achieved by interfering in the frequency step of the frequency increment pipeline stage, so that the corresponding frequency range is excluded from sweep generation. This method of creating notches is not mathematically identical to eliminating the unwanted frequencies on the frequency domain (by means of Fast Fourier Transform) but it is experimentally feasible and effective.

In this design, a 14-bit input Digital-to-Analog converter, at a sample rate of 12.5 MHz, is used for the sweep reproduction in the real world. However, this sweep implementation can be easily customized to fit any DAC, regardless its digital input size and sample rate. Taking full advantage of DAC's potential is a decisive factor which ultimately determines the effectiveness of this implementation. To begin with, let's assume a more generic situation where the DAC that is used receives N-bit digital inputs at a sample rate F_s and the desirable sweep frequency values range from f_{min} to f_{max}. The main goal is for maximum resolution of sweep signal to be achievable, inside a specified range of frequencies. The minimum frequency f_{min} that needs to be replicated defines the required sine resolution, namely the number of sine points that should be available inside FPGA ROM (LUT table). The calculation of this value is quite straightforward and occurs by dividing the sample rate F_s with the minimum required sweep frequency f_{min}. This computation gives the ideal sampling size of a sine period. However, as already referred, only one quarter of a sine period is necessary to be stored. In addition to that, for the sake of memory addressing, it is more convenient for the total of sine points to be a power of two. Ultimately, the formula that gives the number of necessary sine quarter resolution is given below:

$$\mathrm{NextPowerOfTwo}\left(\frac{F_s}{f_{min} * 4}\right)$$

Furthermore, the maximum frequency f_{max}, must not exceed the half of the DAC's sampling rate, according to Nyquist theorem. Finally, sine points should be represented in binary vectors of N-1 bits. The most significant DAC input bit, responds to the sign of the corresponding sine point, and is calculated by some additional digital logic, using the phase register value.

For the purpose of this project, where $F_s = 12.5$ MHz and $f_{min} = 10$KHz, 512 points is the optimal sine quarter resolution. Storing more than the number of sine points that is defined by our minimum frequency is practically a waste of storage space as such a sine resolution cannot be experimentally utilized by our DAC.

FNF

Direct digital synthesis of an FNF waveform is based on the Fast Fourier Transform calculation inside the FPGA. Additional configuration capabilities can be accessed by generating some extra digital logic. More specifically, notch application can be realized by excluding frequency tones in the frequency domain, mainly setting to zero the unwanted values. Additionally, phase modulation for better signal power distribution can be easily modeled by interfering to the imaginary part of the frequency domain points of the FNF signal. Finally, amplitude modifications can occur either in the frequency domain, by changing the amplitude of the complex values, or in the time domain, in a similar manner as sweep amplitude modifications where applied.

As far as the timing specifications are concerned, for an IFFT of 2^{15} points, under the specified FFT core configuration, a delay of approximately 1ms is required. This duration defines the FNF calculation time as any extra signal customizations occur simultaneously with signal reproduction by DAC, in the real world.

Comparison of Direct Digital Synthesis by FPGA against PC software waveform generation:
In this diploma thesis, methods of digital waveform generation both by PC and by FPGA were implemented and tested. Each approach is characterized by different features, thus attentive examination of project needs and specifications is required to take the right decision every time.

One of the main differences is the precision of the calculation that can be potentially achieved in each case. The maximum precision that is achieved by PC software calculation is taken as the reference point to measure this variable. Waveform calculation in PC can eventually occur by means of double precision floating point numbers (64-bit representation of each waveform point). Considering the case of sweep generation, achieving such a resolution inside the FPGA would mean a substantial increase in the resources occupied by the sine wave LUT. On the other hand, in case of FNF waveform synthesis, it would be infeasible by the use of the Xilinx FFT module, as it cannot handle such high precision point representation. Thus, development and instantiation of a custom FFT module would be necessary, a task that would vastly increase the time-to-market and design complexity. Therefore, it is evident that signal resolution comes at no cost in the software waveform synthesis while highly affects the FPGA design in terms of resources, ease of implementation and practicality. It is important to note that, signal precision requirements are always relevant to the project needs and are ultimately defined by the available hardware's capability of representing the synthesized waveform in the real world. It would be meaningless to implement a high resolution waveform calculation method which would be subsequently truncated by the Digital to Analog converter and thus would not have a practical reason of existence.

Furthermore, calculation time, resource utilization and overall design robustness and reliability are some extra points of interest that can accept further observation. To begin with, FPGA devices allow direct digital synthesis in a pipelining manner and thus real-time signal generation. Therefore, no calculation time is necessary and signal is generated and reproduced by DAC on the same time. This is not possible in all cases. For instance, FNF signal synthesis requires the operation of Fourier Transform which adds a calculation delay to the overall process. However such a delay is of no comparison to the inevitable calculation delay and download time required in case of signal generation by software. The fact that signal generation is highly dependent to communication of FPGA with external devices (PC, memory modules etc.) adds extra levels of potential instrument failure. On the contrary, direct digital synthesis by FPGA is a far more trustworthy approach and should be adopted when the product is on its final stage of mass production. Finally, the major advantage of software signal synthesis is the fact that it allows direct and effective application of potential modifications and reinforces the trial-and-error procedure, especially in the early stages of the instrument's development.

6 Lessons learned towards productization

Throughout the long-term process of Omnitrap development, constant upgrades and experimental testing, some conclusions are drawn concerning the ideal final implementation of an interface that could set the bases for potential instrument‘s mass production in the future. The main conditions that such a design should satisfy, as well as corresponding actions that could contribute to this goal, are described below:

- **It should effectively manipulate the instrument components while giving user-friendly and comprehensive high-level access to Omnitrap functionalities.** All in all, it is always important to notify that the instrument is intended for laboratory use by specialized scientists like biologists and chemists, who are unaware of Omnitrap's internal design and working principles. To this end, instrument control via the already mentioned sequence of commands should be modified and upgraded. More specifically, user capabilities through software should be condensed to some predefined experimental processes, that each one represents a unique sequence of commands with a specified purpose. These discrete experiments should have the form of black boxes in the software's user interface, where the term "black box" is used to describe an experimental procedure with a corresponding name and no further details provided on how it is implemented. In this way, user is no longer bothered to generate high precision series of actions by itself in order to use the instrument. Moreover, the instrument's basic principles are protected from the public view. Finally, restricting access to instrument's low-level functionalities highly reduces the possibility of hardware damage due to misuse by unspecialized users and make the process of ensuring both users' and instrument's safety much easier.
- **It has to be robust and resistant to long lasting experimental cycles.** It is a fact that such laboratory instrumentation is common to remain in operation for long periods of time, like days or weeks and in most cases, system reset can lead in significant loss of time and even irreplaceable analyte samples. Our system control is based on the communication between PC and FPGA. The use of a PC for configuring the basis of parameters of every experimental process and receiving the corresponding results is by far the most practical for this purpose. Thus, a communication port between PC and FPGA should always exist and act as a mediator between Omnitrap and user. However, the responsibilities of PC software should be limited to high-level experiment configuration, processing of experiment results and visual representations. For multiple reasons that include human mistakes or operating system failure, connection with PC can be lost. In such events, the instrument should not remain exposed while a second protection layer should exist. The FPGA constitutes a far more trustworthy and stable processing module than a PC, which should always be responsible for controlling the most important hardware components of Omnitrap, generating the real time data that are important for the experiment (i.e. isolation waveforms) and taking decisions in case of

malfunctions (i.e. a high temperature measurement). To this end, concerning the aspects of the design that were tested and improved in this diploma thesis, the superspeed USB 3.0 communication method that was followed is considered ideal for fast and efficient configuration data transfer, while the real time direct digital synthesis of Sweep and FNF isolation waveform by the FPGA would solve the major issue of large data packet download to FPGA and lead to a far more stable and consistent implementation.

- **Instrument's hardware assembly concerning both electronics and mechanical components should be simple, strictly organized and effective** so that the probability of errors during assembly process is diminished. For this purpose, concerning the electronics aspect of the instrument that were included in the realization of the current diploma thesis, USB 3.0 interface and Artix 7 FPGA should be rooted on already existing PCB instrument components so their functionality is not dependent to the production and purchase of development boards from other companies. In addition to that, assembly gets simpler and hardware malfunction probability is highly reduced.

7 Conclusion

To conclude, this thesis attempted to explore and upgrade the overall system that is responsible for the Omnitrap mass spectrometer device control. Several communication methods between PC and FPGA were implemented, so that the best approach is reached that fits appropriately the needs of the experimental process. In addition, for the purposes of ion isolation through excitation, signal direct digital synthesis methods using PC were also explored and compared. All implementations were tested under instrument operation for sufficient timing periods, a fact that adds an extra level of assurance for their efficiency and functionality. Undoubtedly, the overall system is still under development. There is always room for improvements towards productization.

8 Bibliography

1. **Scientific, Thermo Fisher.** https://www.thermofisher.com/gr/en/home/industrial/mass-spectrometry/mass-spectrometry-learning-center/mass-spectrometry-applications-area.html. *https://www.thermofisher.com/.* [Online] Thermo Fisher Scientific.

2. **DIMITRIS PAPANASTASIOU, EMMANUEL RAPTAKIS.** *Segmented Linear Ion Trap for Enhanced Ion Activation and Storage. US20170221694A1* United States, August 3, 2017.

3. en.wikipedia.org. [Online]

4. www.fasmatech.com. [Online]

5. *Analysis of molecular isotopic structures at high precision and accuracy by Orbitrap mass spectrometry.* **John Eiler a, ∗, Jaime Cesarb, Laura Chimiaka, Brooke Dallasa, Kliti Griceb,Jens Griep-Ramingc, Dieter Juchelkac, Nami Kitchena, Max Lloyda, Alexander Makarovc, Richard Robinsd, Johannes Schwietersc.** Pasadena, USA : Elsevier, 2017.

6. **Kowalewski, F. Salewski and S.** *Exploring the Differences of FPGAs and Microcontrollers for their Use in Safety-Critical Embedded Applications.* Antibes Juan-Les-Pins : 2006 International Symposium on Industrial Embedded Systems, 2006. 10.1109/IES.2006.357483.

7. *Stored waveform inverse Fourier transform (SWIFT) ion excitation in trapped-ion mass spectometry: Theory and applications.* **Shenheng Guan, Alan G. Marshall.** s.l. : Elsevier, International Journal of Mass Spectrometry and Ion Processes , 1999. S0168-1176(96)04461-8.

8. *Filtered noise field signals for mass-selective accumulation of externally formed ions in a quadrupole ion trap.* **Douglas E. Goeringer, Keiji G. Asano, Scott A. McLuckey, Don. Hoekman, and Steven W. Stiller.** s.l. : Analytical Chemistry, 1994 . 10.1021/ac00075a001.

9. *Universal Asynchronous Receiver and Transmitter (UART).* **Umakanta Nanda, Sushant Kumar Pattnaik.** Coimbatore, India : International Conference on Advanced Computing and Communication Systems (ICACCS), 2016. 10.1109/ICACCS.2016.7586376.

10. *AMBA AXI and ACE Protocol.* s.l. : ARM. IHI 0022D (ID102711).

11. *AXI UART Lite v2.0.* s.l. : Xilinx, 2017. PG142.

12. *AXI Interconnect LogiCORE IP Product Guide.* s.l. : XIlinx, 2017. PG059.

13. *Microblaze Processor Reference Guide.* s.l. : Xilinx, 2018. UG984.

14. *7 Series FPGAs Memory Interface Solutions User Guide.* 2012.

15. *Dynamic Data Acquisition system using FT2232H.* **Parmar Pranav, Savitanandan Patidar, Mayursinh Thakor, Dhaval Patel.** New Delhi : IEEE, 2015. 10.1109/INDICON.2015.7443321.

16. *FT2232H Dual High Speed USB to Multipurpose UART/FIFO IC Datasheet.* s.l. : FTDI. FT_000061.

17. *Formal Verification of Synhcronizers.* Kapschitz, Tsachy : s.n., 2005. 10.1007/11560548_31.

18. *Integrated Logic Analyzer v6.1.* s.l. : Xilinx, 2016. PG172.

19. www.usb.org. [Online]

20. *Research on data transmission application based on USB3.0 bridge chip on FPGA.* **Dezhuang Ma, Lunhui Deng.** Beijing, China : MATEC Web of Conferences, 2018, Vol. 189. 10.1051/matecconf/201818904002.

21. *FT600Q-FT601Q IC Datasheet.* s.l. : FTDI. FT_001118.

22. *AXI DataMover v5.1.* s.l. : Xilinx, 2017. PG022.

23. *Implementation of DDS Chirp Signal Generator on FPGA.* **Heein Yang, Sang-Burm Ryu, Hyun-Chul Lee, Sang-Gyu Lee, Sang-Soon Yong, Jae-Hyun Kim.** Busan : International Conference on Information and Communication Technology Convergence (ICTC), 2014. 10.1109/ICTC.2014.6983343.

24. *A Study on Look-up Table Based Sine Wave Generation.* **A.J. Salazar, G. Bahubalindruno, G.R. Locharla, H.S. Mendonça, J.C. Alves, J.M. Da Silva.** Porto, Portugal : s.n., 2011.

25. *The Design and Implementation of FFT Algorithm Based on The Xilinx FPGA IP Core.* **Zhu Jin, Luo Jun, Zhang Shuang.** Guilin, China : Atlantis Press, 2012.

26. *Fast Fourier Transform v9.1.* s.l. : Xilinx, 2020. PG109.

27. *Using FPGA to Implement a N-channel Arbitrary Waveform Generator with Various Add-on Functions.* **Jen-Wei Hsieh, Guo-Ruey Tsai, Min-Chum Lin.** Yun-Kan City, Taiwan : IEEE. 10.1109/FPT.2003.1275761.