Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

τομεας τεχνολογιας πληροφορικης και υπολογιστων

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων

Scalable Ultrafast Ultrasound Delay-and-Sum Beamforming design on an FPGA

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Βασίλειου Κυπριώτη

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2023

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Scalable Ultrafast Ultrasound Delay-and-Sum Beamforming design on an FPGA

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Βασίλειου Κυπριώτη

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 29η Σεπτεμβρίου, 2023.

.....	.....	.....
Δημήτριος Σούντρης	Γεώργιος Ματσόπουλος	Σωτήριος Ξύδης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Χ.Π.Α

Αθήνα, Σεπτέμβριος 2023

.....

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

στην οικογένεια μου

Περίληψη

Η υπερηχογραφική απεικόνιση έχει καθιερωθεί ως ένας από τους κυρίαρχους τρόπους απεικόνισης για την διάγνωση και τη θεραπεία ιατρικών παθήσεων, λόγω της μη επεμβατικής φύσης της, καθιστώντας την ακίνδυνη για τον ασθενή. Αυτή η μέθοδος απεικόνισης βασίζεται στο φαινόμενο Doppler, εκπέμποντας και λαμβάνοντας ακουστικά κύματα υψηλής συχνότητας (1-20 MHz), ενώ υποστηρίζει frame rates που δεν ξεπερνούν τα 10-40 fps. Ωστόσο, δεδομένου ότι παρατηρείται μια πληθώρα δυναμικών φαινομένων στο εσωτερικό του ανθρώπινου σώματος, η ανάγκη καταγραφής παροδικών γεγονότων με την αύξηση του frame rate της εφαρμογής καθίσταται επιτακτική. Αυτή η μέθοδος απεικόνισης με υψηλά frame rates είναι γνωστή ως Υπερταχεία Υπερηχοραφική Απεικόνιση (Ultrafast Ultrasound Imaging). Τα υψηλά frame rates θα μπορούσαν να εφαρμοστούν με την αύξηση του beamforming rate του συστήματος υπερήχων, ένα στοιχείο του συστήματος που συχνά εισάγει bottlenecks κατά τις απόπειρες ενσωμάτωσης της υπερταχείας απεικόνισης. Η παρούσα διπλωματική εργασία θα επικεντρωθεί στην υλοποίηση ενός beamforming kernel σε μια συσκευή FPGA, μια ελκυστική επιλογή λόγω της configurable υπολογιστικής αρχιτεκτονικής τους, που επιτρέπει τεράστιες παράλληλες διεργασίες σε πραγματικό χρόνο. Ο βασικός στόχος είναι η δημιουργία ενός hardware πυρήνα που λαμβάνει raw RF σήματα υπερήχων, υλοποιεί τον beamforming αλγόριθμο πάνω σε αυτά, αθροίζει τις συνεισφορές από διαφορετικές γωνίες και τις αποθηκεύει για περαιτέρω επεξεργασία. Ενώ προηγούμενες προσεγγίσεις έχουν σχεδιάσει επιτυχώς τέτοιους πυρήνες σε συσκευές FPGA, είτε στερούνται επαρκούς frame rate και beamforming rate, είτε στερούνται επαρκούς ποσότητας probe channels είτε υστερούν σε scalabity. Πιο συγκεκριμένα, προτείνουμε δύο ultrafast beamforming πυρήνες σε FPGA - έναν που εφαρμόζει interpolation όπου οι εκτιμήσεις είναι ίσες με τη μέση τιμή των δύο γειτονικών υπερηχογραφικών δειγμάτων και έναν που εφαρμόζει linear interpolation. Το dataset που αξιοποιήθηκε για beamforming συλλέχθηκε από το IEEE IUS 2016 Plane-wave Imaging Challenge in Medical UltraSound : PICMUS. Υποστηρίζοντας ultrafast beamforming με 128 κανάλια αισθητήρα, η πρώτη έκδοση του προτεινόμενου πυρήνα υποστηρίζει συχνότητα επανάληψης ακουστικών παλμών (PRF) ίση με 11,116 KHz και beamforming rate ίσο με 2,173 GSamples/s. Από την άλλη, η δεύτερη έκδοση του προτεινόμενου πυρήνα υποστηρίζει συχνότητα επανάληψης ακουστικών παλμών (PRF) ίση με 10,014 KHz και beamforming rate ίσο με 1,957 GSamples/s. Εδραιώθηκε το scalability και για τα δύο αυτά designs, όσον αφορά το πλήθος των probe channels, το πλήθος των εκπεμπόμενων plane waves, καθώς και μια εσωτερική παράμετρο του beamformer μας, η οποία θα αναλυθεί αργότερα σε αυτή τη διπλωματική.

Λέξεις Κλειδιά — Ultrafast Ultrasound Imaging, Beamforming, Delay and Sum, High Frame Rates, FPGA, High Level Synthesis, Vitis

Abstract

Ultrasound imaging has been established as a dominant imaging modality for evaluation and treatment of medical conditions, due to its non-invasive nature that is harmless for the patient. This imaging method is based on the ideas of the Doppler effect, producing and receiving high frequency acoustic waves (1-20 MHz), while supporting frame rates that were limited to 10-40 fps. However, since there is a plethora of dynamic responses observed inside the human body, the need to capture transient events has emerged, by increasing the frame rates of the application. This high frame rate imaging method is known as ultrafast ultrasound imaging. High frame rates could be supported by improving the beamforming rate of the ultrasound system, a component that often introduces bottlenecks during attempts of integrating ultrafast imaging. This thesis will focus on implementing an ultrafast beamforming kernel on an FPGA device, a potential candidate due to their configurable computational architecture that enables immense parallel, real-time processes. The main goal is to create a hardware kernel that receives raw ultrasound signals and beamforms them, compounds the contributions from different angles and saves them for further processing. While previous approaches have managed to design such kernels on FPGA devices, they either lack in adequate frame rate and beamforming rate, amount of probe channels or scalability. More specifically, we propose two ultrafast beamforming FPGA kernels; one that applies an interpolation where estimations equal to the mean value of the two adjacent ultrasound samples, and one that applies linear interpolation. The dataset used for beamforming was collected by the IEEE IUS 2016 Plane-wave Imaging Challenge in Medical UltraSound : PICMUS. While supporting an ultrafast beamforming process with 128 probe channels, the first version of the proposed kernel supports a pulse repetition frequency of 11.116 KHz and a beamforming rate of 2.173 GSamples/s. On the other hand, the second version of the proposed kernel supports a pulse repetition frequency of 10.014 KHz and a beamforming rate of 1.957 GSamples/s. The scalability for both of these designs was established, regarding the amount of utilized probe channels, the amount of plane waves, as well as an internal parameter of our beamformer, that will be explained later in this thesis.

Keywords — Ultrafast Ultrasound Imaging, Beamforming, Delay and Sum, High Frame Rates, FPGA, High Level Synthesis, Vitis

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τους καθηγητές κ. Δημήτριο Σούντρη και κ. Χρήστο Στρύδη για την καθοδήγηση, τις γνώσεις που μου μετέδωσαν και την ευκαιρία να εκπονήσω ένα θέμα διπλωματικής άκρως ενδιαφέρον για εμένα. Θα ήθελα επίσης να ευχαριστήσω τον υποψήφιο διδάκτορα Γέωργιο Σμάραγδο του Erasmus University of Rotterdam για την υποστήριξη και την προθυμία να μου προσφέρει την χρήσιμη βοήθεια του όποτε αυτή ήταν απαραίτητη, καθώς και τα υπόλοιπα μέλη του Department of Neuroscience του Erasmus MC που ήταν πάντα διαθέσιμοι να μου επιλύσουν οποιαδήποτε απορία. Ευχαριστώ επίσης τους στενούς μου φίλους για την υπομονή που υπέδειξαν και την συμπαράστασή τους όλα αυτά τα χρόνια. Τέλος, το μεγαλύτερο ευχαριστώ το οφείλω στους γονείς μου Γεώργιο και Μαρία, καθώς και στον αδερφό μου Γιάννη, που ήταν πάντα δίπλα μου σε κάθε κεφάλαιο της ζωής μου, και δίχως τους οποίους δεν θα βρισκόμουν σε αυτή την θέση.

Κυπριώτης Βασίλειος
Σεπτέμβρης 2023

Contents

Figure List

Table List

Chapter 1

Εκτεταμένη Ελληνική Περίληψη

1.1 Εισαγωγή

Τις τελευταίες δεκαετίες, η απεικόνιση με υπερήχους έχει γίνει μια σημαντική, ταχέως αναπτυσσόμενη μέθοδος ιατρικής απεικόνισης. Λόγω της μη επεμβατικής της φύσης, αποτελεί ένα ευρέως διαδεδομένο διαγνωστικό εργαλείο που είναι ασφαλές για τον ασθενή. Η αβλαβής, μη ιονισμένη ακτινοβολία που χρησιμοποιεί καθώς και το χαμηλό της κόστος, της προσδίδει πλεονέκτημα έναντι άλλων δημοφιλών τεχνικών απεικόνισης, όπως η τομογραφία X-ray ή η μαγνητική τομογραφία (MRI)[1]–[3].

Τα συμβατικά συστήματα υπερήχων βασίζονται στη μετάδοση εστιασμένων υπερηχητικών ακτίνων. Αυτή η τεχνική οδηγεί σε frame rates που περιορίζονται στα 10-40 fps [4]. Ωστόσο, η σημαντική αύξηση των frame rates της απεικόνισης με υπερήχους θα θέσει τα θεμέλια για νέες τεράστιες εφαρμογές στην ιατρική απεικόνιση. Μελλοντικά συστήματα τρισδιάστατης απεικόνισης σε πραγματικό χρόνο ή βελτιωμένη παρακολούθηση της κίνησης της καρδιάς καθ' όλη τη διάρκεια του καρδιακού κύκλου θα είναι εφικτά. Θα επιτραπεί επίσης η απεικόνιση ταχέως μεταβαλλόμενων γεγονότων, όπως η σε βάθος απεικόνιση της αιμοδυναμικής του εγκεφάλου. Τα προαναφερθέντα είναι δυνατά με μια τεχνική υπερήχων που ονομάζεται Ultrafast Ultrasound Imaging [4], [5].

Η εναλλακτική τεχνική που προτείνεται για την ανακατασκευή εικόνων υπερήχων με υψηλά frame rates είναι γνωστή ως Compounding Plane Wave Imaging (CPWI). Η CPWI εξισορροπεί την ποιότητα της εικόνας (π.χ. χωρική ανάλυση, λόγος σήματος προς θόρυβο (SNR)) και τα frame rates απεικόνισης ενώ χρησιμοποιείται συνήθως σε εφαρμογές που απαιτούν υπερηχογράφημα με πολύ υψηλή ταχύτητα. Πρόκειται για μια πολύπλοκη τεχνική με υψηλούς ρυθμούς δεδομένων και υψηλούς υπολογισμούς διαμόρφωσης ηχητικών κυμάτων ή beamforming [6]–[8].

Δεδομένου ότι το beamforming είναι ένα θεμελιώδες, αναπόσπαστο μέρος ενός συστήματος υπερήχων, είναι προφανές ότι ένα σύστημα υπερήχων που υποστηρίζει υψηλά frame rates πρέπει να είναι σε θέση να επιτύχει beamforming rates που να συμβαδίζουν με την υπερταχεία απεικόνιση. Η παρούσα Διπλωματική εργασία θα επικεντρωθεί στη σχεδίαση ενός ultrasound beamforming πυρήνα που προσαρμόζεται σε υψηλά frame rates. Συγκεκριμένα, ο πυρήνας θα αναπτυχθεί και θα δοκιμαστεί σε ένα Field Programmable Gate Array.

1.1.1 Στόχοι της παρούσας Διπλωματικής Εργασίας

Με κίνητρο τα παραπάνω, η παρούσα διπλωματική διερευνά το πρόβλημα της σχεδίασης ενός αποδοτικού beamforming πυρήνα σε hardware, για την υποστήριξη υψηλών frame rate. Ο πυρήνας θα λαμβάνει δείγματα υπερήχων, θα εκτελεί τον αλγόριθμο beamforming πάνω σε αυτά τα δείγματα, θα συνδυάζει τις συνεισφορές από διαφορετικές γωνίες και θα αποθηκεύει τα αποτελέσματα στη DDR της FPGA, για περαιτέρω επεξεργασία. Ο ultrafast beamforming πυρήνας αναμένεται να επιτύχει beamforming rate της τάξης των GSPS, προκειμένου να υποστηρίζει high frame rates (στην περιοχή των KHz). Ο

beamforming αλγόριθμος που εφαρμόζει ο προτεινόμενος πυρήνας είναι ο αλγόριθμος Delay and Sum (DAS). Στόχος της διπλωματικής αυτής είναι η διερεύνηση για σχήματα παραλληλισμού που επιτρέπουν ταχύτερο beamforming, ενώ παράλληλα σέβονται τους διαθέσιμους πόρους του FPGA.

Συνοπτικά, η παρούσα διπλωματική εργασία επιδιώκει να συμβάλει στα εξής:

- Να διερευνήσει τις δυσκολίες της υλοποίησης ενός ultrafast beamformer σε μια FPGA συσκευή. Να προσδιορίσει τα bottlenecks της μνήμης, καθώς και τις απαιτήσεις σε μαθηματικούς υπολογισμούς.
- Να προσδιορίσει τα σχήματα παραλληλισμού του αλγορίθμου Delay and Sum, που επιτρέπουν ισορροπία μεταξύ της κατανάλωσης πόρων και της υψηλής απόδοσης.
- Να σχεδιάσει μια beamforming αρχιτεκτονική σε ένα μόνο FPGA, η οποία επιτυγχάνει υψηλά frame rates, επιτρέποντας την ενσωμάτωση υψηλών ρυθμών σκπομπής και λήψης υπερήχων. Το design θα διερευνηθεί όσον αφορά το scalability του, ώστε να επιτευχθεί η πρόβλεψη της συμπεριφοράς του για διαφορετικά υπερηχητικά probe και περιοχές απεικόνισης.
- Να διερευνήσει την δυνατότητα ενσωμάτωσης linear interpolation, καθώς και phase rotation on-chip. Αυτές οι δύο διαδικασίες θα επιτρέπουν αντίστοιχα, καλύτερη ακρίβεια beamforming και τη δυνατότητα beamforming σε IQ modulated δείγματα υπερήχων.
- Να αξιολογήσει τον προτεινόμενο πυρήνα όσον αφορά τα υποστηριζόμενα frame rates, το πλήθος των καναλιών και το τελικό μέγεθος της εικόνας, καθώς και την ποιότητα των ανακατασκευασμένων εικόνων.
- Να προσφέρει μια σταθερή βάση για να προτείνει μελλοντικές αναβαθμίσεις και βελτιώσεις όσον αφορά την υλοποίηση ενός υπερταχύτατου beamformer, σε ένα FPGA.

1.2 Σχετική Βιβλιογραφία

Η κατάσταση της τεχνολογίας γύρω από την απεικόνιση υπερήχων έχει ήδη εισαγάγει έναν αριθμό από beamformers υλοποιημένων σε hardware που υποστηρίζουν απεικόνιση με υψηλά frame rates. Μια από τις πρώτες προσπάθειες προτάθηκε από τους Jayaraj U Kidav και Sreejeesh S. G [2]. Το FPGA design τους υποστηρίζει 128 κανάλια και σχηματίζει 28 beams ανά μετάδοση ενός plane wave. Αυτή η αρχιτεκτονική καταφέρνει να επιτύχει beamforming rate ίσο με 1120 MSPS και 714 frames per second, αλλά με μικρό μέγεθος εικόνας (64×181 pixels).

Επιπλέον, οι Enrico Boni et al. [5] πρότειναν hardware αρχιτεκτονική μιας συνεχούς, real-time εφαρμογής απεικόνισης υπερήχων στην ερευνητική πλατφόρμα ULA-OP 256. Αυτό το σύστημα αποτελείται από τέσσερα Delay and Sum compute units που λαμβάνουν ultrasound δείγματα από μια Analog Front-End (AFE) πλακέτα. Κάθε AFE ελέγχει 32 κανάλια και κάθε DAS unit επεξεργάζεται μία γραμμή εικόνας, εφαρμόζοντας quadratic interpolation, apodization weights και coherent άθροιση. Αυτή η beamforming αρχιτεκτονική έχει σχεδιαστεί σε ένα FPGA της οικογένειας ARRIA V GX και παράγει εικόνες χρησιμοποιώντας 32 κανάλια. Το σύστημα αυτό καταφέρνει να επιτύχει συνεχές Pulse Repetition Frequency (PRF) ίσο με 3800 Hz, ένα μέγιστο Frame Rate (FR) ίσο με 1100 Hz, ανάλογα με τον αριθμό των plane waves, και beamforming rate ίσο με 467 MSamples ανά δευτερόλεπτο.

Μια παρόμοια σχεδίαση παρέχεται από τους Nicholas A. Campbell et al. [9], οι οποίοι προτείνουν έναν dual-mode ultrafast beamformer που λειτουργεί σε πραγματικό χρόνο και υποστηρίζει ταυτόχρονα είτε focused είτε ultrafast imaging. Ανάλογα με την επιλεγμένη λειτουργία, αυτός ο beamformer μπορεί να παράγει εναλλακτικά focused εικόνες υψηλότερης ποιότητας και πολλές ultrafast εικόνες χαμηλής ποιότητας. Το σύστημα που περιγράφεται από τους συγγραφείς αποτελείται από δέκα Kintex 7 FPGAs (xc7k70tfbg484-2; Xilinx Inc., San Jose, CA, USA). Η τελική εικόνα έχει μέγεθος 768×128 από 64 probe channels. Από την άλλη πλευρά, η υπερταχεία λειτουργία λειτουργεί με παρόμοιο τρόπο, αλλά παράγει μια εικόνα μεγέθους 384×64. Για να συμπιέσουν τις καθυστερήσεις του Delay and Sum και να τις αποθηκεύσουν αποτελεσματικά στη μνήμη, οι συγγραφείς χρησιμοποιούν μια παραλλαγή της μεθόδου κωδικοποίησης delta. Εκφράζουν τις καθυστερήσεις σε σχέση με την καθυστέρηση που αντιστοιχεί στο πρώτο pixel, δημιουργώντας μια γραμμική προσέγγιση για αυτές τις καθυστερήσεις, σε σχέση με την

θέση τους. Το σφάλμα για κάθε pixel περιορίζεται μεταξύ -0.83 ns και +0.83 ns, για τα περισσότερα pixels. Αυτή η αρχιτεκτονική επιτυγχάνει ultrafast frame rate ίσο με 875 Hz για 16 επίπεδα κύματα, 8 probe elements και μέγεθος εικόνας 64×384. Κατά προσέγγιση, χρησιμοποιεί πάνω από το 50% των πόρων της FPGA και επιτυγχάνει beamforming rate ίσο με 917 MSPS.

Σε αντίθεση με το προαναφερόμενο design, οι Zhengchang Kou et al. [7] πρότειναν έναν scalable ultrafast ultrasound beamformer που λειτουργεί σε ένα μόνο FPGA. Αν και αυτό το design δεν έχει ενσωματωθεί σε ένα πραγματικό σύστημα υπερήχων όπως τα προαναφερόμενα designs, επιτυγχάνει beamforming rate ίσο με 4.83 GSPS. Οι συγγραφείς χρησιμοποιούν έναν τρόπο επαναχρησιμοποίησης των τεράστιων ποσοτήτων των καθυστερήσεων, μειώνοντας σημαντικά το μέγεθος του delay profile. Αξίζει να σημειωθεί ότι ο συγκεκριμένος πυρήνας λειτουργεί μόνο με raw RF samples και εφαρμόζει μια λιγότερο ακριβή interpolation τεχνική στα δείγματα εισόδου, καθώς κάθε RF δείγμα προκύπτει από το μέσο όρο των δύο γειτονικών δειγμάτων. Αυτή η μέθοδος επαναχρησιμοποίησης του προφίλ καθυστέρησης βελτιώνει τη χρήση της μνήμης στο εσωτερικό της FPGA και απλοποιεί την αρχιτεκτονική μνήμης, ενώ ταυτόχρονα μειώνει το μέγεθος της επεξεργασίας δεδομένων, επιταχύνοντας σημαντικά τους υπολογισμούς. Ο πυρήνας είναι σε θέση να επιτύχει frame rate ίσο με 14865 Hz, με μέσο beamforming rate 4.83 GSPS για μέγεθος εικόνας 64×1280 και 64 κανάλια.

Ο πίνακας 1.1 συνοψίζει τους προαναφερθέντες hardware-based beamfomers, παρέχοντας μια σύγκριση όσον αφορά τα frame rates, τα MSamples ανά δευτερόλεπτο, το μέγεθος της εικόνας υπερήχων καθώς και το πλήθος των καναλιών που αξιοποιεί κάθε εφαρμογή.

Design	***PRF (Hz)***	***BF Rate (MSPS)***	***Frame Rate (Hz)***	***PW***	***Image Size***	***Aperture Size***
[2]	714	1120	714	1	64×181	64
[5]	3800	467	345	11	96×512	32
[9]	14,000	917	875	16	64×384	8
[7]	14,865	4830	14,865	1	64×1280	64

Table 1.1: Απόδοση των state of the art FPGA ultrafast beamformers

1.3 Implementation του Ultrafast Beamforming πυρήνα

1.3.1 Ορισμός του Προβλήματος

Η συγκεκριμένη ενότητα συνοψίζει τις προσπάθειές μας για την ανάπτυξη ενός επιταχυνόμενου, αυτόνομου beamforming kernel σε μια Alveo U200 FPGA συσκευή. Ο πυρήνας θα υποστηρίζει plane wave imaging σε συνδυασμό με coherent compounding. Στόχος είναι να δημιουργηθεί ένας πυρήνας που θα σέβεται τις προδιαγραφές ενός συστήματος υπερήχων, όπως η απόδοση, η κατανάλωση resources και ισχύος, ώστε να μπορεί να προσαρμοστεί σε ένα τέτοιο σύστημα με ελάχιστες τροποποιήσεις.

Δεδομένου ότι η πλειονότητα των δοκιμών μας πραγματοποιήθηκε με μια κάρτα επιτάχυνσης Alveo U200 Data Center Accelerator Card, προϊόν της Xilinx, χρησιμοποιήσαμε τα εργαλεία Vitis HLS και Vitis Unified Software Platform.

Σε γενικές γραμμές, ο στόχος για την beamforming εφαρμογή μας είναι η επεξεργασία κάθε frame να εκτελείται ταχύτερα από την παραγωγή και την καταγραφή του επόμενου frame. Αυτό σημαίνει ότι η εφαρμογή μας θα πρέπει να υποστηρίζει τα frames per second (FPS) του συστήματος λήψης, του οποίου η έξοδος μεταδίδεται στην είσοδο του πυρήνα μας. Για να πετύχουμε την συγκεκριμένη απόδοση στοχεύουμε στα εξής :

- Τον προ-υπολογισμό των delays off-chip και την φόρτωση τους στις εσωτερικές BRAM της συσκευής FPGA, καθώς ο on-chip υπολογισμός τους είναι χρονοβόρος και καταναλώνει έναν σημαντικά μεγάλο αριθμό από resources.

- Την αναδιαμόρφωση του beamforming αλγορίθμου Delay and Sum, ώστε να απαιτείται ο υπολογισμός και η αποθήκευση ενός σημαντικά μικρότερου πλήθους καθυστερήσεων.
- Η μείωση των επιπέδων ιεραρχίας των loops του αλγορίθμου από 3 σε 2 επίπεδα, με στόχο την ελάττωση των παράλληλων αθροισμάτων που πρέπει να υπολογιστούν σε έναν κύκλο ρολογιού και άρα την επιτάχυνση της εφαρμογής.

1.3.2 Προτεινόμενη αναδιαμόρφωση του αλγορίθμου Delay and Sum

Αρχικά, για τον υπολογισμό των αποστάσεων και καθυστερήσεων που είναι απαραίτητοι για τον αλγόριθμο Delay and Sum (DAS), απαιτούνται οι συντεταγμένες των pixels της εικόνας. Αναφερόμαστε σε αυτές τις δισδιάστατες συντεταγμένες ως πλέγμα και επιλέγονται από τον προγραμματιστή, ανάλογα με την επιθυμητή ποιότητα της ανακατασκευασμένης εικόνας. Ένας συνεπής τρόπος για τη δημιουργία ενός πλέγματος για την ανακατασκευασμένη εικόνα είναι ο ακόλουθος.

Ας υποθέσουμε πως οι συντεταγμένες των στοιχείων του probe μπορούν να υπολογιστούν από την εξίσωση $ch_s = ch_0 + \kappa \cdot d_{ch}$, όπου $\kappa \in [0, n_{channels} - 1]$ και ch_0, d_{ch} είναι χαρακτηριστικά του αισθητήρα υπερήχων. Τότε, επιλέγουμε τις εξής συντεταγμένες για τους δύο άξονες:

- $x_s = x_0 + \lambda \cdot d_{ch}$, όπου $\lambda \in [0, X_{size} - 1]$.
- $z_s = \lambda \cdot \frac{c_s}{f_s}$, όπου $\lambda \in [0, Z_{size} - 1]$, c_s η ταχύτητα του ήχου στο μέσο και f_s η συχνότητα συλλογής δειγμάτων RF από το ανακλώμενο ηχητικό κύμα.

Αυτές οι τρεις εξισώσεις για τις συντεταγμένες κάθε probe και κάθε καναλιού θα φανούν χρήσιμες ακολούθως, καθώς προσπαθούμε να εκμεταλλευτούμε τις γεωμετρικές ιδιότητες του χώρου απεικόνισης, ώστε να πετύχουμε πιο αποδοτικούς υπολογισμούς.

Η διαδικασία υπολογισμού των beamforming delays εξαρτάται μόνο από τη γεωμετρία της περιοχής απεικόνισης και τις ιδιότητες του probe. Μια τυπική προσέγγιση είναι ο εκ των προτέρων υπολογισμός των επιθυμητών καθυστερήσεων που απαιτούνται για το beamforming ενός μεμονωμένου frame και η φόρτωσή τους στη DDR της FPGA, πριν από την εκκίνηση του πυρήνα. Είναι σαφές ότι η προσωρινή αποθήκευση σε ορισμένες FPGA δεν είναι καν δυνατή, καθώς το πλήθος των συγκεκριμένων καθυστερήσεων ξεπερνάει την χωρητικότητα των BRAMs. Για να αντιμετωπίσουμε αυτό το πρόβλημα μνήμης αποφασίσαμε να αναμορφώσουμε τον αλγόριθμο Delay and Sum ως ακολούθως, χρησιμοποιώντας την ιδέα του delay compresion που εισήγαγαν οι West et al. [10]

Ο χρόνος για να διανύσει το ακουστικό κύμα την απόσταση μεταξύ ενός συγκεκριμένου στόχου με συντεταγμένες $X_s = (x_s, z_s)$ και ενός συγκεκριμένου στοιχείου καναλιού $X_i = (x_i, z_i)$ ισούται με :

$$\tau(X_s, X_i, \theta) = \frac{d_{TX}(X_s,\theta)+d_{RX}(X_s,X_i)}{c}, \text{ όπου}$$
$$d_{TX}(X_s, \theta) = z_s \cdot cos(\theta) + x_s \cdot sin(\theta) \text{ και } d_{RX}(X_s, X_i) = \sqrt{(x_s - x_i)^2 + z_s^2}.$$

Για να μειώσουμε τις διαστάσεις των υπολογισμών αποφασίζουμε να αντικαταστήσουμε τη διαφορά $x_s - x_i$ με μία μόνο μεταβλητή Δx.

Οι νέες αποστάσεις εκπομπής και λήψης, συμπεριλαμβανομένης της νέας μεταβλητής Δx, αναθεωρούνται ως εξής:

- $d_{RX}(X_s, X_i) = \sqrt{\Delta x^2 + z_s^2}$
- $d_{TX}(X_s, X_i, \theta) = z_s \cdot cos(\theta) + (\Delta x + x_i) \cdot sin(\theta)$

Ωστόσο, ενώ μειώσαμε τις μεταβλητές της απόστασης λήψης από τρεις σε δύο, προσθέσαμε μία επιπλέον διάσταση στην απόσταση εκπομπής, πράγμα που σημαίνει ότι τελικά οι διαστάσεις που καθορίζουν τις τιμές της καθυστέρησης εξακολουθούν να είναι τρεις. Για να ξεπεράσουμε αυτό το τελευταίο εμπόδιο θα ξαναγράψουμε την απόσταση εκπομπής ως εξής :

$$d_{TX}(X_s, X_i, \theta) = [z_s \cdot cos(\theta) + \Delta x \cdot sin(\theta)] + x_i \cdot sin(\theta) \Rightarrow d_{TX}(X_s, X_i, \theta) = d'_{TX}(X_s, \theta) + x_i \cdot sin(\theta)$$

Επομένως, εάν όλες οι διαφορετικές τιμές $d'_{TX}(X_s, \theta)$, $x_i \cdot sin(\theta)$ είναι αποθηκευμένες στο εσωτερικό της DDR και συνδυάζονται σωστά ανά loop στα διάφορα pixels και κανάλια, οι επιθυμητές καθυστερήσεις για κάθε ζεύγος pixel-καναλιού μπορούν ουσιαστικά να υπολογιστούν, με ένα απλό άθροισμα.

Δεδομένου ότι οι τιμές $x_i \cdot sin(\theta)$ είναι ελάχιστες σε σύγκριση με το σύνολο των τιμών που πρέπει να φορτώσουμε στην FPGA εκ των προτέρων ($n_{channels} \times n_{angles}$), θα μπορούσαμε επίσης να τις αποθηκεύσουμε ως ROM μέσα στην FPGA αντί να τις αποθηκεύσουμε στην DDR. Οι υπόλοιπες τιμές που απαιτούνται για τον υπολογισμό των επιθυμητών καθυστερήσεων θα φορτωθούν από την DDR στις εσωτερικές BRAM της FPGA πριν από την έναρξη της επεξεργασίας κάθε frame. Αυτό σημαίνει ότι πριν ξεκινήσει το beamforming για κάθε frame, πρέπει να φορτωθεί στις Block RAMs της FPGA ένα ποσό $32 \times Z_{size} \times (X_{size} + n_{channels} - 1)$ bits -υποθέτοντας μεταβλητές 32-bit-, σημαντικά λιγότερες από το αρχικό πλήθος των $32 \times Z_{size} \times X_{size} \times n_{channels}$ delay bits.

Κατά συνέπεια, είναι προφανές ότι εκμεταλλευόμενοι τις ιδιότητες της γεωμετρίας του συστήματός μας καταφέρνουμε να αναμορφώσουμε τον αλγόριθμο Delay and Sum με τρόπο που μας επιτρέπει να μειώσουμε σημαντικά το ποσό των καθυστερήσεων που πρέπει να υπολογίσουμε (έως και 20 φορές λιγότερα delays), προκειμένου να υπολογίσουμε την beamformed εικόνα ενός frame.

Ωστόσο, θα πρέπει να έχουμε κατά νου ότι οι Block RAMs στο εσωτερικό της FPGA διαθέτουν μόνο δύο θύρες ανάγνωσης και εγγραφής, γεγονός που μπορεί να περιορίσει τον παραλληλισμό της σχεδίασής μας. Θα αντιμετωπίσουμε αυτό το ζήτημα με κατάλληλο partition των Block RAMs.

1.3.3 Προτεινόμενη υλοποίηση του DAS αλγορίθμου στην FPGA συσκευή

Αρχικά, στο Σχήμα 1.1 παρουσιάζουμε ένα flow control diagram του προτεινόμενου πυρήνα Delay and Sum, για να βοηθήσουμε τον αναγνώστη να ακολουθήσει από την αρχή την περιγραφή της υλοποίησής μας. Κάθε στοιχείο αυτού του διαγράμματος θα εξηγηθεί συνοπτικά αργότερα στην παρούσα ενότητα.

Αφού μειώσαμε τις παραμέτρους των beamforming υπολογισμών από τρεις σε δύο διαστάσεις, εξοικονομώντας σημαντικό όγκο εσωτερικής μνήμης FPGA, πρέπει τώρα να περιγράψουμε τον τρόπο ανάθεσης των αποστάσεων Δx στα αντίστοιχα ζεύγη καναλιών - pixel, ώστε να πραγματοποιηθούν τα σωστά αθροίσματα.

Διατηρώντας το βάθος z καθώς και τη γωνία εκπομπής αμετάβλητες, τότε με βάση τις ανανεωμένες DAS εξισώσεις, κάθε καθυστέρηση υπολογίζεται με την σχετική απόσταση Δx καθώς και τη συντεταγμένη του καναλιού λήψης. Εάν αγνοήσουμε προς το παρόν την τιμή $\frac{x_i \cdot sin(\theta)}{c_{sound}}$, τότε για κάθε πιθανό βάθος εικόνας υπάρχουν ακριβώς $n_{\Delta x}$ διαφορετικές τιμές των delays. Έτσι, για το beamforming των pixels σε ένα συγκεκριμένος image depth, αρχικά δημιουργούμε έναν δισδιάστατο πίνακα μεγέθους $n_{\Delta x} \times n_{channels}$, όπου η i-οστή σειρά αντιπροσωπεύει τα δείγματα από κάθε κανάλι λήψης, τα οποία μπορούμε να εντοπίσουμε μέσω της καθυστέρησης $\tau(\Delta x)$ για την i-οστή τιμή Δx_i. Ένα σύντομο κομμάτι ψευδοκώδικα θα μπορούσε να μας βοηθήσει να οπτικοποιήσουμε αυτή την απλή συλλογή δειγμάτων.

Algorithm 1: Calculate Stacked Samples Array

```
for Δx in (0, Δx_n) do
    index_1 ← Δx delay index
    for channel in (0, n_channels) do
        index_2 ← channel delay index
        index ← index_1 + index_2
        array_samples[Δx][channel] ← buffer_samples[channel][index]
    end for
end for
```

Υποθέτοντας 64 pixels στον άξονα x και 64 κανάλια λήψης, μια οπτικοποίηση ενός τέτοιου πίνακα δίνεται στο σχήμα 1.2.

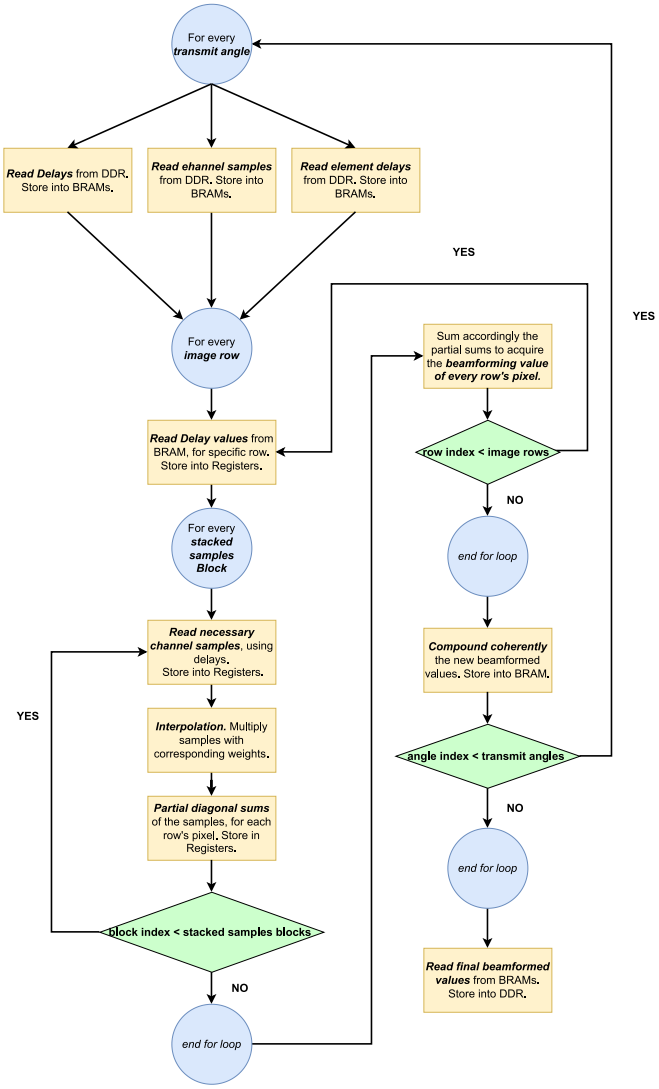


Figure 1.1: Control flow diagram του προτεινόμενου πυρήνα.

Εχουμε λοιπόν στα χέρια μας όλα τα απαραίτητα δείγματα για το beamforming μιας ολόκληρης σειράς εικονοστοιχείων εικόνας. Εν ολίγοις, έχουμε ήδη εκτελέσει τον υπολογισμό της καθυστέρησης και την ανίχνευση των δειγμάτων του καναλιού του αλγορίθμου Delay and Sum, για ένα μόνο βάθος εικόνας. Τώρα το θέμα είναι να συνειδητοποιήσουμε ποια δείγματα αντιστοιχούν σε ποια pixels και να βρούμε έναν αποτελεσματικό τρόπο άθροισης τους.

Παρατηρώντας τις γεωμετρικές ιδιότητες μεταξύ των pixels και των καναλιών, το γεγονός πως η ελάχιστη απόσταση τόσο των pixels όσο και των καναλιών είναι η ίδια, οδηγεί στο συνπέρασμα πως τα αθροίσματα για τον υπολογισμό των beamformed pixels πάνω στον stacked samples πίνακα πρέπει να πραγματοποιηθούν διαγώνια. Τα διαγώνια αυτά αθροίσματα φαίνονται στην εικόνα 1.3, όπου δίνεται το παράδειγμα μιας εφαρμογής για 8 κανάλια και 8 pixels σε κάθε image depth. Τα αθροίσματα πραγματοποιούνται ανάλογα και για περισσότερα κανάλια και pixels.

Να σημειωθεί πως πλέον είναι φανερό πως τα loops για τα αθροίσματα πραγματοποιούνται πάνω σε δύο μεταβλητές, την συντεταγμένη z των pixels καθώς και τις σχετικές αποστάσεις Δx, Μειώσαμε επιτυχημένα δηλαδή τα στάδια ιεραρχίας του loop που υλοποιεί όλα τα αθροίσματα του αλγορίθμου.

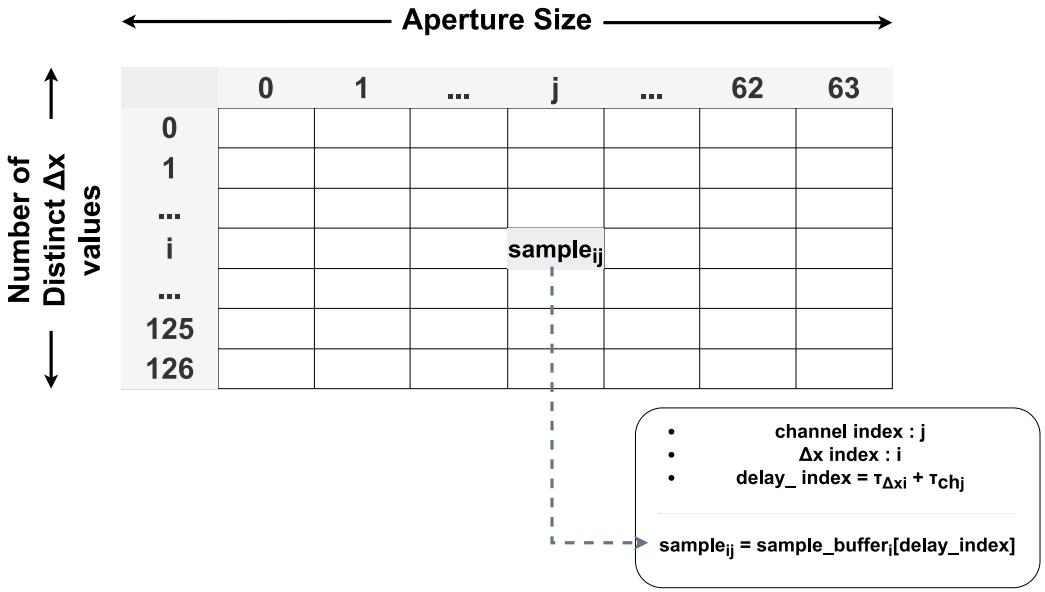


Figure 1.2: Stacked samples array, για συγκεκριμένο βάθος και συγκεκριμένη γωνία εκπομπής.

1.3.4 Beamforming Design για ένα μεμονωμένο Image Depth

Εφόσον έχουμε περιγράψει συνοπτικά την συλλογή των κατάλληλων samples από τις BRAMs και την διαδικασία αθροίσματος, για την υλοποίηση του beamforming σε ένα συγκεκριμένο image depth, θα εξηγήσουμε πως θα υλοποιηθεί αυτή η διαδικασία στη συσκευή FPGA. Αρχικά, να σημειωθεί πως ιδανικά επιθυμούμε να επεξεργαζόμαστε κάθε βάθος εικόνας σειριακά, ενώ το beamforming εσωτερικά σε κάθε image depth να είναι pipelined είτε τα αθροίσματα να εκτελούνται παράλληλα.

Όσον αφορά τη σχεδίαση του πυρήνα μας, υπάρχουν δύο κύριες κατευθύνσεις στις οποίες πρέπει να εστιάσουμε: τη διαχείριση της μνήμης και την επεξεργασία δεδομένων. Ακολούθως θα περιγράψουμε πως υλοποιείται κάθε απαραίτητο βήμα του αλγορίθμου, αυτά που απεικονίζονται και στο control flow diagram 1.1, on-chip.

Ανάγνωση από την εξωτερική μνήμη DDR

Ο beamforming πυρήνας που σχεδιάσαμε αποτελείται από δύο θύρες εισόδου και μία θύρα εξόδου. Οι δύο θύρες εισόδου χρησιμοποιούνται για τη φόρτωση όλων των δειγμάτων που λαμβάνονται από το probe υπερήχων καθώς και των προκαθορισμένων delays για κάθε βάθος και κάθε γωνία. Η θύρα εξόδου προορίζεται για την υποστήριξη της μεταφοράς της beamformed εικόνας για κάθε frame, από τους εσωτερικούς buffers της FPGA στην εξωτερική DDR μνήμη. Αυτά είναι όλα τα δεδομένα που απαιτούνται για το beamforming κάθε frame που εισάγεται στον πυρήνα.

Για την εσωτερική αποθήκευση των προαναφερόμενων δεδομένων χρησιμοποιούμε τρεις buffers:

- buffer εισόδου, μεγέθους $n_{channels} \times n_{samples}$. Εδώ αποθηκεύονται τα δείγματα από τον αισθητήρα υπερήχων για μια συγκεκριμένη γωνία εκπομπής.
- buffer καθυστέρησης, μεγέθους $n_z \times [n_{channels} + n_x - 1]$. Εδώ αποθηκεύονται οι προκαθορισμένες καθυστερήσεις για μια συγκεκριμένη γωνία εκπομπής.
- buffer εξόδου, μεγέθους $n_z \times n_x$. Οι compounded beamformed τιμές αποθηκεύονται εδώ, προκειμένου να μεταφερθούν στη μνήμη DDR.

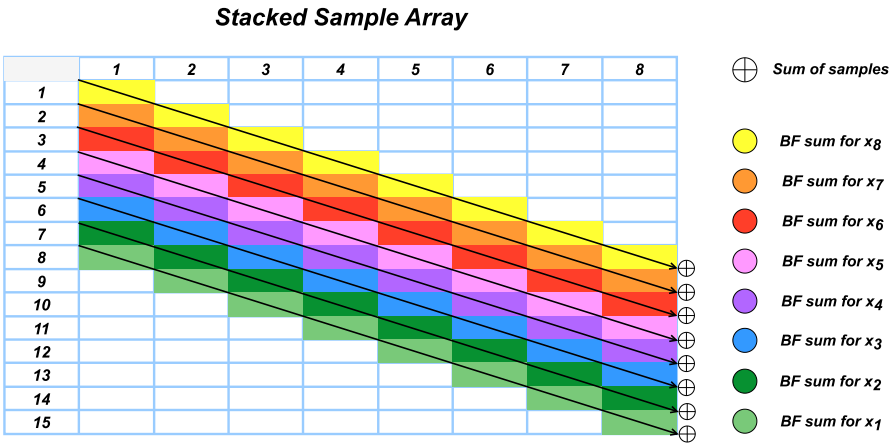


Figure 1.3: Beamforming αθροίσματα για μια συγκεκριμένη οριζόντια γραμμή της εικόνας, μέσω του stacked samples πίνακα. Τα χρωματιστά κελιά συμμετέχουν στα αθροίσματα, ενώ τα κενά κελιά είναι αχρησιμοποίητα.

Interpolation

Σχεδιάσαμε δύο εκδοχές του προτεινόμενου beamformed πυρήνα. Η μία υλοποιεί μια naive Interpolation τεχνική, όπου η εκτιμώμενη τιμή ισούται με τη μέση τιμή των δύο γειτονικών δειγμάτων της (όπως αυτή που υλοποιήθηκε από τους Kou et al. [7]), ενώ η άλλη υλοποιεί linear Interpolation.

Όσον αφορά τον πυρήνα με naive Interpolation, η διαδικασία εκτίμησης μπορεί να εφαρμοστεί κατά τη φόρτωση των δειγμάτων από το DDR. Επομένως, στο control flow διάγραμμα 1.1, μπορούμε να παραλείψουμε τη διαδικασία "Interpolation". Αυτή η μέθοδος Interpolation αποδίδει ικανοποιητικές εκτιμήσεις, αν και μικρότερης ακρίβειας από εκείνες που παράγονται με linear Interpolation. Ωστόσο πετυχαίνει καλύτερη απόδοση και βελτιωμένη διαχείριση των FPGA resources.

Από την άλλη πλευρά, η linear Interpolation τεχνική υλοποιείται κατά την beamforming διαδικασία, υπόσχεται εικόνες με βελτιωμένη ποιότητα αλλά πετυχαίνει χειρότερη απόδοση και διαχείριση των πόρων του FPGA, συγκριτικά με την προαναφερόμενη τεχνική.

Άθροιση stacked samples

Για τον προτεινόμενο beamformer, θα υλοποιήσουμε μια συνάρτηση που επεξεργάζεται εικόνες με 64 pixels, χρησιμοποιώντας 64 probe channels. Για beamforming σε μεγαλύτερα σύνολα δεδομένων, δημιουργούμε πολλαπλά instances αυτής της συνάρτησης, οι οποίες μπορούν να εκτελούνται παράλληλα. Έτσι, υποθέτοντας 64 κανάλια καθώς και 64 pixels για κάθε βάθος εικόνας, η πρόσθεση 64 τιμών για κάθε ένα από τα 64 pixels, σε έναν μόνο κύκλο ρολογιού, δεν μπορεί να υποστηριχθεί από τους πόρους της FPGA συσκευής. Κατά συνέπεια, η διαδικασία άθροισης μιας εικόνας ενός pixel πρέπει να χωριστεί σε στάδια.

Για κάθε κύκλο ρολογιού, επιλέξαμε να αθροίζουμε 8 δείγματα για κάθε pixel. Για να το επιτύχουμε αυτό, χωρίζουμε τον stacked samples πίνακα σε blocks των 8 γραμμών και υπολογίζουμε το μερικό άθροισμα μόνο με τα δείγματα που υπάρχουν (επομένως 8 δείγματα για κάθε pixel). Αφού αποκτηθούν τα μερικά αθροίσματα από κάθε block, αθροίζονται αναλόγως για να ληφθεί η τελική beamformed τιμή για κάθε pixel αυτού του βάθους. Εν τέλει, για την ολοκλήρωση του beamforming κάθε image row απαιτεί $N_{\Delta x}/8$ κύκλους ρολογιού.

Ανάλογα με τη θέση του μπλοκ, τα απαραίτητα δείγματα για άθροισμα μπορούν να βρεθούν σε διαφορετικούς δείκτες του πίνακα. Ένα παράδειγμα δίνεται στο σχήμα 1.4.

Κάθε μερικό άθροισμα, μετά τον υπολογισμό, θα αποθηκεύεται σε εσωτερικά Flip Flops της FPGA, ώστε να μπορούν να διαβαστούν όλα σε έναν κύκλο ρολογιού. Για το beamforming μιας μεμονωμένης γραμμής απαιτείται ο συνδυασμός των μερικών διαγώνιων αθροισμάτων από όλα τα blocks και η άθροιση

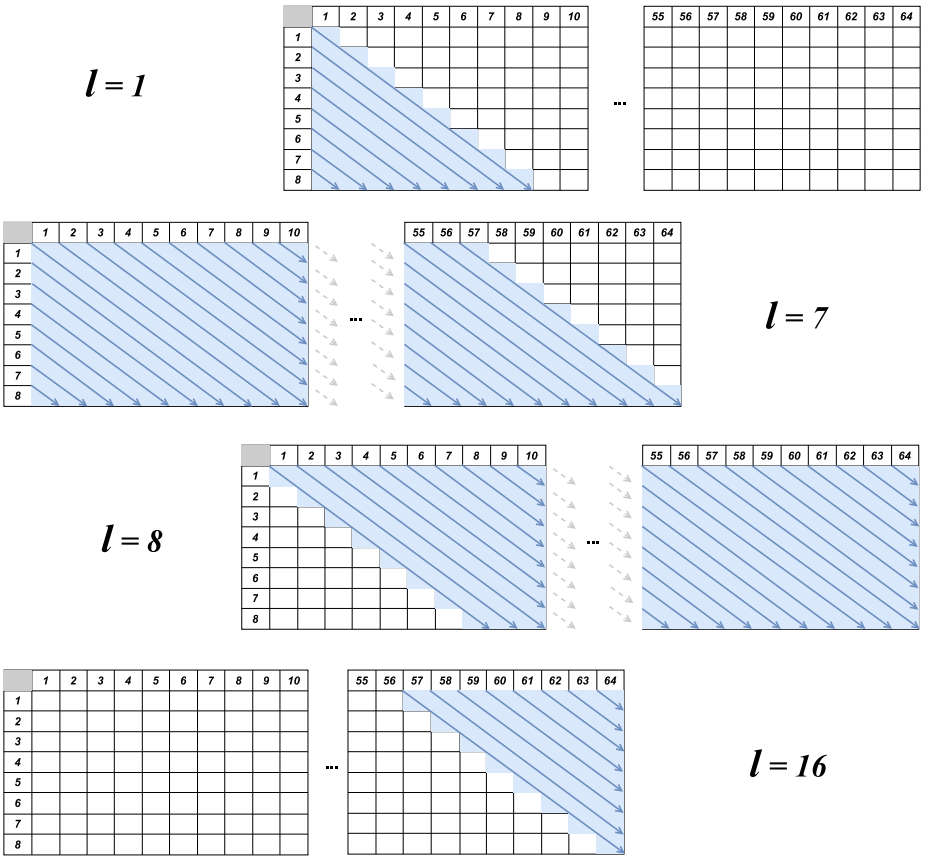


Figure 1.4: Ορισμένα blocks του stacked samples πίνακα και ποια μερικά αθροίσματα αξιοποιούνται. Η μεταβλητή l δείχνει την θέση του block στον αρχικό πίνακα.

τους, ώστε να λάβουμε τελικά την τελική beamformed γραμμή. Τα μερικά και τα τελικά αθροίσματα αποθηκεύονται σε καταχωρητές, ώστε κάθε summing process να εκτελεστεί σε έναν κύκλο ρολογιού, εφόσον όλα τα δείγματα εισόδου που χρησιμοποιούνται για την άθροιση μπορούν επίσης να διαβαστούν σε έναν κύκλο ρολογιού και δεν υπάρχουν data dependencies. Επιπλέον, το μέγεθος των προαναφερθέντων μπλοκ δειγμάτων αποκαλύπτει έναν trade-off μεταξύ της κατανάλωσης πόρων και του latency του design. Τα μεγαλύτερα μπλοκ πετυχαίνουν σε ταχύτερο πυρήνα, ωστόσο απαιτούν περισσότερους πόρους, δηλαδή Flip Flops και Lookup tables, για την εκτέλεση των διαδικασιών άθροισης.

Ανάγνωση δεδομένων από εσωτερικές BRAM

Από τη διαδικασία άθροισης, δεδομένου ότι εκτελούμε την επεξεργασία δεδομένων σε ομάδες των 8 γραμμών από τον stacked samples array, είναι σαφές ότι 8 τιμές καθυστέρησης πρέπει να διαβαστούν σε έναν κύκλο από τους delay buffers. Για να το επιτύχουμε αυτό, εφαρμόζουμε cyclic partition με συντελεστή 8, στη δεύτερη διάσταση του πίνακα καθυστέρησης με μέγεθος $n_z \times n_{\Delta x}$.

Προκειμένου να αποκτήσουμε όλα τα δείγματα για έναν μόνο delay index σε έναν κύκλο ρολογιού, διαμερίζουμε πλήρως τον channel buffer στο dimension των καναλιών. Αυτό σημαίνει ότι τα δείγματα από διαφορετικά κανάλια μπορούν να προσπελαστούν ταυτόχρονα. Ωστόσο, δεδομένου ότι απαιτούνται 8 σειρές δειγμάτων για τον υπολογισμό κάθε ομάδας μερικών αθροισμάτων, τότε πρέπει να δημιουργήσουμε 4 διαφορετικά αντίγραφα των channel samples προκειμένου να καταστεί δυνατή η πρόσβαση σε 8 δείγματα από ένα κανάλι, σε έναν κύκλο ρολογιού.

Αποθήκευση των αποτελεσμάτων σε εσωτερικούς buffers και στη DDR

Μετά το beamforming μιας ολόκληρης γραμμής εικόνας για κάθε γωνία εκπομπής, το επόμενο βήμα είναι το άθροισμα των συνεισφορών κάθε γωνίας για την απόκτηση της τελικής beamformed γραμμής. Έτσι, μετά το beamforming μιας μόνο γραμμής για μια συγκεκριμένη γωνία και την προσωρινή αποθήκευση σε καταχωρητές, αθροίζουμε με τις τιμές που έχουμε ήδη υπολογίσει για προηγούμενες γωνίες στον output buffer που αναφέραμε παραπάνω. Εφαρμόζουμε δηλαδή coherent compounding.

Αφού ολοκληρωθεί το beamforming για κάθε γωνία, το αποτέλεσμα μεταφέρεται στην εξωτερική DDR.

Συνολική σχεδίαση του beamforming πυρήνα

Για να συνοψίσουμε, για ένα μεμονωμένο group από frames που πηγάζουν από μεταδόσεις από διαφορετικές γωνίες εκπομπής, ο beamforming πυρήνας μας υλοποιείται ως ακολούθως.

Algorithm 2: Beamforming Kernel Flow

```
for n in (0, angles) do
  channel_buffer ← read_input_ddr(n)
  delay_buffer ← read_delay_ddr(n)
  for z in (0, image_rows) do
    for l in (0, D/8) do
      delay_registers ← read_delays_from_buffer(delay_buffer, z, l)
      sample_registers ← read_samples_from_buffer(channel_buffer,delay_registers)
      weighted_sample_registers ← interpolation(sample_registers, delay_registers)
      if l < (D/8)/2 then
        partial_sum_registers ← sum_stacked_samples_1(weighted_sample_registers, l)
      else
        partial_sum_registers ← sum_stacked_samples_2(weighted_sample_registers, l)
      end if
    end for
    final_sum_registers ← sum_partial_sums(partial_sum_registers)
    compound_bf_values(output_buffer,final_sum_registers, z)
  end for
end for
write_output_ddr(output_buffer)
```

Θα πρέπει να σημειώσουμε ότι οι συναρτήσεις sum_stacked_samples_1() και sum_stacked_samples_2() αντιστοιχούν σε δύο αθροιστικούς υποπυρήνες που υλοποιούν τα επιμέρους αθροίσματα και η συνάρτηση final_sum_registers() αντιστοιχεί σε έναν αθροιστικό υποπυρήνα που συνδυάζει όλα τα επιμέρους αθροίσματα για την έξοδο του beamforming της γραμμής. Το Σχήμα 1.1 που παρουσιάστηκε προηγουμένως, απεικονίζει τον προαναφερθέντα ψευδοκώδικα, περιγράφοντας το design του πυρήνα και τη ροή δεδομένων της εφαρμογής μας.

Επιπλέον, είναι ασφαλές να πούμε ότι δεν υπάρχουν εξαρτήσεις μεταξύ των μεταβλητών εντός του beamforming loop. Pipeline με *Initiation Interval = 1* του εσωτερικού loop είναι εφικτό, εξασφαλίζοντας την ελαχιστοποίηση της καθυστέρησης, λαμβάνοντας υπόψη τους διαθέσιμους πόρους της FPGA. Αυτό σημαίνει ότι ο εσωτερικότερος βρόχος θα έχει καθυστέρηση ($N_{\Delta x}/8 + single\_loop\_latency - 1$).

Συνοψίζοντας την ανάλυσή μας, περιγράψαμε έναν αναμορφωμένο αλγόριθμο για Delay and Sum beamforming που επιτυγχάνει δύο συγκεκριμένες βελτιώσεις:

- Μείωση του delay profile της beamforming διαδικασίας.
- Μείωση των συνολικών επαναλήψεων κατά τη διάρκεια των beamforming διαδικασιών άθροισης.

Τα αποτελέσματα αυτών των βελτιώσεων συνοψίζονται στον Πίνακα 1.2.

Βελτιστοποίηση	***Bottleneck***	***Αποτέλεσμα Βελτιστοποίησης***
Αναδιαμόρφωση των υπολογισμών των καθυστερήσεων στον αλγόριθμο Delay and Sum, για την μείωση του delay profile που φορτώνεται στην FPGA.	$Z_{size} \times X_{size} \times n_{elements}$ καθυστερήσεις πρέπει να αποθηκευτούν στις BRAMs. Το array partition για να υποστηριχτεί το επιθυμητό σχήμα παραλληλίας, εξαντλεί τα διαθέσιμα resources.	Πλέον $Z_{size} \times (X_{size} + n_{elements} - 1)$ **καθυστερήσεις** πρέπει να φορτωθούν στο FPGA. Array partition που υποστηρίζει υψηλή παραλληλία είναι πλέον εφαρμόσιμο.
Αναδιαμόρφωση του Delay and Sum αλγορίθμου για την ελάττωση των loop επαναλήψεων κατά την διαδικασία της άθροισης.ω	**Τριπλή ιεραρχία loop** που δεν επιτρέπει ένα linear calculation flow για ultrafast imaging. Επαρκής παραλληλισμός δεν είναι εφικτός λόγω περιορισμών στους πόρους του FPGA.	**Η ιεραρχία των loop** για τα αθροίσματα μείωνεται στα **δύο επίπεδα** (πάνω στις μεταβλητές z και Δx). Πλέον μπορούμε να εφαρμόσουμε το απαραίτητο pipeline.

Table 1.2: Βελτιστοποιήσεις για την αντιμετώπιση των σημαντικότερων bottlenecks του DAS αλγορίθμου που περιορίζει το performance για υψηλά frame rates.

1.3.5 Ανάπτυξη της Beamforming Εφαρμογής με τα εργαλεία Vitis και Vitis HLS

Σε αυτή την ενότητα θα περιγράψουμε εν συντομία την ανάπτυξη του beamforming πυρήνα με τη χρήση των εργαλείων Vitis και Vitis HLS, που παρέχονται από την Xilinx.

Αρχικά, ο κώδικας του πυρήνα σε C++ παράγεται με τη χρήση του εργαλείου Vitis High Level Synthesis για τη σύνθεση ενός πυρήνα με αποδεκτή καθυστέρηση, η οποία επίσης δεν υπερβαίνει τους διαθέσιμους πόρους. Αξιοποιούμε τα HLS directives ώστε να καθορίσουμε το scheduling του πυρήνα (παραλληλία και pipeline) καθώς και την μεταφορά δεδομένων και διαχείριση μνήμης εσωτερικά του FPGA (επικοινωνία με εξωτερική DDR, array partition και τύπος μνήμης για κάθε μεταβλητή, όπως BRAM, URAM, Flip Flops).

Στη συνέχεια, το εργαλείο Vitis χρησιμοποιείται για την ανάπτυξη της beamforming εφαρμογής. Γράφουμε ένα host πρόγραμμα σε OpenCL, προκειμένου να ρυθμιστεί το περιβάλλον για την εκτέλεση του πυρήνα, καθώς και για τη διαχείριση της μνήμης μεταξύ της CPU του συστήματος και της FPGA. Μεταγλωττίζουμε επίσης τον beamforming πυρήνα, τον συνθέτουμε και δημιουργούμε ένα bitstream από τον πυρήνα HLS, προκειμένου να τον εκτελέσουμε στην πλακέτα. Το host πρόγραμμα είναι υπεύθυνο για την εκτέλεση του πυρήνα και τη συλλογή των δεδομένων εξόδου. Ας σημειωθεί ότι κάθε σύνθεση και κάθε bitstream που δημιουργούμε προορίζονται για το Alveo U200, ένα FPGA της Xilinx.

Καθώς η συγκεκριμένη συσκευή αποτελείται από 3 SUper Logic Regions (SLRs), κατά τη διάρκεια του linking stage με τον v++, ορίζουμε πόσα instances του πυρήνα μας θα πρέπει να δημιουργηθούν και καθορίζουμε σε ποια Super Logic Region (SLR) θα τοποθετηθούν. Αποφασίσαμε να σχεδιάσουμε έναν πυρήνα που υποστηρίζει beamforming αξιοποιώντας 64 κανάλια και μέγεθος image line ίσο με 64 pixels. Δεδομένου ότι το dataset που χρησιμοποιήσαμε για το evaluation υποστηρίζει 128 κανάλια και 128 pixels στον άξονα x, δημιουργήσαμε τέσσερα πανομοιότυπεα instances του beamforming πυρήνα μας. Δύο compute units αντιστοιχίζονται στο SLR0 και οι άλλες δύο στο SLR2, προκειμένου να αποφύγουμε την επικοινωνία μεταξύ των SLRs, η οποία αναπόφευκτα θα αυξήσει τη συχνότητα ρολογιού. Στο Σχήμα 1.5 απεικονίζεται το αρχιτεκτονικό διάγραμμα της beamforming εφαρμογής στην Alveo U200.

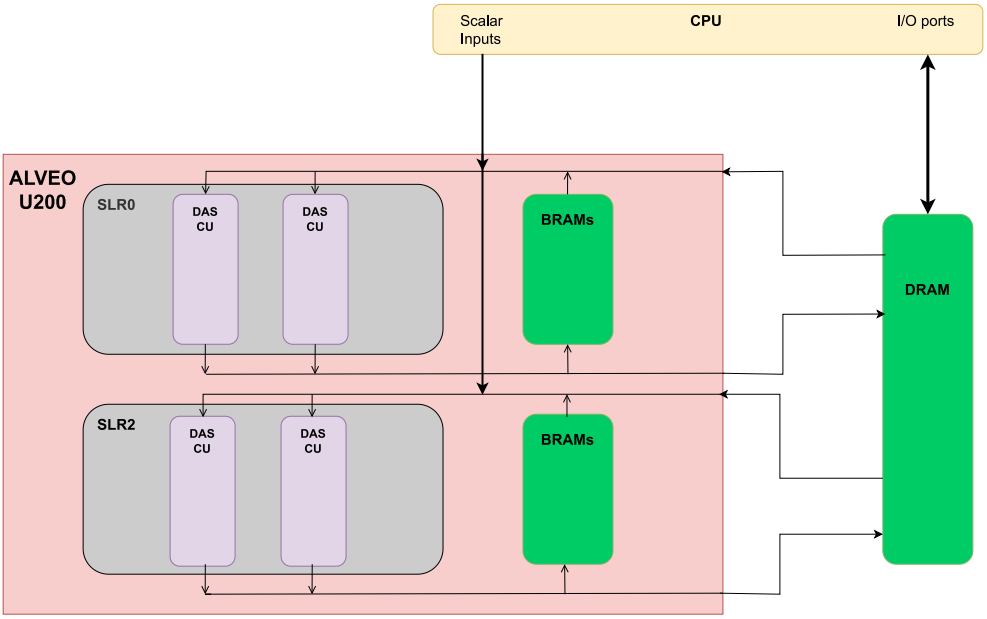


Figure 1.5: Architectural diagram του προτεινόμενου πυρήνα.

1.4 Αποτελέσματα και Αξιολόγηση

Οι δύο εκδοχές του beamforming πυρήνας μας αξιολογήθηκαν με την χρήση ενός dataset από την ιστοσελίδα του PICMUS, the IEEE IUS 2016 Plane-wave Imaging Challenge in Medical UltraSound [11]. Η αξιολόγησή μας πραγματοποιήθηκε με κριτήριο την ορθότητα, την απόδοση, το scalability των design μας καθώς και με την σύγκριση με τους state of the art FPGA beamformers.

Τα χαρακτηριστικά του προαναφερόμενου dataset παρουσιάζονται στον Πίνακα 1.3.

Aperture Size	128 channels
Sampling Frequency (Hz)	2308000
Collected samples per channel	1527
Plane Waves	75

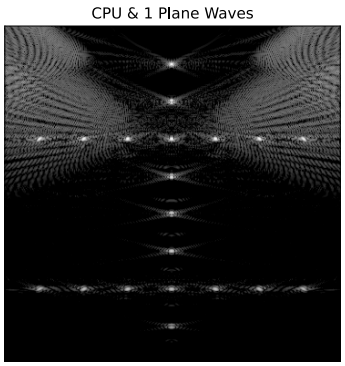
Table 1.3: Τα χαρακτηριστικά τπου dataset που χρησιμοποιήθηκε για validation και evaluation του beamforming πυρήνα μας.

Οι εικόνες που παράγονται από το παραπάνω dataset, έπειτα από beamforming σε CPU, για 1,3,11,38 και 75 γωνίες εμφανίζονται στην Εικόνα 1.6.

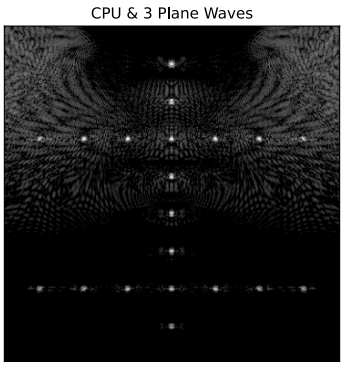
1.4.1 Beamforming αποτελέσματα από τον προτεινόμενο πυρήνα

Για το beamforming του συνόλου δεδομένων δημιουργήσαμε όπως προαναφέρθηκε, 4 instances του πυρήνα μας που εφαρμόζει Delay And Sum με 64 κανάλια, σε 609 γραμμές εικόνας με 64 pixels η καθεμία για 75 γωνίες. Το αποτέλεσμα είναι μια beamformed εικόνα μεγέθους 128×609, χρησιμοποιώντας 128 κανάλια. Αρχικά, εκτελούμε το beamforming με 75 επίπεδα κύματα.

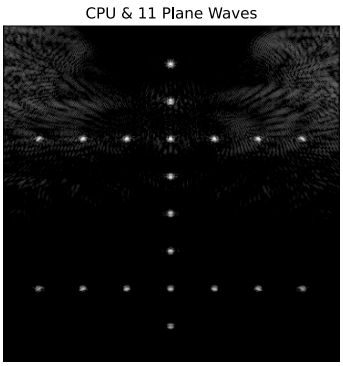
Εκτελέσαμε την εφαρμογή για τον πυρήνα που υλοποιεί την Naive Interpolation τεχνική και για τον πυρήνα που υλοποιεί την Linear Interpolation τεχνική. Η απόδοση και η αξιολόγηση των δύο designs παρουσιάζονται ακολούθως.



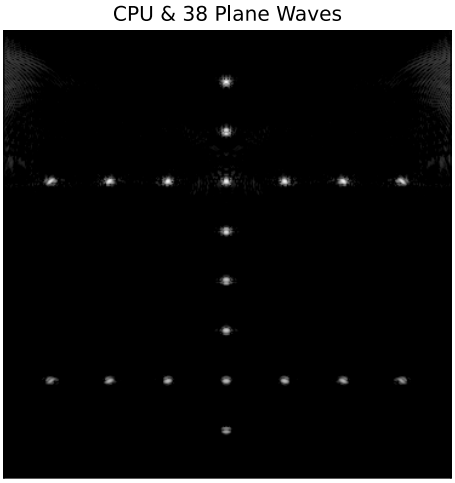
(a) 1 Plane Wave



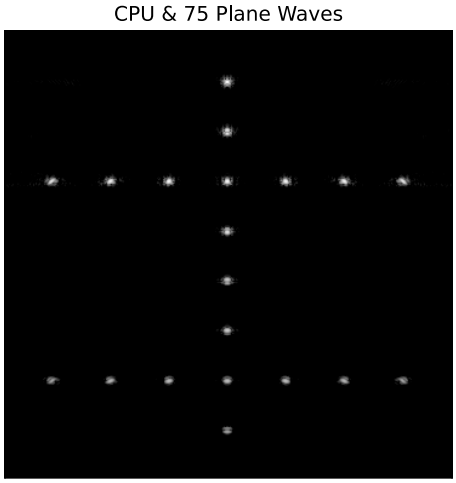
(b) 3 Plane Waves



(c) 11 Plane Waves



(d) 38 Plane Wave



(e) 75 Plane Waves

Figure 1.6: Beamformed εικόνες με 1,3,11,38 και 75 γωνίες, σε CPU

Naive Interpolation πυρήνας

Ο πίνακας 1.4 συνοψίζει την απόδοση του πυρήνα μας για 75 επίπεδα κύματα, ενώ ο πίνακας 1.5 την κατανάλωση πόρων του πυρήνα DAS, όχι μόνο σε ολόκληρο το Alveo U200, αλλά και την κατανάλωση ανά SLR. Οι πληροφορίες αυτών των δύο πινάκων αποκτήθηκαν με profiling της εφαρμογής μας με τα εργαλεία Vitis και Vitis Analyzer.

Clock Frequency (MHz)	300
Clock Cycles	1,996,397
System Runtime (ms)	7.135
Kernel Runtime (ms)	6.747
Frame Rate (Hz)	148.214
Pulse Repetition Frequency (Hz)	11,116.052
Beamforming Rate (MSamples/s)	2173
Power Consumption (Watt)	10.800

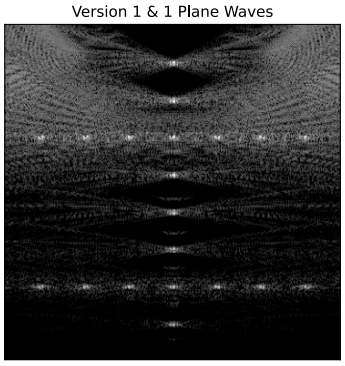
Table 1.4: Απόδοση του DAS Beamforming kernel, με την *naive* interpolation τεχνική, για 75 γωνίες εκπομπής.

Resource	*1 CU/Full device*	*4 CUs/Full Device (%)*	*1 CU/Per SLR (%)*	*2 CUs/Per SLR (%)*
LUT	36185 (3.06%)	12.24%	11.14%	22.28%
BRAM	271 (12.55%)	50.20%	42.48%	84.96%
URAM	96 (12%)	36%	30 %	60%
Flip Flops	38289 (1.75%)	7%	5.29%	10.58%
DSP	3 (0.04%)	0.12%	0.13%	0.26%

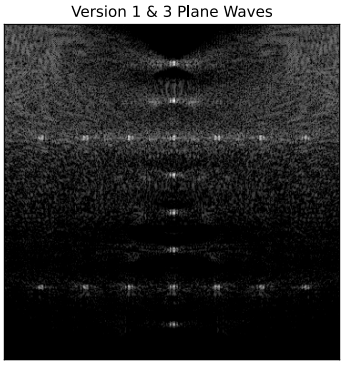
Table 1.5: Κατανάλωση του DAS Beamforming kernel, με *naive* interpolation,για 75 γωνίες. Το design μας αποτελείται από 4 Compute Units, όπου 2 Compute Units τοποθετούνται στο ίδιο SLR.

Ο προτεινόμενος DAS beamforming πυρήνας επιτυγχάνει beamforming rate στην περιοχή GSamples/s, ή ομοίως υποστηρίζει Pulse Repetition Frequency (PRF) στην περιοχή των KHz. Επομένως, πληροί τα κριτήρια ενός beamforming πυρήνα για ultrafast ultrasound εφαρμογές. Επιπλέον, οι πόροι με τη μεγαλύτερη χρήση είναι οι BRAM και οι URAM. Το γεγονός αυτό υποδεικνύει ότι το design μας είναι memory constrained.

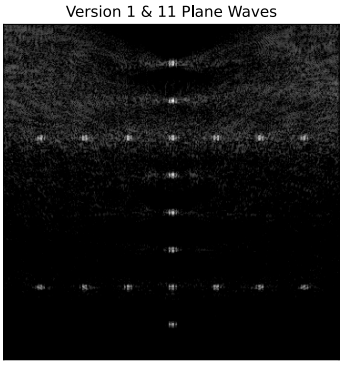
Στο Σχήμα 1.7 παρουσιάζονται οι εικόνες εξόδου του πυρήνα μας για εκπομπές 1, 3, 11, 38 και 75 επίπεδων κυμάτων.



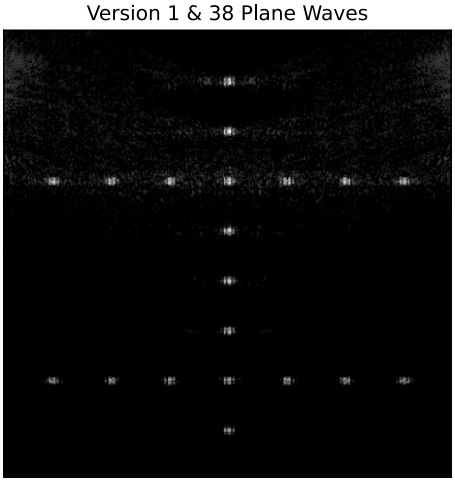
(a) 1 Plane Wave



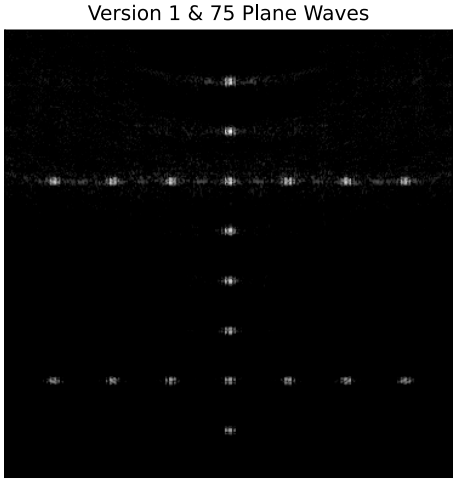
(b) 3 Plane Waves



(c) 11 Plane Waves



(d) 38 Plane Wave



(e) 75 Plane Waves

Figure 1.7: Beamformed εικόνες με 1,3,11,38 και 75 plane waves, στο FPGA με τον πρώτο προτεινόμενο kernel.

Οι εικόνες εξόδου είναι παρόμοιες με αυτές που παράγονται από την CPU, αποδεικνύοντας την ορθότητα του προτεινόμενου design. Ωστόσο, είναι ορατή η μείωση της ποιότητας της εικόνας. Αυτό συμβαίνει καθώς το Interpolation αποδίδει χαμηλότερη ακρίβεια εκτίμησης στις εκτιμώμενες τιμές.

Linear Interpolation πυρήνας

Οι πίνακες 1.6 και 1.7 περιγράφουν την απόδοση του προτεινόμενου πυρήνα, καθώς και τη κατανάλωση των πόρων του FPGA.

Clock Frequency (MHz)	300
Clock Cycles	2,218,847
System Runtime (ms)	7.899
Kernel Runtime (ms)	7.489
Frame Rate (Hz)	133.530
Pulse Repetition Frequency (Hz)	10,014.750
Beamforming Rate (MSamples/s)	1957
Power Consumption (Watt)	11.921

Table 1.6: Απόδοση του DAS Beamforming kernel, με την *linear* interpolation τεχνική, για 75 γωνίες εκπομπής.

Resource	*1 CU/Full device*	*4 CUs/Full Device (%)*	*1 CU/Per SLR (%)*	*2 CUs/Per SLR (%)*
LUT	122316 (10.35 %)	41,14%	34.46%	68.92%
BRAM	286 (13.24 %)	52.96%	44.83%	89.66%
URAM	96 (12 %)	48%	30%	60%
Flip Flops	151127 (6.92 %)	27.68%	20.89	41.78%
DSP	1026 (15.02 %)	60.08%	45.30 %	90.60%

Table 1.7: Κατανάλωση του DAS Beamforming kernel, με *linear* interpolation,για 75 γωνίες. Ξενά, το design αποτελείται από 4 Compute Units, όπου 2 Compute Units τοποθετούνται στο ίδιο SLR.

Σε σύγκριση με τον προαναφερθέντα προτεινόμενο πυρήνα, αυτή η έκδοση επιτυγχάνει μεγαλύτερη καθυστέρηση, άρα μικρότερο frame rate, ενώ καταναλώνει περισσότερους πόρους FPGA.

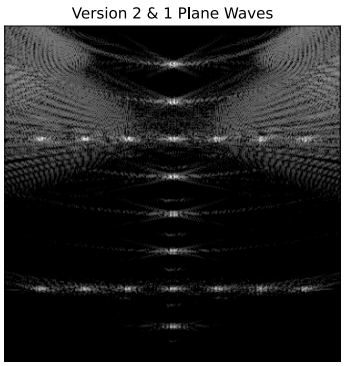
Επιπλέον, σε κάθε κύκλο ρολογιού, οι τιμές του stacked samples block πολλαπλασιάζονται με ένα αντίστοιχο βάρος λόγω linear Interpolation. Έτσι, αναμένεται η χρήση των DSP cores που χρησιμοποιούνται για υπολογισμούς, να αυξηθεί σημαντικά. Επιπλέον, η αύξηση της χρήσης των LUT και των Flip Flops εξηγείται από την ανάγκη προσωρινής αποθήκευσης των Interpolation weights και των δειγμάτων του καναλιού πριν από τον πολλαπλασιασμό, σε καταχωρητές.

Από την άλλη πλευρά, η κατανάλωση πόρων μνήμης (BRAMs, URAMs) παρέμεινε σχεδόν αμετάβλητη, δεδομένου ότι οι παράμετροι της εφαρμογής (αριθμός pixels, κανάλια, δείγματα ανά κανάλι) δεν τροποποιήθηκαν. Παρατηρείται μια μικρή αύξηση στην κατανάλωση των BRAMs λόγω της αύξησης της ακρίβειας των bit των τιμών καθυστέρησης (από 16 bit σε 32 bit).

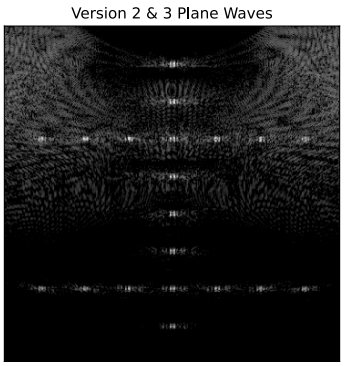
Στο Σχήμα 1.8 απεικονίζονται οι beamformed εικόνες από τον προτεινόμενο πυρήνα, για εκπομπές 1, 3, 11, 38 και 75 επίπεδων κυμάτων.

Η αύξηση της καθυστέρησης και της κατανάλωσης πόρων επιτρέπει ένα σημαντικό πλεονέκτημα, τη βελτίωση της εικόνας εξόδου. Σε σύγκριση με τις εικόνες εξόδου από την πρώτη έκδοση του design μας, η βελτίωση της ποιότητας της εικόνας είναι σημαντική.

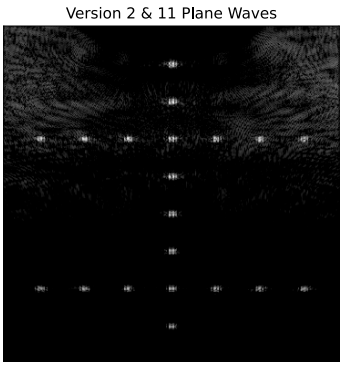
Συνοψίζοντας την αξιολόγησή μας, οι προτεινόμενοι πυρήνες είναι ικανοί να υποστηρίξουν την απεικόνιση υπερήχων Ultrafast Ultrasound, καθώς η υποστηριζόμενη PRF είναι στην περιοχή των KHz. Το πρώτο



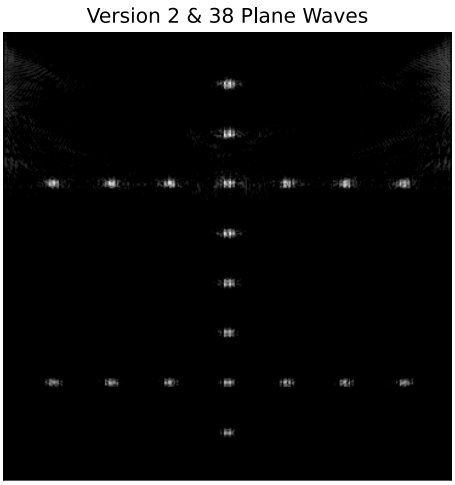
(a) 1 Plane Wave



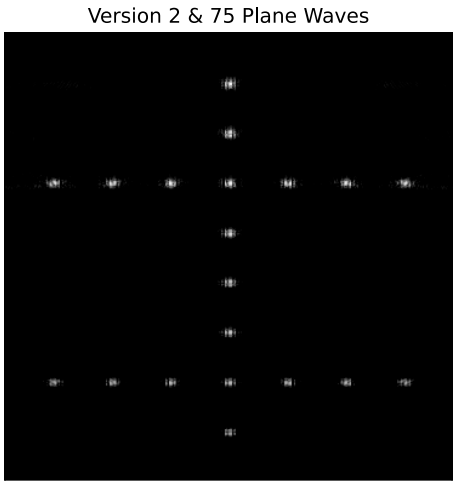
(b) 3 Plane Waves



(c) 11 Plane Waves



(d) 38 Plane Wave



(e) 75 Plane Waves

Figure 1.8: Beamformed εικόνες με 1,3,11,38 και 75 plane waves, στο FPGA με τον δεύτερο προτεινόμενο kernel.

design παράγει χειρότερη ποιότητα εικόνας, αλλά αποτελεί ταχύτερη και φθηνότερη επιλογή, ενώ το design που εφαρμόζει linear Interpolation παράγει εικόνες υψηλότερης ποιότητας, επιτυγχάνοντας όμως μεγαλύτερη χρήση πόρων και χαμηλότερο frame rate.

1.4.2 Scalability του προτεινόμενου πυρήνα

Στη συνέχεια θα εξετάσουμε το scalability του πρώτου πυρήνα για διάφορες τιμές των plane waves, καναλιών και μεγέθους του stacked samples block. Εστιάζουμε στην πρώτη έκδοση του πυρήνα καθώς έχει χημηλότερη πολυπλοκότητα υλικού, καθιστώντας έτσι ευκολότερο και πιο αποδοτικό το profiling της εφαρμογής. Ωστόσο, μπορούμε να υποθέσουμε με ασφάλεια ότι ο πυρήνας που εφαρμόζει γραμμικό interpolation έχει την ίδια συμπεριφορά όσον αφορά τον scalability.

Plane Waves

Εφαρμόσαμε beamforming στο προαναφερόμενο dataset αξιοποιώντας 1, 3, 11, 38 και 75 επίπεδα κύματα. Η καθυστέρηση του πυρήνα για κάθε γωνία παρουσιάζεται στο γράφημα της Εικόνας 1.9.

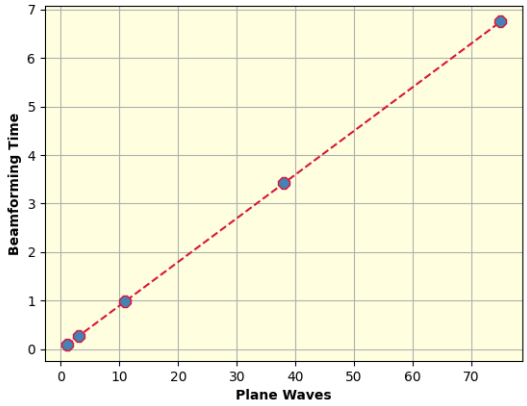


Figure 1.9: Γράφημα της καθυστέρησης του design και του πλήθους των γωνιών εκπομπής.

Δεδομένου ότι τα δεδομένα που αποκτώνται από διαφορετικά plane waves επεξεργάζονται διαδοχικά και όχι παράλληλα, το frame rate της εφαρμογής αυξάνεται γραμμικά σε σχέση με το πλήθος των επίπεδων κυμάτων. Αυτό επιβεβαιώνεται και από το παραπάνω γράφημα. Το scalability του πυρήνα είναι γραμμικό όταν μεταβάλλουμε τον αριθμό των γωνιών εκπομπής.

Κανάλια

Προκειμένου να αναλυθεί το scalability του προτεινόμενου design,δημιουργήσαμε bitstream για διάφορες ποσότητες καναλιών, μαζί με διαφορετικά μεγέθη μπλοκ. Στο γράφημα 1.10 απεικονίζεται η μεταβολή της καθυστέρησης του πυρήνα για διαφορετικό αριθμό καναλιών, ενώ στα bar plots 1.11 παρουσιάζεται η κατανάλωση των resources.

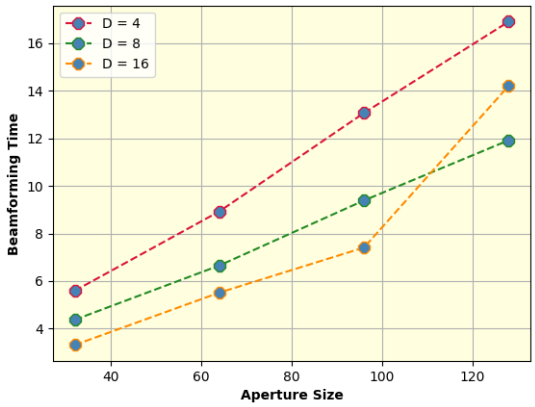
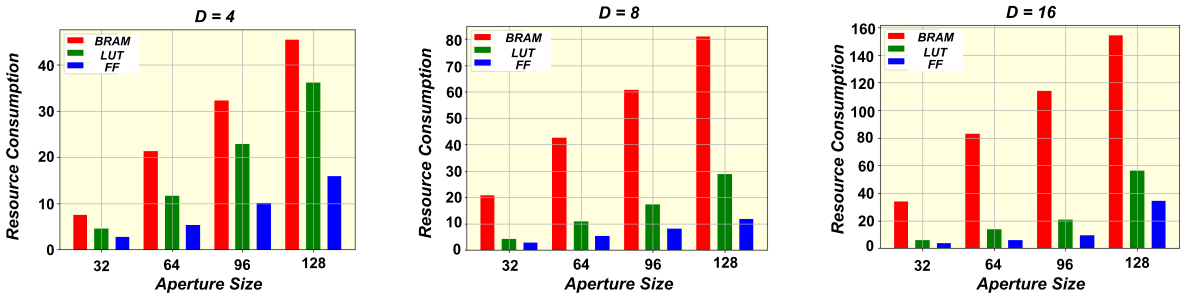


Figure 1.10: Γράφημα της καθυστέρησης της εφαρμογής σε συνάρτηση με το πλήθος καναλιών, διατηρώντας σταθερό το μέγεθος block D (4, 8 και 16 γραμμές).

- Η χρήση των BRAM και URAM αυξάνεται σχεδόν γραμμικά με την αύξηση των καναλιών.
- Η χρήση LUT και Flip Flop αυξάνεται επίσης με την αύξηση των καναλιών. Ένας μεγαλύτερος αριθμός καναλιών οδηγεί σε περισσότερα pixels σε μια σειρά εικόνας, με αποτέλεσμα να υπολογίζονται περισσότερα αθροίσματα.
- Όσον αφορά την καθυστέρηση, είναι προφανές ότι με την αύξηση των καναλιών της σχεδίασης, αυξάνεται και η καθυστέρηση. Ωστόσο, η καθυστέρηση αυξάνεται με ρυθμό μικρότερο από τον



(a) $D = 4$ (b) $D = 8$ (c) $D = 16$

Figure 1.11: Bar Plots που απεικονίζουν τα resources που καταναλώνονται για κάθε πλήθος καναλιών, ενώ διατηρούμε σταθερή την τιμή D

.

ρυθμό αύξησης των καναλιών. Επομένως, εάν οι διαθέσιμοι πόροι το επιτρέπουν, είναι προτιμότερο να δημιουργηθεί ένα ενιαίο compute unit που διαμορφώνει μια εικόνα με περισσότερα κανάλια, παρά να διαμορφώνεται η ίδια εικόνα με πολλαπλά compute units και λιγότερα κανάλια, οι οποίες εκτελούνται διαδοχικά.

Συνοψίζοντας τις παρατηρήσεις μας, είναι σαφές ότι με την αύξηση των καναλιών δεν αυξάνονται μόνο οι πόροι που καταναλώνονται, αλλά και η καθυστέρηση της εφαρμογής. Κατά συνέπεια, το design μας φαίνεται να είναι scalable όσον αφορά το πλήθος των καναλιών του probe, αφού οι μεταβολές που παρατηρούνται έχουν σταθερό ρυθμό. Ας σημειωθεί ότι ο ρυθμός αύξησης των πόρων με την αύξηση των καναλιών φαίνεται σταθερός, ανεξάρτητα από το μέγεθος block.

Μέγεθος Block

Τέλος, θα εξετάσουμε το scalability του πυρήνα μας για διαφορετικά μεγέθη block. Παρουσιάζεται ένα γράφημα της καθυστέρησης του πυρήνα σε σχέση με το μέγεθος D των blocks (Σχήμα 1.12), καθώς και ένα ραβδόγραμμα (Σχήμα 1.13) που απεικονίζει την SLR κατανάλωση των BRAM, LUT και Flip Flop, ανά ποσοστό.

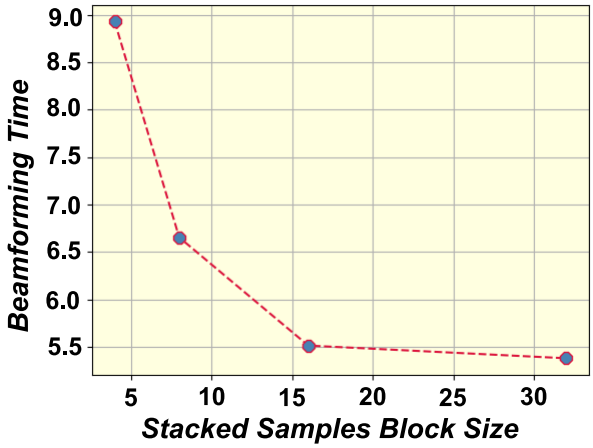


Figure 1.12: Γράφημα της καθυστέρησης του design σε συνάρτηση με το μέγεθος των stacked samples blocks.

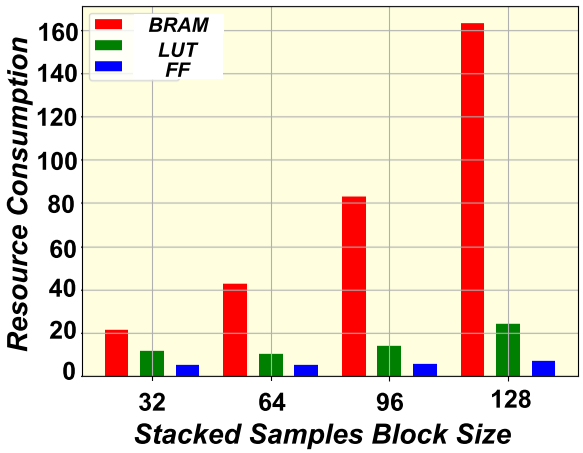


Figure 1.13: Bar plot που απεικονίζει την κατανάλωση των resources για κάθε εξεταζόμενο μέγεθος *D*.

Παρατηρούμε πως:

- Ενώ το μέγεθος block αυξάνεται, οι απαιτήσεις BRAM του design αυξάνονται επίσης, με εκθετικό ρυθμό.
- Χρησιμοποιούνται περισσότερα LUTs και Flip Flops καθώς αυξάνεται το μέγεθος block. Για μεγαλύτερες τιμές του block size, οι συναρτήσεις που υλοποιούν το διαγώνιο άθροισμα απαιτούν περισσότερους πόρους, καθώς αθροίζουν περισσότερα δείγματα.
- Η καθυστέρηση της σχεδίασης μειώνεται καθώς αυξάνεται το μέγεθος των blocks. Το latency μειώνεται ταχύτερα σε σύγκριση με την μείωση του block size, άρα επιθυμούμε μεγαλύτερα μεγέθη block εφόσον ιδανικά το επιτρέπουν τα διαθέσιμα resources του FPGA.

Ένα πράγμα που ξεχωρίζει από το γράφημα, είναι ότι ενώ αυξάνεται το μέγεθος D, η κατανάλωση BRAM αυξάνεται ραγδαία σε σύγκριση με την κατανάλωση LUT και FF. Αυτό ενισχύει τον ισχυρισμό μας ότι το προτεινόμενο design είναι memory constrained. Συνολικά, είναι προφανές ότι πράγματι το βέλτιστο μέγεθος block, για beamforming μιας εικόνας με 128 κανάλια είναι οι 8 γραμμές, καθώς παρέχει την ελάχιστη καθυστέρηση χωρίς να καταναλώνει περισσότερους πόρους FPGA από τους διαθέσιμους.

1.4.3 Σύγκριση με state of the art FPGA Beamformers

Στον Πίνακα 1.8 παρουσιάζουμε την απόδοση των δύο προτεινόμενων DAS beamforming πυρήνων FPGA καθώς και των state of the art design που παρουσιάστηκαν στην Ενότητα 1.2.

Σχολιάζοντας τα δεδομένα αυτού του πίνακα παρατηρούμε τα εξής:

- Τα προτεινόμενα designs αυτής της διπλωματικής εργασίας επιτυγχάνουν PRF και beamforming συγκρίσιμο με τα state of the art ultrafast ultrasound συστήματα, ενώ ταυτόχρονα χρησιμοποιούν το μεγαλύτερο πλήθος καναλιών.
- Σε σύγκριση με την δουλειά των [2] και [5], οι πυρήνες μας επιτυγχάνουν βελτιωμένη απόδοση.
- Σε σύγκριση με το design των Campbell et al. [9], οι δύο πυρήνες μας επιτυγχάνουν υψηλότερο beamforming rate αλλά χαμηλότερο PRF. Ωστόσο, χρησιμοποιούν πολύ μεγαλύτερο πλήθος καναλιών (128 στοιχεία έναντι 8 στοιχείων), σε συνδυασμό με μεγαλύτερο μέγεθος εικόνας.

Design	*PRF (Hz)*	*BF Rate (MSPS)*	*Frame Rate (Hz)*	*PW*	*Image Size*	*Aperture Size*
Proposed kernel (1st Version)	11,116	2173	148.21	75	128×609	128
Proposed kernel (2nd Version)	10,014	1927	133.53	75	128×609	128
[2]	714	1120	714	1	64×181	64
[5]	3800	467	345	11	96×512	32
[9]	14,000	917	875	16	64×384	8
[7]	14,865	4830	14,865	1	64×1280	64

Table 1.8: Σύγκριση με state of the art FPGA ultrafast beamformers

- Σε σύγκριση με το design των Kou et al. [7], παρόλο που τα δικά μας design επιτυγχάνουν συγκρίσιμες τιμές PRF και beamforming rate, εξακολουθούν να επιτυγχάνουν χαμηλότερη απόδοση. Ωστόσο, τα design μας χρησιμοποιούν διπλάσιο αριθμό στοιχείων του probe. Επιπλέον, η δεύτερη έκδοση του προτεινόμενου πυρήνα υλοποιεί linear interpolation, μια βελτιωμένη τεχνική interpolation συγκριτικά με αυτή που χρησιμοποιείται από τους [7]. Επομένως, η δεύτερη έκδοση του προτεινόμενου πυρήνα μπορεί να επιτυγχάνει χειρότερες επιδόσεις, αλλά εφαρμόζει linear interpolation και αξιοποιεί 128 κανάλια, ανακατασκευάζοντας εικόνες με υψηλότερη ποιότητα.

Συμπερασματικά, οι προτεινόμενοι hardware beamforming πυρήνες της παρούσας διπλωματικής εργασίας επιτυγχάνουν frame rates που μπορούν να συγκριθούν και να βελτιώσουν τους hardware beamformers του state of the art, ενώ χρησιμοποιούν ένα πλήθος καναλιών που είναι τουλάχιστον δύο φορές μεγαλύτερο από τα σχέδια που περιγράφονται στον πίνακα 1.8.

1.5 Συμπεράσματα και Μελλοντικές προεκτάσεις

Στην παρούσα διπλωματική εργασία, επιχειρήσαμε να σχεδιάσουμε έναν Delay and Sum πυρήνα σε ένα μόνο chip FPGA, που λειτουργεί σε υψηλά frame rates. Η έξοδος είναι μια δισδιάστατη B-mode εικόνα, η οποία ανακατασκευάζεται μετά τη σύνθεση υπερηχητικών σημάτων που εκπέμπονται από διάφορες γωνίες εκπομπής. Προτείνοντας μια αναδιαμορφωμένη εκδοχή του Delay and Sum αλγορίθμου, καταφέραμε να μειώσουμε το delay profile του, εξασφαλίζοντας την επιτάχυνση του. Σχεδιάσαμε έτσι τον beamformer που υποστηρίζει RF δεδομένα εισόδου και παρέχει στην έξοδο την beamformed, compounded εικόνα με 128 κανάλια. Προτείναμε δύο εκδόσεις του πυρήνα- μία που εφαρμόζει μια λιγότερο ακριβή interpolation τεχνική και μία που εφαρμόζει γραμμικό interpolation. Η πρώτη επιτυγχάνει χαμηλότερη καθυστέρηση και μικρότερη κατανάλωση πόρων από τη δεύτερη, ωστόσο παράγει εικόνες με χειρότερη ποιότητα.

Μετά την αξιολόγηση των αποτελεσμάτων από τις εκτελέσεις των προτεινόμενων beamforming πυρήνων, καταλήξαμε στο συμπέρασμα ότι τα design μας υποστηρίζουν εφαρμογές υπερηχητικής απεικόνισης με υπερταχείς ρυθμούς. Πιο συγκεκριμένα, και οι δύο πυρήνες εκτελούν beamforming σε εικόνες με 128 probe channels, μετά από λήψη 1527 δειγμάτων για κάθε κανάλι, σε υψηλά frame rates. Η πρώτη έκδοση που εφαρμόζει naive interpolation επιτυγχάνει PRF ίσο 11116,052 Hz και beamforming rate ίσο με 2173 MSamples/s. Η δεύτερη έκδοση που εφαρμόζει linear interpolation επιτυγχάνει PRF ίσο με 10014,75 Hz και beamforming rate ίσο με 1957 MSamples/s. Είναι σαφές ότι και οι δύο πυρήνες υποστηρίζουν υψηλά frame rates.

Επιπλέον, μετά από σύγκριση των προτεινόμενων πυρήνων με state of the art FPGA beamformers, καταλήξαμε στο συμπέρασμα ότι η δουλειά μας επιτυγχάνει παρόμοιες επιδόσεις ή υπερτερεί των εξεταζόμενων design, όσον αφορά το PRF, το beamforming rate, το πλήθος καναλιών ή την ποιότητα της τελικής εικόνας. Επιπλέον, τα προτεινόμενα design είναι scalable, αφού μπορεί κανείς να παράγει το ίδιο design για διαφορετικές παραμέτρους υπερήχων, με ελάχιστες αλλαγές, διατηρώντας μια προβλέψιμη απόδοση.

Με βάση την εμπειρία που αποκτήσαμε κατά την ανάπτυξη του beamforming πυρήνα, είμαστε σε θέση να συζητήσουμε προτάσεις για μελλοντική έρευνα, με γνώμονα την εργασία μας:

- Βελτίωση των μεταφορών μνήμης στο εσωτερικό της FPGA, καθώς όπως εξηγήθηκε προηγουμένως, το προτεινόμενο design περιορίζεται σε μεγάλο βαθμό από την διαθέσιμη μνήμη. Κάθε image line απαιτεί μόνο ένα ορισμένο σύνολο δειγμάτων του καναλιού για το beamforming, ανάλογα με το βάθος της. Εκμεταλλευόμενοι αυτή την παρατήρηση, το μέγεθος των buffer εισόδου που χρησιμοποιούνται κατά το beamforming θα μπορούσε να μειωθεί, μειώνοντας έτσι την κατανάλωση BRAM.
- Μεταφορά του προτεινόμενου πυρήνα και του τροποποιημένου αλγορίθμου DAS σε ένα FPGA που προσφέρει περισσότερους πόρους, καθώς και μεγαλύτερη υπολογιστική ισχύ. Ένας κατάλληλος υποψήφιος θα ήταν το Versal ACAP, που εισήγαγε η Xilinx [12], ο οποίος όχι μόνο παρέχει περισσότερους πόρους από το FPGA που χρησιμοποιήθηκε κατά τη διάρκεια της παρούσας διπλωματικής, αλλά περιέχει και μηχανές AI που επιτυγχάνουν υπολογιστική πυκνότητα σε χαμηλή ισχύ, η οποία επιτρέπει την επεξεργασία σήματος υπερήχων σε πραγματικό χρόνο και με χαμηλή καθυστέρηση [13].
- Υλοποίηση των προτεινόμενων πυρήνων με float point variables, αντί για fixed point variables. Θα πετύχουμε σε αυτή την περίπτωση καλύτερη ακρίβεια στην κατασκευή της εικόνας.
- Διερεύνηση διαφορετικών τύπων interpolation, όπως πολυωνυμικό interpolation υψηλότερου βαθμού ή spline interpolation.
- Τροποποίηση του προτεινόμενου πυρήνα για την υποστήριξη δεδομένων εισόδου IQ. Δεδομένου ότι αυτή η τροποποίηση θα συμπεριλάμβανε phase rotations, που περιλαμβάνουν τον υπολογισμό συναρτήσεων ημιτόνου και συνημιτόνου, θα απαιτηθεί μια πλατφόρμα με περισσότερους διαθέσιμους πόρους.
- Ενσωμάτωση διαφόρων διαδικασιών βελτίωσης εικόνας on-chip. Για παράδειγμα, envelope detection ή log compression είναι οι πιο ευρέως διαδεδομένες. Αυτό θα μπορούσε να επιτευχθεί σε ένα FPGA που παρέχει περισσότερες μηχανές DSP ή με την πρόσφατα επερχόμενη πλατφόρμα Versal της Xilinx [12], η οποία προσφέρει μηχανές AI που επιτρέπουν δυνατότητες DSP υψηλής απόδοσης και πραγματικού χρόνου [13].

Chapter 2

Introduction

During the last decades, ultrasound imaging has become a major, rapidly emerging, medical imaging modality. Due to its non-invasive nature, it has become a widespread diagnostic tool that is safe and gentle to the patient. Ultrasound equipment operates with harmless non-ionized radiation, it is portable and offers real-time, dynamic images, whilst having a moderate cost, giving it an edge over other popular imaging techniques, like X-ray tomography or Magnetic Resonance Imaging (MRI) [1]–[3].

The early development of ultrasound's real-time capabilities is considered the main technological breakthrough that established its proliferation. However, standard conventional ultrasound systems are based on the transmission of focused beams, whose echoes are beamformed during reception. This technique leads to frame rates that are limited to 10-40 fps, due to the fact that every formed scan line corresponds to a single transmit event, limiting their potential [4]. Moreover, conventional US imaging suffers from additional inherent limitations. Applying dynamic focusing during reception rather than transmission, results into non-uniform images, whereas the collection of scan lines during different time instances, could allow the appearance of artifacts, while visualizing fast hemodynamic events [5].

Nevertheless, increasing the frame rate of ultrasound imaging will lay the foundation for immense new applications of medical ultrasound. This will result into future real-time 3D imaging systems or improved tracking of heart motion throughout the cardiac cycle. It also allows visualization of transient events, such as mechanical shear wave propagation for elastography, in-depth imaging of brain hemodynamics, or even indirect imaging of the mechanical effects of electric action potentials. High frame rates are also used in image enhancement methods such as video integration or composite imaging methods. The aforementioned are possible with an ultrasound technique called Ultrafast Ultrasound Imaging [4].

Ultrafast ultrasound has been the main motivation behind many novel ultrasound imaging applications such as shear wave elastography, high-resolution Ultrasound Localization Microscopy (ULM) and Functional Ultrasound (fUS). Traditional line-by-line or multi-line focused beam scanning cannot provide sufficient frame rates for applications that require high-speed tracking of tissue motion (e.g., blood flow, shear wave motion, microbubble motion) within a large field of view (FOV). The alternative technique, proposed for reconstructing ultrasound images with high frame rates is known as Compounding Plane Wave Imaging (CPWI). CPWI balances image quality (e.g. spatial resolution, Signal-to-Noise Ratio (SNR)) and imaging frame rate and it is typically used in applications requiring ultrafast ultrasound. It is a complex technique with high data rates and high beamforming computations. Ultrafast ultrasound typically requires a peak data rate of 112 Gbps and a peak sampling rate of 8 GSPS for a 128-channel system with a 62.5 MHz 14-bit analog-to-digital converter (ADC). In order to achieve continuous, non-blocking ultrafast ultrasound imaging, a sustained average beamforming rate of the beamformer is not required to achieve the peak data rate because the duty cycle (acquisition time/acquisition interval) of ultrasound data acquisition is not 100%. Depending on the application,

this duty cycle is typically between 10% and 30% [6]–[8].

Since beamforming is a fundamental, inextricable part of an ultrasound system, it is evident that an ultrasound system which supports high frame rates must be able to achieve beamforming rates that keep up with the ultrafast imaging. This thesis will focus on designing a beamforming kernel that adapts to high frame rates. Specifically, the kernel will be deployed and tested on a Field Programmable Gate Array.

2.1 Problem Statement

Motivated from the aforementioned, the present thesis explores the problem of designing an efficient, hardware-based beamforming kernel for high frame rates. The main idea around the design is to create a hardware kernel that receives raw ultrasonic samples and beamforms them, compounds the contributions from different angles and stores them on the FPGA's DDR for further processing. Generally speaking, the beamforming kernel is supposed to receive streams of ultrasound samples from previous components of the ultrasound system. However, since the proposed design is a standalone kernel that is not integrated and tested on an actual ultrasound system, the ultrasound samples will be stored in advance in the DDR and will be read from the beamforming kernel, after launch. The ultrafast beamforming kernel is expected to achieve a beamforming rate in the range of GSPS, in order to support high frame rates (in the KHz range).

The beamforming algorithm that the proposed kernel applies is the Delay and Sum algorithm (DAS). This algorithm is known to calculate numerous traveltimes of the wavefront, between several pairs of pixels and transducer elements. The primary way to speed up the beamforming process, is to execute the aforementioned calculations in parallel, as they do not share any dependencies. The thesis goal is to explore for parallelization schemes that allow faster beamforming while also respecting the available FPGA resources. Moreover, it is important to locate the bottleneck of our design while increasing the variables of the application, whether that is the memory requirements or the executed operations per second. Towards this goal, after setting a specific size for the dataset the proposed kernel beamforms, we will additionally test its performance for different sizes of the transducer array, the image pixels etc. in order to explore the scalability of the proposed design. The desired result would be to create a fully scalable design, that applies beamforming to different datasets, with minimal changes. Designing a scalable ultrafast beamformer is a coveted goal in the medical ultrasound field, since the main desire is to beamform the received ultrasonic signal regardless of the experimental setup and the circumstances (ultrasound probe, insonified area etc), conditions that tend to alter from experiment to experiment. This explains why the importance of the scalability of our design is so critical.

Lastly, after validating the correctness of the proposed kernel and testing its scalability, the goal is to explore the integration of additional enhancing processes on the proposed kernel. Mainly, we will attempt to implement a linear interpolation to the given ultrasound samples, aiming to acquire more accurate beamformed values in the output. Furthermore, we will expand the design to also support IQ demodulated samples as an input, improving the versatility of the beamforming kernel, while obtaining the signal's envelope at the output, rendering the beamformed signal easier to visualize.

2.2 Thesis Objectives

This thesis seeks to make the following contributions:

- Explore the challenges of implementing an ultrafast beamformer on an FPGA device. Identify the memory bottlenecks, as well as the requirements in mathematical operations.
- Identify the parallelization schemes of the Delay and Sum beamforming algorithm, that offer a balance between resource consumption and efficient performance.
- Design a hardware based beamforming architecture on a single FPGA, that achieves high beamforming rates, allowing the integration of high ultrasound acquisition rates. The design will be

explored regarding its scalability, to achieve the goal of predicting its behaviour for different ultrasonic probes and scanned areas.

- Explore the feasibility of integrating linear interpolation, as well as phase rotation on-chip. These two processes will allow respectively, better beamforming accuracy and the opportunity of beamforming IQ modulated ultrasound samples.
- Evaluate the proposed kernels in terms of the achieved frame rates, aperture and final image size, as well as the quality of the reconstructed images.
- Provide a solid baseline to propose future upgrades and improvements regarding the design of an ultrafast beamformer, on an FPGA.

2.3 Thesis Structure

This thesis is organised in 6 chapters. The current chapter introduces the reader to the topic of the thesis, the motivation that encourages its composure and the goals it pursues. Chapter 3 provides a brief insight on the related work in the ultrasound imaging field, focusing on high frame rate systems. After discussing the benefits of high frame rates into several medical fields, state of the art software based and hardware based ultrafast beamformers are presented. In Chapter 4 we recount the basic theoretical concepts needed for a clear understanding of the remainder of the thesis and its purpose. Chapter 5 consists of the methodology behind the implementation of the proposed Delay and Sum beamforming kernel. Specifically, we present a reformation of the classic Delay and Sum algorithm that reduces the memory requirements of the FPGA kernel, as well as the architectural flow of the proposed design. Furthermore, in Chapter 6 we evaluate and compare our beamforming kernel with similar state of the art DAS kernels, that achieve high frame rates and are deployed not only on an FPGA, but also on a GPU. Lastly, in Chapter 7 we report the conclusions stemming from the evaluation of our work and we propose future research directions that could emerge from this thesis.

Chapter 3

Related Work

Ultrasound imaging is an enormous scientific field with lots of prior art, as well as many unanswered topics. In the present chapter, we will provide a brief but also detailed presentation of a number of previous works around ultrafast ultrasound imaging and it is organized as follows:

In section 2.1 fundamental findings about ultrafast imaging and coherent plane wave compounding are presented in conjunction with the breakthroughs they introduce in several imaging modes. Section 2.2 refers to plane wave imaging and Delay and Sum Beamforming on GPUs, the benefits that this device offers but also the drawbacks. Lastly, section 2.3 presents a handful of FPGA designs for Delay and Sum beamforming around ultrafast imaging, to try and highlight the performance of each architecture. Nevertheless, since this thesis revolves around Delay and Sum Beamforming for Plane Wave Imaging, this last chapter will provide a good baseline for analyzing the results of our implementation.

3.1 Ultrafast Ultrasound Imaging

As mentioned in the previous chapter, Ultrafast plane wave imaging (PWI) is a method that provides the potential of an ultrasound system that supports frame rates in the kilohertz range. Combined with coherent plane wave compounding, this imaging technique allows the reconstruction of high quality images with a reduced time of signal acquisition, as a cause of the limited amount of plane waves used. This unveils many emerging opportunities for clinical applications of ultrasound imaging in many fields, as the high frame rates could conduce to overcoming natural barriers regarding visualization of the human body. Imaging fields that could benefit from the described method are Functional Ultrasound Imaging of Brain Activity, Imaging of Natural waves, Shear Wave Elastography, Ultrafast Contrast Imaging, Ultrafast Doppler Imaging and many more.

In an attempt to compare the conventional Focused Ultrasound imaging with a High Frame Rate (HFR) Plane Wave imaging, Gabriel Montaldo et al. [4] created a theoretical model predicting that with PWI, a similar image quality compared to standard B-mode is possible, regarding contrast, SNR and resolution, by emitting 10 times less ultrasonic waves. After conducting experiments comparing the two methods, they proved that indeed by using a small number of plane waves they managed to produce images with similar SNR and contrast, better resolution while keeping the frame rate at a much higher level. 45 plane waves produce a similar image with their proposed Multi-Focused technique.

Additionally, the authors of [4] decided to apply plane wave imaging with coherent compounding into transient elastography with low-frequency shear waves at very low speed. Due to their low propagation speed, these waves are useful to assess the elastic properties and stiffness of soft tissues, but to catch any possible displacements a high frame rate is required (over 1 KHz), something possible through ultrafast imaging. After visualizing an heterogeneous PVA (Poly-vinil Alcohol) phantom by coherently

compounding plane waves, the authors noticed that the tissue displacement estimates are improved and the quality of these estimates is only affected by the side lobe effect. Namely, the coherent plane wave compounding allows a gain around 10 dB in contrast and a spatial resolution two times better than single beam insonification, improving the transient elastography imaging mode. Moreover, this approach ensures an improvement of the SuperSonic Shear Imaging (SSI) mode, regarding the spatial extent of the elasticity image and the accuracy local elasticity estimation. This could be extremely helpful for echographic imaging in body parts that usually reveal spatial heterogeneities, like the breast, muscles and the heart.

Along the same lines Jeremy Bercoff et al. [14] described the contribution of Ultrafast imaging into full blood flow visualization, in slow and fast blood flows. The first observation the authors make, is that in Doppler-based flow analysis, PWI could reduce the signal acquisition time by 16 times. As far as fast blood flows are concerned, the high frame rates are more efficient, as they could provide better temporal resolution for transitory and turbulent flows. Even for the fastest sequence, this method outperforms focusing imaging regarding image quality, resulting into better spatio-temporal continuity. On the other hand, for slower blood streams, the plane wave imaging method offers improved flow detection and definition as each pixel is derived from several temporal samples, enabling a higher sensitivity. By increasing the plane wave range, although the gain in acquisition time is reduced, the final image is enhanced regarding contrast and resolution. All in all, the authors make it clear that Ultrafast compound Doppler imaging could introduce breakthrough performances in flow analysis.

Emilie Mace et al. [8] propose a real-time functional ultrasound (fUS) imaging on the brain, based on high frame rate plane wave imaging. As the authors outline, ultrasound imaging has the potential to support an in-depth imaging of brain hemodynamics. However, conventional Doppler imaging cannot provide the necessary sensitivity in order to detect blood flow through small vessels inside the brain, even if the signal attenuation and aberration by the skull can be surpassed. To overcome this limitation, the authors analyze in depth a power Doppler imaging method, which is based on acquiring multiple micro-Doppler images in a high rate. The high frame rates enable the collection of more temporal samples for a certain pixel in the same acquisition time, increasing the technique's sensitivity and consequently detecting more brain microvessels. Through experiments conducted on animals, Emilie Mace et al. [8] noticed that plane wave imaging provided a clear improvement to the images' contrast and SNR.

We provided a glimpse of the capabilities of Coherent Plane wave compounding in medical images, and the breakthroughs it could possibly induce in a handful of real-time imaging modalities. Nevertheless, it is crucial to implement this technique into systems that support highly parallelized signal acquisition processes in order to perform ultrafast imaging sequences. Nowadays, many platforms provide this opportunity with the most popular being the Verasonics platform and the ULA-OP 256. Furthermore, the processing rate of these ultrafast signals must be higher than the acquisition rate, in order to support real-time imaging. The first devices that dispose the necessary computational power to support ultrafast imaging applications were the GPUs and later on designs of these applications expanded on ASICs and FPGA devices.

3.2 Ultrafast Beamforming on GPU devices

Recently, the emergence of GPU devices has rapidly propelled the development of real-time ultrasound systems. Their Single Instruction Multiple Data (SIMD) architecture enables the possibility of high computational and efficient parallel task execution, benefiting significantly the acceleration of ultrafast imaging applications. Dongwoon Hyun et al. [15] made an attempt to provide an open source GPU-based beamformer that supports plane wave imaging. Their repository provides functions for data management, data processing with computational graphs, as well as neural network processing with deep learning. After testing their GPU kernels, the authors managed to reconstruct images of size around 456×305 pixels with 25 plane waves and 47-55 frames per second.

Upgrading the GPU-based ultrafast ultrasound imaging, Billy Y. S. Yiu et al. [16] presented one of the first attempts of implementing an ultrafast imaging processor on GPU devices. Their system

utilizes two GPU devices for analytic signal conversion and delay and sum beamforming and one GPU for coherent compounding. The DAS kernels are separated into block of threads, and each block is allocated to calculate the beamformed pixels of a two dimensional sub-grid of the final image. After a batch of low resolution images is complete, it is sent to the GPU designated for coherent compounding. The final image is produced from 49 plane wave and consists of 512×255 pixels, with an aperture size of 32, 64 and 128 channels. The maximum frame rate achieved by this approach equals approximately to 4750 Hz and is achieved with an aperture size of 32 channels and an image depth of 5cm. On the other hand, the minimum frame rate equals to 750 Hz and is achieved for an image depth of 15 cm, by utilizing 128 channels.

Proving the feasibility of their architecture, the authors of [16] created a real-time scanning platform implementing high frame rate ultrasound for color-encoded speckle imaging [17]. They integrated their high performance GPU beamforming cores into the system's architecture for fast, real-time signal processing. The live implementation was possible by the use of a 192-channel programmable ultrasound front-end module, which is receiving signals emitted with a high pulse repetition frequency (PRF) and is streaming the received samples to the GPUs, using a streaming link with a capacity of 4.8 GB/s. The graphical cores are able to perform with a real-time throughput of 55 fps, a value that is suitable for live video display.

To reinforce the claim about GPU devices being appropriate for high frame rate ultrasound systems, two additional ultrafast architectures with GPU-based beamformers will be introduced. Pascal Alexander Hager et al. [1] designed an ultrafast imaging platform named UltraLight, based on a digital 64-channel ultrasound probe. A NVIDIA GTX1080 GPU is integrated to the system, responsible for performing ultrafast beamforming and compounding of plane waves, as well as rendering B-mode images. The digital probe is configured to insonify 31 plane waves from different angles and the system captures 2048 samples per channel with a frequency of 20 MHz, covering a depth of 6 cm. Furthermore, Christoph Risser et al. [18] presented an ultrafast beamforming platform called DiPhAS, that has the potential to support 3-D and even 4-D ultrasound imaging. The system consists of 256 parallel channels both for generation and reception of the ultrasonic waves, that can be extended to a 1024-element matrix transducer with the help of an integrated 1:4 multiplexer. The samples are transferred to a PC via PCI-Express and are transferred to the GPU for parallel processing. DiPhAS platform provides the opportunity for optimized volumetric beamforming and 3-D/4-D real-time applications.

Despite their powerful performance in parallel execution, GPUs face a major bottleneck in transferring data from the ultrasound system to the PC [7], [15]. As stated previously, FPGA devices are able to overcome this obstacle, making the data transfer from the acquisition module negligible. At the same time, these hardware devices are able to provide powerful, parallel computational performance with low power consumption, which makes them a viable alternative to GPUs for accelerating ultrasound applications.

3.3 Ultrasound Beamforming on an FPGA platform

Previously mentioned on Chapter 4, high frame rate imaging is the newest alternative to conventional ultrasound imaging methods, providing several options to overcome a handful of their inherent limitations regarding image quality and frame rate. Imaging based on the transmission of focus beams can result to non uniform images with lower resolution and possible artifacts. Additionally, methods like this require many transmit-receive events for the construction of a single frame, thus increasing the acquisition time and reducing the system's frame rate. Conversely, high frame rate imaging methods utilize transmission of multiple simultaneous focused beams or plane waves from different transmit angles. While this approach could conduce to improving the aforementioned drawbacks, it could also invite undesired artifacts that originate from different points of the insonified area. To reduce their effect on the final image, compounding is applied on the combination of echoes from different steering angles.

State of the art around ultrasound imaging has already introduced a number of beamformers designed on hardware that support high frame rate imaging. One of the earliest attempts into a hardware

based, high frame rate ultrasound system was proposed from Jayaraj U Kidav and Sreejeesh S. G [2]. Their FPGA design supports an aperture size of 128 channels and forms 28 beams per plane wave transmission. It is based on a 28-read and 1-write port BRAM that stores all the delays necessary for beamforming, loaded from the external SRAM. This architecture manages to achieve an output beamforming rate of 1120 MSPS and a Frame Rate of 714 per second, but with a small image size of 64×181 and an aperture size of 64 channels.

Furthermore, Enrico Boni, et al. [5] proposed the hardware architecture of a continuous, real-time compounded ultrasound imaging application in the ULA-OP 256 research platform. This system consists of four Delay and Sum processing units that receive samples from an Analog Front-End board (AFE). Each AFE controls 32 transducer elements and each DAS unit is responsible for beamforming a single image line, also applying quadratic interpolation, apodization weights and coherent summing. In each DAS beamformer, 32 delay and apodizations cells are included, one for every channel. After the entire beamforming procedure is over, all data are sent to the DDR memory and then to DSPs that apply quadrature demodulation, low-pass filtering, downsampling and compounding to the beamformed data. This beamforming architecture is designed on an FPGA of the ARRIA V GX Family, and it produces images with a subaperture size equal to 32 channels. This system manages to achieve a continuous pulse repetition frequency (PRF) of 3,800 Hz, a maximum Frame Rate (FR) of 1100 Hz, depending on the number of plane waves, and its beamforming units are able to produce up to 467 MSamples per Second.

A similar design is provided by Nicholas A. Campbell et al. [9], who propose a real time dual-mode high frequency beamformer that supports focused and ultrafast imaging. Depending on the mode selected, this beamformer can alternate between producing focused images of a higher quality and many low quality Ultrafast images. The system described by the authors is hardware based and consists of ten Kintex 7 FPGAs (xc7k70tfbg484-2; Xilinx Inc., San Jose, CA, USA), one utilized for the transmit portion of the system and the rest nine from the system's receiver. Eight of the FPGAs are used to beamform the samples originated from 64 transducer elements, and a group of eight channels is assigned to each FPGA. Regarding the focused imaging mode, samples are upsampled before being interpolated and beamformed, with the Delay and Sum algorithm. The beamformed results are then downsampled, producing a line with 768 pixels. The final image has a size of 768×128 from 64 transducer elements. On the other hand, the ultrafast mode works in a similar way but produces an image of size 384×64, after compounding the beamformed data with the next received diverging wave. Additionally, all DAS processors have access to stored, pre-calculated delay values. To compress the delays and store them in memory efficiently, the authors utilize a variation of the delta encoding method. They rewrite the delays relative to the delay corresponding to the first pixel, creating a linear approximation for these delays, with respect to their index. The error for each pixel is bounded between -0.83 ns and +0.83 ns, for most pixels. This architecture achieves an Ultrafast Frame Rate of 875 Hz for 16 diverging waves, 8 probe elements and an image size of 64×384. Approximately, it uses over 50% of the FPGA's resources and it achieves a beamforming rate of 917 MSPS. Lastly, at 10 Hz for one focused image the system is able to generate 68 UF images.

In contrast with the aforementioned design, Zhengchang Kou et al. [7] have proposed a scalable ultrafast ultrasound beamformer that operates on a single FPGA. Although this design it has not been integrated into an actual ultrasound system like the aforementioned designs, it achieves a 4.83 GSPS beamforming rate. The authors introduce an alternative way of handling the enormous amounts of delay values by revising the delay and sum algorithm, in order to decrease the delay size profile. The beamforming architecture of this kernel executes a linear beamforming of the image's rows, where each row is processed in eight clock cycles. With the reformed DAS algorithm, pixels at the same depth use the same set of delay values, enabling delay reuse. It is noteworthy that this specific kernel functions only with raw RF samples and applies a more naive interpolation to the input samples, as each interpolated row of raw RF data occurs from the mean of two original rows. This delay profile reuse method improves the memory utilization inside the FPGA and simplifies the memory architecture, while at the same time it reduces the magnitude of data processing, speeding up significantly the calculations. The kernel is able to achieve a Frame Rate of 14865 Hz, with an average beamforming rate of 4.83 GSPS for an image size of 64×1280 and an aperture size of 64 transducer elements. The

image quality did not show any drops in CNR and lateral resolution which indicates that the design did not sacrifice image quality for processing speed.

Table 3.1 sums up the aforementioned hardware-based beamfomers, providing a comparison regarding their frame rates, MSamples per second, the ultrasound image size as well as the aperture size.

Design	***PRF (Hz)***	***BF Rate (MSPS)***	***Frame Rate (Hz)***	***PW***	***Image Size***	***Aperture Size***
[2]	714	1120	714	1	64×181	64
[5]	3800	467	345	11	96×512	32
[9]	14,000	917	875	16	64×384	8
[7]	14,865	4830	14,865	1	64×1280	64

Table 3.1: Performance of state of the art FPGA ultrafast beamformers

The designs mentioned above will be the guideline regarding evaluating the performance of our design. However, it is important to keep in mind that we cannot form a precisely accurate comparison between our kernel and the aforementioned designs, since there are slight differences into the beamforming choices each designer makes. Some architectures also support apodization, upsampling and downsampling, IQ demodulation and coherent compounding, while others prefer to execute these processes outside of the FPGA, into some DSP devices for example. Moreover, most of these designs are also integrated and tested into an actual ultrasound system, whereas our kernel is not. On the other hand, one thing that is worth mentioning is that all of the architectures above decide to pre-calculate the delays necessary and loading the into an external memory, rather than calculating them on chip. This suggests how computational heavy those calculations are, rendering them unsuitable for an FPGA device.

Chapter 4

Background

In the current chapter, background information that is needed for a proper understanding of the rest of the thesis is presented. In Section 4.1, we discuss the basics of ultrasound imaging, as well as the physics behind the whole procedure. Sections 4.2 and 4.3 provide a detailed insight on the methods used to insonify the scanned area for the conventional ultrasound technique and the ultrafast imaging respectively. A thorough definition of ultrasound beamforming is given in section 4.4, section 4.5 covers the standard delay and sum beamforming, a fundamental algorithm for ultrasound image reconstruction and section 4.6 describes certain alternative beamforming algorithms. Moreover, to add to the image processing analysis, section 4.7 includes further image enhancing techniques like apodization, IQ demodulation and f-number usage and also presents the log-compression and the envelope detection, post-processing procedures that benefit the visualization of the ultrasound image. After analyzing the fundamentals of ultrasound imaging and digital beamforming, an illustration of a standard hardware-based, ultrasound system and its components is provided in Section 4.8. Lastly, in Section 4.9, a brief reference regarding FPGAs and their contribution in medical imaging is given, in conjunction with the Alveo U200 Data Center Accelerator Card characteristics and with a short definition of the AXI protocol and the benefits it offers into efficient memory transactions.

4.1 Physics of Ultrasound Imaging

Ultrasound imaging is a non-invasive medical imaging modality that helps a physician evaluate, diagnose and treat medical conditions and it is commonly used due to its low cost, portability, and real-time capability. This imaging method is based on the ideas of the Doppler effect, as it works on the principle of producing and receiving high frequency acoustic waves (1-20 MHz), aiming at the visualization of the inside of the human body.

These ultrasound pulses are generated by applying a voltage potential at a piece of piezoelectric material, called transducer element, which vibrates and eventually insonifies the area of interest. These piezoelectric devices convert electric energy to acoustic waves and vice versa. When an electric current is applied to their piezoelectric crystals, they begin to vibrate generating sound waves with a frequency in the ultrasound range. Correspondingly, when these crystals are hit with the reflection of the ultrasound wave they emit, they vibrate once again, transforming these mechanical vibrations into electric current [19]. An abstract view of an ultrasonic probe, like the one we have just described, is illustrated in Figure 4.1.

Therefore, to produce an acoustic wavefront and cover the entirety of the desired medical area, a series of these transducer elements is used, called ultrasonic probe, whose characteristics determine the central frequency of the transmitted acoustic wave. If this wave hits a material target, the pulse is attenuated and partially reflected as a result of the difference in acoustic impedance between different tissues in the insonified area. The backscattered echoes are also tracked by the ultrasonic probe in

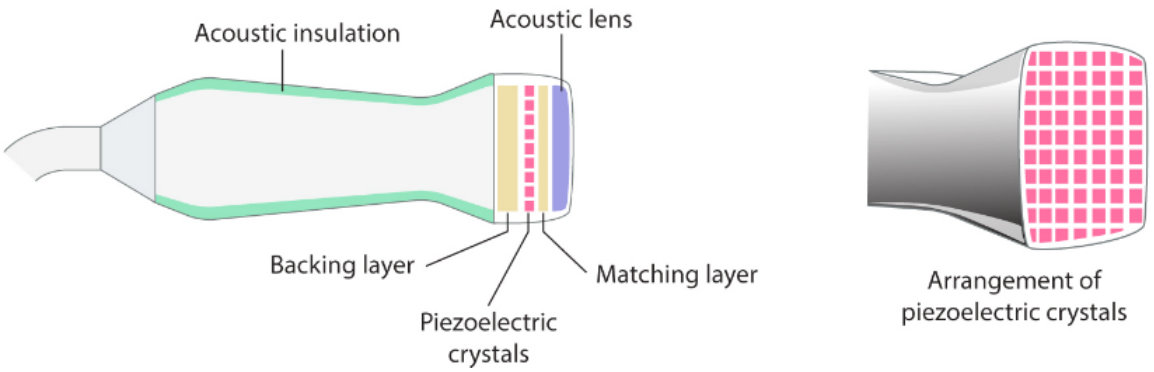


Figure 4.1: The piezoelectric material that generates and tracks ultrasonic waves [19].

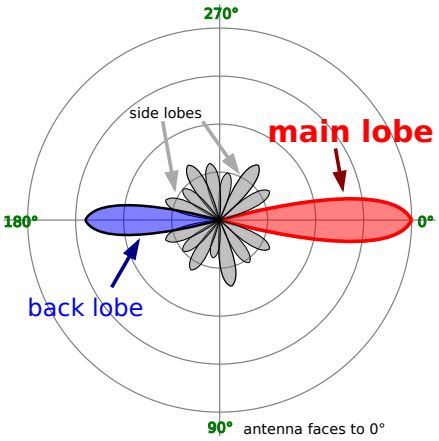
the form of a small voltage potential. The electric signal is then sent to the ultrasound system for processing and image reconstruction [19].

The probe is always connected to a back-end system, which includes the analog front-end electronics and the digital processing and control unit. After receiving the information from the echoes, the signal is amplified and digitized by the analog front-end. The digitized signal is then processed with an algorithm called beamforming, which allows the signals, after some additional post-processing steps, to be constructed in an image presenting the geometrical information of the scatterers in the depth and the lateral axis [20], [21]. Due to the fact that intensity of the displayed pixel corresponds to the signal value, and thus the amount of the signal reflected by a scatterer, this imaging mode is also called brightness mode or B-mode.

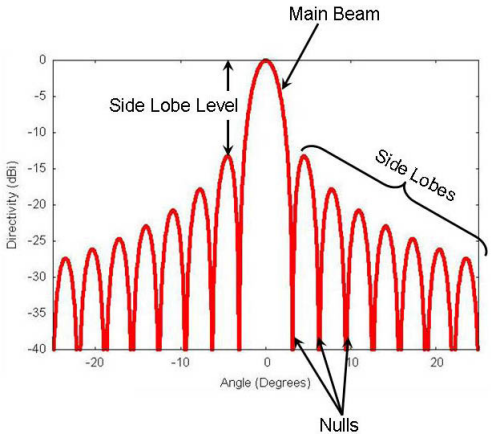
4.1.1 Side Lobes and Grating Lobes artifacts

One major downside of the antennas used for ultrasound, is that they cannot radiate signals coherently equally in every direction, due to the nature of electromagnetic radiation. Instead, most antennas use a radiation pattern, that includes points in different angles where signal strength reaches a local maximum. These points are known as lobes. These lobes are separated by nulls, angles where the signal strength equals to zero (Figure 4.2a). Ultrasound imaging utilizes directional antennas, whose objective is to emit the ultrasonic signal in a single direction. To achieve that, the lobe in that direction is designed to have a higher strength compared to the other lobes. The lobe with the highest strength is called the main lobe and the other lobes are called side lobes (Figure 4.2b) [22]. Side lobes are usually considered as a source of image artifacts, because when the signal originated from these lobes encounters a strong reflector, the received echo appears falsely to originate from the central beam, which emits in a different angle. These backscattered echoes create spurious indications in the final image [23]. A special type of side lobes are the grating lobes and they are born due to spatial aliasing effects. They can regularly be found on discrete aperture antennas, like phased arrays, where the element spacing is greater than a half wavelength. It is important to distinguish grating lobes from side lobes, because their amplitude reaches the strength level of the main lobe [22], thus needing different techniques to diminish their effect. Figure 4.2c provides an illustration of main, side and grating lobes.

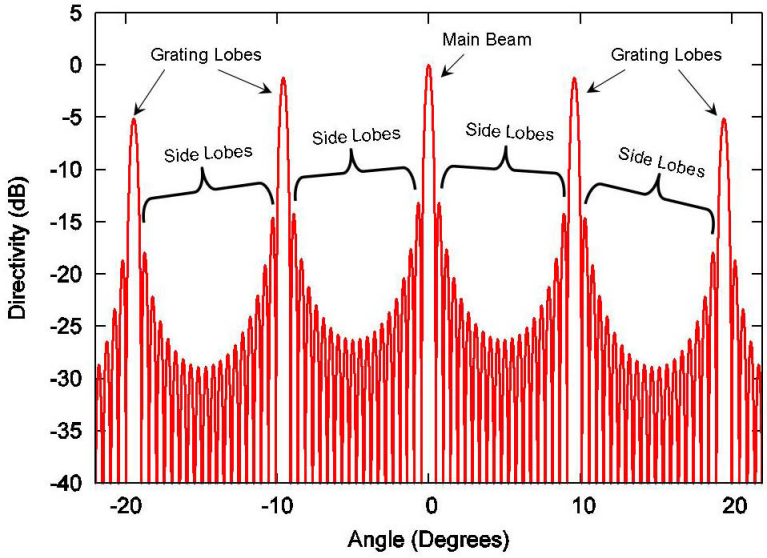
Side lobes are an unfortunate reality for directional antennas and their effect cannot be eliminated completely. The main way to minimize their intensity during transmission is to precisely control the power emitted by each transducer element, in a way that the furthest elements from the centre of the probe produce less power [24]. Nonetheless, designing low power sidelobes affects significantly the system's cost. On one hand, lower sidelobes result into a higher beamwidth, thus requiring a larger antenna. On the other hand, the precise control of the element's signal power is by itself a costly procedure regarding design cost and production. This creates a crucial need to diminish the side lobe effect by appropriately processing the signal after reception.



(a) Main lobe and side lobes



(b) Strength levels of main lobe compared to side lobes



(c) Grating lobes

Figure 4.2: Illustrations of the main lobe, the side lobes and the grating lobe for a typical directional antenna [22].

4.2 Conventional Ultrasound Imaging

The main technique, that was developed for ultrasound imaging, was the insonification of the scanned area with sequential focused ultrasonic beams. Figures 4.3a and 4.3b depict how focused beams are transmitted from the ultrasonic probe. Typically, a standard two dimensional image is reconstructed with 64-512 scan lines and each scan line is collected with a single receive-transmit event. Multiple scan lines are not received at the same time. As a consequence, conventional ultrasound systems are designed to receive the echo of one beam at a time, process the image line and then proceed to the next beam. However, ultrasonic waves cannot travel faster than the speed of sound inside the human body. This is the main factor that restricts the temporal resolution of the reconstructed image, as ultrasound pulses can be transmitted at pulse repetition frequencies around 5-10 KHz, depending on the organ [25]. Therefore, for a two dimensional image with a depth equal to D, the time required to receive every focused beam's echo equals to :

$$T_{rec} = \frac{N_{beams} \cdot 2 \cdot D}{c_{sound}}$$

Under the given circumstances, if the processing time of an image line is less than the acquisition time of the focused beam, the maximum frame rate that can be achieved equals to :

$$FR_{max} = \frac{1}{T_{rec}}$$

Taking into consideration that an acceptable depth for ultrasound imaging is around 5-15 cm, then the maximum frame rate that can be achieved is around *100 fps*. However, a handful of medical applications require rapid dynamic responses of biological tissues, which can only be observed and analyzed with higher frame rates, like Functional Ultrasound Imaging of Brain Activity, Shear Wave Elastography, Ultrafast Contrast Imaging and more [4], [8], [14]. To improve the temporal resolution of conventional ultrasound imaging, an alternative imaging technique is essential. To satisfy this cause, experts have resorted to an imaging technique called ultrafast ultrasound imaging, which allows frame rates in the KHz range.

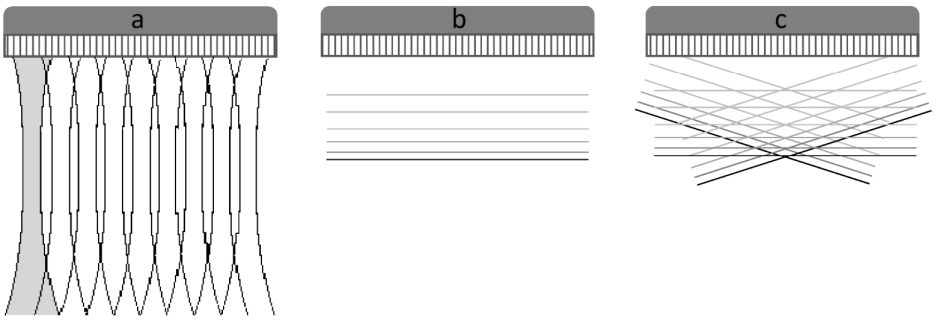
4.3 Ultrafast Ultrasound Imaging

The imperative need of drastically increasing the frame rates of the ultrasound system for medical applications that require high temporal resolution, resulted in the emergence of Ultrafast imaging. Its fundamentals rely on emitting multiple unfocused ultrasonic beams, such as plane waves and diverging beams [26]–[28]. Figures 4.3a and 4.3b illustrate how plane waves are emitted from the ultrasonic probe, in contrast with focused beam patterns. These waves are generated by applying a plane or circular delay to the transducer elements [28]. Rather than applying a line-by-line pulse-echo imaging, this high frame rate imaging technique enables the reception of full-field ultrasonic echoes with a single emission, decreasing the necessary emissions for visualizing the entirety of the insonified area [27]. To reconstruct the desired image, applying the beamforming algorithm to the received signals is essential. Beamforming encourages the formation of focused beams in reception, which are used to retrieve the localization of echoes, enabling the possibility of creating multiple scan lines at the same time [29]. Therefore, in an extreme case the output image could be reconstructed by a single unfocused ultrasonic wave transmission, increasing significantly the ultrasound's frame rate [27]. Indicatively, depending on the chosen imaging depth, this method could allow frame rates up to 10000 fps, around 100 times higher than the frame rate of the conventional ultrasound technique [29].

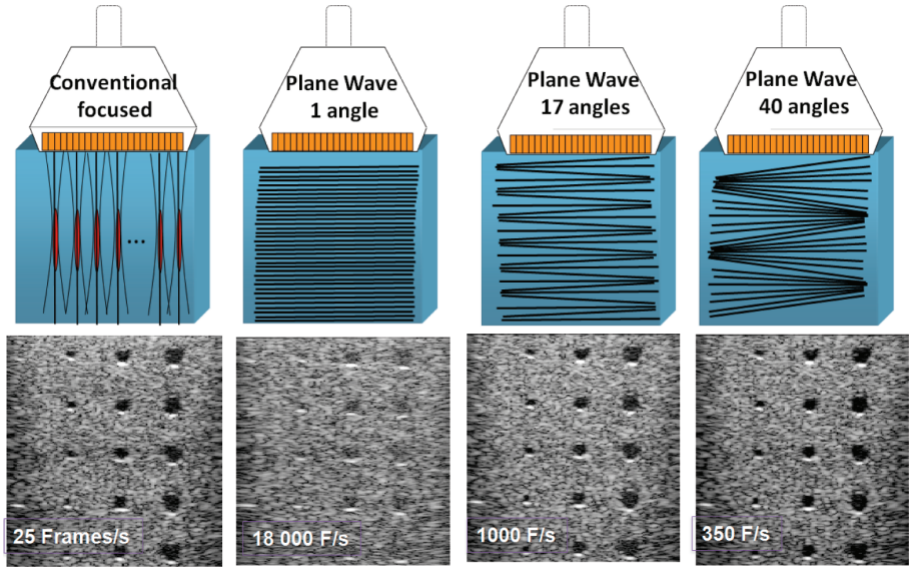
Despite the significant increase in frame rates, plane wave imaging introduces a trade-off regarding image quality. The lack of focusing results into lower spatial resolution and contrast of the reconstructed images, because focalisation is created inherently in reception and the unfocused waves illuminate a wide imaging range, producing unwelcome echoes that play the role of noise. To overcome the regression in image quality, a coherent compounding of multiple tilted plane waves is proposed. By summing coherently point spread functions (PSFs) from different steered angles, the compounded output is improved, because only the echoes originated from a scatterer are coherently summed, and the other parts are thus canceled. Using this method, the transmit focalisation is retrospectively achieved by the coherent summation [27].

It is apparent that the more distinct plane waves someone utilizes, the higher quality of image is achieved. On the other hand, increasing the amount of plane waves needed for reconstructing a single image inevitably reduces the application's frame rate. Therefore, a new interesting trade-off between high image quality and high frame rates is introduced. However, it is proved that even if coherent combinations of compounded plane-wave transmissions limits the maximum frame rate, the achievable frame rate is still substantially higher, allowing concurrently high spatial image resolution with 5-10 times less ultrasonic wave emissions, compared to the conventional focused imaging technique [4].

Ultrafast ultrasound imaging introduces several innovations around medical imaging. It provides the potential of observing and capturing transient physiologic events, that are altered in a rapid pace,



(a) a) Line by line acquisition, by using focused beams. b) Plane wave imaging with a single emission. c) Plane wave imaging with several plane waves [30]



(b) a) Conventional focused imaging (128 beams, 25 fps). b) Plane wave imaging (18000 fps). c) Plane wave imaging from 17 transmit angles (1000 fps). d) Plane wave imaging from 40 transmit angles (350 fps) [26].

Figure 4.3: Comparing ultrasound wave emissions during conventional US imaging and PWI.

such as blood flow, brain and cardiac activity, arterial pulse wave propagation and many more. Novel imaging modalities are benefited highly from UF imaging, with Shear Wave Elastography (SWE), Ultrafast Doppler Imaging and functional US (fUS) imaging of the brain only being some of them. Figure 4.4 given by Mickael Tanter and Mathias Fink [26], sums up perfectly our analysis. Moreover, this imaging technique allows the distinction between tissue and blood, since the area of a tissue is known to be more spatially coherent than blood. Consequently, ultrafast ultrasound avails not only the tracking of high velocity flows but also the discrimination of tissue movements from blood flow [31], [32].

4.4 Ultrasound Beamforming

As mentioned, beamforming is an essential stage of the image reconstruction process in numerous fields, including wireless communications, acoustics, radar, sonar and of course the medicine field. It is mainly a radio frequency (RF) management technique, directing radio and sound waves for signal transmission and reception.

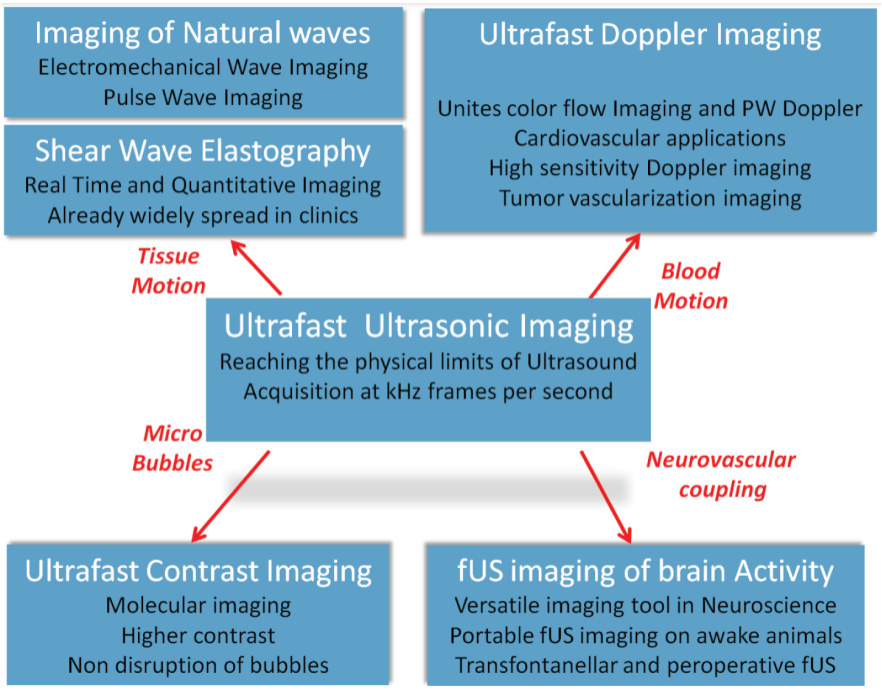


Figure 4.4: Medical applications benefiting from the evolution of ultrafast imaging [26].

Beamforming is applied in both the transmitted and the received acoustic wave and acts as a spatial filtering process. In medical imaging, it deals with the focusing of the transmitted acoustic wave at the desired region and the recombination of the received ultrasound signals for the purpose of reconstructing images. Instead of sending the signal in all directions – how a signal would traditionally be transmitted – the travel path of the acoustic energy of the wave can be focused on a certain area, in order to send a high amplitude wavefront and maximize the signal strength [33], [34]. To achieve this goal, the beamforming technique is incorporated with an array transducer that divides the transducer surface into sub-regions, called elements. Each element is geometrically focused, thus its focal point is determined by its curvature. By controlling each element separately, it is feasible to produce time controlled excitations, in order to accordingly plan the diffraction of each individual wavefront generated from the elements and create a focused acoustic pulse. By adjusting the pattern of delays applied, the focal region can easily be determined [34]. This process is called transmit focusing. The Figure 4.5 demonstrates this whole idea.

On the other hand, in the receiving end, the reflected waves (raw channel data) are delayed in a way that the contributions from each element, coming from the same given point of the medium, are summed up together. In more detail, having an array of elements means that all the signals from a certain focal region will arrive at each transducer element at different times. The differences in signal arrival time should be compensated for, by applying predefined delays to the electrical signals after they are received. This process of delaying the received signals from the same focal point in order to align their phases and enhance the energy coming from a desired direction is called Receive Beamforming, and is demonstrated in Figure 4.6 [34], [36]. One could clearly understand that the delays applied are only dependent on the geometry of the insonified area and completely independent from the received signal amplitude. This fact means that the delays could easily be pre-calculated even before the transmission of the ultrasound wave, which could be helpful at the design of a fast and efficient beamforming system.

In this thesis we will focus entirely on the Receive instead of the Transmit Beamforming, as we will search for an efficient, real-time implementation on a Field Programmable Gate Array (FPGA). One can thus navigate through the different techniques using the following question as a compass: which

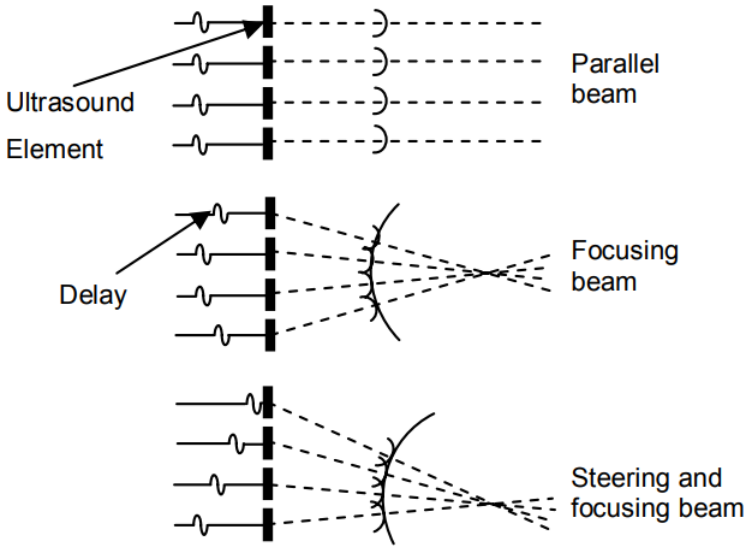


Figure 4.5: Transmit delay compensation for beam steering and focusing [35]

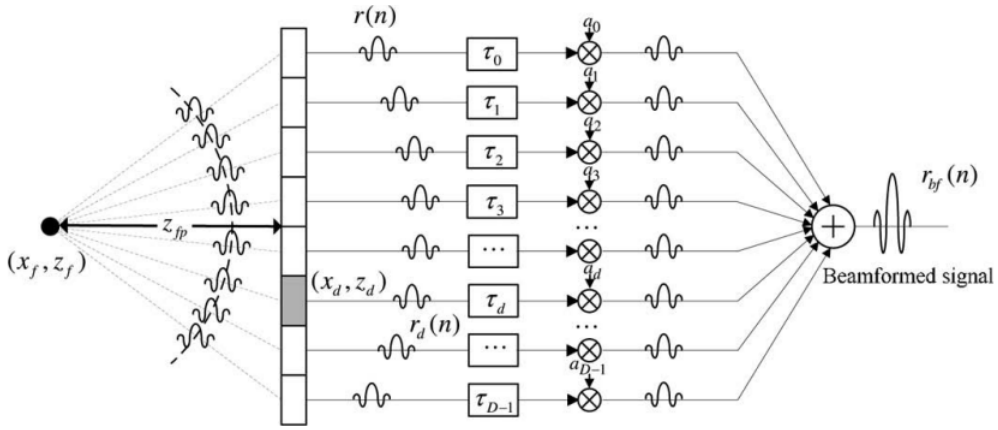


Figure 4.6: Receive Beamforming demonstration for a single target [20].

imaging features are important to my application of interest, and which features can I sacrifice? There is, in fact, no ultimate beamforming approach, and the answer to the previous question strongly depends on what one wants to see in the images.

4.5 Delay and Sum Beamforming

Delay and Sum (DAS) Beamforming is the most conventional and widespread beamforming technique, mainly due to its simplicity and its compatibility with real time applications. On an attempt to provide a succinct description of its fundamentals, we would point out that while applying delay and sum on a received signal, radio waves of the different antennas are time-shifted with predefined delays (the greater the receive angle, the greater the delays), so that they can be summed to increase the signals in the desired receive direction while attenuating those from undesired directions, thus aiming to the reduction of undesired sound on the final image.

4.5.1 Geometric Analysis

We will describe the geometric properties of DAS Beamforming under the assumption of a plane wave emission from the ultrasound probe, only because high frame rate ultrasound is most often used in conjunction with the emission of plane or circular waves. The main goal of this beamforming technique is to calculate the traveltime of the transmitted ultrasonic signal from the transducer to every scatterer, and back from the scatterer to every element of the transducer. That procedure will help us determine what part of an element's received signal corresponds to a specific scatterer. By combining all these signals through a weighted summation, we obtain the beamformed value of a certain scatterer on the insonified area. Consequently, if we follow this process for every desired scatterer on the scanned area, we successfully acquire the beamformed image [36].

To provide the simple fundamental equations of delay-and-sum beamforming, we assume that the speed of sound in the medium is uniform (a standard and safe assumption). By applying the simplest of physics laws, the aforementioned traveltime of the wavefront, for a specific target of coordinates $X_s = (x_s, z_s)$ and a specific probe element $X_i = (x_i, z_i)$, is calculated by the following fraction :

$$\tau(X_s, X_i, \theta) = \frac{d_{TX}(X_s,\theta)+d_{RX}(X_s,X_i)}{c}$$

where **c** equals to the speed of sound inside the medium and $d_{TX}(X_s)$ and $d_{RX}(X_s, X_i)$ equal respectively to the transmit distance between the wave's source and the target, and the receive distance between the target and the probe element. A visualization of these distances is provided by Figure 4.7.

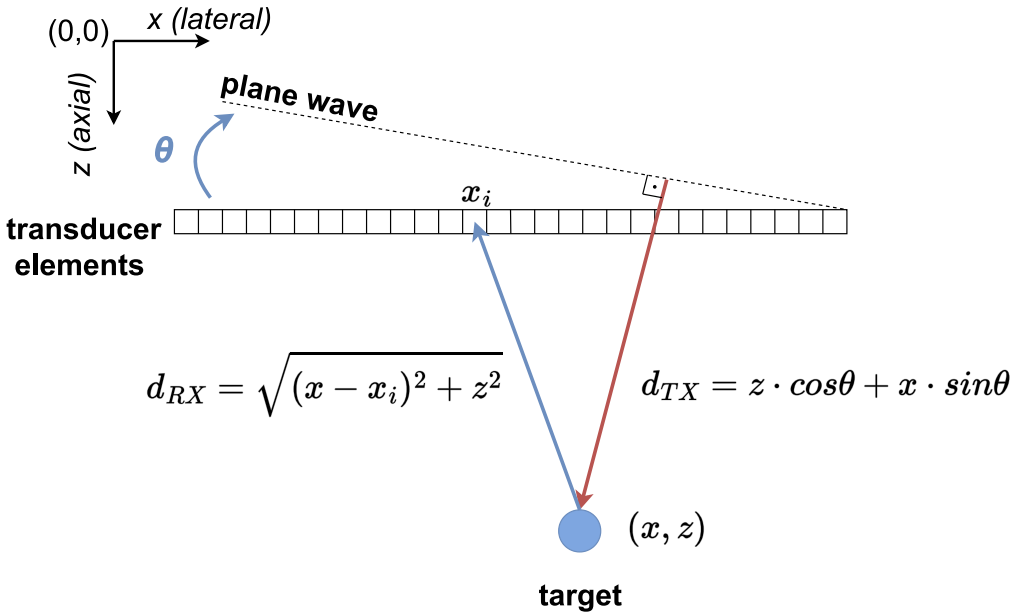


Figure 4.7: Transmit and Receive distances for a single pair target-element, assuming plane wave emission from a steering angle θ.

As we assume that the source emitting the ultrasonic wave is placed on the start of the geometric axes, the transmit distance is only dependent on the coordinates of the scatterer and the angle the waveform is transmitted from. Therefore, the transmit distance for a single scatterer $X_s = (x_s, z_s)$ from a steering angle θ equals to :

$$d_{TX}(X_s, \theta) = z_s \cdot cos(\theta) + x_s \cdot sin(\theta) \text{ (Figure 4.7)}$$

On the other hand, the receive distance does not depend on the transmitting angle but it is defined by the coordinates $X_s = (x_s, z_s)$ of the target and the coordinates $X_i = (x_i, z_i)$ of the probe element. Assuming that the ultrasound probe is placed onto the x-axis, for every element $z_i = 0$, so $X_i = (x_i, 0)$. As a result, by using simple pythagorean and geometry rules, the desired receive distance equals to :

$$d_{RX}(X_s, X_i) = \sqrt{(x_s - x_i)^2 + z_s^2} \text{ (Figure 4.7)}$$

If we define as $s_i(t)$ the ultrasound signal the element X_i obtains, then the beamformed value $s_{bf}(X_s)$ for a single pixel $X_s = (x_s, z_s)$ of the final image is calculated from the equation :

$$s_{bf}(X_s) = \sum_{\theta \in \text{angle\_set}} \sum_{i \in \text{channel\_set}} s_i(\tau(X_s, X_i, \theta)) \tag{4.5.1}$$

However, something important we should point out is the fact that each element's signal is sent as an input for processing from the analog front-end (AFE). This means that these signals are amplified and digitized, resulting into not having a continuous waveform but rather samples of the waveform, chosen according to the sampling frequency of the AFE. Consequently, it is highly likely that some values $s_i(\tau(X_s, X_i, \theta))$ needed for beamforming are not present in the element's sample pool, and as a result we should estimate them from the available samples. To achieve that, an estimation method called interpolation is usually applied [36]. Another observation is that these traveltimes discussed above, are completely independent of the input data coming from each probe channel and are defined exclusively from the geometry of the insonified area. Therefore, the desired delays could be pre-computed in advance, before the launch of the beamforming process, instead of being calculated on the fly. This note can be the origin of valid questions regarding which of the two aforementioned delay calculation strategies is optimal, and the answer varies depending on the beamforming system's specifications.

4.5.2 Q-Point Interpolation

Interpolation is a method used in the mathematical field, to construct an estimation of desired data points based on the range of a specific discrete set of known data points. There is a plethora of interpolation techniques like linear, polynomial, spline interpolation and more. Basically, when implementing a q-point interpolation, the discrete set used for estimating consists of the q closest points to the data point we aim to estimate. Generally, if we have n data points, there is exactly one polynomial of degree at most q-1 going through all the data points. So a q-point interpolation is a polynomial interpolation where a polynomial of degree q-1 is used [37]. By using the q closest points to a desired point of unknown value, we can determine a polynomial of q-1 degree. If we assume that the unknown point is also included into that polynomial, then we can acquire its estimated value from the polynomial's equation. The higher the degree of that polynomial, the more precise is the estimation but at the same time the more complex are the calculations.

For the purpose of implementing a beamforming FPGA kernel we will focus entirely on a 2-point interpolation, also known as linear interpolation. The accuracy it provides is enough for the requirements of the reconstruction of an ultrasound image and also we need to keep its complexity as low as possible, so we do not make the interpolation process a resource hungry procedure for the FPGA. A brief description of the linear interpolation between two known points is the following. Assuming the two known points $X_0(x_0, y_0)$, $X_1(x_1, y_1)$ and the point $X(x, y)$ where $x_0 \leqslant x \leqslant x_1$ and y is the value we aim to estimate. Then, from the equation of the linear function that connects the two points, the desired estimation equals to $y = y_0 \cdot (\frac{x_1 - x}{x_1 - x_0}) + y_1 \cdot (\frac{x - x_0}{x_1 - x_0})$.

If we consider that X_0 and X_1 are consecutive samples from the sample pool of a single probe element, then $x_1 - x_0 = 1$, and also $y_s = s_i(\tau(X_s, X_i, \theta))$ for the pixel X_s. Consequently, the interpolated beamformed value for the pixel X_s coming from the probe element X_i equals to :

$$y = s_i(\tau(X_s, X_i, \theta)) = s_i(\tau(X_0, X_i, \theta)) \cdot (x_1 - x) + s_i(\tau(X_1, X_i, \theta)) \cdot (x - x_0),$$
$$\text{where } \tau(X_0, X_i, \theta) \leqslant \tau(X_s, X_i, \theta) \leqslant \tau(X_1, X_i, \theta).$$

This value y is the estimated value for the examined scatterer's echo, received by a specific transducer element. This estimation method yields a satisfactory accuracy for beamforming purposes, especially during the observation of human tissues, which are known to be spatially coherent, thus echoes from neighbouring scatterers have a similar amplitude [36].

4.6 Additional Beamforming Techniques

In general, digital beamforming algorithms can be distinguished into two categories; non-adaptive and adaptive beamformers. Non-adaptive beamformers are characterised by a data-independent nature. Regardless of the properties of the received ultrasonic waves, the delay propagation applied is only defined by the geometry of the scanned area. Delay and sum is the most common and widespread non-adaptive beamformer, thanks to its low complexity and its real-time compatibility. Nevertheless, being a blind reconstructing method affects the resolution of the output images, revealing high levels of grating sidelobes and being prone to clutter and noise.

An expansion of the DAS algorithm called Delay Multiply and Sum (DMAS) is also commonly used, yielding improved image quality in the expense of calculations complexity. This method can be considered as a way to apply more efficient weight vectors for sidelobe control. Basically, instead of just coherently summing the responses from each transducer element, the delayed signals are first combinatorially coupled and multiplied and the results are then summed coherently [38], [39]. Assuming an amount of N channels, the DAS equation 4.5.1 is transformed as follows :

$$s_{DMAS}(X_s) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} s_j(\tau(X_s, X_j)) * s_i(\tau(X_s, X_i))$$

This procedure can be viewed as an auto-correlation calculation of the received waves, meaning that the output of the beamformer provides the coherence of the detected signals, and each sample is weighted based on other samples [38]. This diminishes the effect of blindness of the DAS algorithmm resulting into improved resolution and contrast by limiting the sidelobe levels [39]. Despite their better efficiency regarding image quality, the constant multiplications combined with the summations make it a challenge to design a real-time DMAS hardware architecture.

On the other hand, in adaptive beamformers the aperture weights are calculated by utilizing the information from the recorded wavefield. In simple words, these methods strive to calculate the most optimal weights according to the received data, removing the blindness that non-adaptive beamformers suffer from. Some of the most well-known adaptive beamforming algorithms are the Minimum Variace (MV) beamforming and the Minimum Mean Squared Error (MMSE) beamforming. Both of these techniques utilize a mathematical metric to optimize the applied weights. MV beamforming aims to choose weights that minimizes the variance of the weighted summations of the delayed signals (equation 4.5.1) [40], [41], while the MMSE beamforming opts to use weights that minimizes the mean-squared error of the output signals [42]. Both of these methods are proved to drastically decrease the sidelobe effect and reduce the artifact intensity in the final image, resulting into an improved image quality. However, the dynamic weight calculation adds a non-negligible computational burden to the ultrasound application making it challenging to design a real-time adaptive beamformer [40], [41]. Additionally, a potential drawback of adaptive beamformers is the lack of robustness against errors in the wavefield parameters, such as errors in acoustic velocity, due to their data-dependent nature [40].

All in all, the focus of this thesis is to provide a real-time, low cost digital beamforming designed on a single FPGA. Since adaptive beamformers are not yet suitable for a real-time, low cost application, we decided that the simple, yet powerful Delay and Sum algorithm is the best candidate for the needs of our beamforming kernel.

4.7 Additional Image Enhancing Processing

To improve ultrasound image quality there are a handful of processing techniques that could be applied to the received signal, before it gets processed or even during beamforming. Additionally, to successfully visualize the output image, the beamformed signal should be further processed into the post-beamforming stages. We will briefly describe the more common image enhancing techniques applied into ultrasound imaging.

4.7.1 Apodization

In general, apodization is called the modification of the shape of a mathematical function. Ultrasound applications use this technique in an attempt to reduce the artifacts originated from side and grating lobes. To achieve that, each channel signal is multiplied by a window function whose main purpose is to suppress the high spatial frequency components of the side lobes [43]. These apodization weights are usually applied during dynamic receive focusing, transforming the beamforming equation 4.5.1 into :

$$s_{bf}(X_s) = \sum_{i \in channel\_set} w_i(X_s) \cdot s_i(\tau(X_s, X_i))$$

It is crucial to detect the optimal apodization weight set for every imaging point. The most prevalent apodization windows are the Hamming and the Hanning functions, mainly smoothening and tapering off the edges of the transducer [43], [44]. This function is applied to the whole transducer or only to a portion of it, discarding element signals outside the apodization window. Figure 4.8 demonstrates the side lobe suppression achieved with a Hanning apodization window, applied from Chi Hyung Seo et al. [45]. This lobe reduction method, despite the fact that it enhances the reconstructed image's quality by improving its contrast, depending on the utilized weights, might result into decreasing the resolution by widening the main lobe.

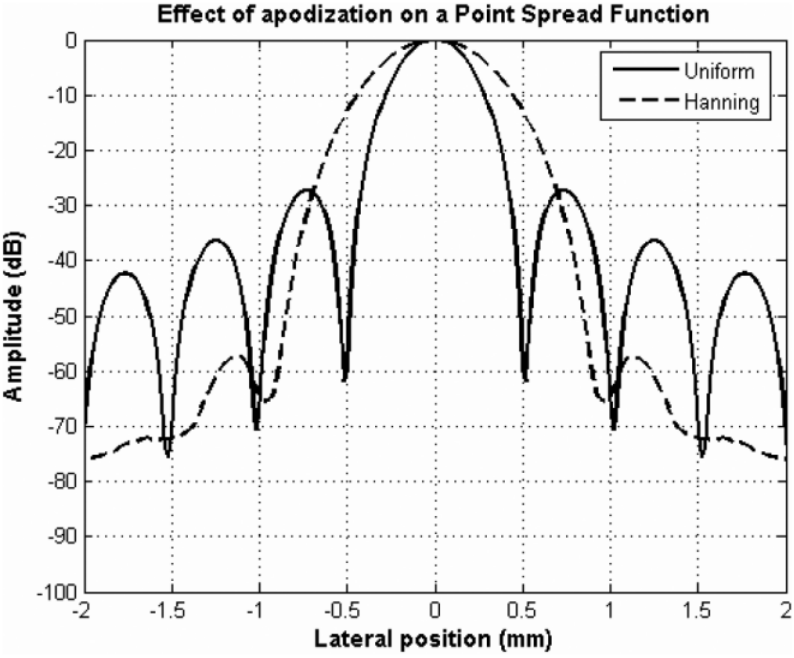


Figure 4.8: Point Spread Function after applying Uniform and Hanning apodization weights [45].

4.7.2 IQ Demodulation

IQ demodulation of a Radio Frequency signal mixes the RF signal with an in phase and a quadrature phase sinusoid function, generating the complex base-band signal [46], [47]. This procedure attenuates signal components from a different frequency and amplifies signal content of that frequency. This is beneficial in order to remove the carrier signal and obtain the signal's envelope, since the magnitude of the I (real part) and Q (Imaginary part) components that are created, contain amplitude and phase information used to acquire B-mode or Doppler images [46]. Moreover, quadrature demodulation reduces the out of band noise in the final image, something that could be considered as a drawback as it is often advantageous for the emitted pulse to span a range of frequencies [47].

The IQ demodulation process is illustrated in the Figure 4.9, provided by Johan Kirkhorn [46].

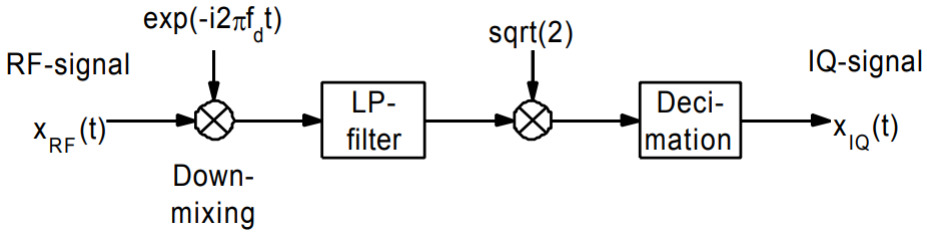


Figure 4.9: IQ demodulation process of an RF signal [46].

It is clear that quadrature demodulation consists of three main steps; down-mixing, low-pass filtering and decimation. Down-mixing mixes the RF signal with the two aforementioned sinusoid signals, giving a complex signal as an output, whose frequency spectrum is no longer symmetric around zero. Furthermore, low-pass filtering is applied to remove the negative frequency spectrum and decimation reduces the sampling rate of the signal in half. To effectively apply beamforming on an IQ demodulated signal, the summing equation 4.5.1 is transformed as follows :

$$IQ_{bf}(X_s) = \sum_{i \in channel\_set} IQ_i(\tau(X_s, X_i)) \cdot e^{(2i\pi f_d \tau(X_s, X_i)}$$

, where f_c equals to the demodulation frequency.

To reconstruct the image from the complex, beamformed signal, either a B-mode image could be obtained from log-compression or a Doppler image from temporal phase shift analysis [46].

4.7.3 F-number

In ultrasound imaging, the lateral resolution is strongly dependent on the wavelength, as well as the channel aperture size. Specifically, the lateral resolution is the smallest with a large aperture and a short wavelength, namely a high frequency. To quantify this, the F-number is defined as the ratio of the imaging depth to the aperture size. This value is a good indication of the directivity of the transducer elements. These elements are not omnidirectional, thus the received signals from all directions are not uniform. For an ultrasound probe with an aperture size of W and a wavelength of λ, if the ratio W/λ is high, then the element has a high directivity; the signals received in front of it have higher strength. Since this ratio is proportional to 1/F-number, then the F-number combined with the element's position on the aperture indicate if the signal of an element should be considered in the beamforming process of a specific image pixel (depending on the depth of that pixel). Vincent Perrot et al. [36] describe in depth the benefit of using the f-number for an improved contrast-to-noise ratio (CNR) and lateral resolution in the final image.

4.7.4 Log Compression and Envelope Detection

After acquiring the beamformed signal of the RF echoes, an envelope detection is applied to enhance the final image. Envelope statistics are tightly related to the microstructures of a tissue and they can describe tissue alterations that are not usually present on an ultrasound B-mode image. To successfully analyze the pathological state of a tissue, envelope detection is a useful process before the construction of the ultrasound image. To realize the extraction of the envelope, a commonly used function is the Hilbert transformation or some approximations of it, for instance the finite impulse response (FIR) or the infinite impulse response (IIR) Hilbert filter method [48].

Along with the aforementioned technique, the beamformed signal could go through a log-compression function. In medical ultrasound imaging, an important aspect of the output signal is the dynamic range. This value determines the ratio between the largest and the smallest signal level, and it can exceed the range of dB that is visible for the human eye (30 dB). To reduce the dynamic range, the acquired envelope could be compressed with a non-linear signal compression technique called log-compression. This could be useful as it allows to display weak signals that contain valuable clinical information [49].

However, both of these processes can be computational heavy for an FPGA device and they are usually conducted either on DSP slices or on software with a CPU or a GPU.

4.8 Description of a standard Ultrasound System

In order to study into the design of a beamforming kernel for the FPGA, it is necessary to analyze each and every component of the ultrasound system individually, as the performance of the kernel and therefore the performance of the whole system is determined by the architecture of the system and the specifications of its components. To begin with, as we have already discussed, the acoustic wavefront is transmitted from the series of transducer elements. Its unique characteristics such as the shape, the energy distribution, the focus of this wavefront and more, are determined by the user and the features he desires to favor at the final image. The delays for the transmit beamformer are stored at a pre-programmed memory and are loaded from the TX beamformer in order to determine the firing patterns for each element. This component creates the pulses and consequently controls the aforementioned parameters of the wavefront. Afterwards, to increase its strength, the output of the beamformer is amplified by a high-voltage pulser (HV pulser) and it is directed via the RX/TX switch to the transducer elements, which produce the acoustic wavefront. The backscattered echoes are picked up by the same transducer elements and through the RX/TX switch they are sent to the analog front-end (AFE) which amplifies and digitizes them. The reason why this switch intervenes between the transducer array and the analog front-end is to control the flow of pulses the AFE receives, because excessive amounts of pulses are able to overload the input ports of the AFE and potentially destroy it. To avoid that, the RX/TX switch is programmed to allow data at correct moments between transmission and receiving. Lastly, the digital back-end that receives the digitized samples from each elements carries out processes for the B-mode image generation that include the receive beamforming and other image processing techniques and the output is either stored or displayed at an operator connected to the back-end computing system. Figure 4.10 displays a schematic of the ultrasound system we described, where each block comprises one of the system's components [20], [21].

It is noteworthy that real-time ultrafast imaging requires computational units that can process the imaging sequences and transfer the immense amount of data necessary faster than the acquisition rate. CPU beamformers are not an option as they cannot provide an adequate speed. Clinical application of high frame rate imaging were firstly made possible with the evolution of GPU devices, as their parallel computational power provides higher beamforming speed. However, there is a memory transfer overhead between the rest of an ultrasound architecture and the GPU, as the GPU device cannot communicate directly with the hardware components of the acquisition system [7], [15]. Thus, Field Programmable Gate Arrays emerged as possible candidates to overcome this bottleneck, as they can directly interface with the Analog Front-End chips, minimizing the memory transfer latency.

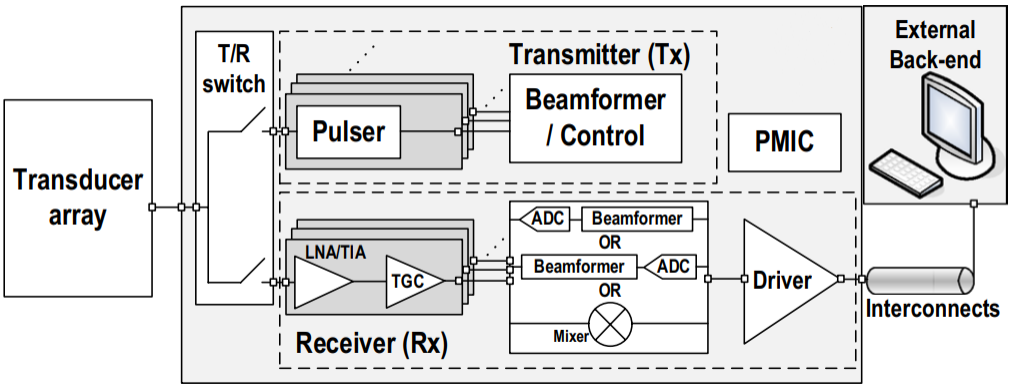


Figure 4.10: Block diagram of a standard ultrasound imaging platform [21].

4.9 Field Programmable Gate Arrays in Ultrasound Imaging

Field Programmable Gate Arrays are commonly used in the medical field for applications regarding diagnostic, monitoring and even therapy. As Xilinx states, "Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing" [50]. Therefore, due to their re-configurable characteristics, an ultrasound system based on a Field Programmable Gate Arrays (FPGAs), gives better flexibility over traditional microprocessors and DSPs in terms of implementation. Additionally, an FPGA provides higher throughput comparing to general purpose processors. As Artificial Intelligence (AI) cores are integrated into the FPGA technology, ultrasound applications with an increasing channel scaling and with very high frame rates are starting to become a reality.

FPGAs emerge as potential candidates for implementing a real-time ultrafast ultrasound imaging platform. Their configurable computational architecture enables immense parallel beamforming processes in real-time. Their configurable logic blocks consist of several flip-flops and look-up tables that implement logic DAS functions, whereas their DSP cores are suitable for executing the complex mathematical expressions necessary for ultrafast DAS beamforming [7]. Meanwhile, their fully programmable memory with the integrated on-chip Block RAMS, supports the growing memory demands of an ultrasound application with high frame rates. Moreover, the high speed interfaces that they provide, like gigabit ethernet and the PCI express, allow the beamforming process to keep up with the high-speed data requirements of the system. The interfaces that are present on an FPGA are able to communicate directly with the analog front-end components of an ultrasound system, minimizing the RF data streaming overhead from the acquisition module to the FPGA [7], [51].

The approach of this thesis is to design an ultrafast ultrasound beamformer on an AMD Alveo U200 Data Center accelerator card, that is capable of supporting high frame rates. We will make an attempt to highlight the benefits of a hardware-based ultrafast beamformer, as well as the constraints that emerged during the implementation of our kernel.

4.9.1 Alveo U200 Data Center Accelerator Card

Providing information straight from the documentation file of the Alveo U200 Accelerator Card, located on the Xilinx's website, we point out that this accelerator card is a custom-built Ultrascale+ FPGA that runs on the Alveo architecture [52]. As Xilinx references, "The Alveo U200 card uses the XCU200 FPGA, which uses AMD stacked silicon interconnect (SSI) technology to deliver breakthrough FPGA capacity, bandwidth, and power efficiency. This technology allows for increased density by combining multiple super logic regions (SLRs)" [52]. This device consists of 3 Super Logic Regions (SLRs) and it is connected into four 16 GB DDR4 memories with a transfer rate of 2400 MT/s and a bandwidth of

77 GB/s, and into 16 lanes of PCI Express with a transfer rate of 8 GT/s [52]. The following Figure 4.12, included in the board's documentation sheet, illustrates the floorplan of the device.

Figure 4.11: Alveo U200 Data Center Accelerator Card. Image provided by the official website of Xilinx [52].

Table 4.1 presents the available resources included on the Alveo U200, per SLR and in total.

Resources	**SLR0**	**SLR1**	**SLR2**	**Total**
Look-Up Tables (LUTs) (K)	365	162	365	892
Flip-Flops (FF) (K)	746	339	746	1831
36 Kb Block RAMS	695	376	695	1766
DSP Slices	2275	1317	2275	5867
288 Kb Ultra RAMs	320	160	320	800

Table 4.1: Available Resources on the Alveo U200 Accelerator Card [52].

Since the FPGA comprises three separate SLRs, while designing the beamforming kernel, we should take extra care when mapping the computation units to the SLRs. If the resources consumed by the kernel exceed the available resources on a single SLR, then the kernel must be mapped into the rest of the SLRs too, inducing SLR crossing while different RTLs on-chip communicate. This SLR crossing will increase the clock period, thus increasing the design's latency, so a preferred practice is to avoid SLR crossing if possible. Moreover, if multiple kernel instances are mapped into different SLRs, then scheduled read/write processes between them will have the same effect.

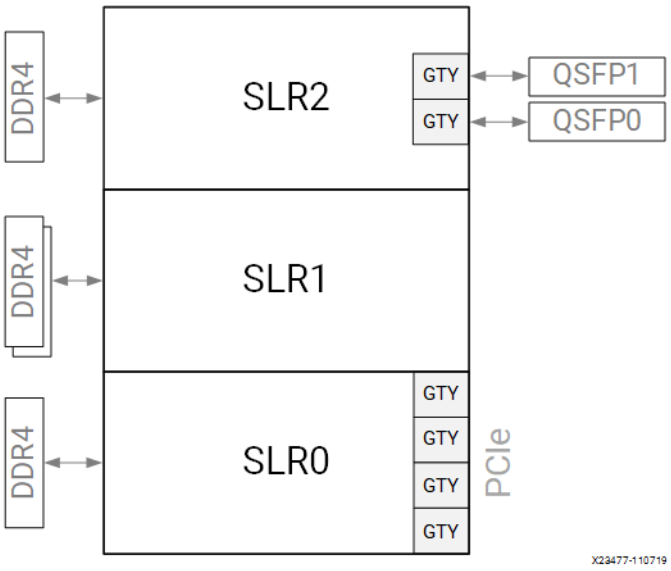


Figure 4.12: Floorplan of the Alveo U200 Accelerator Card, as provided by Xilinx [52].

4.9.2 Advanced Microcontroller Bus Architecture (AMBA)

A System-on-Chip (SoC) like a SoC FPGA consists of a variety of functional blocks. As a result it is important to find a consistent way of connecting and managing them. To achieve that, ARM introduced The Advanced Microcontroller Bus Architecture, or AMBA. AMBA is an open-standard, on-chip interconnect specification for the connection, the management and the communication of functional blocks in system-on-a-chip (SoC) designs. It facilitates the development of multi-processor designs with large numbers of controllers and components with a bus architecture. The transactions conducted on this bus architecture are defined by a burst-based protocol, the AXI protocol [53]. The popularity of AMBA is interpreted by the several benefits it provides, such as :

- Independent read and write channels that can provide low-cost Direct Memory Access (DMA), improving bandwidth during simultaneous read and write processes.
- Multiple outstanding addresses, allowing parallel processing of transactions and increasing performance.
- No strict timing relationship between address and data operations. The memory manager does not have a time restriction to provide the data after issuing a transaction.
- Out-of-order transaction completion.
- Burst-based transactions with only the start address issued. For any following transfers, the next transfer address is calculated based on the burst type.
- Minimization of silicon infrastructure while providing high performance and low power for on-chip communication.

It is evident that the AXI protocol owns multiple characteristics that provide high-bandwidth, low-latency memory transactions, allowing the design of high-frequency systems with high performance communication. This is the primary reason we opted to adopt the AXI protocol for the I/O ports of our design [53].

Furthermore, it should be mentioned that an efficient way to transfer data through the AXI protocol is by bursting. This transfer mode is an optimization that aims to maximize the throughput bandwidth while accessing the DDR memory. Instead of reading/writing a single value from or to the DDR, memory is transferred through data chunks in an attempt to transfer the same amount of data in the

minimum amount of time, improving the throughput of load/store operations. The larger the chunks of data, the higher the throughput. AXI interfaces must have a width that is a power of 2, while the maximum width equals to 512 bits. If a kernel is compiled to operate at a frequency of 300 MHz, then in theory it can achieve a DDR bandwidth of (512 bits $\cdot$ 300 Mhz)/1 sec $\simeq$ 17 GB/s. Bursting typically allows a 4-5 times improvement in throughput and it is generally useful on DDR ports with contention from competing kernels [54].

We will explain how we implemented the bursting method for the memory transfers, through the AXI ports of our beamforming kernel, in Chapter 5.

Chapter 5

Implementation

This chapter details our efforts to develop an accelerated, standalone beamforming kernel for an Alveo U200 FPGA. The kernel will support ultrafast plane wave imaging with coherent compounding for raw RF input samples. The goal is to create a kernel that respects the specifications of an ultrasound acquisition system such as throughput, resource and power consumption as well as input and output streaming, so it can be adapted into such a system with minimum modifications.

For the implementation of our Beamforming kernel we've chosen to work with High Level Synthesis (HLS), being able to synthesize our Beamforming function written in C++ programming language into RTL code, for implementation in the programmable logic (PL) region of our FPGA. Since the majority of our testing was conducted with an Alveo U200 Data Center Accelerator Card, a product of Xilinx, we used the Vitis HLS and Vitis Unified Software Platform tools for synthesis, place, and route of our Beamforming IP core and its integration to our accelerated application.

To successfully design a Beamforming IP core one should have a clear understanding of the limitations the developer needs to face, coming not only from the requirements of an ultrasound application but also from the capabilities of an FPGA device. The most decisive of the limitations that contributed to the design choices we made during the implementation stages of our kernel are the following:

- A kernel implemented with AXI input and output ports can support a data bus width specified as a power of 2, up to 512 bits wide, which means that a kernel can read or write up to 64 bytes per clock cycle per port. Moreover, each AXI port can accept 64 read transactions and 32 write transactions. In burst transfer mode, the upper bound for total Bytes transferred per transaction is 4096 Bytes for read transactions and 2048 Bytes for write transactions.

- The Block RAM (BRAM) inside an FPGA supports up to two I/O ports, either two read ports or one read and one write port. Direct corollary is that during one clock cycle, Block RAMs are limited to support a maximum of two read operations, or one read and write operation, on the condition that those two operations don't refer two the same address. If one desires to execute more parallel read or write processes, he must partition the Block RAM, which means that the resource usage of BRAMs inside the FPGA will increase.

- In general, the aim for our beamforming application is to process each frame faster than it takes to produce and record the next frame. This means that our application should support the frames per second (FPS) of the acquisition system whose output is streaming to our beamforming kernel's input. This is defined by the transmission rate of the ultrasonic probe we're using to produce and record the ultrasound waves and its backscattered echoes. A usual practice is to acquire and process a batch of frames at once, so the processing of a batch should be complete before acquisition of the next is done.

5.1 Optimizations on the Delay And Sum algorithm

The processing time of a single image frame depends heavily on the desired size of the final beamformed image as well as the aperture size, namely the width of transducer elements array used for beamforming. Since we are focusing on reconstructing 2-D images, the image size is actually dependent on two parameters, the number of pixels on the horizontal geometric axis (referred as x-axis) and the number of pixels on the vertical geometric axis (referred as z-axis). Thus, the beamforming calculations are done over three design parameters, for a single frame. As a result, this creates a three-level loop hierarchy which clearly does not allow a linear calculation flow for real-time imaging, as the processing delay surpasses the desired threshold. Consequently, we understand that we need to parallelize our calculations in a certain way, in order to achieve the desired latency per frame for the beamforming process. As explained in Chapter 4, an FPGA could be a potential hardware candidate to accomplish this goal.

However, to move towards this direction we firstly need to establish a consistent way of determining the grid based on which we will reconstruct the beamformed image of the insonified area.

5.1.1 Grid Construction

There are several parameters determined by the earlier components of the ultrasound system that need to be accounted before starting the receive beamforming procedure. Specifically, during the transmission and the reception of the acoustic wavefront the number of transducer elements are set, depending on which elements are enabled to record the backscattered echoes. Also, the probe geometry determines not only the number of transducer elements but also the distance between two sequential elements, defining in this way the whole set of coordinates for the probe channels. Additionally, the analog front-end (AFE) determines the number of samples stored from each transducer element, as well as the sampling frequency. Lastly, an important parameter is the speed of sound inside the insonified area, which is usually considered homogeneous and has a value of $1540 \sim 1550$ m/s [36]. These are the values one should take into consideration to design a beamformer for a specific area. It is important to understand that these values are dependent only on the geometry of the scanned area, the geometry of the probe and the properties of the AFE, which means that the beamforming procedure is completely independent on the backscattered echoes the transducer elements receive. As a result, the beamforming kernel could be reused multiple times for an area with the same characteristics, as long as the attributes of the signal collection and the sampling procedure remain unchanged.

In order to calculate the traveltimes necessary for the Delay and Sum algorithm, the coordinates of the image's pixels are needed. We refer to those 2-D coordinates as grid and they are chosen by the developer, depending on the desired quality of the reconstructed image. A consistent way to create a grid for the reconstructed image is the following. For each dimension, to define the set of coordinates of the pixels we need to determine three values ; the coordinate of the first pixel, the distance between two sequential pixels and the number of pixels on each dimension. When we achieve that, we will be able to determine the size of the final reconstructed image as well as the relative coordinates of every pixel inside the beamformed image. Let it be noted that the amount of pixels on the final image, thus its resolution, are the developer's choice and it is programmed directly during the kernel implementation. The user is not able to influence this parameter.

For the horizontal axis, the one that the ultrasonic probe is placed, we decide to set the number of pixels equal to the number of transducer channels available and their minimum distance equals to the minimum distance between two transducer elements. In this way we set the horizontal image size equal to the size of the channel array, reassuring that each target on the scanned area will be monitored from at least one probe channel. However, this doesn't mean that the horizontal image size will equal to the aperture size, because it is a common practice to omit the information from the leftmost and rightmost transducer elements, as they might contain backscattered echoes from targets outside the desired area, and thus increasing the noise on the reconstructed image. Additionally, since the centre of our ultrasound probe is placed on the beginning of our two dimension grid (coordinates of (0,0)), then the set of transducer elements coordinates is a set of discrete values $[-L/2, L/2]$, where L equals to

our probe's length. We decide to keep this quality for the coordinate set of the pixels on the horizontal axis, so this discrete set is defined as $[-X_{size}/2, X_{size}/2]$, where X_{size} is the horizontal image size. To sum up, the three values needed for the horizontal dimension of the image grid are the following :

- Number of total pixels $\rightarrow X_{size} = elements$, where elements equal to the active elements on the ultrasonic probe.
- Coordinate of the first pixel $\rightarrow x_0 = -X_{size}/2$.
- Distance between two consecutive pixels $\rightarrow d_x = d_{ch}$, where d_{ch} equals to the distance between two consecutive transducer elements, and it is a probe characteristic.

Consequently, the coordinates for a pixel on the horizontal axis is given from the equation $x_s = x_0 + \lambda \cdot d_x$, where $\lambda \in [0, X_{size} - 1]$.

Similarly, for the vertical axis, we decide to set the number of pixels equal to the number of RF samples available at each transducer element. The first pixel is placed at the beginning of the axis (z=0) for simplicity, whereas the image size on this dimension will be calculated from the speed of sound, the total number of samples collected, as well as the sampling frequency. The last possible pixel is placed at the last possible position the acoustic wave arrived, during the sampling procedure. If we refer to the sampling frequency as f_s, to the number of samples as $n_{samples}$ and to the speed of sound as c_s, then the acoustic wavefront traveled a distance equal to $\frac{c_s \cdot (n_{samples}-1)}{f_s}$. This exact value is the coordinate of the last pixel of the vertical axis of our grid. As we pointed out, since the number of pixels on the grid equal to $n_{samples}$ and the first pixel is placed at the beginning of the axis (z=0), then the coordinate of the last pixel equals to $d_z \cdot (n_{samples} - 1)$. From this two equations we can see that $d_z = \frac{c_s}{f_s}$. To sum up, the three values needed for the vertical dimension of the image grid are the following :

- Number of total pixels $\rightarrow Z_{size} = N_{samples}$.
- Coordinate of the first pixel $\rightarrow z_0 = 0$.
- Distance between two consecutive pixels $\rightarrow d_z = \frac{c_s}{f_s}$.

As a result, the coordinates for a pixel on the vertical axis is given from the equation $z_s = \lambda \cdot d_z$, where $\lambda \in [0, Z_{size} - 1]$.

By utilizing the two equations for the two coordinates of each pixel we can successfully construct the geometrical grid for the final beamformed image, a grid that is needed to calculate the transmit and receive distances between pixels and transducer elements. A visualization of the described image grid is given on Figure 5.1.

The same type of equation emerges also for the coordinates of the transducer elements, which can be calculated from the equality $ch_s = ch_0 + \kappa \cdot d_{ch}$, where $\kappa \in [0, n_{channels} - 1]$ and ch_0, d_{ch} are properties of the ultrasound probe.

These three equations for the coordinates of each pixel and each channel will be really useful onwards, as we attempt to reform the delay-and-sum algorithm for efficient calculations. We should point out that these are the equations we used to construct the image grid on our own design. This means that these images sizes are a design choice, in order to have a baseline when testing our kernel and at the same time taking advantage of the geometric symmetry of the reconstructed area, as we will analyze later. The final image could indeed consist of more or less pixels. In both cases, if the new distances between the pixels on both dimensions remain a multiple of the distances d_x, d_z then the analysis remains the same. For the sake of simplifying our analysis, we chose $Z_{size} = N_{samples}$, $X_{size} = N_{channels}$.

5.1.2 Optimizing Delay Calculation

The most significant bottleneck of the beamforming application is the calculation of the traveltimes for every pixel of the final image and every probe element of the aperture used for beamforming. From the equation 4.5.1 discussed on Chapter 4, it is clear that for each unique pair of pixel-element a specific

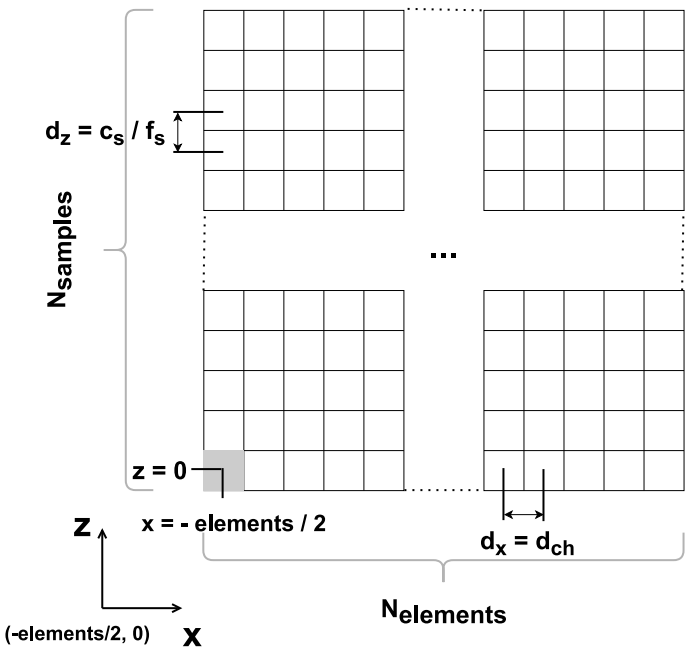
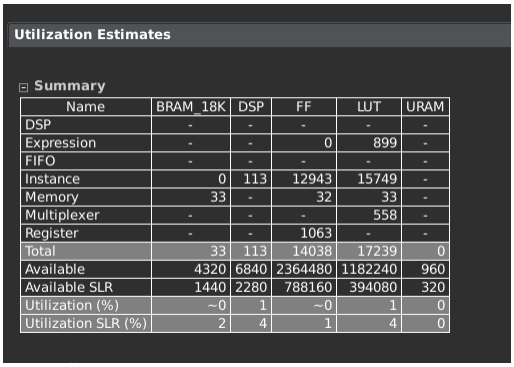


Figure 5.1: Grid that describes the coordinates of the pixels present on the reconstructed image.

delay value should be calculated, which means that for the beamforming of a single frame, an amount of $X_{size} \times Z_{size} \times n_{elements}$ delays need to be calculated. If we assume that each frame of the collected RF data contains 64 channels and we aim at reconstructing a 128×256 image, which is a realistic dataset size for the experiments conducted throughout this thesis, then we need to calculate 2,097,152 delay values. As we discussed, there are two possible ways to deal with this; live calculation on chip or pre-define the delays offline and load them on the FPGA. We will try to elaborate on both options and explain why the first is not feasible for high frame rates support and try to make the second one viable, regarding our application's requirements.

5.1.3 Delay Calculation on-chip

It is safe to assume that calculating a single delay value for a pair of pixel-channel is a computational heavy process, because besides additions and multiplications there is also one square root, one cosine and one sine calculation. Especially for the FPGA these mathematical expressions consume a significant amount of resources and they demand more clock cycles to be completed. On Figure 5.2 we present the HLS synthesis of an RTL design that takes as an input the coordinates of a pixel and the coordinates of a certain tranducer element, and gives as an output the corresponding delay value. The HLS synthesis is conducted assuming an Alveo U200, an accelerator card provided by Xilinx. This certain FPGA will be the baseline of our experiments. By observing the resource usage of that certain function, we are able to see that it consumes around 1% of the LUTs in total. This number is of course acceptable for a standard FPGA, however since we want to calculate the beamformed values of many pixel-elements pairs simultaneously, we need to instantiate this certain kernel many times. For example, if we wish to parallelize the beamforming of the pixels of a certain row of the reconstruction grid, and the beamforming of each of these pixels is conducted as a pipeline over the total transducer elements, we will need at least 128 such kernel instances to calculate delays at the same time, as much as the pixels are on every row of the reconstructrion grid. This means that the total resource usage, only from the delay calculating kernels, will exceed the 100% limit, consuming more than the available resources of the Alveo U200. Consequently, this approach does not seem feasible for an application with low latency and high parallelization requirements.



Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	899	-
FIFO	-	-	-	-	-
Instance	0	113	12943	15749	-
Memory	33	-	32	33	-
Multiplexer	-	-	-	558	-
Register	-	-	1063	-	-
Total	33	113	14038	17239	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	~0	1	~0	1	0
Utilization SLR (%)	2	4	1	4	0

Figure 5.2: HLS synthesis for the delay calculation of a single pixel-element pair.

5.1.4 Loading Pre-Defined Delays

As we discussed on Chapter 4, the delay calculation process is completely independent from the RF values collected from the probe, and is only dependent on the geometry of the insonified area and the probe properties. A standard approach is to calculate offline the desired delays needed for beamforming of a single frame, and loading them on the DDR of the FPGA, before running the beamforming kernel. Taking into consideration the previous example, for a 256×128 image size, with a 64 element aperture size and 32-bit delay values, we need to buffer 67,108,864 bits, which equals to 65,536 Kbits. A standard FPGA usually contains a number of 18Kbit and 36Kbit BRAMs. To buffer these delays, we will need more than 3640 18Kb BRAMs or more than 1820 36KB BRAMs, or a combination of the two type. It is clear that buffering on some FPGAs is not even possible due to resource restrictions, and if it is then this means that the resource usage will be close to 100%, not allowing any other Block RAM utilization, which will be essential for also buffering other input/outputs and other in between variables for calculations. Consequently, we either need to save these values in the DDR and keep reading them when it is needed, which will restrict the design regarding latency due to memory bandwidth and limited port count and port size, or we need to explore a way to reduce the dimensions of our problem. To tackle this memory bottleneck we decided to reform the Delay and Sum algorithm as follows, utilizing the delay compression idea introduced by West et al. [10].

5.1.5 Reformed Delay and Sum algorithm

To revisit the delay calculation equations, the travel time for a specific target of coordinates $X_s = (x_s, z_s)$ and a specific probe element $X_i = (x_i, z_i)$ equals to :

$$\tau(X_s, X_i, \theta) = \frac{d_{TX}(X_s,\theta)+d_{RX}(X_s,X_i)}{c}, \text{ where}$$
$$d_{TX}(X_s, \theta) = z_s \cdot cos(\theta) + x_s \cdot sin(\theta) \text{ and } d_{RX}(X_s, X_i) = \sqrt{(x_s - x_i)^2 + z_s^2}.$$

It is obvious that for a fixed angle, thus for a single frame, the delay is defined from three variables; the coordinates x_s, z_s of the pixel and the coordinate x_i of that specific element. To reduce the dimensions of the calculations we decide to replace the difference $x_s - x_i$ with a single variable Δx. Now, the amount of delays needed to be loaded in the FPGA are equal to $Z_{size} \times n_{\Delta x}$ and not $X_{size} \times Z_{size} \times n_{elements}$ anymore. If we can show that $n_{\Delta x} < X_{size} \times n_{elements}$, by a significant amount, then we reduced the dimensions of the delay values from three to two.

As we recall from the grid construction section, the coordinates of the pixels and the transducer elements, on the x-axis, are given respectively by the following equations :

- $x_s = x_0 + \lambda \cdot d_x$, where $\lambda \in [0, X_{size} - 1]$
- $x_i = ch_0 + \kappa \cdot d_{ch}$, where $\kappa \in [0, n_{channels} - 1]$

Since we showed that $d_x = d_c h = d$, then the different values of Δx are given by the equation :

$$\Delta x = x_s - x_i = x_0 + \lambda \cdot d - ch_0 - \kappa \cdot d \Rightarrow \Delta x = \Delta x_0 + \mu \cdot d, \text{ where } \Delta x_0 = x_0 - ch_0, \mu = \lambda - \kappa.$$

Since $\mu = \lambda - \kappa$ and $\lambda \in [0, X_{size} - 1]$, $\kappa \in [0, n_{channels} - 1]$ then $\mu \in [-(n_{channels} - 1), X_{size} - 1]$. This means that the variable μ produces $(X_{size} + n_{channels} - 1)$ different values for the variable Δx, an amount that is significantly less than $X_{size} \times n_{channels}$. This means that by taking advantage of the geometry characteristics of the probe as well as the scanned area, we managed to reduce the dimensions that the delay values are dependent to, from three to two.

The new transmit and receive distances, including the new variable Δx, are revised as :

- $d_{RX}(X_s, X_i) = \sqrt{\Delta x^2 + z_s^2}$
- $d_{TX}(X_s, X_i, \theta) = z_s \cdot cos(\theta) + (\Delta x + x_i) \cdot sin(\theta)$

However, while reducing the variables of the receive distance from three to two, we added one extra dimension to the transmit distance, meaning that eventually the dimensions that determine the delay values are still three. To overcome this last obstacle we will rewrite the transmit distance into a sum of two parts, one that is dependent on the variables $z_s, \Delta x$ and one that is dependent on the variable x_i. Specifically :

$$d_{TX}(X_s, X_i, \theta) = [z_s \cdot cos(\theta) + \Delta x \cdot sin(\theta)] + x_i \cdot sin(\theta) \Rightarrow d_{TX}(X_s, X_i, \theta) = d'_{TX}(X_s, \theta) + x_i \cdot sin(\theta)$$

Therefore, if all the different values $d'_{TX}(X_s, \theta)$, $x_i \cdot sin(\theta)$ are stored inside the DDR, and they are combined correctly while looping through the different pixels and channels, the desired delays for every pixel-channel pair can effectively be calculated, with just a summation. Assuming an image of size 128×256, an aperture size of 64 channels and a plane wave emission from 10 different angles, then we will need to store $Z_{size} \times (X_{size} + n_{channels} - 1) \times n_{angles} = 488,960$ values for $d'_{TX}(X_s, \theta)$ and $n_{channels} \times n_{angles} = 640$ values for $x_i \cdot sin(\theta)$. Assuming 32-bit variables for the entirety of the design, then we would need to store $32 \cdot (488,960 + 640) = 15,667,200$ bits which is around 2 MB, way less than the DDR's capacity of a standard FPGA.

Since the values $x_i \cdot sin(\theta)$ are only a few compared to the entirety of the values we need to load in the FPGA in advance $(n_{channels} \times n_{angles})$, we could also store them as a ROM inside the FPGA instead of storing them in the DDR. The rest of the values needed to calculate the desired delays will be loaded from the DDR to the internal BRAMs of the FPGA before the start of each frame processing. This means that before the beamforming of each frame starts, an amount of $32 \times Z_{size} \times (X_{size} + n_{channels} - 1)$ bits -assuming 32-bit variables- must be loaded to the FPGA's Block RAMs. To provide an insight of the Block RAMs needed for this application, assuming the aforementioned dataset we will need either 85 18-Kbit BRAMs or 43 36-Kbit BRAMs for the beamforming of a single frame.

Consequently, it is evident that by exploiting the geometry properties of our system we manage to reform the Delay and Sum algorithm in a way that allows us to reduce significantly the amount of delays we need to calculate, in order to beamform a whole image frame. Specifically, we manage to reduce the BRAM usage for delay storage more than 20 times for the dataset size we mentioned above, making it clear that now the Block RAMs present in a standard FPGA are more than enough to support our application. Additionally, there is room for further BRAM utilization for storing input/output values, in between variables while processing and for other usages. We decide to pre-define these delay values, store them in the DDR for every distinct angle and load the necessary delays into the Block RAMs before the launch of the beamforming process of each frame. We load the delays that correspond to the angle of that specific frame and after its beamforming is over, we load the next set of delays and so on and so forth. By doing this, we avoid calculating these values on the FPGA, saving many LUTs and Flip Flops in the expense of some Block RAMs, allowing us to deploy bigger datasets on the platform.

However, we should keep in mind that the Block RAMs inside the FPGA only have two read and write ports, which can restrain the parallelism of our design. We will address this issue later as we discuss the parallelism levels we plan to apply to our calculations, but partitioning the Block RAMs will be the key to overcome this obstacle.

5.2 Delay and Sum kernel

First of all, on Figure 5.3 we present a control flow of the proposed Delay and Sum kernel, to assist the reader during his understanding of our implementation. Every component of this diagram will be explained later on the current Chapter.

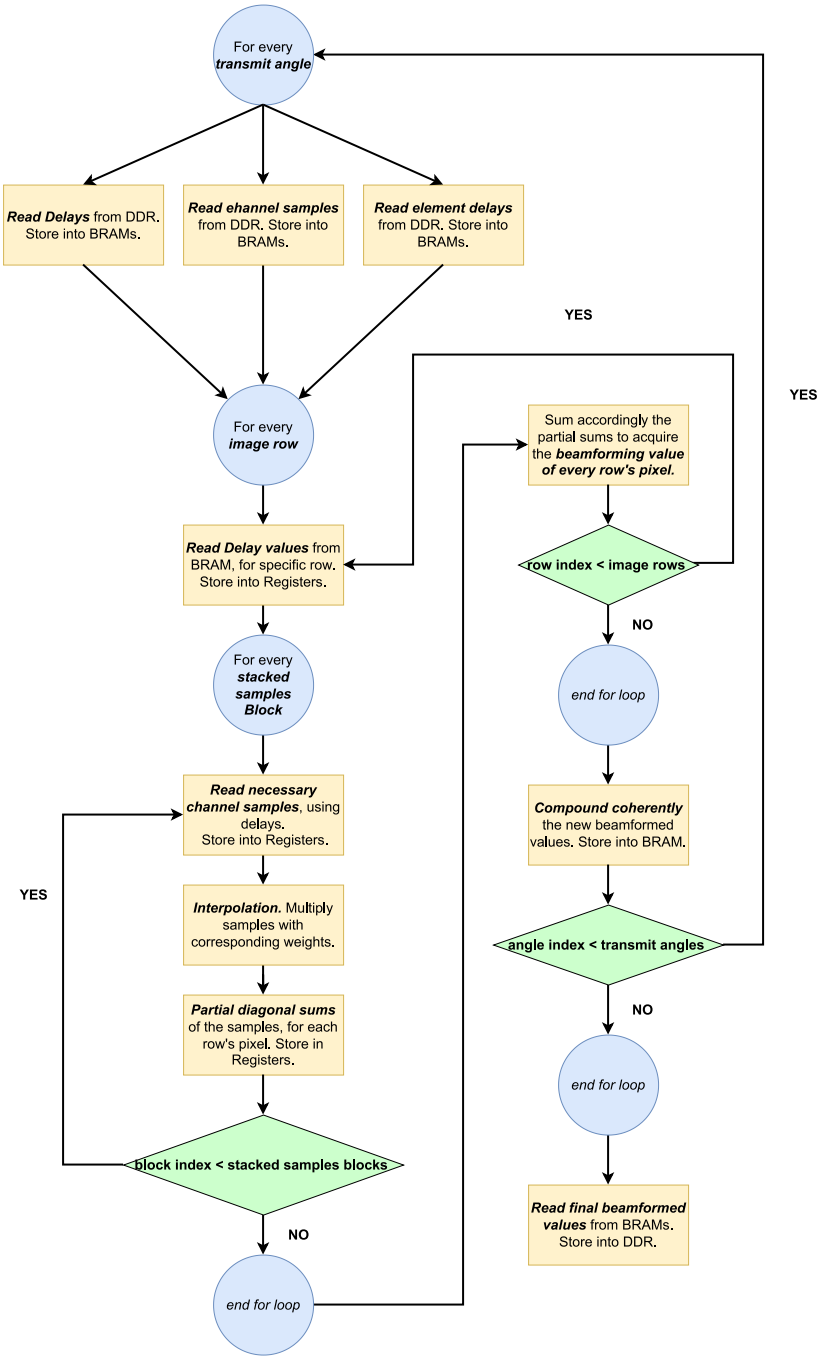


Figure 5.3: Control flow diagram of the proposed kernel.

On section 5.1, we described how it is possible to reduce the parameters of our beamforming calculations from three to two dimensions, saving an important amount of internal FPGA memory. Now we need to describe how to assign the lateral distances Δx to the corresponding pairs of channels - pixels.

To accomplish this, we make the two following observations.

- Assuming that the spacing d between two consecutive pixels equals to the spacing between two adjacent probe elements, we proved on the previous section that two consecutive Δx values also share a distance of d. This means that for a certain depth z, if a pair pixel-element with coordinates (x_n, x_m) corresponds to a relative distance Δx of index i (where i belongs in the set $[0, X_{size} + n_{channels} - 1]$), then the following equations regarding elements, channels and their relative lateral distance are true :
  - $(x_{n+1}, x_{m+1}) \rightarrow \Delta x_i$
  - $(x_n, x_{m+1}) \rightarrow \Delta x_{i+1}$
  - $(x_{n+1}, x_m) \rightarrow \Delta x_{i-1}$

  In short, this means that by increasing laterally both the coordinates of the pixel and the receive element by the minimum possible distance d, then their relative distance Δx remains unchanged. Likewise, by increasing only the element's position by d or only the pixel's position by d, then their lateral distance is increased or reduced accordingly by d. This can be shown even clearer on Figure 5.4.

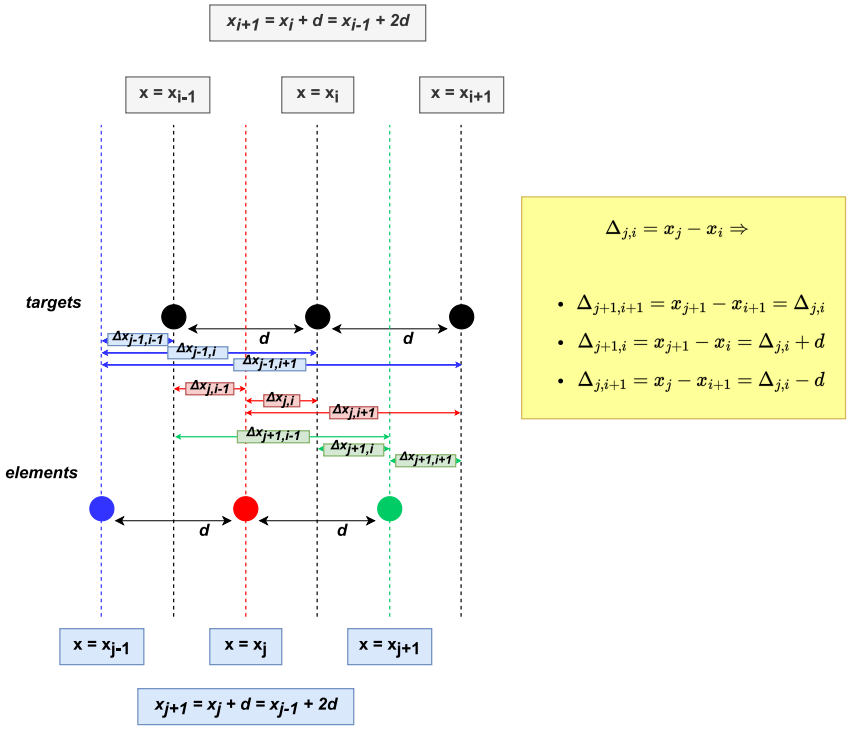


Figure 5.4: Lateral distances Δx and their mathematical relations, for adjacent pixels and adjacent transducer elements.

We will explain now how we exploit these observations to design an efficient HLS function, regarding resources and latency, that performs beamforming for a certain image row. Then we will use the same function for every image depth sequentially to obtain the beamformed image for the specific frame and we will repeat the same process for every possible frame our kernel receives from the ADC.

5.2.1 Beamforming of an image row

The main parallelization scheme for this kernel has to do with the way we sum the samples that correspond to every image pixel, coming from each probe element. To a specific image depth z, for

a specific angle, an amount of $X_{size} + n_{channels} - 1$ different distances Δx correspond. For instance, assuming that our final image will consist of 128 pixels for each row and we use 128 probe elements for beamforming, then for each depth we can acquire 255 distinct Δx values. As we proved on section 5.1.5 each delay value is equal to :

$$\tau(X_s, X_i) = \frac{\sqrt{\Delta x^2 + z_s^2} + z_s \cdot cos(\theta) + (x_i - \Delta x) \cdot sin(\theta)}{c_{sound}}$$

Keeping the depth z as well as the steering angle unchanged, then we see that each delay is calculated depending on two values, the relative distance Δx as well as the receive channel's coordinate. If we disregard the value $\frac{x_i \cdot sin(\theta)}{c_{sound}}$ for now, then by naming the remaining delay portion as $\tau_{\Delta x}(X_s, X_i)$ then :

$$\tau_{\Delta x}(X_s, X_i) = \frac{\sqrt{\Delta x^2 + z_s^2} + z_s \cdot cos(\theta) - \Delta x \cdot sin(\theta)}{c_{sound}}$$

This means that for every possible image depth there are exactly $n_{\Delta x}$ different $\tau_{\Delta x}$ values. In other words, for each depth there are $n_{\Delta x}$ distinct delay indexes provided by the multiple lateral distances between pixels and channels. And for a certain Δx value the corresponding index remains the same, regardless of the pair channel - pixel. Thus, to beamform the pixels on a single depth, at first we create a 2-D array of size $n_{\Delta x} \times n_{channels}$ where the i-th row represents the samples from each receive channel, which we can locate through the delay $\tau_{\Delta x}$ for the i-th Δx value. Given a certain value Δx_i and a certain channel ch_j, then the sample location on the channel's input buffer can be found by acquiring the index results, from adding the partial index calculated from $\tau_{\Delta x}$ with the partial index calculated from $\frac{x_i \cdot sin(\theta)}{c_{sound}}$. A brief piece of pseudocode could help us visualize this simple sample acquisition.

Algorithm 3: Calculate Stacked Samples Array

```
for Δx in (0, Δx_n) do
    index_1 ← Δx delay index
    for channel in (0, n_channels) do
        index_2 ← channel delay index
        index ← index_1 + index_2
        array_samples[Δx][channel] ← buffer_samples[channel][index]
    end for
end for
```

Assuming 64 pixels on the x-axis and 64 receive channels, a visualization of such an array is given on Figure 5.5.

Since the acquired samples emerge from the delays calculated from the relative distances Δx, this means that we have in our hands all the necessary samples to beamform a whole row of image pixels. In short, we have already executed the delay calculation and the channel samples detection of the Delay and Sum algorithm, for a single image depth. Now it is a matter of realizing which samples correspond to which pixels and finding an efficient way of summing them together, in order to apply the summation step of the beamforming algorithm.

5.2.2 Summation step Description of DAS

In order to explain which pixel corresponds to every sample on the stacked samples array we described on the previous subsection, we will use the first remark we made on section 5.2.

Assuming a row of the stacked samples array, then every sample stored on that row correlates to the same relative lateral distance value Δx_i. So, for every probe channel there is exactly one pixel on this specific depth, where their lateral distance equals to Δx_i. And as we explained previously, consecutive channels are related to consecutive pixels on the same depth with the same lateral distance. On that row, if the sample from the x_j channel is used for the beamforming of the pixel x_i, then the sample from the x_{j+1} channel must be used for the beamforming of the x_{i+1} pixel.

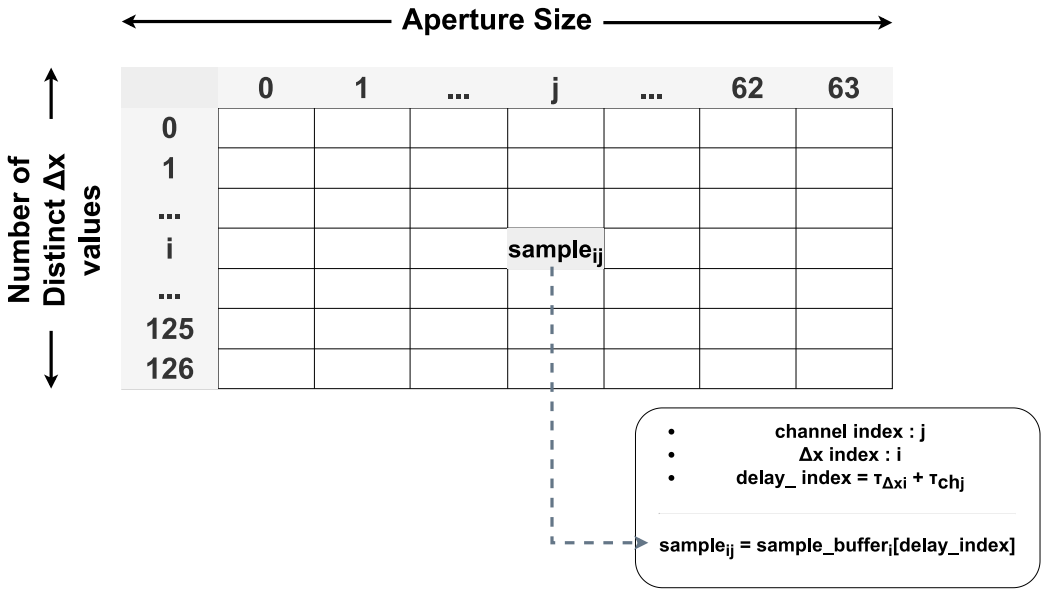


Figure 5.5: Stacked samples array, for a fixed depth and a fixed steering angle. Note that the buffer "sample buffer" represents the internal BRAM where we store the samples of the j-th receive channel.

Additionally, let's assume two consecutive rows of the stacked samples array, with indexes k and k+1. These rows are associated with two consecutive Δx values. According to the third point of the first remark on the section 5.2, sequential receive channels are related to the same pixel, with consecutive lateral distances. Assuming that for the x_i pixel we utilize the sample from the channel xh_j from the row k, then for the same pixel, from the row k+1, the sample from the channel x_{j+1} must be used for its beamforming.

Lastly, assuming a column of the stacked samples array, then all samples present on that column originate from the same receive channel. From the second point of the same remark, we understand that consecutive samples from the same column, namely samples that belong to consecutive rows and thus correlate with sequential Δx values, should be used for the beamforming of consecutive pixels, but with a decreasing step.

Our observations are illustrated in Figure 5.6.

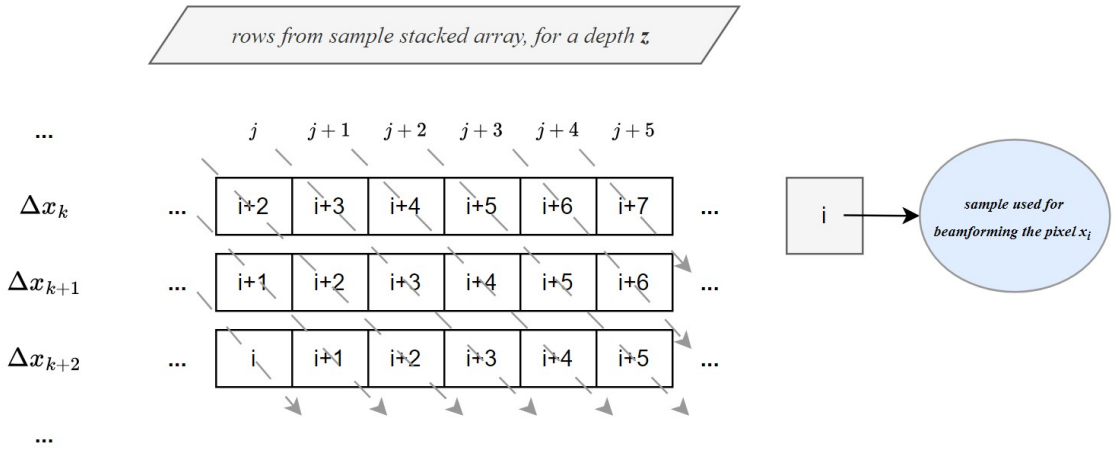


Figure 5.6: Sample distribution from our stacked sample array to the corresponding pixels.

To sum up our comments, it is clear that in order to obtain the beamformed value for each pixel we should add up diagonally the necessary samples, where each summing process starts from the first sample of each row. Since $\Delta x = x_{channel} - x_{pixel}$, then the smallest lateral distance Δx always corresponds to the first channel and the last pixel on that certain depth. This means that the first diagonal sum (whose first sample belongs to the first row of the stacked samples array) is equal with the beamformed value of the last pixel of this depth, and as the position of these sums increases vertically, the index of the pixel whose beamforming value equals with that specific sum, decreases laterally. This whole summing procedure is described from the following equation.

$$bf_i = \sum_{j=0}^{n_{channels}} s(i+j, j).$$

The signal s represents the stacked sample array and the value bf_i equals to the beamformed value for the pixel with coordinates (x_i,z).

Moreover, to visualize even better how the summing step of the algorithm should be implemented, we provide the two following figures. Firstly, in Figure 5.7 we assume a smaller dataset with 8 pixels on the x-axis and 8 channels ($n_{\Delta x} = 15$), for better understanding, whereas in Figure 5.8 we assume 64 pixels on the x-axis and 64 channels ($n_{\Delta x} = 127$), the actual dataset size of our baseline kernel.

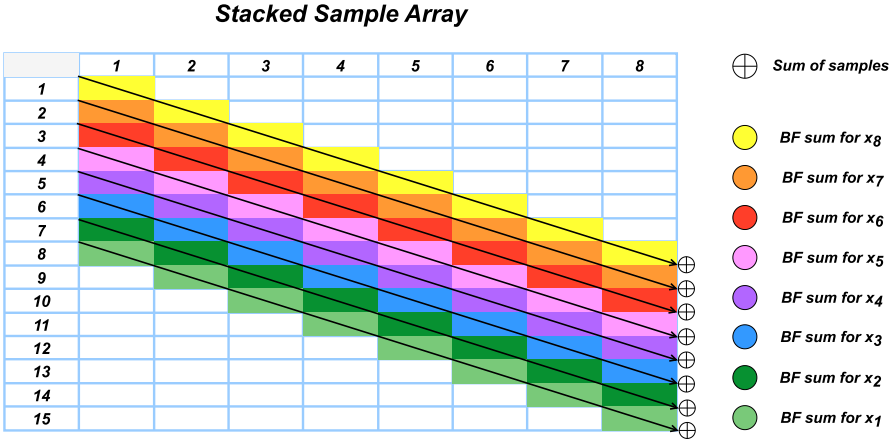


Figure 5.7: Sum beamforming process for a single image line, through the stacked sample array. Coloured tiles represent the samples that are used for beamforming, whereas white tiles represent the unused samples.

From the images, one can understand that there are no reuses of samples during summing, each sample is read only once. This means that for a single depth, all the beamformed values can be calculated at the same time, since there are no dependencies. The goal would be to obtain every result from summing in one clock cycle, after gathering all the necessary wave samples from the BRAM of each channel. If we apply the same procedure for every possible image depth and every angle, then we calculated successfully the beamformed image of the scanned area for ever desired angle.

In the following subsection we will describe in detail how we read and gather the necessary delays and input samples for the beamforming of a single depth, how we implement the summing process on the FPGA and how we store the resulting beamformed values.

5.2.3 Beamforming Design for a single Image Depth

Regarding our kernel design, there are two main directions we should focus on ; memory management and data processing. Regarding memory transfer, data are stored in advance in the DDR. We will explain how to efficiently transfer them from the external memory to the internal Block RAMs, and afterwards how to read and write from/to the internal FPGA memory during the data processing, in an optimized way regarding resource consumption but also fast processing. As long as data processing

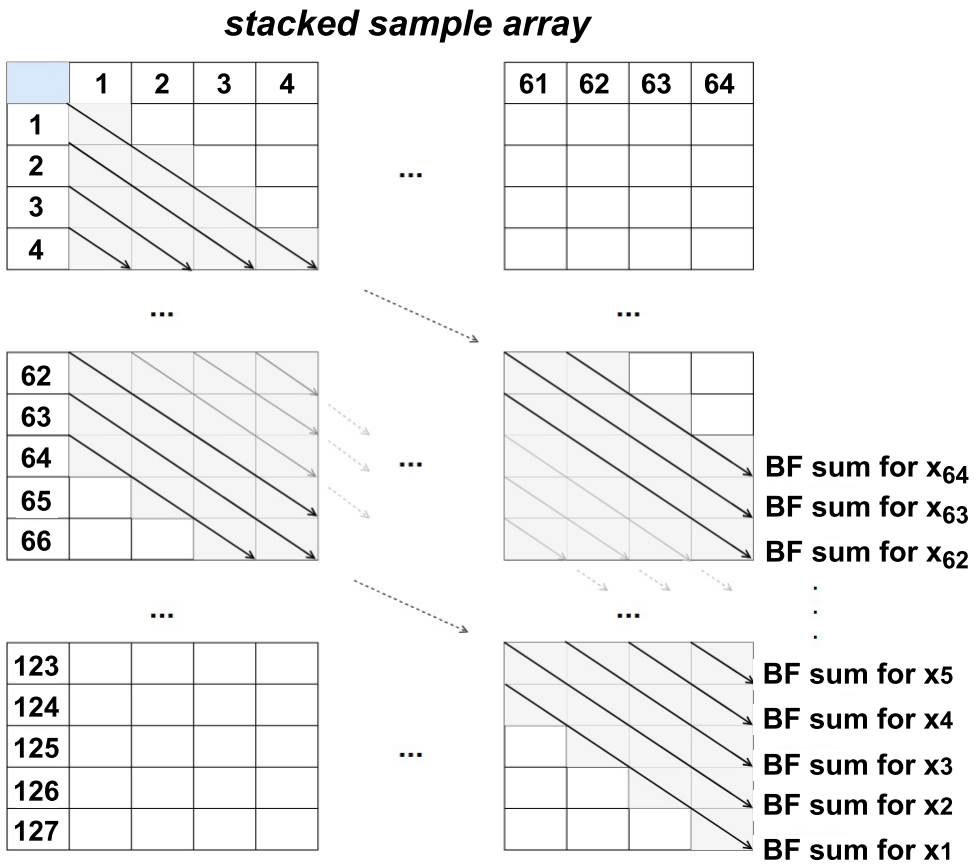


Figure 5.8: Sum beamforming process through the stacked sample array, for the dataset of our baseline kernel. Grey tiles represent the samples that are used for beamforming. Each arrow indicates which samples are summed for the beamforming of the corresponding pixel.

is concerned, we will describe how to implement the summing functions on chip, aiming to minimize the clock cycles needed for the beamforming of a single image row.

Reading from the external DDR memory

The beamforming kernel we designed consists of two input ports and one output port. The two input ports are utilized for loading all the samples received from the ultrasound probe as well as the pre-defined delays for every depth and every angle. The output port is destined to support the transfer of the beamformed image for every frame, from the internal FPGA buffers to the external DDR. Delays have already been calculated offline, for every possible image depth and every possible transmit angle. These are all the values needed for the beamforming of every frame inserted on the kernel. Before the beamforming kernel starts processing input data, we load these delays on the DDR memory and when the kernel is executed, these values will be read and stored inside the Block RAMs of the FPGA.

Since we are designing a standalone kernel and not a kernel that is tested on an actual ultrasound system, we opt to also store the probe samples in advance inside the external DRAM, and after execution is finished, we transfer the beamformed frame in the DDR. There, it can be read from the CPU to be further processed, like applying envelope detection and log-compression, so that the final image is ready to be visualized. Normally, input samples would be streamed from the Analog to Digital Converter component to the beamforming kernel, which would decrease the latency needed to read the input even more, since the delay from the DRAM's bandwidth would be avoided.

The memory transfers are executed over AXI interfaces. As stated in the introduction of this section, according to the AXI protocol, every AXI port must have a width defined as a power of 2, up to 512 bits wide. So indeed, to minimize our transfers from the DDR to internal Block RAMs the port width of every I/O port is set as the maximum possible, 512 bits. If the design supports 16-bit values for data processing, then during a single read/write process 32 values can be transferred at once. This means that every memory transfer to or from the DDR is executed in bursts of 512 bits, so with every clock cycle a maximum of 512 bits per port can be moved towards and from the DDR memory. Burst memory transfer is important in order to make the best use of the available bandwidth.

To make the Vitis HLS tool understand that the kernel ports are defined with a width of 512 bits, we create specific datatypes for every port. The input samples are represented as 16-bit, fixed point values, where only 2 bits define the numbers above the decimal point and 14 bits define the values under the decimal point. Delay values are either represented as short integers (16 bits) for the simple interpolation kernel, or as 32-bit fixed point values for the linear interpolation kernel, where 13 bits are used to define the numbers above the decimal point. Lastly, the output, beamformed samples, as well as the in between values during processing, are implemented as 32-bit, fixed point values, where also 13 bits define the numbers above the decimal points. To achieve these bit accuracies, we create custom fixed point datatypes, by using the "ap_fixed.h" hls library.

In this way we vectorize the read/write processes from/to the DDR, which means that with every clock cycle we read a single value of 512 bits which includes 32 alligned 16-bit values or 16 alligned 32-bit values. We access the DDR memory before the beginning of each frame's beamforming process so we can read the necessary samples of the receive channels, as well as the necessary delay values for the transmit angle. After the beamforming is executed for every possible angle and the values are compounded, the beamformed image is stored into the DDR. To store internally these values we use three buffers :

- input buffer, of size $n_{channels} \times n_{samples}$. The samples from the ultrasound probe for a specific angle are stored here.
- delay buffer, of size $n_z \times [n_{channels} + n_x - 1]$. The pre-defined delays for a specific transmit angle are stored here.
- output buffer, of size $n_z \times n_x$. The compounded beamformed values are stored here, in order to be moved to the DDR memory.

Interpolation

Aiming to compare with the state of the art hardware beamformers, presented on the Related Work Chapter (Chapter 3, we designed two versions of the proposed ultrafast, beamforming kernel. One implements a naive interpolation, where the estimated value equals to the mean value of its two adjacent samples (like the one implemented by Kou et al. [7]), whereas the other one implements a linear interpolation, as described in Chapter 4.

Regarding the naive interpolation kernel, it can be applied while loading the samples from the DDR, rather than during the loop over the stacked samples blocks. Instead of storing the actual samples on the BRAMs before starting the beamforming calculations, one can store the mean value of every two consecutive samples. Therefore, on the control flow diagram 5.3, we can skip the "Interpolation" process. This interpolation method yields satisfactory estimations, as proven on Chapter 6, although worse than the ones produced by a linear interpolation.

On the other hand, linear interpolation should be applied during the loop over the stacked samples blocks, since it utilizes the delays to form the necessary weights. These delays determine how close or far the estimated sample is from its neighbour samples. Thus, the interpolation process progresses as follows. As explained, assuming a 8×64 block, each block carries samples that correspond to 64 transducer elements and 8 Δx delay values. After reading the necessary Δx delays for the specific Block, as well as the element delays $x_i \cdot sin(\theta)$, for that specific angle, we calculate all the delays for every element-Δx pair, by summing up the two delays. In order to apply linear interpolation, for

each one of those delays, the neighbour samples are located by applying a floor function to the delay, whereas the weight equals to how far the estimated sample is from the neighbour samples. Since linear interpolation requires the contribution of two samples, lower and higher, we create two copies of the stacked samples block. Each sample is multiplied by its own weight and then the two stacked samples block are ready to undergo through the diagonal summing functions. After diagonally summing the samples, the contributions from both blocks, lower and higher, are summed too. As demonstrated in Chapter 6, the linear interpolation yields more accurate estimations.

Summing Stacked Samples

First of all, a description of the aforementioned summing processes will be provided, to make data interactions with input and delay buffers more clear. The beamformed value for every pixel in a single frame receives feedback from every transducer element, which means that it is a result of summing together a number of samples equal to the aperture size. For the proposed beamforming design, we will implement a function that beamforms image rows with 64 pixels, using an aperture size of 64 channels. To beamform larger datasets, we are able to create multiple instances of this function, that can run in parallel. Thus, assuming an aperture size of 64 channels as well as 64 pixels for every depth, adding up 64 values for each and every one of the 64 pixels, in a single clock cycle, cannot be supported from the resources of the FPGA. As a consequence, the summing process of a single pixel image should be split into stages.

For every clock cycle, we chose to sum together 8 samples for every pixel. To achieve that we also split the stacked sample array into blocks of 8 rows, and we calculate the partial sum only with the samples that are present (thus 8 samples for every pixel). Starting from the first block of samples, we repeat the adding until every necessary partial sum is obtained. We will refer to the amount of distinct lateral distances Δx in a single depth as D. This means now that instead of one clock cycle, the summing process for a single row demands $D/8$ clock cycles. After the partial sums from every block of samples are acquired, they are summed up accordingly to obtain the final beamformed value for every pixel of this depth.

To implement the summing processes we create three separate functions. It is clear that from every block of the stacked samples array, we only need a handful of samples for the partial sums of the row's pixels. Depending on the block's position, the necessary samples can be found on different array indexes. An example is given in Figure 5.9.

Creating a single function for every block, that has a variable as an input indicating the block's position, would not work, because a "for loop" would be needed to implement the summations. However, the HLS tool cannot schedule this operation in one clock cycle. Creating one separate function for every block is not an option either because that would increase significantly the LUT consumption.

To avoid these obstacles, at first we separate the stacked samples array into two halves, upper and lower half. Regarding the samples we use for summing from every part, the upper array resembles a lower triangular matrix and the lower array an upper triangular matrix 5.10.

This means that if two blocks belong in the same sub-array, then their partial diagonal sums are executed in the same direction. For these two sub-arrays, we locate in advance the block that has the most partial sums useful for the final beamforming of the row's pixels (last block for the upper sub-array and first block for the lower one), creating the two sub-kernels calculating these sums. For every other block we still use the same function for its partial sums, depending on which sub-array they belong. On these blocks there will still be some unnecessary partial sums calculated, but we will not utilize them while summing the partial sums. This allows us to create only two instances of these functions for calculating the diagonal partial sums, that will be re-used during the beamforming execution. Even though we calculate some values that are not needed, this eventually saves a significant amount of Lookup Tables. Figures 5.11 and 5.12 illustrate the two summing functions described above.

Each partial sum, after calculation, will be stored in internal FPGA Flip Flops, so they can all be read in one clock cycle. For example, assuming 64 channels and 64 pixels for each row, then for each block

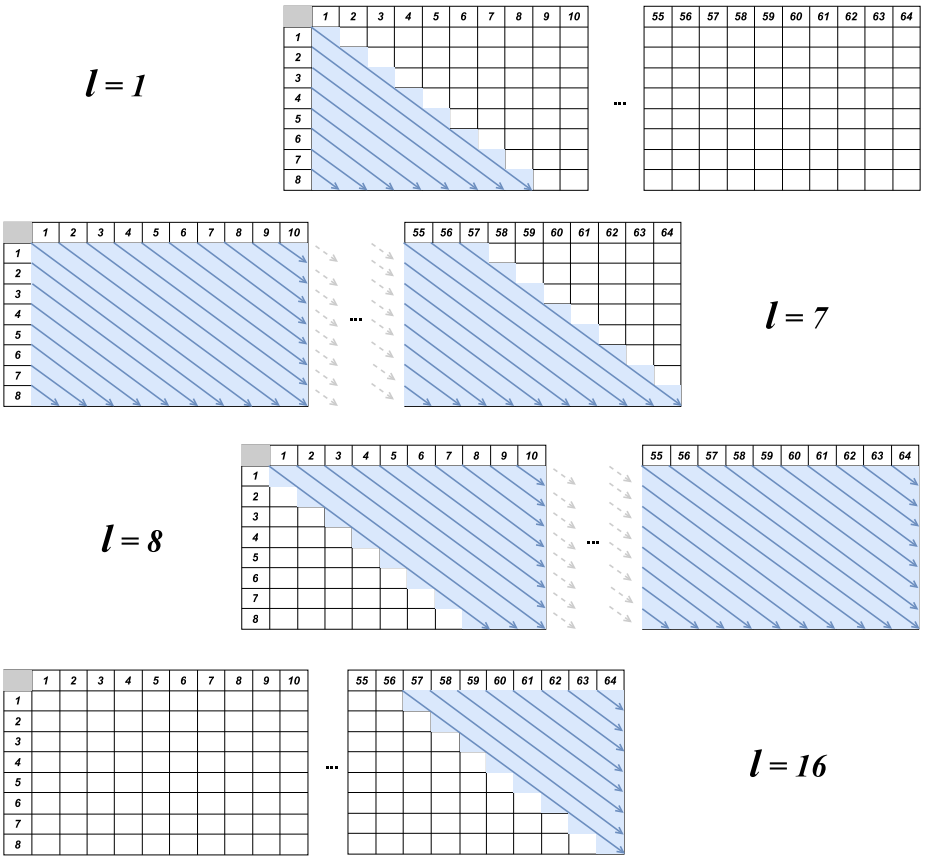


Figure 5.9: Certain blocks from the stacked sample array and which partial sums are utilized from these specific blocks. Variable l indicates the location of the block on the initial array.

of 8 rows, exactly 64 partial sums will be calculated. Since the defined blocks equal to 16, 1024 values will be calculated. Assuming 16-bit values, a beamforming of a whole row demands 16384 registers for the partial sums.

The third and final function for the beamforming of a single row is the one that will combine accordingly the diagonal sums from all the blocks of the stacked array, sum them up to eventually obtain the final beamformed row. Since we explained that not every one of the partial sums is needed for the beamforming of each row, then this function defines which of the partial sums will be needed for the final summations and also execute them. Table 5.1 indicates which partial sums are utilized for the beamforming of every one of the 64 pixels. Final sums will be stored into Flip Flops which means that an additional amount of $64 \times 16 = 1024$ registers are required.

Pixel Index	***Sample Pairs of Indexes***
63	{0,7}, {1,15}, {2,23}, {3,31}, {4,39}, {5,47}, {6,55}, {7,63}, {8,63}
62	{0,6}, {1,14}, {2,22}, {3,30}, {4,38}, {5,46}, {6,54}, {7,62}, {8,62}
61	{0,5}, {1,13}, {2,21}, {3,29}, {4,37}, {5,45}, {6,53}, {7,61}, {8,61}
60	{0,4}, {1,12}, {2,20}, {3,28}, {4,36}, {5,44}, {6,52}, {7,60}, {8,60}
59	{0,3}, {1,11}, {2,19}, {3,27}, {4,35}, {5,43}, {6,51}, {7,59}, {8,59}
58	{0,2}, {1,10}, {2,18}, {3,26}, {4,34}, {5,42}, {6,50}, {7,58}, {8,58}
57	{0,1}, {1,9}, {2,17}, {3,25}, {4,33}, {5,41}, {6,49}, {7,57}, {8,57}
56	{0,0}, {1,8}, {2,16}, {3,24}, {4,32}, {5,40}, {6,48}, {7,56}, {8,56}
55	{1,7}, {2,15}, {3,23}, {4,31}, {5,39}, {6,47}, {7,55}, {8,55}, {9,63}
54	{1,6}, {2,14}, {3,22}, {4,30}, {5,38}, {6,46}, {7,54}, {8,54}, {9,62}

53	{1,5}, {2,13}, {3,21}, {4,29}, {5,37}, {6,45}, {7,53}, {8,53}, {9,61}
52	{1,4}, {2,12}, {3,20}, {4,28}, {5,36}, {6,44}, {7,52}, {8,52}, {9,60}
51	{1,3}, {2,11}, {3,19}, {4,27}, {5,35}, {6,43}, {7,51}, {8,51}, {9,59}
50	{1,2}, {2,10}, {3,18}, {4,26}, {5,34}, {6,42}, {7,50}, {8,50}, {9,58}
49	{1,1}, {2,9}, {3,17}, {4,25}, {5,33}, {6,41}, {7,49}, {8,49}, {9,57}
48	{1,0}, {2,8}, {3,16}, {4,24}, {5,32}, {6,40}, {7,48}, {8,48}, {9,56}
47	{2,7}, {3,15}, {4,23}, {5,31}, {6,39}, {7,47}, {8,47}, {9,55}, {10,63}
46	{2,6}, {3,14}, {4,22}, {5,30}, {6,38}, {7,46}, {8,46}, {9,54}, {10,62}
45	{2,5}, {3,13}, {4,21}, {5,29}, {6,37}, {7,45}, {8,45}, {9,53}, {10,61}
44	{2,4}, {3,12}, {4,20}, {5,28}, {6,36}, {7,44}, {8,44}, {9,52}, {10,60}
43	{2,3}, {3,11}, {4,19}, {5,27}, {6,35}, {7,43}, {8,43}, {9,51}, {10,59}
42	{2,2}, {3,10}, {4,18}, {5,26}, {6,34}, {7,42}, {8,42}, {9,50}, {10,58}
41	{2,1}, {3,9}, {4,17}, {5,25}, {6,33}, {7,41}, {8,41}, {9,49}, {10,57}
40	{2,0}, {3,8}, {4,16}, {5,24}, {6,32}, {7,40}, {8,40}, {9,48}, {10,56}
39	{3,7}, {4,15}, {5,23}, {6,31}, {7,39}, {8,39}, {9,47}, {10,55}, {11,63}
38	{3,6}, {4,14}, {5,22}, {6,30}, {7,38}, {8,38}, {9,46}, {10,54}, {11,62}
37	{3,5}, {4,13}, {5,21}, {6,29}, {7,37}, {8,37}, {9,45}, {10,53}, {11,61}
36	{3,4}, {4,12}, {5,20}, {6,28}, {7,36}, {8,36}, {9,44}, {10,52}, {11,60}
35	{3,3}, {4,11}, {5,19}, {6,27}, {7,35}, {8,35}, {9,43}, {10,51}, {11,59}
34	{3,2}, {4,10}, {5,18}, {6,26}, {7,34}, {8,34}, {9,42}, {10,50}, {11,58}
33	{3,1}, {4,9}, {5,17}, {6,25}, {7,33}, {8,33}, {9,41}, {10,49}, {11,57}
32	{3,0}, {4,8}, {5,16}, {6,24}, {7,32}, {8,32}, {9,40}, {10,48}, {11,56}
31	{4,7}, {5,15}, {6,23}, {7,31}, {8,31}, {9,39}, {10,47}, {11,55}, {12,63}
30	{4,6}, {5,14}, {6,22}, {7,30}, {8,30}, {9,38}, {10,46}, {11,54}, {12,62}
29	{4,5}, {5,13}, {6,21}, {7,29}, {8,29}, {9,37}, {10,45}, {11,53}, {12,61}
28	{4,4}, {5,12}, {6,20}, {7,28}, {8,28}, {9,36}, {10,44}, {11,52}, {12,60}
27	{4,3}, {5,11}, {6,19}, {7,27}, {8,27}, {9,35}, {10,43}, {11,51}, {12,59}
26	{4,2}, {5,10}, {6,18}, {7,26}, {8,26}, {9,34}, {10,42}, {11,50}, {12,58}
25	{4,1}, {5,9}, {6,17}, {7,25}, {8,25}, {9,33}, {10,41}, {11,49}, {12,57}
24	{4,0}, {5,8}, {6,16}, {7,24}, {8,24}, {9,32}, {10,40}, {11,48}, {12,56}
23	{5,7}, {6,15}, {7,23}, {8,23}, {9,31}, {10,39}, {11,47}, {12,55}, {13,63}
22	{5,6}, {6,14}, {7,22}, {8,22}, {9,30}, {10,38}, {11,46}, {12,54}, {13,62}
21	{5,5}, {6,13}, {7,21}, {8,21}, {9,29}, {10,37}, {11,45}, {12,53}, {13,61}
20	{5,4}, {6,12}, {7,20}, {8,20}, {9,28}, {10,36}, {11,44}, {12,52}, {13,60}
19	{5,3}, {6,11}, {7,19}, {8,19}, {9,27}, {10,35}, {11,43}, {12,51}, {13,59}
18	{5,2}, {6,10}, {7,18}, {8,18}, {9,26}, {10,34}, {11,42}, {12,50}, {13,58}
17	{5,1}, {6,9}, {7,17}, {8,17}, {9,25}, {10,33}, {11,41}, {12,49}, {13,57}
16	{5,0}, {6,8}, {7,16}, {8,16}, {9,24}, {10,32}, {11,40}, {12,48}, {13,56}
15	{6,7}, {7,15}, {8,15}, {9,23}, {10,31}, {11,39}, {12,47}, {13,55}, {14,63}
14	{6,6}, {7,14}, {8,14}, {9,22}, {10,30}, {11,38}, {12,46}, {13,54}, {14,62}
13	{6,5}, {7,13}, {8,13}, {9,21}, {10,29}, {11,37}, {12,45}, {13,53}, {14,61}
12	{6,4}, {7,12}, {8,12}, {9,20}, {10,28}, {11,36}, {12,44}, {13,52}, {14,60}
11	{6,3}, {7,11}, {8,11}, {9,19}, {10,27}, {11,35}, {12,43}, {13,51}, {14,59}
10	{6,2}, {7,10}, {8,10}, {9,18}, {10,26}, {11,34}, {12,42}, {13,50}, {14,58}
9	{6,1}, {7,9}, {8,9}, {9,17}, {10,25}, {11,33}, {12,41}, {13,49}, {14,57}
8	{6,0}, {7,8}, {8,8}, {9,16}, {10,24}, {11,32}, {12,40}, {13,48}, {14,56}
7	{7,7}, {8,7}, {9,15}, {10,23}, {11,31}, {12,39}, {13,47}, {14,55}, {15,63}
6	{7,6}, {8,6}, {9,14}, {10,22}, {11,30}, {12,38}, {13,46}, {14,54}, {15,62}
5	{7,5}, {8,5}, {9,13}, {10,21}, {11,29}, {12,37}, {13,45}, {14,53}, {15,61}
4	{7,4}, {8,4}, {9,12}, {10,20}, {11,28}, {12,36}, {13,44}, {14,52}, {15,60}
3	{7,3}, {8,3}, {9,11}, {10,19}, {11,27}, {12,35}, {13,43}, {14,51}, {15,59}

2	{7,2}, {8,2}, {9,10}, {10,18}, {11,26}, {12,34}, {13,42}, {14,50}, {15,58}
1	{7,1}, {8,1}, {9,9}, {10,17}, {11,25}, {12,33}, {13,41}, {14,49}, {15,57}
0	{7,0}, {8,0}, {9,8}, {10,16}, {11,24}, {12,32}, {13,40}, {14,48}, {15,56}

Table 5.1: Table that indicates how partial sums are combined to calculate the final beamformed value of every pixel. A sample with indexes {i,j} originates from the i-th block and the j-th channel.

Let it be noted that since partial and final sums are stored into registers, then every one of these 3 sub-kernels can be executed in one clock cycle, as long as all the input samples used for summing can also be read in one clock cycle. Moreover, the size of the aforementioned sample blocks reveals a trade-off between resource consumption and design latency. Larger blocks translate to a faster design, however they require more resources, namely Flip Flops and Lookup tables, to execute the summing processes.

Reading data from internal Block RAMs

As mentioned in the previous subsection, the beamforming of a single image row requires all the delay values that correspond to that specific depth, and are stored inside the delay buffer. The goal is to read every necessary delay value in one clock cycle, and save them into Flip Flops, in order to be able to access them separately later. However, the internal BRAMs provide only two I/O ports which means that normally only two of these delays could be read in a single clock cycle. To overcome this, we partition the delay buffer in order to read the necessary delay index values simultaneously. From the summing procedure, since we execute the data processing in groups of 8 delay rows, it is clear that 8 delay values must be read in one cycle. To achieve that, we apply a cyclic partition with a factor of 8, on the second dimension of the delay array with size $n_z \times n_{\Delta x}$. Depending on which block is processed, the necessary delay index values are read in one clock cycle, they are stored in registers so they can be accessed simultaneously and they will be used to locate the necessary samples for row beamforming, from the channel buffer.

The only step left to start the summing procedure, is to retrieve the necessary sample for each stacked samples block and store them into registers. In order to acquire all the samples for a single delay index in one clock cycle, we fully partition the channel buffer on the channel dimension. This means that samples from different channels can be accessed at the same time. However, since 8 rows of samples are needed for the calculation of every group of partial sums, then optimally 8 samples from each probe channel should be accessed in one clock cycle, but each channel buffer includes only two I/O ports. This means that either we need to create 4 different copies of the channel buffers in order to make it possible to access 8 samples from one channel, in one clock cycle. This option consumes extra Block RAMs but keeps the latency as low as possible. We will analyze later both of the options. After all the necessary samples are acquired and stored into Flip Flops, the summing procedure for the specific block is ready to begin.

Storing Results into Internal Buffers and the DDR

After beamforming a whole image row for every transmit angle, the next step is to compound them to acquire the final beamformed row. Compounding is a simple process, the beamformed values of a certain pixel for each angle are summed together. So, after we beamform a single row for a certain angle and store the results temporarily into registers, we compound them with the values we have already calculated for previous angles in the output buffer we mentioned above, a buffer whose values are initialized to 0 before the beamforming process starts. In order to finish the compounding for a single depth in one clock cycle, we partition the output array in a way that pixels with the same x coordinate are stored in the same BRAM.

After the beamforming is over, the output buffer holds the compounded beamformed image for every single transmit angle. The final step is to store these values in the external DDR. To achieve an efficient

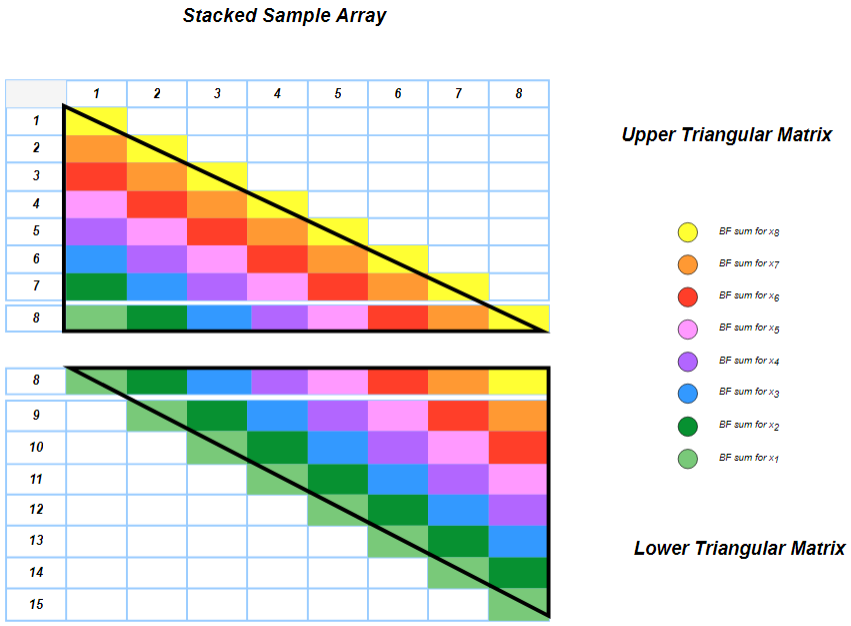


Figure 5.10: Splitting the stacked sample array into two halves. First half represents a lower triangular matrix and the second half represents a higher triangular matrix. The stacked sample array is the same with the array located on Figure 5.7

communication with the DDR, again we transfer the output values in batches of 512 bits, namely 32 values of 16-bits.

Overall Beamforming Kernel Design

To summarize, for a single batch of frames including the transmissions for each transmit angle, the beamforming kernel is designed as follows.

We should note that the functions sum_stacked_samples_1() and sum_stacked_samples_2() correspond accordingly to the two summing sub-kernels we described above and the function final_sum_registers() corresponds to the third summing sub-kernel we described, the one that combines all the partial sums to output the beamformed row. Figure 5.3 presented earlier, illustrates the aforementioned pseudocode, describing the kernel's design and the dataflow of our application.

Moreover, since we described a way to read from the channel and delay buffers in one clock cycle, and since the functions sum_stacked_samples() write into different registers for every iteration of the innermost loop, then it is safe to say that there are no dependencies between variables inside that specific loop. Pipelining the innermost loop with an *Initiation Interval = 1* is feasible, ensuring that the latency is minimized, taking into consideration the available FPGA resources. This means that the innermost loop will have a latency of $(D/8 + single\_loop\_latency - 1)$. However to achieve this, as mentioned above, we should create four different copies of the channel buffer, which would increase the BRAM utilization. If for a certain dataset size the available BRAMs cannot support the application, then we could also use the available URAMs or LUTRAMs inside the FPGA, to reduce the BRAM usage. On the other hand, if we opt to not create copies of the channel buffer then the innermost loop cannot be pipelined due to read dependencies. Then, we can only pipeline the reading process from the channel buffer, which will eventually require 4 clock cycles to finish. This would increase the latency of our beamforming kernel but would reduce the BRAM usage. We will analyze more about the kernel's latency on Chapter 6.

To sum up our analysis, we described a reformed Delay and Sum beamforming algorithm that achieves two specific improvements:

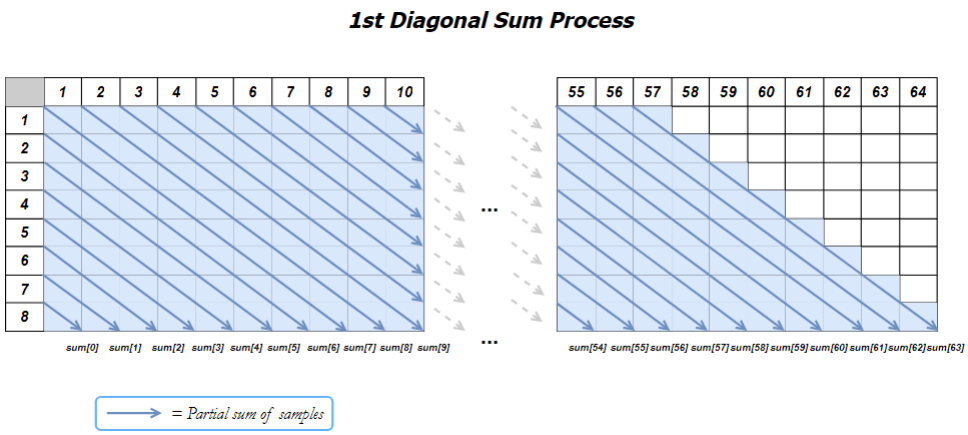


Figure 5.11: First summing function, for the blocks that belong to the upper sub-array of the stacked sample array.

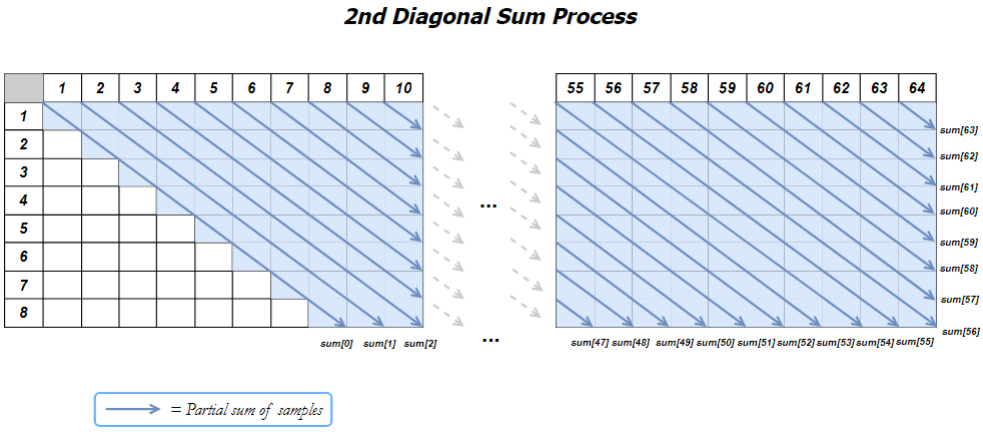


Figure 5.12: Second summing function, for the blocks that belong to the lower sub-array of the stacked sample array.

- Reduce the Delay profile of the beamforming process.
- Reduce the total iterations during beamforming summing processes.

The effects of these improvements are summarized on Table 5.2 .

5.3 Vitis and Vitis HLS kernel Development

In this section, we will briefly describe the beamforming kernel development using the Vitis and the Vitis HLS tools, provided by Xilinx. First of all, the C++ kernel code is generated using the Vitis High Level Synthesis tool to synthesize a kernel with an acceptable latency, that also does not exceed the available resources. Afterwards, the Vitis tool is used to develop the beamforming application. A host program is written, by utilizing the OpenCL API calls, in order to setup the environment for kernel execution as well as memory management between the system's CPU and the FPGA. After our application passes successfully the Software and Hardware Emulations, we compile the beamforming kernel, synthesize it and implement a bitstream from the HLS kernel, in order to execute it on the board. The host program is responsible for the kernel execution and the output data collection. Let it

Algorithm 4: Beamforming Kernel Flow

```
for n in (0, angles) do
  channel_buffer ← read_input_ddr(n)
  delay_buffer ← read_delay_ddr(n)
  for z in (0, image_rows) do
    for l in (0, D/8) do
      delay_registers ← read_delays_from_buffer(delay_buffer, z, l)
      sample_registers ← read_samples_from_buffer(channel_buffer,delay_registers)
      weighted_sample_registers ← interpolation(sample_registers, delay_registers)
      if l < (D/8)/2 then
        partial_sum_registers ← sum_stacked_samples_1(weighted_sample_registers, l)
      else
        partial_sum_registers ← sum_stacked_samples_2(weighted_sample_registers, l)
      end if
    end for
    final_sum_registers ← sum_partial_sums(partial_sum_registers)
    compound_bf_values(output_buffer,final_sum_registers, z)
  end for
end for
write_output_ddr(output_buffer)
```

be noted that every synthesis and every bitstream we create are destined for the Alveo U200, a Xilinx FPGA.

5.3.1 Programming with Vitis HLS

After writing the C++ kernel code, the first step is to run a C synthesis with the Vitis High Level Synthesis tool. However, the plain C++ code will not be efficient running into an FPGA. Hence, it is important to guide the compiler with HLS directives in order to achieve the desirable scheduling and memory management for our kernel. Below we describe how we control the kernel scheduling as well as the memory management of the design.

Loop Management

As previously mentioned, the main data processing consist of two loops. The outtermost one loops over the images rows, and should be executed linearly, without being pipelined. The innermost one, referred as block loop, loops over the several stacked samples array blocks and should be pipelined. Every other loop inside the block loop should be unrolled in order for its processes to be conducted in one clock cycle. Moreover, the loop processes regarding reading and writing on the DDR memory should also be pipelined.

Internal Memory Management

Concurrently, it is important to partition certain buffers in order to achieve multiple read/write processes in one clock cycle. Also, by fully partitioning a one-dimensional array, we guide the HLS tool to resolve this memory into individual register. This is useful because as noted previously, our design benefits from the FPGA's Flip Flops. Moreover, the Ultra RAMs should be utilized in order to reduce the consumption of the Block RAMs/ Also, the I/O processes regarding those memories should be pipelined into two clock cycles, by using registers on the URAM's output, to achieve the desirable clock frequency. I/O processes regarding the URAM are in general slower compared to the BRAM.

Optimization	*Bottleneck*	*Optimization Improvement*
Revising the delay calculations to reduce the size of the delay profile that needs to be loaded on the FPGA.	$Z_{size} \times X_{size} \times n_{elements}$ **delays** need to be stored into the FPGA's BRAMs. The array partition to support the parallelization scheme we aim to apply, exceeds the available resources.	Now $Z_{size} \times (X_{size} + n_{elements} - 1)$ **delays** need to be loaded in the FPGA. Array partition that will support highly parallelized summing processes is possible.
Reforming the DAS algorithm to reduce iterations during sum processes	**Three-level loop hierarchy** that disallows a linear calculation flow for ultrafast imaging. Adequate parallelization is not possible due to resources restriction.	**Loop hierarchy** through summations is reduced to **two levels** (over variables z and Δx). We can apply now the necessary pipeline.

Table 5.2: Optimizations to tackle important bottlenecks of the DAS algorithm that restrict performance for high frame rates.

Mapping Kernel I/O Ports

Input and output ports of the beamforming kernel should be implemented with the AXI4 protocol, in order to achieve efficient burst reads and writes with the DDR memory.

As a baseline, we decided to design a kernel that supports beamforming with an aperture size of 64 channels and an image row size of 64 pixels. In order to increase the aperture size or the image row size one should create many identical beamforming computational units on board. For instance, since the dataset that we used for validation supports 128 channels and 128 pixels on the x-axis, we created four identical instances of our beamforming kernel, in order to completely beamform the input frames.

5.3.2 Deploying the Beamforming Application with Vitis

After writing the kernel C++ code, synthesizing it with Vitis HLS and achieving the desirable estimations regarding latency and resource consumption, the next step is to develop the accelerated application using the Vitis tool, and generating the bitstream.

During the v++ linking process it is important to define how many instances of our kernel should be created and specify to which Super Logic Region (SLR) they should be mapped. We decided to create four instances of the beamforming kernel. Every beamforming kernel consists of four memory ports: inp, delays, channel_delays, outp. Two compute units are mapped to SLR0 and the other two at SLR2, in order to avoid a compute unit to be mapped to multiple SLRs since this will cause SLR crossing, which will unavoidably increase the clock frequency. Furthermore, the memory ports of the compute units mapped at SLR0 are mapped to the DDR[0] and the memory ports of the compute units mapped at SLR2 are mapped to the DDR[3]. On Figure 5.13 the architectural diagram of our FPGA beamforming design is illustrated.

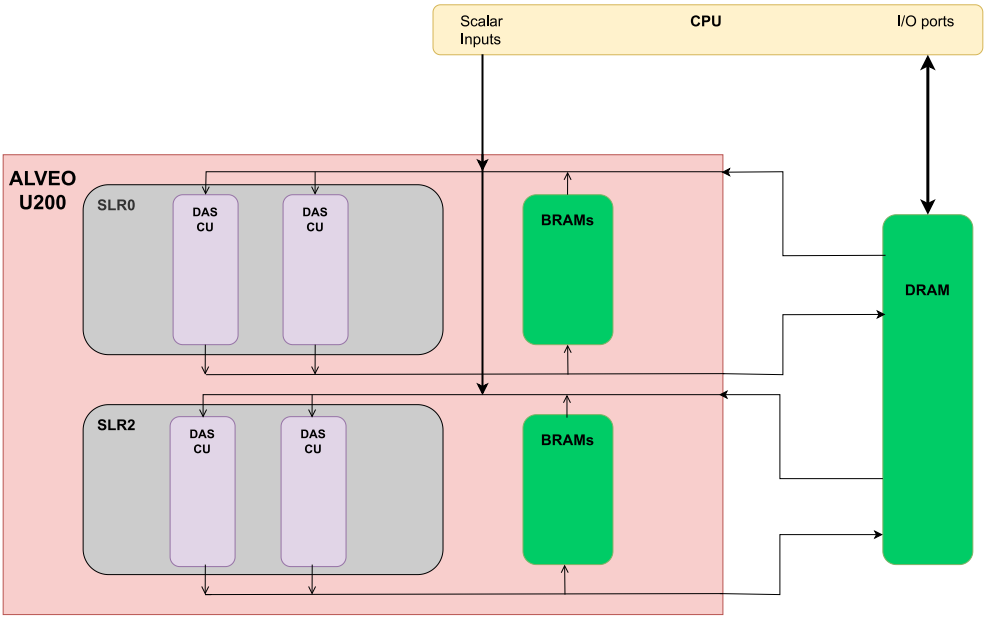


Figure 5.13: Architectural diagram of the proposed kernel.

Chapter 6

Results & Evaluation

This chapter presents the validation and evaluation results of our beamforming application, after testing it with ultrasound datasets. Specifically, the kernel will be evaluated regarding performance, throughput, resources and energy consumption, as well as final image resolution and it will be compared to other state of the art FPGA beamformers, but also to a naive DAS beamformer deployed on a GPU, provided by the Center for Ultrasound and Brain imaging (CUBE) laboratory from the Erasmus University of Rotterdam.

6.1 Kernel Validation on RF Ultrasound Datasets

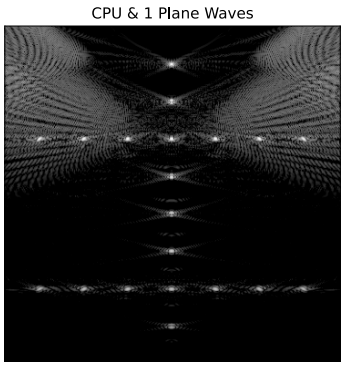
Our beamforming design was tested with the following dataset provided by the website of PICMUS, the IEEE IUS 2016 Plane-wave Imaging Challenge in Medical UltraSound [11]. This dataset contains samples collected by emitting several plane waves into the insonified area, with the help of the Verasonics Vantage 256 Ultrasound System. Its characteristics can be found on the following Table 6.1.

Aperture Size	128 channels
Sampling Frequency (Hz)	2308000
Collected samples per channel	1527
Plane Waves	75

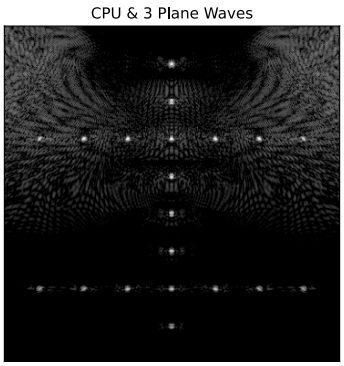
Table 6.1: Table describing the characteristics of the dataset used for validation and evaluation of our beamforming kernel.

This means that the final image will consist of 128 pixels on the horizontal dimension. Regularly, from the grid construction description in section 3.1.1, the final image will include 1527 rows, as many as the samples collected from each channel. However, the dataset had already constructed the final image grid with 609 rows, so we kept it that way. Thus, the output image from the kernel will be a 128×609 image produced from RF input samples with an aperture size of 128 transducer elements. Moreover, the kernel will receive 1527×128 channel samples and 609×255 pre-calculated delay values as inputs stored in the DDR, and will produce 128×609 compounded, beamformed values.

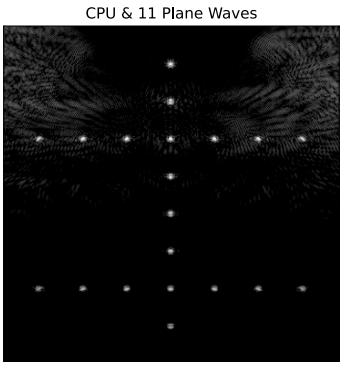
First of all, on Figure 6.1 we present the beamformed images we produced with a CPU, for several number of angles (1, 3, 11, 38, 75). These images will be utilized to evaluate the images our kernel produces regarding correctness and image quality. The main difference of the CPU beamformed images compared to the ones produced by our FPGA kernels is that the CPU uses float values for calculations, whereas the proposed kernels implement fixed point datatypes. Therefore, the CPU output is expected to achieve equal or better accuracy.



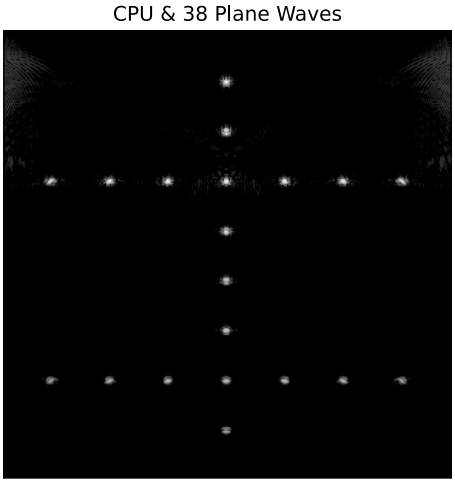
(a) 1 Plane Wave



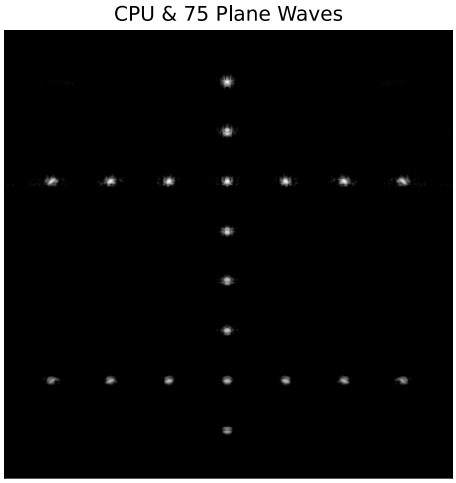
(b) 3 Plane Waves



(c) 11 Plane Waves



(d) 38 Plane Wave



(e) 75 Plane Waves

Figure 6.1: Beamformed images with 1,3,11,38 and 75 plane waves, on a CPU

As mentioned on Chapter 5, we implemented two versions of the beamforming kernel. One that applies a simple interpolation on the input samples, where every necessary sample equals with the mean value of its two adjacent samples, and one that applies linear interpolation. We expect the second version to output improved images, since the applied interpolation yields better estimation precision, but with the cost of higher hardware complexity.

We will evaluate the designs of the two versions of our Delay and Sum beamforming kernel, which both use four beamforming compute units. Each compute unit beamforms 609 row images with a size of 64 pixels and an aperture size of 64 channels.

6.1.1 DAS Beamforming Kernel with a naive Interpolation

To beamform the desired dataset we created four instances of our kernel that applies Delay And Sum with an aperture size of 64, on 609 image rows with 64 pixels each. Figure 6.2 presents how the four compute units and their memory ports are mapped on the DDR of the Alveo U200. This image is generated with the Vitis Analyzer tool.

Two "bf_kernel" instances are mapped into SLR0 and the other two "bf_kernel" instances are mapped into SLR2. The first two beamform the samples for the first 64 channels and the other two for the rest 64 channels. The result is a beamformed image of size 128×609, by using an aperture size equal to 128 channels. At first, we execute the beamforming with 75 plane waves.

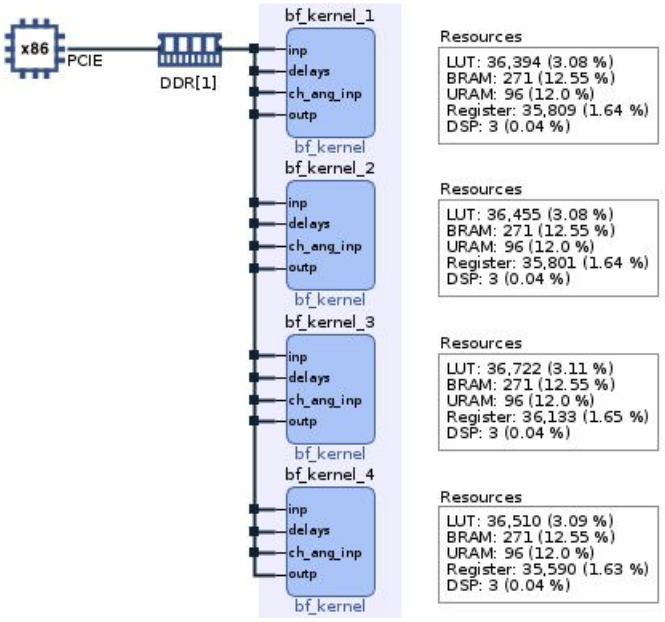


Figure 6.2: System diagram of the Alveo U200, after deploying the four beamforming compute units.

Table 6.2 sums up entirely the performance of our kernel for 75 plane waves, whereas Table 6.3 presents the resource consumption of our DAS kernel, not only on the entire Alveo U200, but also the consumption per SLR. The information from these two tables was acquired by profiling our application with the Vitis and Vitis Analyzer tools.

Clock Frequency (MHz)	300
Clock Cycles	1,996,397
System Runtime (ms)	7.135
Kernel Runtime (ms)	6.747
Frame Rate (Hz)	148.214
Pulse Repetition Frequency (Hz)	11,116.052
Beamforming Rate (MSamples/s)	2173
Power Consumption (Watt)	10.800

Table 6.2: Performance of our DAS Beamforming kernel, with the *naive* interpolation, for 75 plane waves

From Table 6.2, the values *Clock Frequency, Clock Cycles, System Runtime, Kernel Runtime, Power Consumption* are obtained by profiling the application, whereas the rest of the information are calculated from the aforementioned. *Frame Rate* = 1/Kernel Runtime, *Pulse Repetition Frequency* = Frame Rate$\times$75 angles and *Beamforming Rate* = Aperture Size$\times$Samples per channel$\times$Frame Rate$\times$ angles.

One can observe that the proposed DAS beamforming kernel achieves a beamforming rate in the GSamples/s range, or equally a Frame Rate in the KHz range, for a single plane wave. This fulfills the criteria of a hardware-based beamforming kernel for an Ultrafast Ultrasound application. Moreover, it is clear that the resources with the highest usage are the BRAMs and the URAMs. This fact indicates that our design is memory constrained, something that needs to be confirmed by tuning the parameters of the kernel, like the aperture size, the beamformed rows, as well as the size of the stacked samples blocks, and examining the resources' consumption.

Figure 6.3 provides the output images of our kernel for 1, 3, 11, 38 and 75 plane wave emissions.

Resource	*1 CU/Full device*	*4 CUs/Full Device (%)*	*1 CU/Per SLR (%)*	*2 CUs/Per SLR (%)*
LUT	36185 (3.06%)	12.24%	11.14%	22.28%
BRAM	271 (12.55%)	50.20%	42.48%	84.96%
URAM	96 (12%)	36%	30 %	60%
Flip Flops	38289 (1.75%)	7%	5.29%	10.58%
DSP	3 (0.04%)	0.12%	0.13%	0.26%

Table 6.3: Resource consumption of our DAS Beamforming kernel, with the *naive* interpolation, for 75 plane waves. Our design consists of 4 Compute Units, where 2 Compute Units are mapped to the same SLR.

Output images are similar with the ones produced from the CPU, establishing the correctness of the proposed design. However, some reduction on the image's quality is visible. This happens due to the naive interpolation applied on the probe's signal samples, as it yields a lower estimation precision on the estimated values. To increase the quality of the output image we proposed the addition of the linear interpolation procedure in our design.

6.1.2 DAS Beamforming Kernel with a Linear Interpolation

As previously mentioned on Chapter 5, this version of the Delay and Sum beamforming kernel follows the same flow as the initial design, but with the additional step of multiplying the input samples collected from the channel buffers, with their corresponding weights, before executing the diagonal sum functions. Following the same design flow, four separate instances of the DAS kernel are created, with two of them being mapped on SLR0 and the other two on SLR2, creating a system diagram similar to the on Figure 6.2.

Tables 6.4 and 6.5 describe the performance of the proposed kernel, as well as the resource utilization.

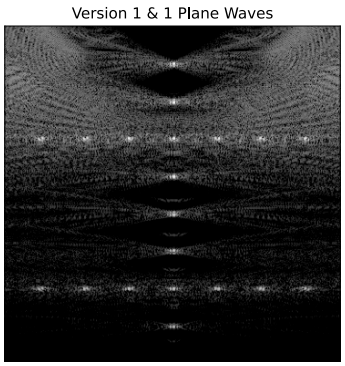
Clock Frequency (MHz)	300
Clock Cycles	2,218,847
System Runtime (ms)	7.899
Kernel Runtime (ms)	7.489
Frame Rate (Hz)	133.530
Pulse Repetition Frequency (Hz)	10,014.750
Beamforming Rate (MSamples/s)	1957
Power Consumption (Watt)	11.921

Table 6.4: Performance of our DAS Beamforming kernel, with the *linear* interpolation, for 75 plane waves

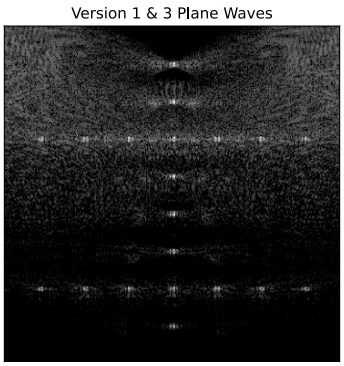
Resource	*1 CU/Full device*	*4 CUs/Full Device (%)*	*1 CU/Per SLR (%)*	*2 CUs/Per SLR (%)*
LUT	122316 (10.35 %)	41,14%	34.46%	68.92%
BRAM	286 (13.24 %)	52.96%	44.83%	89.66%
URAM	96 (12 %)	48%	30%	60%
Flip Flops	151127 (6.92 %)	27.68%	20.89	41.78%
DSP	1026 (15.02 %)	60.08%	45.30 %	90.60%

Table 6.5: Resource consumption of our DAS Beamforming kernel, with the *linear* interpolation, for 75 plane waves. Again, the design consists of 4 Compute Units, where 2 Compute Units are mapped to the same SLR.

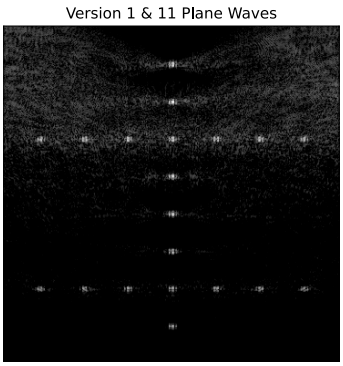
Compared with the aforementioned proposed kernel, this version achieves a higher latency, thus a



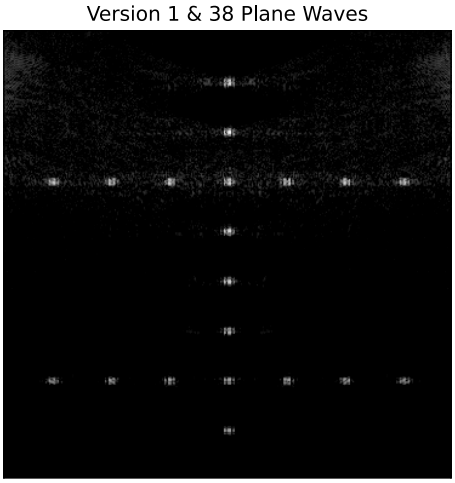
(a) 1 Plane Wave



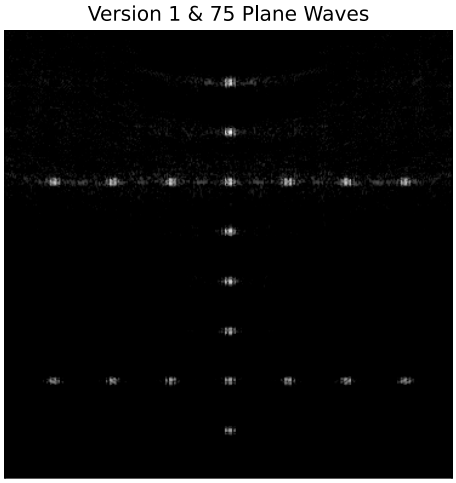
(b) 3 Plane Waves



(c) 11 Plane Waves



(d) 38 Plane Wave



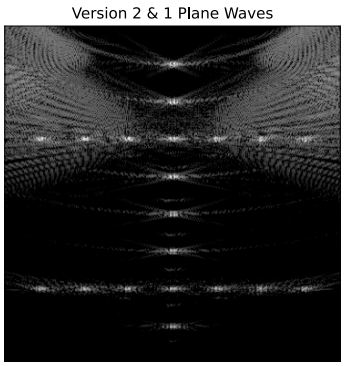
(e) 75 Plane Waves

Figure 6.3: Beamformed images with 1,3,11,38 and 75 plane waves, on the FPGA with the first proposed kernel.

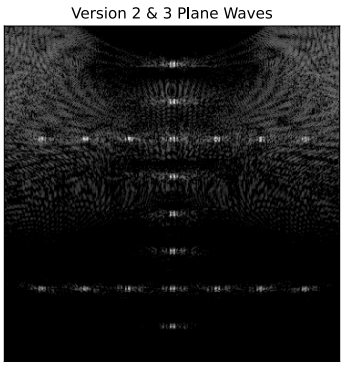
smaller frame rate, while consuming more FPGA resources. This behaviour is expected, since an additional computational step is added to design flow, causing an increase to the time needed to beamform a whole image row. Furthermore, every clock cycle, the values of the stacked samples block are multiplied by its corresponding weight. Since, the size of this block equals to $D \times \text{subaperture} \times 2 = 512$, then 1024 multiplications are being executed every clock cycle, due to the fact that every sample is estimated from its two adjacent samples, hence requiring two multiplications. Thus, at least 1024 DSPs are expected used for these calculations, increasing the DSP utilization. Additionally, the increase of the LUT and Flip Flops usage is explained by the need to temporary save the weights and the channel samples before multiplication, into registers. On the other hand, memory resource consumption (BRAMs, URAMs) remained almost unchanged, since the parameters of the application (number of pixels, aperture size, samples per channel) were not modified, hence the necessary buffers to store ultrasound samples, delays, and beamformed samples, have the same size. A slight increase is observed on the utilized BRAMs, and that happens due to the increase of the bit accuracy of the delay values (from 16 bits to 32 bits).

Figure 6.4 illustrates the beamformed images from the proposed design, for 1, 3, 11, 38 and 75 plane wave emissions.

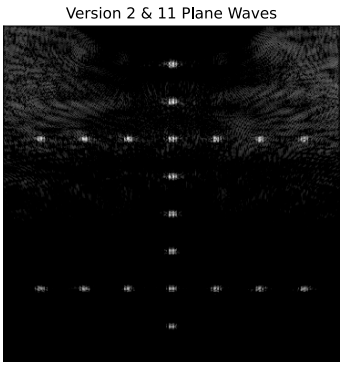
The increase in latency and resource consumption introduce an important benefit, the enhancement of the output image. Comparing with the output images from the first version of our design, the



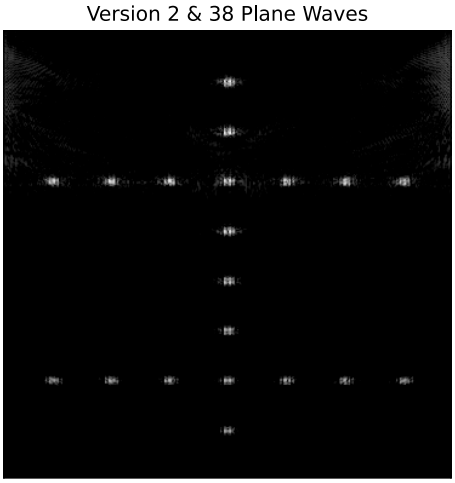
(a) 1 Plane Wave



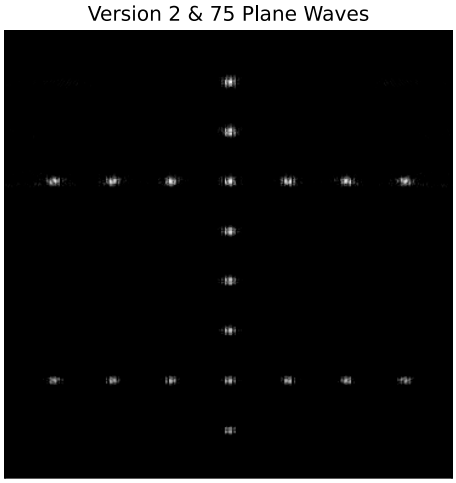
(b) 3 Plane Waves



(c) 11 Plane Waves



(d) 38 Plane Wave



(e) 75 Plane Waves

Figure 6.4: Beamformed images with 1,3,11,38 and 75 plane waves, on the FPGA with the second proposed kernel.

improvement on the image quality is significant, which means that the linear interpolation does indeed produce better estimation for the channels samples. Additionally, the importance of using several plane waves during ultrafast imaging is reinforced in both designs, considering that the increase in image resolution is achieved with the increase of the plane wave emissions, as we can see on Figures 6.3 and 6.4. Lastly, comparing these produced images with the one produced from our CPU, it is obvious that images produced with the same amount of plane waves, do not show significant differences.

To sum up our evaluation, the proposed kernels are capable of supporting Ultrafast Ultrasound imaging, since the supported Frame Rate is in the KHz range, for a single plane wave emission. The first design produces worse image quality but it is a faster and a cheaper option since it consumes less resources, whereas the design that applies linear interpolation produces higher quality images, while achieving a higher resource utilization and a lower frame rate. However, its frame rate is still acceptable for ultrafast, ultrasound applications. Therefore, after establishing the correctness of the two proposed designs, we will explore our kernel's scalability, by tuning different parameters.

6.2 Scalability of the Beamforming Kernel

The goal of this section is to outline the scalability of our design regarding performance, resource utilization and image quality. We will focus on the first version of our beamforming kernel, which

applies the simple interpolation, since it has a lower hardware complexity, thus making it easier and more time-efficient to profile the application. However, we can safely assume that the beamforming kernel that applies linear interpolation, has the same behaviour regarding scalability, since its design is the same with the initial kernel, with an additional step of weight integration.

In order to examine the scalability of our design, we will tune a handful of the kernel's parameters: the number of plane waves, the size of the stacked samples blocks and the aperture size. Below, we present the results collected from running different versions of the kernel, by highlighting which is the tuned parameter. We should point out that when tuning the the stacked samples block size and the aperture size, we examined the kernel's performance and resource consumption with the Vitis HLS tool. The latency is estimated by the total cycles the design requires, as well as the frequency it achieves and the resource utilization is collected after exporting the RTL on hardware, by utilizing the Vivado tool. After creating the bitstream for the proposed kernels mentioned above, we ascertained that these results provide an accurate insight on the design's characteristics. Therefore, we decided to use the export RTL feature of the Vitis HLS tool rather than creating the whole bitstream for the Alveo U200 platform, since it allows a more time efficient analysis. Moreover, while mentioning the Block RAMs consumed, we are referring to the 36-Kb BRAMs that are available into the FPGA.

Plane Waves

The aforementioned dataset was beamformed by utilizing 1, 3, 11, 38 and 75 plane waves. The output images received from the proposed beamformer, have already been presented on Figure 6.3. The performance of our kernel is described on Table 6.6 below.

Plane Waves	System runtime (ms)	Kernel runtime (ms)	Clock Frequency (MHz)	Average On-Chip Power (Watt)
1	0.471	0.089	300	10.800
3	0.670	0.267	300	10.800
11	1.382	0.989	300	10.800
38	3.789	3.461	300	10.800
75	7.135	6.747	300	10.800

Table 6.6: Performance for different amounts of plane waves

Additionally, since the kernel runtime is related to the achieved Frame Rate, Figure 6.5 provides a graph of the frame rate (x-axis) in relation to the transmitted plane waves.

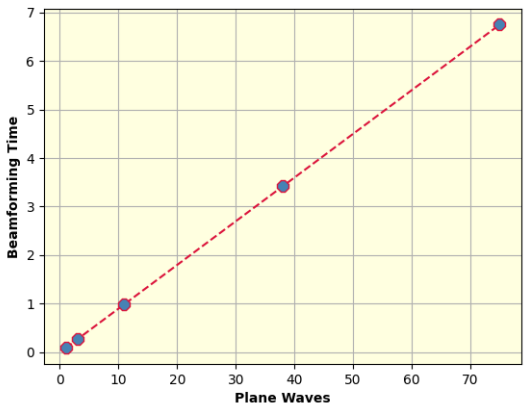


Figure 6.5: Graph of the design's latency with respect to the transmitted plane waves.

Since the data acquired from different plane waves are processed sequentially and not in parallel, the frame rate of our beamforming application increases linearly with respect to the amount of plane waves.

Furthermore, we should point out that the resources consumed from our beamforming design do not increase with the increase of the plane waves. This behaviour is expected, because the kernel reuses the same RTL that beamforms a single plane wave, for the processing of the additional plane waves. If one wishes to speed up the kernel for multiple plane waves, then he could create multiple instances of this RTL and process several plane waves in parallel. For example, by creating two instances of our kernel, the kernel latency will be halved at the expense of extra resources, whose consumption will be doubled.

It is obvious that the image resolution gets higher when more plane waves are utilized. This showcases the importance of emitting multiple plane waves for ultrafast ultrasound imaging.

Stacked Samples Block Size

A reasonable question emerges about the proposed design, regarding the size of the Blocks that we describe in Chapter 5. By changing their size, the latency of the design is affected in conjuction with the resources consumed. We opted to set their size D equal to 8 rows, since it requires four copies of the input samples, a reasonable amount that allows us to create four instances of the baseline kernel that fit into the FPGA. However, in order to examine which is the optimal size D, that achieves low latency without consuming all of the FPGA's resources, we created four separate design's of the proposed kernel, that beamforms 609 image rows of 64 pixels, with an aperture size of 64 channels. Each design is implemented with a Block size of 4, 8, 16 and 32 rows. The resource consumption for the four aforementioned versions, as well as their achieved latency, are written in Table 6.7.

Block Size	LUT	FF	BRAM	URAM	DSP	Kernel Latency (ms)
4	41139	38899	136	96	3	**8.93**
8	36185	38289	271	96	3	**6.65**
16	48851	40400	528	96	2	**5.51**
32	85323	50540	1040	96	4	**5.38**

Table 6.7: Resource consumption and Latency of the proposed design, for different Stacked Samples Block size. The resource consumption that surpasses the available resources on a single SLR is marked with red text.

By examining the information collected from the execution of these four designs, a handful of observations could be made about the influence of the selected Block size on the application's performance.

- While the size D increases, the BRAM demands of the design also increase exponentially. This is expected, since every Block row needs to be read in one clock cycle. Taking into consideration that every FPGA BRAM is able to provide two read ports, $D/2$ copies of the input samples are necessary as well as higher array partition for every buffer.
- More LUTs and Flip Flops are used as the size D increases. For higher values of D, the functions that implement the diagonal sum require more resources, since they sum together more samples.
- The latency of the design decreases as the size D increases. By implementing bigger stacked samples blocks, more samples are summed into every diagonal sum, hence decreasing the diagonal sums, and consequently the clock cycles, needed for calculating the final beamformed value. However, the latency improvement is not linear, due to the fact that calculating the diagonal sums for a single block requires more than one clock cycle as size D increases. Graph 6.6 proves this claim.
- The compute unit with $D=16$ barely fits into an SLR, thus only 2 of these compute units fit into the Alveo U200 (SLR1 has less resources than SLR0 and SLR2). Additionally, the compute unit with $D=32$ does not fit into a single SLR, which means that a maximum of two similar

compute units fit into the Alveo U200, without avoiding SLR crossing. As a result, the clock period is increased, increasing at the same time the application's latency. Meanwhile, regarding the designs with *D=4* and *D=8*, two of those compute units can be mapped to SLR0 and SLR2, both allowing the beamforming of an 128×609 image, with an aperture size of 128 channels, in the desired kernel runtime.

For better visualization of the aforementioned observations, a Graph of the kernel's latency with respect to the block size D is presented (Figure 6.6, as well as a bar plot (Figure 6.7) that illustrates the BRAM, LUT and Flip Flop SLR consumption, by percentage.

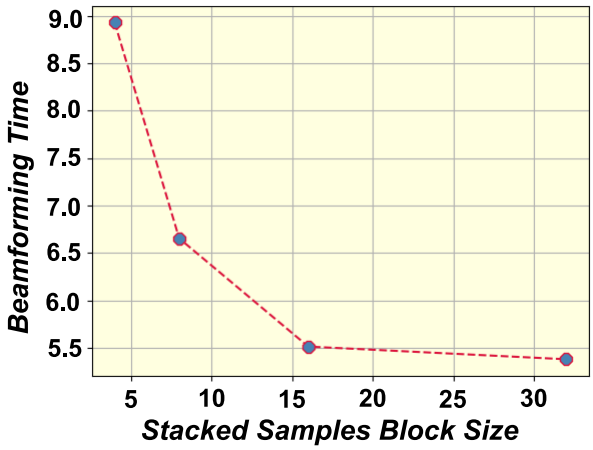


Figure 6.6: Graph of the design's latency with respect to the stacked samples block size.

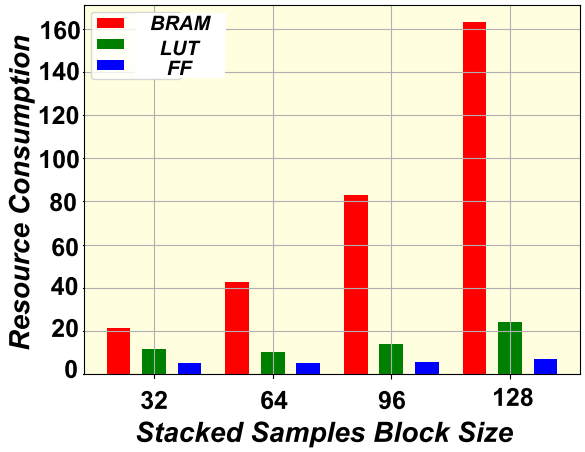


Figure 6.7: Bar Plot representing the resources consumed for every examined size *D*.

One thing that stands out from Graph 6.7, is that while increasing the size *D*, the BRAM consumption increases rapidly compared to the LUT and FF consumption. This strengthens our claim that the proposed design is memory constrained.

All in all, it is obvious that indeed the optimal Block size *D*, to beamform an ultrasound image with an aperture size of 128 is the D = 8 rows, since it provides the minimum latency without consuming

more FPGA resources than the ones available. If someone has less FPGA resources available, then he could consider either reducing the aperture size to 64 channels, while keeping the Block size D = 8, or reducing the Block size to D = 4, while maintaining the aperture size equal to 128 channels. At the same time, if the goal is to beamform the image with an aperture size of 64 channels, then the blocks sizes of D=16 or D=32 could be better options, since two of these compute units are able to fit into the Alveo U200, while reducing the design's latency at the same time.

Aperture Size

One of the most important parameters during ultrasound imaging, that influences heavily the image quality, is the aperture size used for beamforming. A higher amount of utilized transducer elements results into more information on the output image, thus yielding better image quality. Therefore, in order to analyze the scalability of the proposed design, it is important to examine the performance of the design for different amount of transducer elements. To achieve that, bitstreams are created for several amount of channels, together with different Block sizes. We also tuned the Block size variable, due to the fact that increasing the aperture size also increases the distinct Δx values, hence increasing the size of the rows on the stacked samples blocks. The resource utilization for these designs, as well as the achieved latency, are described into Table 6.8.

Aperture Size & Block Size	**LUT**	**FF**	**BRAM**	**URAM**	**DSP**	**Kernel Latency (ms)**
128 & 4	128308	115441	290	176	3	**16.91**
128 & 8	101836	85444	516	176	3	**11.92**
128 & 16	199720	248368	984	176	2	**14.21**
96 & 4	81393	73395	206	128	3	**13.07**
96 & 8	61222	59026	388	128	3	**9.39**
96 & 16	73320	67828	728	128	2	**7.41**
32 & 4	16308	19654	78	48	3	**5.59**
32 & 8	14858	19592	132	48	3	**4.37**
32 & 16	21073	24490	216	48	2	**3.31**

Table 6.8: Resource consumption and Latency for an aperture size of 32,96,128 channels and a block size of 4,8,16 rows. The resource consumption that surpasses the available resources on a single SLR is marked with red text.

From the aforementioned table, several observations are made about the influence of the aperture size, into the performance of the beamforming application.

- BRAM and URAM utilization increases with the increase of the aperture size. As expected, more transducer elements translate to more ultrasound samples, delay values and image pixels, that are stored into buffers, thus increasing linearly the BRAM and URAM consumption.
- LUT and Flip Flop utilization also increases with the increase of the aperture size. A greater channel count leads to more pixels on an image row, as well as rows with more samples on the stacked samples blocks. As a result, not only a higher amount of diagonal sums is calculated for each block, but also more beamformed values emerge for each image row. To complete these calculation, more LUTs and Flip FLops are required.
- Regarding latency, it is obvious that by increasing the aperture size of the design, the latency also increases. However, the latency increases in a rate smaller than the increase rate of the transducer elements. Therefore, if the available resources allow it, it is preferable to create a single compute unit that beamforms an image with a larger aperture size, rather than beamforming the same image with multiple compute units with a smaller aperture size, that run sequentially. An

exception to the aforementioned is the design with an aperture size of 128 channels and a block size $D = 16$. In this situation, increasing the size D does not achieve the expected behaviour regarding latency, and the design with $D = 8$ achieves better performance. This is happening due to the fact that the diagonal sums cannot be executed in one clock cycle, thus not allowing a fully pipelined design with II = 1 and increasing the design's latency. This behaviour can be seen clearly on Graph 6.8.

Moreover, two graphs are provided, for better visualization of the claims above. Graph 6.8 illustrates the alteration of the kernel's runtime for different aperture sizes, while Bar Plot 6.9 presents the resource consumption for the designs described on Table 6.8 and how the aperture size affects it.

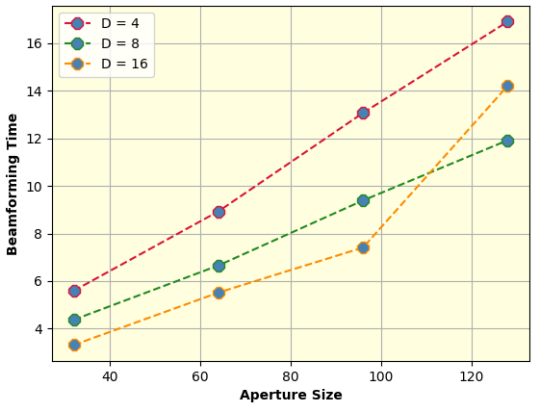
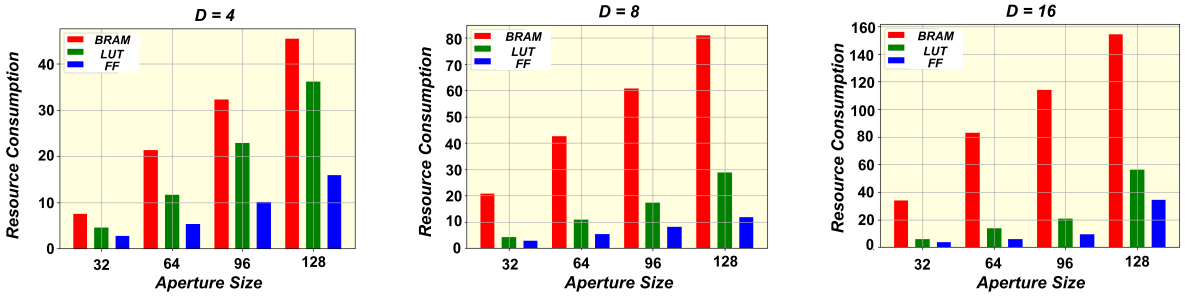


Figure 6.8: Graph of the design's latency with respect to the aperture size, while keeping the size D fixed to a certain value (4, 8 and 16 rows).



(a) $D = 4$ (b) $D = 8$ (c) $D = 16$

Figure 6.9: Bar Plot representing the resources consumed for every examined aperture size, while keeping unchanged the value D.

To sum up our observations, it is clear that by increasing the aperture size of the design not only the resources consumed are increased, but also the latency of the design. Consequently, our design appears to be scalable as far as the aperture size of the beamforming is concerned. Let it be noted that the rate the resources are increased with the increase of the aperture size seems steady, no matter the size D. Once again, one can observe that the first FPGA resources that are fully utilized are the Block RAMs, proving that the design is indeed memory constrained.

To continue with the evaluation of the proposed Delay and Sum kernel, comparisons of our design with software and hardware based beamforming kernels follow.

6.3 Comparison with state of the art DAS Beamformers

An important step of evaluating the proposed kernel is to compare its performance to state of the art Delay and Sum beamformers. At first, the proposed design will be compared with other hardware-based DAS kernels developed on an FPGA. Additionally, we will display the performance of the naive DAS GPU beamformer, provided by the Center for Ultrasound and Brain imaging (CUBE) laboratory from Erasmus University of Rotterdam, while beamforming the ultrasound signals from the dataset used for evaluating the proposed design.

6.3.1 Comparison with FPGA DAS designs

On Table 6.9 we present the performance of the two proposed FPGA delay and sum beamforming kernels, as well as the state of the art beamforming FPGA designs, described in Chapter 3.

Design	***PRF (Hz)***	***BF Rate (MSPS)***	***Frame Rate (Hz)***	***PW***	***Image Size***	***Aperture Size***
Proposed kernel (1st Version)	11,116	2173	148.21	75	128×609	128
Proposed kernel (2nd Version)	10,014	1927	133.53	75	128×609	128
[2]	714	1120	714	1	64×181	64
[5]	3800	467	345	11	96×512	32
[9]	14,000	917	875	16	64×384	8
[7]	14,865	4830	14,865	1	64×1280	64

Table 6.9: Comparison with state of the art FPGA ultrafast beamformers

At first glance, it is obvious that the proposed designs achieve a PRF and a beamforming rate comparable to the state of the art ultrafast systems, while using at the same time the largest aperture size. Compared to the designs proposed by [2] and [5], our kernels outperforms them regarding PRF, beamforming rate, image size and aperture size. Additionally, compared to the design introduced by Campbell et al. [9], the two kernels described in this thesis achieve a higher beamforming rate but a lower PRF. However, they utilize a much more higher aperture size (128 elements to 8 elements), in combination with a higher image size, providing an importantly higher potential for an improved quality of image. One can understand that the difference in PRF does not justify the difference in aperture size (16 times higher), hence sacrificing frame rate for a significant improvement on image quality. Lastly, compared to the design introduced by Kou et al. [7], even though our design achieves comparable PRF and beamforming rate values, it still achieves a lower performance. However, it is important to point out that our design utilizes double the amount of transducer elements. As previously explained, the aperture size influences linearly the latency of our design. Thus, by reducing the aperture size to 64 channels for our kernel, the latency would face a decrease by almost two times, achieving in that way PRF and MSPS values that surpass the ones achieved from the design proposed by Kou et al. [7]. Furthermore, the second version of the proposed kernel implements a linear interpolation, an improved interpolation to the one utilized by [7] and the first version of the proposed kernel. Therefore, the second version of the proposed kernel might achieve worse performance, but implements a linear interpolation and an aperture size of 128 transducer elements, reconstructing images with higher quality.

In an attempt to take into consideration the aperture size of the design, while evaluating the Pulse Repetition Frequency, we create a composite metric equal to $PRF \times$ channels. We refer to this metric as effective PRF. Table 6.10 presents the effective PRF for each design we used for comparisons.

Design	$PRF \times$ ***Aperture Size*** $(KHz \times channels)$
Proposed kernel (1st Version)	1422.848
Proposed kernel (2nd Version)	1281.792
[2]	45.696
[5]	121.600
[9]	112.000
[7]	951.360

Table 6.10: $PRF \times$ channels for the proposed designs, as well as the state of the art FPGA ultrafast beamformers.

In conclusion, the proposed designs offer an improved solution for ultrafast beamforming, regarding achieving a high PRF, while at the same time maintaining a high aperture size. However, it is crucial to mention that this composite metric would only be accurate if the design's latency is linearly dependent to the Aperture Size. This is the case for the designs proposed on this thesis, but might not be valid for the rest of the examined designs. Additionally, it is worth to mention that the design by Zhengchang Kou et al. [7], that achieves a comparable *effective PRF*, beamforms almost double the image rows compared to our designs. Taking into consideration the amount of image depths beamformed, and assuming that the design they propose has a latency proportional to the beamformed image rows, then our design would still be outperformed.

Since the authors of the aforementioned state of the art beamformers do not provide the datasets they used to conduct their experiments, this proposed metric is an attempt to equally compare in the best way possible, our designs with the state of the art ultrafast FPGA beamformers.

All in all, the proposed DAS hardware kernels in this thesis achieves frame rates that can compare and outperform the state of the art hardware-based, ultrafast beamformers, while utilizing an aperture size that is at least two times higher than the designs described on Table 6.9.

6.3.2 Comparison with a GPU beamformer, provided by CUBE

Since Graphic Processor Units are a suitable candidate for a software beamformer for high frame rates, as explained in Chapter 4, it would be productive to compare the proposed designs of this thesis to a software beamformer deployed on a GPU. To accomplish this, we present the performance of a naive Delay and Sum beamformer implemented on a GPU, after beamforming the dataset used in the current chapter. Table 6.11 presents the runtime of the aforementioned GPU kernel.

Kernel Runtime (ms)	254.607
Frame Rate (Hz)	3.928
PRF (Hz)	294.600
Beamforming Rate (MSamples/s)	57.581

Table 6.11: Performance of the naive GPU beamformer provided by CUBE, for 75 plane waves.

Comparing this implementation to the second version of the proposed kernel (improved quality, worse performance), it is evident that our kernel achieves a significant improvement on the Pulse Repetition Frequency, as well as the beamforming rate. Our design is able to provide a frame rate that is approximately 50 times higher, making the implementation of ultrafast imaging far more feasible. It is proved that the proposed kernels achieve a better performance than a simple DAS implementation on a GPU device.

Chapter 7

Conclusions & Future Work

In this thesis, we attempted to design a Delay and Sum beamforming kernel on a single FPGA chip that operates in high frame rates. The output is a two dimensional B-mode image that is reconstructed after compounding ultrasonic signals emitted from several transmit angles. At first, we addressed the problem of locating the appropriate parallelization schemes to accelerate the standard DAS algorithm. After realizing that the three dimensions the problem revolves around (coordinates on the z-axis, coordinates on the x-axis, channels), cannot support the necessary parallelization to reduce the design's latency at the desired levels, while maintaining the LUT utilization under 100%, we revised the standard DAS algorithm. Thus, the calculations are executed over two dimensions. After that, we designed the beamformer that supports raw RF input data and beamforms the insonified area with an aperture size of 128 transducer elements. We proposed two versions of the kernel; one that applies a naive interpolation on the necessary samples and one that applies linear interpolation. First one achieves lower latency and less resource consumption than the second one, however it produces images with worse quality. The two proposed designs were compared with state of the art FPGA and software based ultrafast beamformers.

7.1 Conclusions on the evaluation results

After evaluating the results from the executions of the proposed beamforming kernels, we came to the conclusion that our designs are able to support ultrafast ultrasound imaging applications. More specifically, both kernels beamform images with an aperture size of 128 transducer elements, after receiving 1527 samples for each element, with a high frame rate. The first version that applies a naive interpolation achieves a pulse repetition frequency of 11116.052 Hz and a beamforming rate of 2173 MSamples/s. On the other hand, the second version that applies linear interpolation achieves a pulse repetition frequency of 10014.75 Hz and a beamforming rate of 1957 MSamples/s. It is clear that both kernels support high frame rates since the achieved PRF is in the KHz range or equivalently their beamforming rate is in the GSamples/s range.

Moreover, after comparing the proposed kernels with the state of the art FPGA designs presented in Chapter 4, we came to the conclusion that our work achieves similar performance and even outperforms the examined state of the art designs, regarding PRF, beamforming rate and aperture size. The PRF and beamforming rate are a metric of the application's frame rate, while a high aperture size is important for the quality of the reconstructed image. Especially, assuming that the latency of the application is proportional to aperture size (something that is true for our design), if we convert each examined design to the same aperture size (128 channels for instance), then the designs proposed on this thesis outperform these specific state of the art hardware beamformers.

Lastly, after tuning the amounts of plane waves, the aperture size and the stacked samples block size, we infer that design's latency, as well as the resource consumption, are proportional to these parameters.

Increasing the plane waves increases linearly the kernel's latency, whereas increasing the aperture size or the stacked samples block size increases the latency and the FPGA resources utilization, but not linearly. This means that we can roughly predict the behaviour of our design for different dataset sizes. Therefore, the scalability of the proposed kernels is established since we can create the same design for different dataset size, with minimum changes, and expect a certain behaviour. Additionally, we pointed out that the proposed designs are memory constrained, hence the first FPGA resources that reach saturation after increasing the size of certain parameters, are the FPGA's BRAMs. This could be a useful observation for future improvement of the design, especially since we have already portrayed the reasons why the memory requirements of our design are high.

All in all, this thesis proposes a beamforming kernel implemented on a single FPGA, that supports high frame rates and outperforms the examined state of the art hardware beamformers, when all designs are converted to utilize the same aperture size and the same plane waves amount. Moreover the proposed design is scalable, since one is able to produce the same design for different ultrasound parameters, with minimal changes, while maintaining the expected performance. The proposed designs beamform raw RF ultrasound samples with an aperture size of 128 transducer elements and achieve a pulse repetition frequency of 11116.052 Hz (1st Version) and 10014.75 Hz (2nd Version), as well as a beamforming rate of 2173 MSamples/s (1st Version) and 1957 MSamples/s (2nd Version).

7.2 Future Work

During the implementation of the proposed design and subsequent experimentation, multiple obstacles occurred that restrict its performance, as well as the quality of the images it produces. As future tangents to our work, various optimizations and modifications could be applied that aim to a further improvement of the proposed design. Most of them would focus on a most efficient memory handling, a faster beamforming kernel or an improved image quality. Based on the experience gained while developing our hardware based beamforming kernel, we are able to discuss suggestions for future research, driven by our work :

- Improving the memory transfers inside the FPGA. As previously explained, the proposed design is heavily memory constrained. Before the launch of the beamforming calculations for a single plane wave, the kernel reads from the DDR the whole input frame. However, each image row requires only a certain set of the channel samples for its beamforming, depending on its depth. By exploiting this observation, the size of the input buffers used during the beamforming of a certain row could be reduced, thus reducing the BRAM consumption. This could open possibilities for further parallelization, hence increasing the frame rates of the beamforming kernel.

- Transferring the proposed kernel and the modified DAS algorithm on an FPGA that offers more resources, as well as more computational power. A suitable candidate would be the Versal ACAP, introduced by Xilinx [12], that not only provides more resources than the FPGA used during this thesis, but also contains AI engines which achieve a compute density at low power that allows real-time, low latency ultrasound signal processing [13].

- Implementing the proposed kernels with float point variables, rather than fixed point datatypes. Float point operations would achieve an improvement on the beamformed values accuracy, hence on the reconstructed image's quality. On the other hand, this modification would result into higher resource consumption. In an FPGA with more resources, it would be interesting to explore the trade-off between image resolution and resource utilization.

- Exploring different types of interpolation, such as a polynomial interpolation with a higher degree or a spline interpolation. Analyzing the improvement on the image's resolution and the effect on the application's latency and on the resource and energy consumption.

- Modifying the proposed kernel to support IQ input data. Since this modification would require phase rotations, that include calculating sine and cosine functions, a platform with more resources available will be needed.

- Integrating several image enhancing processes on-chip, that are applied on the beamformed signal, improving the image's visualization. For instance, envelope detection and log compression are the most widely used post-beamforming processes, that include operations like calculating the Hibbert transformation and the logarithm of the beamformed signal. This could be accomplished into an FPGA that provides more DSP engines or with the recently upcoming Xilinx's Versal platform [12], that offers AI engines that allow "high-performance, real-time DSP capabilities" [13].

Appendix A

Vitis and Vitis HLS tools

This Appendix will attempt to provide more info about the development of the proposed kernels, with the Vitis & Vitis HLS tools.

A.1 Vitis HLS kernel Development

To efficiently produce a C++ kernel with the Vitis HLS tool, the use of directives is necessary. On the Table A.1 it is described how the use of HLS directives contributes into efficient scheduling and memory management.

HLS Directive	***Influence in scheduling or memory management***
#pragma HLS pipeline	Pipelines the block loop during data processing as well as loops for memory transactions with the DDR.
#pragma HLS unroll	Unrolls the innermost loops of pipelined loops. Used for operations inside a loop with no dependencies, that can executed in parallel.
#pragma HLS array_partition	Cyclic Partition for channel, delay and output buffers that should be implemented as Block RAMS or Ultra RAMS. Complete partition for in between arrays that should be implemented as individual registers.
#pragma HLS bind_storage	Used to control three characteristics of the internal FPGA memories. Defines if a buffer is implemented as a BRAM or a URAM, forces buffers to be implemented as dual-port RAMs and determines the default latency for the binding of this memory.
#pragma HLS interface	Implements I/O kernel ports with the AXI4 protocol, by using the mode=m_axi option
#pragma HLS inline	Implements the three summing functions of the kernel as inlined and not as separate RTLs, which allows operations within the function to be shared and optimized more effectively with the calling function. This might increase area but at the same time increases the clock's frequency.

Table A.1: HLS Directives used to optimize the Delay and Sum beamforming kernel

A.2 Designing with the Vitis tool

A Vitis application consists of three main components :

- The host code, written in OpenCL.
- The kernel code, written in C/C++ and synthesized with Vitis HLS.
- The linker, the component that creates the bistream for the desired FPGA and basically integrates the kernels on the base platform.

A.2.1 OpenCL Host Application

In general, the structure of the OpenCL API host code is divided into three sections:

1. Setting up the OpenCL environment.
2. FPGA command execution including executing kernels and buffer transfer to/from the device.
3. Post processing and FPGA cleanup.

Regarding the environment setup, the standard OpenCL structures need to be initialized. First of all, a platform is identified composed of one or more Xilinx devices where the host application needs to identify the corresponding devices, in order to choose at what device the application should be executed. The server where we build and run our applications includes a platform of two Xilinx accelerator boards of the Alveo family and the device chosen from the host code is the Alveo U200. Afterwards, we create a context utilizing the *clCreateContext* API, that contains the Xilinx device and is able to communicate with the host machine. At the same time, a command queue is created from the *clCreateCommandQueue* API, that supports an out-of-order execution of multiple kernels, which is useful since we aim to execute multiple identical kernels in parallel. After all these steps, the host application is ready to load the FPGA binary (.xclbin file) using the *clCreateProgramWithBinary* API.

After the FPGA bitstream is loaded on the board, the next step is to prepare the kernels for execution. We utilize the OpenCL API *clCreateKernel* to access the four different instances of the beamforming kernel that supports 64 channels and 64 pixels on each image row. It is also important to setup the kernel arguments. For every beamforming kernel, three buffers are created, two input and one output buffer. These buffers are a pointer to a memory object created with the context associated with the program and kernel objects. We define the two input buffers, that contain the channel and the delay data, as **read only**, and the output buffer that contains the compounded beamformed data, as **write only**. After initializing the two input buffers, all of them are allocated on the device and mapped on the FPGA's DDR and the input values are transfered from the local host machine to the FPGA's external memory. The command queue is now ready to execute all the computational units in parallel. After beamforming is over, the beamformed data are transfered from the FPGA memory to the local host machine. We utilize the *chrono* C library to measure the system runtime.

Lastly, at the end of the host code, all the resources should be released by unmapping the allocated buffers from the FPGA. After execution, four groups of beamformed data are calculated. Values that correspond to the same groups of 64 pixels, out of the 128, but from different channels, are summed on the machine's CPU in order to acquire the complete beamfromed values for those pixels. Eventually, we end up with two groups of beamformed values for image rows of size equal to 64 pixels, that originate from 128 channels, which means that the final image consists of image rows with a size of 128 pixels and the beamforming was conducted with an aperture size of 128. The output is saved on text file and is used for further processing and image visualization with the help of Matlab functions.

A.2.2 Kernel Synthesis and Linking

In order to create the bitstream that programs the Alveo board with our beamforming kernels we should run the Synthesis and Implementation steps of Vivado. These steps happen automatically through Vitis. The tool creates the necessary makefiles to at first synthesize the kernel code through

Vitis HLS, like previously mentioned, and then after a successful C Synthesis the linker is responsible for executing Vivado Synthesis, Implementation and bitsream generation.

During the v++ linking process we define how many instances of our kernel are created and we specify to which Super Logic Region (SLR) they are be mapped. Moreover, the linker options give the opportunity to specify which ports of the compute units are mapped to which FPGA memory resources like the DDR, HBM and PLRAM. Finally, during the HLS code compilation as well as the linking phase the clock frequency of the kernels could be defined using the v++ compiler. To tune the aforementioned options, Vitis provides a linking configuration file which is used to guide the Vitis compiler through the Vitis phase. An example of our config file is the following.

```
[advanced]
misc=solution_name=link

[connectivity]
nk=bf_kernel:4:bf_kernel_1.bf_kernel_2.bf_kernel_3.bf_kernel_4
slr=bf_kernel_1:SLR0
slr=bf_kernel_2:SLR0
slr=bf_kernel_3:SLR2
slr=bf_kernel_4:SLR2

sp=bf_kernel_1.inp:DDR[0]
sp=bf_kernel_1.delays:DDR[0]
sp=bf_kernel_1.ch_ang_inp:DDR[0]
sp=bf_kernel_1.outp:DDR[0]

sp=bf_kernel_2.inp:DDR[0]
sp=bf_kernel_2.delays:DDR[0]
sp=bf_kernel_2.ch_ang_inp:DDR[0]
sp=bf_kernel_2.outp:DDR[0]

sp=bf_kernel_3.inp:DDR[3]
sp=bf_kernel_3.delays:DDR[3]
sp=bf_kernel_3.ch_ang_inp:DDR[3]
sp=bf_kernel_3.outp:DDR[3]

sp=bf_kernel_4.inp:DDR[3]
sp=bf_kernel_4.delays:DDR[3]
sp=bf_kernel_4.ch_ang_inp:DDR[3]
sp=bf_kernel_4.outp:DDR[3]
```

After linking compilation, the binary file for the design is generated and it is loaded in the FPGA by the host application, as stated above. After executions are complete we can utilize the Vitis Analyzer to profile our application, measure kernel time execution and power consumption and track memory transfers between host-device and DDR-internal FPGA memory.

Bibliography

[1] Hager, Pascal Alexander, Speicher, Daniel, Degel, Christian, and Benini, Luca, "Ultralight: An ultrafast imaging platform based on a digital 64-channel ultrasound probe," en, 2017. DOI: 10.3929/ETHZ-B-000219261.

[2] J. U. Kidav and S. S. G, "An FPGA-accelerated parallel digital beamforming core for medical ultrasound sector imaging," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 69, no. 2, pp. 553–564, Feb. 2022. DOI: 10.1109/tuffc.2021.3126578.

[3] C. Zhang, X. Geng, F. Yao, *et al.*, "The ultrasound signal processing based on high-performance cordic algorithm and radial artery imaging implementation," *Applied Sciences*, vol. 13, no. 9, 2023, ISSN: 2076-3417. DOI: 10.3390/app13095664.

[4] G. Montaldo, M. Tanter, J. Bercoff, N. Benech, and M. Fink, "Coherent plane-wave compounding for very high frame rate ultrasonography and transient elastography," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 56, no. 3, pp. 489–506, Mar. 2009. DOI: 10.1109/tuffc.2009.1067.

[5] E. Boni, L. Bassi, A. Dallai, *et al.*, "Architecture of an ultrasound system for continuous real-time high frame rate imaging," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 64, no. 9, pp. 1276–1284, Sep. 2017. DOI: 10.1109/tuffc.2017.2727980.

[6] P. Song, J. D. Trzasko, A. Manduca, *et al.*, "Improved super-resolution ultrasound microvessel imaging with spatiotemporal nonlocal means filtering and bipartite graph-based microbubble tracking," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 65, no. 2, pp. 149–167, 2018. DOI: 10.1109/TUFFC.2017.2778941.

[7] Z. Kou, Q. You, J. Kim, *et al.*, "High-level synthesis design of scalable ultrafast ultrasound beamformer with single FPGA," *IEEE Transactions on Biomedical Circuits and Systems*, pp. 1–12, 2023. DOI: 10.1109/tbcas.2023.3267614.

[8] E. Mace, G. Montaldo, B.-F. Osmanski, I. Cohen, M. Fink, and M. Tanter, "Functional ultrasound imaging of the brain: Theory and basic principles," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 60, no. 3, pp. 492–506, Mar. 2013. DOI: 10.1109/tuffc.2013.2592.

[9] N. A. Campbell and J. A. Brown, "A real-time dual-mode high-frequency beamformer for ultrafast and focused imaging," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 69, no. 4, pp. 1268–1276, Apr. 2022. DOI: 10.1109/tuffc.2022.3151218.

[10] B. L. West, J. Zhou, C. Chakrabarti, and T. F. Wenisch, "Delay compression: Reducing delay calculation requirements for 3d plane-wave ultrasound," in *2019 IEEE International Ultrasonics Symposium (IUS)*, 2019, pp. 1278–1281. DOI: 10.1109/ULTSYM.2019.8925725.

[11] H. Liebgott, A. Rodriguez-Molares, F. Cervenansky, J. Jensen, and O. Bernard, "Plane-wave imaging challenge in medical ultrasound," Tech. Rep. Denmark, 2016.

[12] *Versal AI Core Series*, Accessed: 10-7-2023.

[13] *AI Engines and Their Applications (WP506)*, Accessed: 10-7-2023.

[14] J. Bercoff, G. Montaldo, T. Loupas, *et al.*, "Ultrafast compound doppler imaging: Providing full blood flow characterization," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 58, no. 1, pp. 134–147, Jan. 2011. DOI: 10.1109/tuffc.2011.1780.

[15] D. Hyun, Y. L. Li, I. Steinberg, M. Jakovljevic, T. Klap, and J. J. Dahl, "An open source GPU-based beamformer for real-time ultrasound imaging and applications," in *2019 IEEE International Ultrasonics Symposium (IUS)*, IEEE, Oct. 2019. DOI: 10.1109/ultsym.2019.8926193.

[16] B. Y. S. Yiu, I. K. H. Tsang, and A. C. H. Yu, "GPU-based beamformer: Fast realization of plane wave compounding and synthetic aperture imaging," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 58, no. 8, pp. 1698–1705, Aug. 2011. DOI: 10.1109/tuffc.2011.1999.

[17] B. Y. S. Yiu, M. Walczak, M. Lewandowski, and A. C. H. Yu, "Live ultrasound color-encoded speckle imaging platform for real-time complex flow visualization *In Vivo*," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 66, no. 4, pp. 656–668, Apr. 2019. DOI: 10.1109/tuffc.2019.2892731.

[18] C. Risser, H. J. Welsch, H. Fonfara, H. Hewener, and S. Tretbar, "High channel count ultrasound beamformer system with external multiplexer support for ultrafast 3d/4d ultrasound," in *2016 IEEE International Ultrasonics Symposium (IUS)*, IEEE, Sep. 2016. DOI: 10.1109/ultsym.2016.7728714.

[19] *The ultrasound transducer - ECG & ECHO*, Accessed: 10-7-2023.

[20] K. Chatar and M. George, "Analysis of existing designs for fpga-based ultrasound imaging systems," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 9, pp. 13–24, Jul. 2016. DOI: 10.14257/ijsip.2016.9.7.02.

[21] J. Lim, "Circuits on miniaturized ultrasound imaging system-on-a-chip: A review," *Biomedical Engineering Letters*, vol. 12, no. 3, pp. 219–228, May 12, 2022. DOI: 10.1007/s13534-022-00228-w.

[22] *Sidelobes - Wikipedia*, Accessed: 10-7-2023.

[23] A. Murphy and D. McGrath, *Side lobe artifact*, Aug. 2016. DOI: 10.53347/rid-47372.

[24] A. Bole, A. Wall, and A. Norris, "Chapter 2 - the radar system – technical principles," in *Radar and ARPA Manual (Third Edition)*, A. Bole, A. Wall, and A. Norris, Eds., Third Edition, Oxford: Butterworth-Heinemann, 2014, pp. 29–137, ISBN: 978-0-08-097752-2. DOI: https://doi.org/10.1016/B978-0-08-097752-2.00002-7. [Online]. Available:

[25] J. Bercoff, "Ultrafast ultrasound imaging," in *Ultrasound Imaging*, I. V. Minin and O. V. Minin, Eds., Rijeka: IntechOpen, 2011, ch. 1. DOI: 10.5772/19729. [Online]. Available:

[26] M. Tanter and M. Fink, "Ultrafast imaging in biomedical ultrasound," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 61, no. 1, pp. 102–119, Jan. 2014. DOI: 10.1109/tuffc.2014.2882.

[27] H. Hasegawa and C. de Korte, "Special issue on ultrafast ultrasound imaging and its applications," *Applied Sciences*, vol. 8, no. 7, p. 1110, Jul. 10, 2018. DOI: 10.3390/app8071110.

[28] *Ultrafast Ultrasound Imaging*, Accessed: 10-7-2023.

[29] E. Tiran, T. Deffieux, M. Correia, *et al.*, "Multiplane wave imaging increases signal-to-noise ratio in ultrafast ultrasound imaging," *Physics in Medicine and Biology*, vol. 60, no. 21, pp. 8549–8566, Oct. 21, 2015. DOI: 10.1088/0031-9155/60/21/8549.

[30] G. Matrone, A. S. Savoia, G. Caliano, and G. Magenes, "Ultrasound plane-wave imaging with delay multiply and sum beamforming and coherent compounding," *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 3223–3226, 2016. [Online]. Available:

[31] M. Couade, "The advent of ultrafast ultrasound in vascular imaging: A review," *Journal of Vascular Diagnostics and Interventions*, p. 9, May 2016. DOI: 10.2147/jvd.s68045.

[32] O. Villemain, J. Baranger, M. K. Friedberg, *et al.*, "Ultrafast ultrasound imaging in pediatric and adult cardiology: Techniques, applications, and perspectives," *JACC: Cardiovascular Imaging*, vol. 13, no. 8, pp. 1771–1791, 2020, ISSN: 1936-878X. DOI: https://doi.org/10.1016/j.jcmg.2019.09.019. [Online]. Available:

[33] L. Demi, "Practical guide to ultrasound beam forming: Beam pattern and image reconstruction analysis," *Applied Sciences*, vol. 8, no. 9, p. 1544, Sep. 2018, ISSN: 2076-3417. DOI: 10.3390/app8091544. [Online]. Available:

[34] R. Göbl, "Receive Beamforming in Medical Ultrasound—A Review of Aperture Data Processing," Sep. 2022. DOI: 10.36227/techrxiv.21202514.v1. [Online]. Available:

[35] M. Tong, C. Chu, and S. Chauhan, "High intensity ultrasound phased array for surgical applications," Tech. Rep. apply. 2006.
[36] V. Perrot, M. Polichetti, F. Varray, and D. Garcia, "So you think you can das? a viewpoint on delay-and-sum beamforming," *Ultrasonics*, vol. 111, p. 106 309, 2021, ISSN: 0041-624X. DOI: https://doi.org/10.1016/j.ultras.2020.106309.
[37] *Interpolation - Wikipedia*, Accessed: 10-7-2023.
[38] G. Matrone, A. S. Savoia, G. Caliano, and G. Magenes, "The delay multiply and sum beamforming algorithm in ultrasound b-mode medical imaging," *IEEE Transactions on Medical Imaging*, vol. 34, no. 4, pp. 940–949, 2015. DOI: 10.1109/TMI.2014.2371235.
[39] M. Mozaffarzadeh, A. Mahloojifar, M. Orooji, S. Adabi, and M. Nasiriavanaki, "Double-stage delay multiply and sum beamforming algorithm: Application to linear-array photoacoustic imaging," *IEEE Transactions on Biomedical Engineering*, vol. 65, no. 1, pp. 31–42, Jan. 2018. DOI: 10.1109/tbme.2017.2690959. [Online]. Available:
[40] J. F. Synnevag, A. Austeng, and S. Holm, "Adaptive beamforming applied to medical ultrasound imaging," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 54, no. 8, pp. 1606–1613, Aug. 2007. DOI: 10.1109/tuffc.2007.431.
[41] B. Luijten, R. Cohen, F. De Bruijn, H. Schmeitz, Y. Eldar, and R. Van Sloun, "Adaptive ultrasound beamforming using deep learning," IEEE, 1909.
[42] Y. C. Eldar, A. Nehorai, and P. S. L. Rosa, "A competitive mean-squared error approach to beamforming," *IEEE Transactions on Signal Processing*, vol. 55, no. 11, pp. 5143–5154, Nov. 2007. DOI: 10.1109/tsp.2007.897883.
[43] M. K. Jeong and S. J. Kwon, "A new method for assessing the performance of signal processing filters in suppressing the side lobe level," *Ultrasonography*, vol. 40, no. 2, pp. 289–300, Apr. 2021. DOI: 10.14366/usg.20032.
[44] A. Ibrahim, F. Angiolini, M. Arditi, J.-P. Thiran, and G. De Micheli, "Apodization scheme for hardware-efficient beamformer," in *2016 12th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, 2016, pp. 1–4. DOI: 10.1109/PRIME.2016.7519547.
[45] C. Seo and J. Yen, "Sidelobe suppression in ultrasound imaging using dual apodization with cross-correlation," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 55, no. 10, pp. 2198–2210, Oct. 2008. DOI: 10.1109/tuffc.919.
[46] J. Kirkhorn, *Introduction to iq demodulation of rf-data*, 1999. [Online]. Available:
[47] H. Zhou and Y.-f. Zheng, "Anefficient quadrature demodulator for medical ultrasound imaging," *Frontiers of Information Technology & Electronic Engineering*, vol. 16, no. 4, pp. 301–310, Apr. 2015. DOI: 10.1631/fitee.1400205.
[48] J. Chang, J. Yen, and K. Shung, "A novel envelope detector for high-frame rate, high-frequency ultrasound imaging," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 54, no. 9, pp. 1792–1801, Sep. 2007. DOI: 10.1109/tuffc.2007.463.
[49] Y. Lee, J. Kang, and Y. Yoo, "Automatic dynamic range adjustment for ultrasound b-mode imaging," *Ultrasonics*, vol. 56, pp. 435–443, Feb. 2015. DOI: 10.1016/j.ultras.2014.09.012.
[50] *What is an FPGA? Field Programmable Gate Array*, Accessed: 10-7-2023.
[51] U. Alqasemi, H. Li, A. Aguirre, and Q. Zhu, "Fpga-based reconfigurable processor for ultrafast interlaced ultrasound and photoacoustic imaging," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 59, no. 7, pp. 1344–1353, 2012. DOI: 10.1109/TUFFC.2012.2335.
[52] *Alveo U200 and U250 Data Center Accelerator Cards Data Sheet (DS962)*, Accessed: 10-7-2023.
[53] *AMBA AXI Protocol Specification*, Accessed: 10-7-2023.
[54] *Vitis Unified Software Platform Documentation: Application Acceleration Development (UG1393)*, Accessed: 10-7-2023.