MASTER

Ultrasound Beamforming on a FPGA

Bakthavatsalam, Yeshika

Award date:
2020

Technische Universiteit Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems Group

Ultrasound Beamforming on a FPGA

Master Thesis

Yeshika Bakthavatsalam

Supervisors and Committee Members:
Marc Geilen
Annamarie Stanton
Massimo Mischi
Emad Ibrahim

version 3

Eindhoven, October 2020

Abstract

Breast Cancer is the most common form of cancer in women, accounting for 25.1% of all cancers. Breast ultrasound is superior to other technologies in diagnosis due to quicker advancement in the underlying technology and higher precision in the resulting image.

To reconstruct the ultrasound image, beamforming is the technique used. Beamforming reconstructs the ultrasound signals generated from a common source that are received by multiple transducer elements of the Ultrasound array into the ultrasound image. This is done by phase alignment and summation of signals. Beamforming involves processing time delays of every element of the transducer array and applying a time delay to each element's received signal. There is a constant need to optimize the computation of time delays to improve efficiency in terms of resources, cost and area. Assuming the field of view to be taken as an $n \times m$ size image, every pixel of the image is considered a point source, thereby calculating the time delay for every element on the transducer array for each point source of the ultrasound image for multiple frames requires billions of calculations and is computationally intensive, this leads to issues related to memory, power and throughput.

To tackle the time delay calculations for every ultrasound signal, this thesis proposes using a FPGA. The FPGA has the ability to stream multiple channels of data and compute beamforming calculations in a parallel or a pipelined way. Also the FPGAs compactness, its high speed and ability to perform DSP operations make it an ideal candidate for this task. This thesis concentrates on understanding the complexity of ultrasound beamforming and using the CORDIC algorithm to optimize the use of hardware resources. It also focuses on constructing the hardware design and understand the requirements to compute the transmit and receive time delays using HDL coder and Xilinx System Generator while also performing High Level Synthesis for FPGAs. This research aims to pave the path towards scaling beamforming on a FPGA towards the application of compound imaging of breast tissues; a technique that uses electronic beam steering of a transducer array to acquire multiple overlapping scans of an object from different viewing angles.

Reconstructing the transmit delay and receive delay of ultrasound beamforming was performed using HDL coder. It was observed that the CORDIC equivalent of cos and sin (used in the transmit delay design) reduced the number of DSPs used for computation but increased the number of CLB LUTs and CLB registers. With area optimizations such as resource sharing, the CLB logic could be reused thereby decreasing the utilization of system logic cells on the whole. The square root operation (used in the receive delay design) exhausts FPGA resources, specifically the CLB LUTs. To solve this problem, the CORDIC equivalent square root operation was proposed. Comparing the MATLAB square root block and Xilinx System Generator's CORDIC 6.0 square root IP block ,the latter had a reduction in CLB LUT utilization. There was a trade off between the resolution of the ultrasound image and the number of receiving channels of the ultrasound probe. There was a trade off between the quality of the ultrasound image and the resource utilization of an FPGA. In conclusion, ultrasound beamforming is computationally intensive and the CORDIC equivalent along with design optimizations like area, speed and pipelining will yield in a reduction of FPGA resources.

This thesis consists of 10 chapters. The first chapter is an introduction on the process of ultrasound beamforming for breast tissue. The second chapter discusses the concepts and tools used during the course of this thesis. The third chapter provides a literature study on the complexity of ultrasound beamforming. The fourth chapter describes the problem statement and research

question. Chapters five, six and seven explore the implementation and results for beamforming using CORDIC equivalents for transmit and receive delay calculations. The eighth chapter discusses the trade offs with respect to resource utilization, cost and area of the FPGA. The ninth chapter answers the research question and the final tenth chapter concludes this thesis.

Contents

List of Figures

Chapter 1

Introduction

Breast cancer is the most common form of cancer in women. There is a never-ending need to improve technology for a more accurate diagnosis. Compared to MRI and X-Ray, breast Ultrasound is superior due to ultrasound signals being able to penetrate dense breast tissues [13]. A majority of the detected lesions are breast cysts; the nature of cysts is that they are circular and contain in them suspended particles that result in reduced resolution and speckle artefact while imaging. Poor resolution and speckle artefact can be tackled by compounding the RF signals coming from each element of the ultrasound. Compounding is an ultrasound technique that uses electronic beam steering of the transducer array to rapidly acquire several overlapping scans of an object from different view angles. Figure 1.1 shows a comparison of an ultrasound image of a breast cyst before and after application of compound imaging [31].

Beamforming requires the computation of time delays for each element of the transducer array for both transmitting and receiving signals. The output image is generated in the following way. A sound wave is first transmitted through the body by the transmitting transducers. The reflected echoes from the transmitted wave are then received by the receiving transducers. These signals are delayed and summed together by each of its elements to reconstruct the ultrasound image. This process is called **beamforming**. The most critical part of the beamforming is the delay calculation, which is the summation of the delay profiles along with the echo samples. The computation of transmit time delay requires millions of computations. Similarly, while performing beamforming, computing the receive time delays requires trillions of calculations [15]. These are computationally intensive and consume a lot of resources. Therefore, computing time delays for each element of the ultrasound array is the bottleneck of the ultrasound image reconstruction process.

The ultrasound reconstruction process produces an image composed of bright dots representing the ultrasound echos, known as a **B-mode ultrasound image**. A B-mode ultrasound image is generated with the help of a linear phased array. A linear phased array probe scans the object along the length of the probe to create a cross-sectional profile without moving the transducer. The transducer array collects the backscattered signals from each transmission, known as time-domain signals. The individual signals are added together into a single signal. This algorithm is called the **Delay and Sum algorithm** and is central to every image reconstruction algorithm for the ultrasound beamforming [28]. Figure 1.2 shows the Delay and Sum.

Beamforming runs into major computation and bandwidth challenges due to the large number of calculations that need to be performed for every signal [3]. Computation of time delays is the bottleneck of ultrasound beamforming and to tackle this, offloading the delay calculation to an FPGA's seems to be a more promising approach with regard to cost, power and flexibility in choosing the hardware blocks according to mathematical computations. Additionally, it is lightweight, portable and efficient due to parallelization and pipelining [15]. The time delays computed for beamforming are represented in a 2D/3D delay matrix depending on the array geometry. A time delay matrix is a representation of discrete-time delays applied to each element of the linear array for each pixel of the ultrasound image. In a 2D time-delay matrix $m \times n$, m

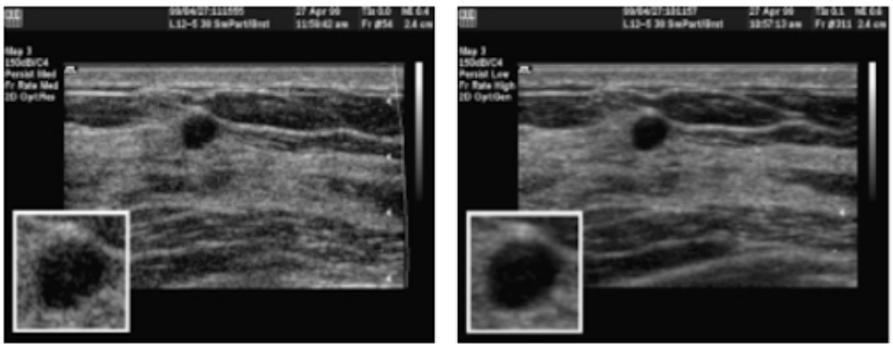

Figure 1.1: On the left is a normal B-mode ultrasound image of a breast cyst. On the right is the reconstruction of the same breast cyst after compounding where the speckle artefact is eliminated

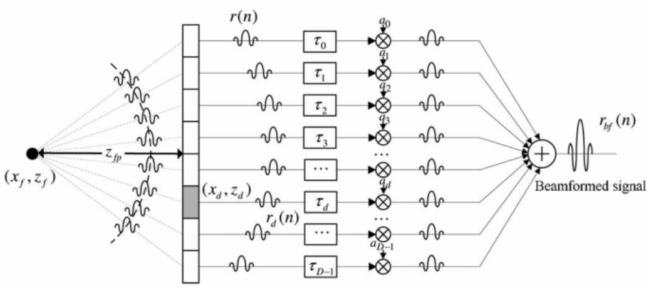

Figure 1.2: Delay and Sum [16]

represents the number of pixels, and n represents each element of the ultrasound array. Some of the work proposing delay calculations on an FPGA is explained in detail in the following literature survey in Chapter 3.

The research question that piques interest is: What is the effect of the CORDIC algorithm on the beamformed ultrasound signal and on an FPGA and what FPGAs can be proposed to perform time delay calculations?

Chapter 2

Background

2.1 Ultrasound Beamforming

This section describes the theory behind ultrasound beamforming. The MATLAB scripts and hardware design are based on the concepts and equations mentioned in this chapter. The type of beamforming performed here is using the delay and sum algorithm using a pixel-based beamforming approach [11] [20]. The incoming data is a $m \times n \times p$ matrix, where m is the number of samples, n is the number of receive elements and p is the number of transmitting elements. Channel data are beamformed directly into a grid of pixels. Figure 2.1 represents the pixel-based approach of beamforming where a 4x4 pixel image is converted into a grid format and each block represents a single source. There are 128 elements in the ultrasound probe, the ultrasound signal is transmitted by the first 21 elements. The signals after being reflected from the source are received by all 128 elements of the array. Beamformed signal is calculated for each pixel of the image for each element transmitting the ultrasound signal therefore there are 16×128 beamforming signals for a single element transmitting the ultrasound signal.

The equations for beamforming are represented below where the first equation is the beamformed signal for a single element transmitting the ultrasound signal and all 128 elements receiving the echo. S_t is the set of signals that create a low-quality image for each transmitting signal. Coordinates [x,z] represent the pixels and elements of the array, respectively. m is the total number of elements of the array (128 elements), y_m is the delayed signal from element m to M and w_m are predefined receive weights. Weights for beamforming are not addressed here as the focus of the thesis is the time delays.

$$S_t[x,z] = \sum_{m=0}^{M-1} w_m[x,z] y_m[x,z] \tag{2.1}$$

In the equation below, the images are coherently compounded into an image of higher quality. $S_t[x,z]$ represents the signals making up a lower quality image and $S[x,z]$ represents the final beamformed signal that is of a higher quality. T is the number of transmit signals (in this case 21) and w_t is predefined transmit weight.

$$S[x,z] = \sum_{t=0}^{T-1} w_t[x,z] S_t[x,z] \tag{2.2}$$

The total time delay i.e between the transmitting element to the point source and back to the receiving element, is given by the equation below where total time delay is represented by τ and the transmit and receive time delays are represented by τ_{tx} and τ_{rx} respectively. Time delays are subtracted from each of the $y_m[x,z]$ signal.

$$\tau = \tau_{tx} + \tau_{rx} \tag{2.3}$$

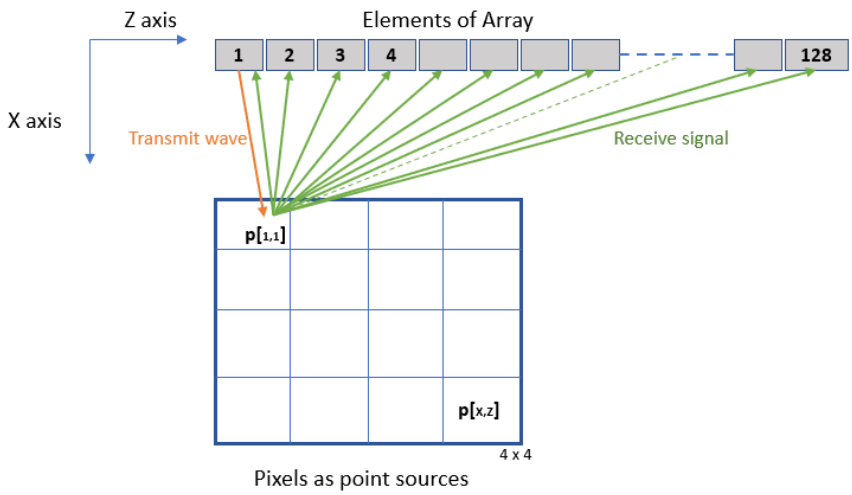

Figure 2.1: Pixels are considered as point sources and bearmorming is performed using a pixel-based approach

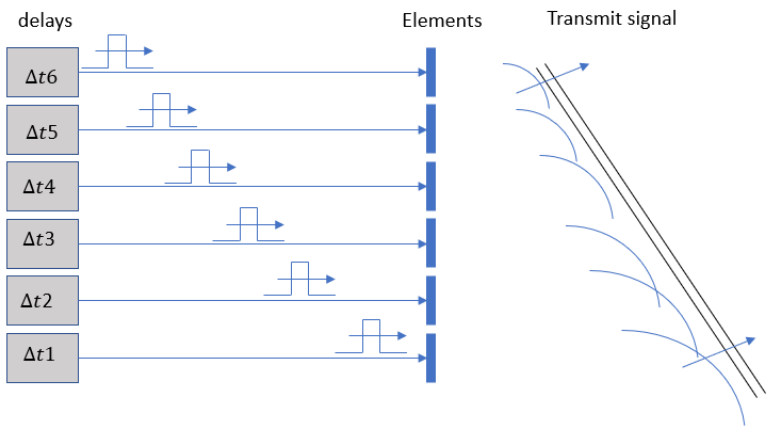

Figure 2.2: Time Delays for transmit signals [27]

2.1.1 Transmit Delay

In ultrasound arrays, the transducer is composed of piezoelectric elements where each element can be separately driven in the time domain to transmit a signal and the echo signal is received by each element of the same array. Each element of the array has a relative time delay of Δt_d from its subsequent element, this allows for ultrasound transmit beam to be steered by an angle. Electronically steering by applying time delays to each element is represented in Figure 2.2.

Transmit delay is calculated by the equation given below, where $\tau_{tx}[x, z]$ represents the transmit delay of a pixel with coordinates x and z where x is the pixel number in the x-axis and z is the element of the array that is transmitting the ultrasound signal in the z-axis. $scan_{zaxis}$ represents a vector containing values that of each pixel that is to be scanned along the z-axis the same goes for $scan_{xaxis}$ along the x-axis. T represents the number of transmission waves, (in this case 21), z represents the elements of the array which is also 21. Since the ultrasound probe consists of 1D elements, it has one angle of freedom, i.e the azimuth angle. However, the angle of elevation (which is zero in this case) is also considered here since this algorithm is flexible for a 2D array. According to the example considered, τ_{tx} is a 16×21 matrix array.

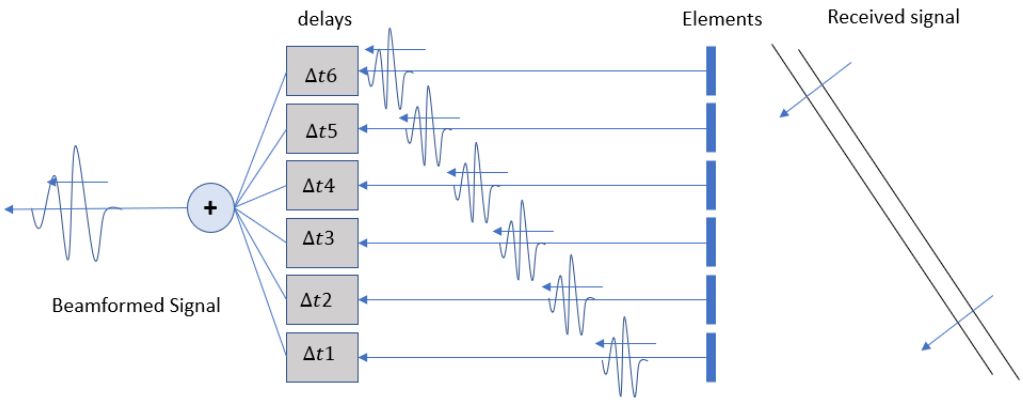

Figure 2.3: Apply receive delays and coherently sum signals to produce a beamformed signal [27]

$$\tau_{tx}[x,z] = \sum_{z=0}^{T-1} scan_{zaxis}.cos(azimuth_z).cos(elevation_z) \\ +scan_{xaxis}.sin(azimuth_z).cos(elevation_z) \tag{2.4}$$

2.1.2 Receive Delay

Relative time delays for each element can also be used to modify characteristics of receiving signals in the array. The figure 2.3 shows the echos of planar waves arriving at an array and this wave strikes each element of the array in succession and a series of electric pulses are generated. The relative time delays Δt_i are applied to these received signals so that all of the signals can be coherently added. Then these are summed together to produce a single large output signal.

The equation below represents the calculation of receive delay. Receive delay is the delay between pixel p and the receiving element x_j where c is the speed of sound. Receive delay is to be calculated for all pixels required to form the output image and all elements of the array that are receiving the signal. According to the example considered in this section, the receive delay would be a matrix of the size 16×128.

$$\tau_{rx} = \frac{|x_j - p|}{c} \tag{2.5}$$

The equation below represents a more detailed mathematical version of the equation above, where x_m represents the difference between the data at a pixel point [x] and the virtual source points represented as pixels in a grid, the same thing goes for z_m along the z-axis and y_m along the y axis. The equation below represents the receive delay for p pixels of the ultrasound image and M elements that are receiving the signal.

$$\tau_{rx} = \frac{\sqrt{x_m^2 + z_m^2 + y_m^2}}{c} \tag{2.6}$$

2.2 Quality Measurement

2.2.1 SNR

Signal to noise ratio compares the level of a desired signal to the level of background noise. In ultrasound imaging it is hard to differentiate between what is noise in an image and what is not. Also, there is no reference image that can be considered as noise as the ultrasound image changes

in every scan. In this case, the signal to noise ratio is defined as the ratio of mean and the standard deviation of an image, represented by the equation below where μ is the mean and σ is the standard deviation. Each pixel of the output image represents a complex valued beamformed signal.

$$SNR = \frac{\mu}{\sigma} \tag{2.7}$$

The mean μ is calculated as the mean of the absolute value of all pixels of the output image divided by the maximum absolute value of all pixels, represented in the equation below. Here, pixels is the vector containing all pixels of the ultrasound image, max is a MATLAB function used to calculate the maximum of a given matrix and abs is a MATLAB function that returns the absolute value of a matrix.

$$\mu = mean|\frac{pixels}{max(abs(pixels)}| \tag{2.8}$$

Subsequently, σ is the standard deviation of the absolute value or all pixels divided by the maximum absolute value of all pixels. It is calculated as below,

$$\sigma = std|\frac{pixels}{max(abs(pixels)}| \tag{2.9}$$

2.2.2 Contrast Ratio

To measure the quality of the image Ultrasound Toolbox presents a tool that can measure the contrast ratio between regions [23]. The contrast ratio (CR) is defined as,

$$CR = |\mu_{ROI} - \mu_B| \tag{2.10}$$

Where μ_{ROI} is the mean intensity value in dB from the region of interest and μ_B is the mean intensity value of the background region. Figure 2.4 is an example of the contrast ratio being calculated. Region 1 is the cyst and Region 2 is the background. The Contrast Ratio (CR) for this example is 28.7. The purpose of using CR as a quality measurement standard in this thesis is to understand the effect CORDIC and Fixed-point optimizations have on the output image. When the contrast ratio decreases then the ability to differentiate the cyst from its background decreases. In further chapters, it is observed that as optimizations increase, the quality of the image decreases i.e the cyst blends into the background and it gets harder to distinguish the boundary of the cyst from its background. This tool will enable one to find a limit as to how much optimization can be done.

With SNR and CR, the ultrasound image's quality can be determined. The highest quality image generated is considered to be the image generated using the floating point data type. The SNR and CR of the floating point data type is used as a reference for further experimentation with CORDIC and fixed point data type.

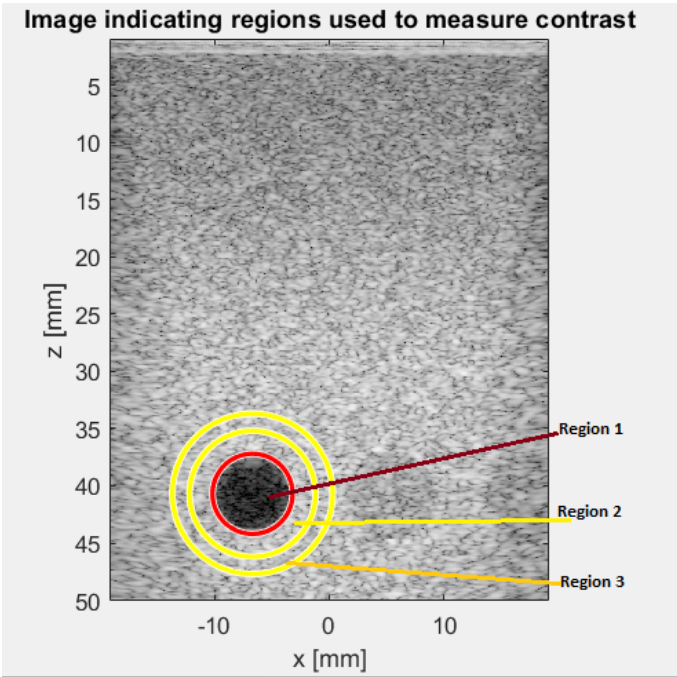

Figure 2.4: Example : Contrast Ratio is measured as the difference between mean intensity of region 1 (cyst) to it's background region 2,3

2.3 Matlab-UltraSound Toolbox

UltraSound Toolbox [25] is an add on toolbox for MATLAB [17]. Ultrasound toolbox presents the Ultrasound File Format(UFF) which is an HDF5 data format open to any programming language for storage of channel and beamformed data. The toolbox is made of both MATLAB and C++ code with GPU support. UFF and USTB (UltraSound Toolbox) provide a structure to process 2D and 3D ultrasound data.

A high-level description of the process of beamforming using Ultrasound Toolbox is represented in figure 2.5 Channel data in the form of a .uff extension can be adhered to any process/algorithm, in this case, the Delay and Sum beamforming[21]. Data can be simulated or read from a file (.uff extension), followed by defining a scan using an ultrasound probe configuration and then is further processed using the Delay and Sum algorithm. Transmit and receive apodization weights are calculated, however in this thesis weights is not of focus and hence default values are used. Apodization involves varying the amplitude across the aperture of the transducer such that elements of the center of the probe are excited with a higher voltage compared to the other elements on the side. Apodization weights are weights given to different elements of the probe that imitate this voltage change.

Transmit and receive delays are calculated using the equations mentioned in the previous section. Finally, the Delay and Sum algorithm is used to compute the beamformed signal. The beamformed data then stored back to the UFF file. The beamformed data can be represented as an ultrasound image of size $m \times n$ pixels, where each pixel represents a beamformed signal. The dataset used is that of a hypoechoic cyst i.e. the cyst is brighter than that of the background. The dataset used is recorded on an Alpinion scanner with a 128 element L3-8 (linear 3-8 Mhz probe) probe and stored in a .uff extension. The dataset consists of 21 plane waves that are being transmitted from the ultrasound probe and the echo is received by all 128 elements of the probe. This dataset is an ideal candidate for the application of compound imaging.

The Data structure of the combination of UFF-USTB is represented in figure 2.6 and figure 2.7. Figure 2.7 represents the `uff` data structure in its simplest form. The `uff` class has several

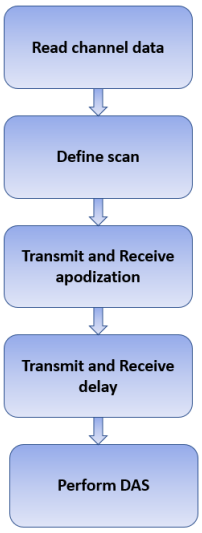

Figure 2.5: Beamforming using Ultrasound Toolbox and Uff data structure

subclasses like that of channel data, scan, beamformed data etc, each of which has several objects. The channel data class has objects like sampling frequency of the probe, probe geometry (element spacing) and holds the data (3 dimensional). objects can be individually accessed and worked on. The scan consists of the geometry of the grid that represents the pixels. In the dataset considered the output image is a 512×512 image therefore the data structure `uff.scan` holds the geometry of each pixel on the x-axis and the z-axis. The class called probe contains the geometry of the probe i.e the placement of the elements of the array and the spacing between them. The class called point contains the focus point of the source. In this implementation, since we consider plane wave imaging, the point sources are set to infinity. The class called wave consists of a sequence of transmit waves, this class has several objects that describe the transmission of the ultrasound wave such as transmit delay and the geometry of the elements that are transmitting. The beamformed data consists of the beamformed signals. [1]. The resulting beamformed data is a M x N image where each pixel is represented by a complex-valued beamformed signal.

The reason behind choosing this dataset is that the hypoechoic data is collected using an actual ultrasound probe and stored in the `uff` data structure. This enables one to expect the intermediate and final results to match the results of an actual ultrasound scan. The beamformed signal and the output image are used as a reference for when further experimentation is done. The beamformed signal is used as a reference floating-point signal to compare the difference in results when beamforming is performed using the CORDIC algorithm and fixed-point notation. The beamformed image is used as a reference image so that the effect of optimization techniques can be observed (for SNR and CR).

[1] https://www.ustb.no/ius2017-abstract

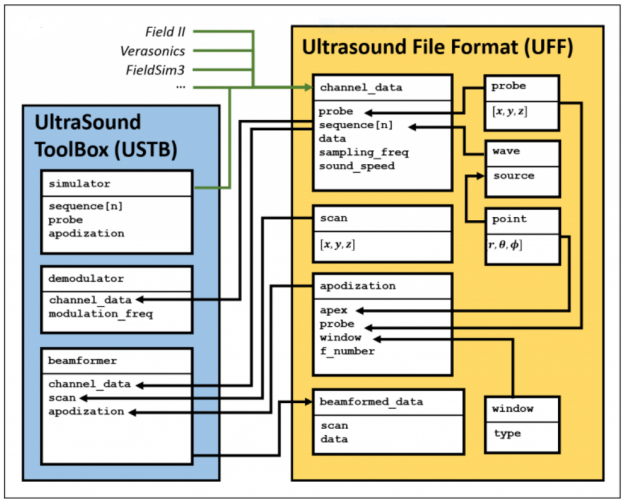

Figure 2.6: Data structure in UFF and USTB

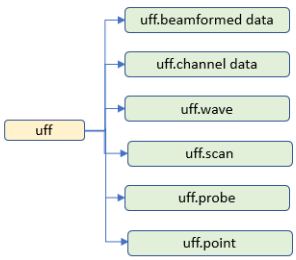

Figure 2.7: Uff data structure

2.4 CORDIC Algorithm

CORDIC is an iterative algorithm used to compute trigonometric, square root and division functions for DSP applications. It was first introduced in 1959 by Volder [30] and over the past years, there have been significant improvements and variants that improve the speed and basic design. The variations all aim to reduce the resources used for computation, increase the speed, reduce the complexity of hardware and to reduce the number of iterations [22]. The algorithm of CORDIC proposed by Volder is derived from the two rotation equations,

$$x' = xcos\phi - ysin\phi \tag{2.11}$$

$$y' = ycos\phi + xsin\phi \tag{2.12}$$

The equations represent a vector (x,y) that is rotated by an angle of ϕ. This is modified further to replace the multiplication operation by iterative shift operations. The modified equation is given by,

$$x_{i+1} = x_i - y_i.m.d_i.2^{-i} \tag{2.13}$$

$$y_{i+1} = y_i - x_i.m.d_i.2^{-i} \tag{2.14}$$

$$z_{i+1} = z_i - d_i.tan^{-1}(2^{-i}) \tag{2.15}$$

where m = -1,0,1 and represents the rotation in the specific coordinate system (m = 0 for linear, m = 1 for circular, m = -1 for hyperbolic) and d_i = +1 if the vector has to be rotated counterclockwise (if $z_i > 0$) and d_i = -1 for clockwise rotation (if $z_i < 0$) This forms the basic equation to compute various mathematical functions using CORDIC.

The CORDIC algorithmic rule performs the rotation of a vector in a sequence of micro-rotations that aim to converge to the required angle that the vector is subjected to rotate at. The arithmetic rule reduces the multiplication and simplify overall hardware requirements. In general, a single multiplication is mapped to a single DSP in a FPGA, hence reducing/eliminating the multiplication operation (as done in CORDIC) reduces the number of DSP slices that are required to perform beamforming on an FPGA. The CORDIC algorithm is chosen here as an optimization to reduce the multiplications required to calculate the cos, sin and square root operations while computing the transmit and receive delay. The basic hardware required to compute a result using CORDIC is shift registers, lookup table, adder/subtractors. The adder/subtractor perform the addition and subtraction of binary numbers. The register performs the bit-shift operation following the algorithmic rule. The constants equivalent to angle values are obtained from the look-up table. Since CORDIC is an iterative algorithm, increasing the number of iterations can produce more accurate results but doing so also increases the expense of the computation i.e number of adder/subtractor blocks and shift registers. An increase in iteration also adds latency. To solve the issue of latency, pipelining can be adopted.

The CORDIC architecture consists of bit shift registers and adder/subtractors and can extend its architecture to perform CORDIC iterations in a parallel and pipelined way. As CORDIC iterations are identical it is convenient to map them to pipelined architectures. When pipelined CORDIC circuits are used, high throughput is achieved. For an N bit, CORDIC input N stage pipeline can be obtained for N cycles. In future improvements and optimizations, CORDIC pipelining can be explored to increase efficiency concerning throughput and speed.

2.4.1 MATLAB CORDIC

Matlab offers a wide range of mathematical operations like cos, sin, square root, tan etc to be computed using the CORDIC algorithm. A high-level understanding of MATLAB CORDIC (the cordic functions offered by MATLAB) is discussed here as this is sufficient to understand the optimizations it presents for hardware resources.

CORDICOS and CORDICSIN

In the equations mentioned below, the MATLAB function used to compute the CORDIC equivalent value. Theta can be a signed or unsigned scalar, vector or a matrix. The values of theta can be real and in the range [–2π 2π). niters is the number of iterations the CORDIC algorithm computes for. niters is a positive scalar value, for fixed-point operation, the maximum number of iteration is one less than the word length (WL-1) of theta and for floating-point operation, the maximum value is 52 (when theta is of a double data type) or 23 (when theta is of a single datatype). The result is stored in y i.e y is the CORDIC-based approximation of the cosine and sine of theta. When the input (theta) is of fixed-point type, the output has the same word length(WL) and the fractional length is equal to word length - 2 (FL-2).

$$y = cordiccos(theta, niters) \tag{2.16}$$

$$y = cordicsin(theta, niters) \tag{2.17}$$

CORDICSQRT

The CORDIC algorithm is also used to estimate the square root of a value. The equation below represents the MATLAB CORDIC square root function where the input u is a positive scalar, vector or a matrix of fixed-point notation. The input should ideally have a value within the range of 0.5 to 2 for the algorithm to be most accurate. In the case where the input is outside this range, the `cordicsqrt` applies to pre and post normalization process to give accurate results. The number of iterations can be specified in the place niters, this must be a positive scalar value. The number of iterations should be a maximum of Wordlength - 1. The output y has the fixed point notation like that of the input u.

$$y = cordicsqrt(u, niters) \tag{2.18}$$

In subsequent chapters, the effect of the fixed point notation of theta, u and the number of iterations on the beamformed signal and output image are discussed. The objective would be to find the word length, fractional length and number of iterations that will yield an ultrasound image with an acceptable resolution and SNR. For reference, the floating-point cos, sin and square root are considered. The CR and SNR calculated using the floating-point cos sin, square root and are considered ideal values and used as a reference.

2.5 Simulink and Fixed-point designer

Simulink

Simulink is a tool provided by MATLAB that can build and test simulations and create a hardware design using a Model-Based approach. It supports system-level design, simulation and automatic code generation. Simulink can read and store data from and to the MATLAB workspace. Simulink has a graphical editor, block libraries (here the 'block' is the Simulink subsystem that contains the hardware design that is to be synthesized.) supported by DSP toolbox and is integrated with MATLAB such that MATLAB algorithms can be incorporated into models and simulation results can be exported for further analysis. Here Simulink plays an important role in creating a hardware design for the transmit and receive delay. Simulink paves a way to work with HDL coder for FPGA synthesis and VHDL code generation.

Fixed point designer

Digital hardware can be represented in fixed-point or floating-point data types. The dynamic range of floating-point is more than that of fixed point and for very small numbers (the decimal point at 10^{-7}) floating-point yields better results than that of the fixed point. However, fixed-point is still used for hardware design. The reason being :

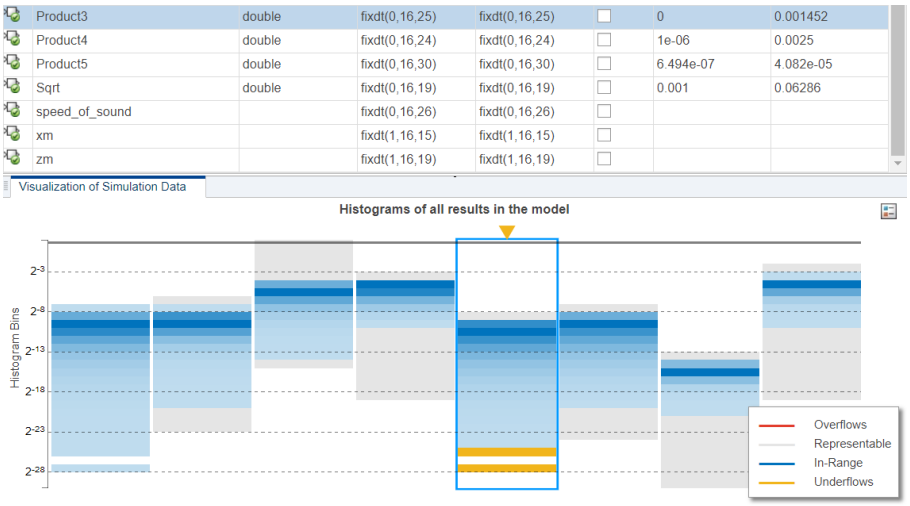

Figure 2.8: Example : Dynamic range of fixed-point designer tool

- Size and Power Consumption: the complexity of logic circuits with fixed-point is lesser than that of floating-point. The chip size and power consumption are lesser compared to floating-point. For applications that require small and portable devices, fixed-point is a better choice.
- Memory Usage and Speed: In general fixed-point calculations require less memory and less processor time to perform as compared to floating-point.
- Fixed-point hardware is more cost-effective than floating-point hardware and thus can result in significant savings

In this thesis fixed-point designer tool by MATLAB is used to convert the floating-point design to fixed-point design. Fixed-point designer provides data types and tools for optimizing and implementing fixed-point design for beamforming. Fixed point designer collects a set of ranges and suggests fixed point notations accordingly such that overflow or underflow do not happen. Figure 2.8 is an example that shows how fixed point designer tool works. Simulink hardware blocks like product4, product5 give results that do not cause overflows or underflows, the same is not for the case of product3. Overflow and underflow both cause errors in the result and therefore affect the beamformed signal and reduce the quality of output image. Therefore the required fixed-point numerical accuracy can be set using fixed point designer according to the requirement of the hardware before actually implementing this on the hardware. A good analysis on fixed point fraction length and optimizing the fractional length keeping in mind not to have overflow and underflow is performed here, these values of fixed-point are then chosen and can be implemented in hardware. Production of C and HDL code can be generated from fixed-point models.

2.6 HDL Coder

HDL coder is a high-level synthesis tool that allows for creating a hardware design of DSP algorithms and performing FPGA synthesis of it. FPGA synthesis results in the code generation of VHDL and provides resource and timing summaries for the hardware design. It offers high-level design blocks, MATLAB functions that can be used to perform simulation and generate synthesizable VHDL or Verilog source code. In this thesis HDL coder is used to reconstruct the transmit and receive delay that is computed during the beamforming process. The bottlenecks are observed and optimizations are performed. A block diagram of the working of HDL coder is shown in figure 2.9 [2]. MATLAB stores the input vectors in the workspace. Simulink is used to design the hardware

[2] https://www.bdti.com/InsideDSP/2012/09/05/MathWorks

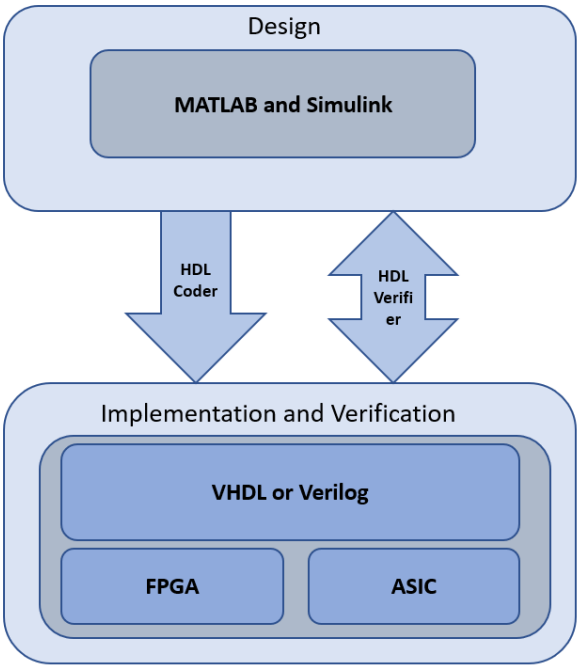

Figure 2.9: HDL coder high level description

required to perform mathematical computations for transmit and receive delay. The HDL coder tool then converts the hardware design to VHDL or Verilog code that is targeted to an FPGA or ASIC device. The HDL tool also offers to create a test bench for verification purposes such that the hardware design can be checked for correctness.

Figure 2.10 represents a more detailed depiction HDL Code generation.

HDL coder properties can be configured, here the target FPGA or ASIC device can be set, optimizations can be explored, the type of report one wants to generate can be set, for verification purposes, test bench configurations can be set and in general global settings can be found here. Next, block properties can be set. Here pipelining options like adaptive pipelining, input and output pipelining and clock rate pipelining can be explored. Input streaming and resource sharing factors can also be set to appropriate values. These configurations mentioned here are discussed in more details in further chapters.

Finally, HDL workflow advisor can be used to perform synthesis and generate VHDL code. The workflow advisor performs several tests and checks compatibility and configurations during these checks. Once all checks are passed then synthesis and code generation is performed. Once the target device is set and HDL mode is set then the HDL code is generated and a report is produced. In this report, a detailed description of hardware resources used to perform the whole computation is generated i.e the number of multipliers, adder/subtractor, one-bit registers, memory is all presented. As and when optimization techniques are included in the hardware design, the resource utilization reduces however this is not always the case, experimentation is done and its results are discussed in detail in further chapters. The final step is FPGA synthesis where the source code is generated and the utilization of resources for the target device are tabulated in the form of a resource summary; here, the number of Lookup tables, DSP slices, RAM are presented. Also, a timing summary is displayed where the required timing, data path delay and slack time are all presented. Changes in hardware design and optimizations with design all yield different timing and resource summaries.

The use of HDL coder her threefold. First, to understand the hardware resources required to compute parts of beamforming. Second to generate VHDL or Verilog source that can be

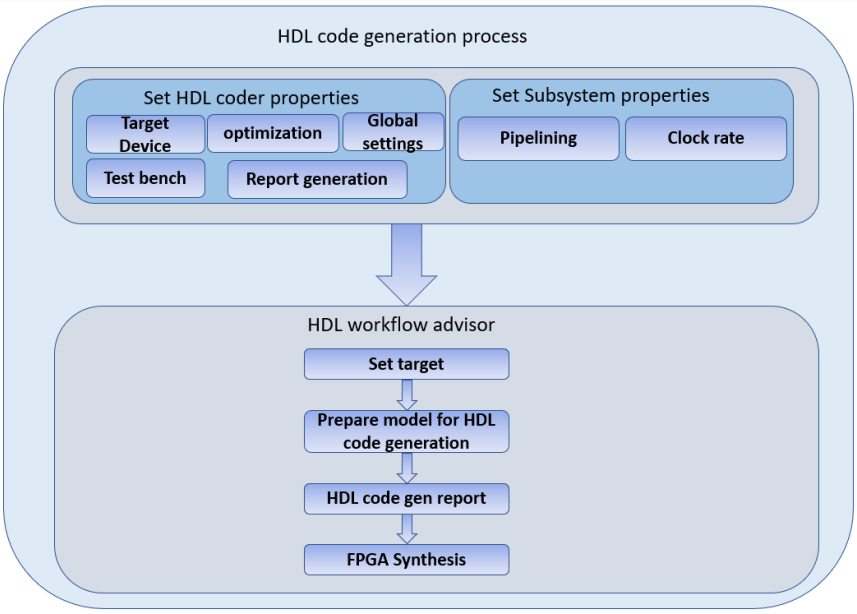

Figure 2.10: HDL code generation in detail

used to work on the FPGA (also hardware in the loop can be performed). Thirdly, the effect of optimizations like pipelining, resource sharing, using CORDIC and using a fixed-point notation can be explored such that the best result can be implemented on an FPGA. The HDL coder is a great tool to map DSP algorithms to hardware devices and perform synthesis.

2.7 Xilinx System Generator

System Generator for DSP is the industry's architecture level design tool to define, test and implement high-performance DSP algorithms on Xilinx devices. This is designed as an add-on toolbox for MathWorks Simulink [7]. SysGen is a tool provided by Xilinx's Vivado. System generator provides IP that enables us to create RTL code for the hardware design of beamforming. It also allows for flexibility in working with hardware blocks, like implementing optimization techniques for conserving hardware resources like LUTs, memory, shift registers etc. Combined with simulation and verification, the quality of the algorithms can also be easily assessed. The RTL code and functions generated can easily be transferred onto one of the Xilinx boards that are compatible with Vivado.

Xilinx system generator allows for building and debugging of high-performance systems, using Xilinx optimized RTL IPs as blocks within Simulink for signal processing (e.g., FIR filters, FFTs) one can define memory blocks (FIFO, RAM, ROM), digital logic and arithmetic logic. The system generator also supports bit and cycle-accurate fixed-point, double, and custom precision floating-point. This is useful to compare the precision for transmit and receive delays.

RTL generation and target specific Xilinx IP cores from the Xilinx Blockset. For further research and implementation, one can generate the RTL code for the beamforming algorithm used, thereby having the ability to transfer and implement beamforming onto a Xilinx board. Several devices are supported to work with Xilinx system generator, these can be found in the System Generator for DSP website[3]. Among these, there is one in particular which is useful and efficient for Ultrasound applications, which is the Kintex UltraScale series. We target this board in specific

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

because of its current availability and also because it can support as many as 32×32 receive channels, which sufficient to stream data in.

Chapter 3

State of the art and literature study

A key bottleneck of delay calculations in the receive beamforming is investigated in A. Ibrahim et al. [14] where the computation of fine-grained time delays among the digitized receive echos are released by implementing smarter time delay calculation and present their feasibility analysis on an FPGA platform. This paper presents studies that show the existing challenge faced by time delay computation in ultrasound beamforming and how to use an FPGA to tackle this. The smart time delay calculations perform beamforming on FPGA by computing a reference lookup table for a single point on the Z-axis and calculate the time delay for subsequent points steered from this reference point and proposing an FPGA hardware logic of two additions and the square root operation. It also states that a typical 2D system requires a pre-computed delay LUT of a few million coefficients but a 3D system ranges beyond billion coefficient, therefore, implying problems arising in storage. Results show that it is possible to perform 3D delay calculations on a Virtex FPGA by tuning fixed-point representations. Like that of this paper, the computationally intensive transmit delay and receive delay were identified. For future improvements, this idea can be adopted for the calculation of square root.

R. Sampson et al. [26] tackle the key challenge of large 3D ultrasound systems. They need to compute over 100 billion round trip delays to convert the RF channel data into beamformed images. They do this by introducing an algorithm that computes on the fly delays as opposed to the standard 2D approach of pre-computing delays and storing them in LUTs (which require large storage space). What piques interest for us is the flow of data generated by Field II simulation and hardware modelling done on MATLAB, showing that on the fly computation of delays is a possibility to perform ultrasound beamforming. This algorithm implemented on hardware by prototyping a single channel of beamformer design on an Altera DE1 FPGA board. The proposed work claims to have a higher speed, low power consumption up and better image quality. The idea of utilizing MATLAB HDL coder for synthesis was stemmed from this research. However, during the course of this thesis it seemed that on the fly computation was not a practical approach to calculate the receive delay, however for the transmit delay it is possible.

A.Ibrahim et al. [15] describe an imager that tackles the bottleneck of a large number of computational time-delay calculations by providing a highly scalable architecture. The architecture is down-scaled for 2D imaging or further up-scaled for 3D ultrasound imaging. The platform supports both real-time inputs over an optical cable and test data feed sent by a laptop running Matlab. This work presents a complete fully digital imaging system capable of both 2D and 3D reconstruction capable of utilizing up to 1024 transducer channels in a single Kintex UltraScale KU040 FPGA. Thus, this work justifies why an FPGA performs efficiently and why the optimization of beamforming of Ultrasound signals is feasible, thus in this thesis, the FPGA was chosen

as the hardware used to offload the calculations on to.

An extension of the said work is scaled down to consume lower power; the proposed parallel architecture uses 32×32 delays that pre-calculated and summed thus utilizing only 32x32 channels probe for reconstruction [3]. In this thesis, 32 receiving elements was also chosen since having 128 receive elements was computationally intensive.

Other methods to tackle the time delay computation problem is to have more efficient algorithms, for example, *I. K. Holfort et al.* [12] investigates a minimum variance (MV) approach for nearfield beamforming of broadband. They compare the performance of MV beamformer with the traditional Delay and Sum beamforming. Results indicated that they outperformed the traditional beamformer in terms of SNR. Another algorithm prominent in the study of breast tissues based off of Delay and Sum algorithm would be delay multiply and sum algorithm (DMAS) as mentioned by *G. Matrone et al*[18] that also outperforms the traditional DAS algorithm in contrast resolution. In DMAS a single scan line is considered just like in DAS and each element transducer receives the echo and DAS is applied, once the signals are in phase they are combinatorially coupled and multiplied. It would be interesting to note what effect these algorithms have on the hardware design of the FPGA beamformer i.e. how would one design the hardware when using superior algorithms containing multiple multiplications, squaring and summing operations. For future high quality imaging, the DMAS algorithm can be adopted instead of DAS. In this thesis the DAS algorithm was chosen as it is simpler to reconstruct the hardware design.

J. Chen et al. [5] look into the design considerations of real-time adaptive beamforming on FPGA and GPU. Their algorithm, MV adaptive beamforming, is more computationally demanding than delay and sum beamformers because it computes weights in real-time with the response from input data providing a throughput of 20 fps-100 fps. This work provides proof that Field 2 simulated data can be used to perform beamforming on Xilinx and that hardware designs can be analysed using Xilinx System Generator. Field II simulation of a cyst is used to simulate the data and the beamforming design uses Simulink and Xilinx System Generator. Ideally, the input channel data is taken from Matlab workspace (field II) and goes into FPGA MV beamformer and the output is sent back to Matlab workspace, which is similar to the way the data flow in the hardware designs of this thesis work. This proposed work uses MV adaptive beamforming which promises speedup but does not see an improvement in image quality primarily due to numerical errors generated from a different sequence of floating-point operations. In this thesis, the data set used is also simulated from Field 2. The decision of using fixed-point design instead of floating-point design was due to the poor results obtained using a floating-point design in this research paper. Floating-point design used too many resources and slowed down processing.

M. Almekkawy et al. [2] presents a resource-optimized dynamic digital beamformer for an ultrasound system based on FPGA. This study uses a 64 channel receive-beamformer embedded into an Altera Arria V FPGA chip. This work provides sufficient knowledge on the hardware resources used for Altera Arria V FPGA, thus enabling the choice of hardware to be Xilinx Ultrascale for our work. The reason for not choosing Altera Arria V FPGA is because Xilinx Ultrascale series is superior in several aspects [24], especially the utilization of LUT .

M. A. Hassan et al. [9], is an approach that is similar to this thesis. Here, a system with two 8-channel blocks is reconstructed by a single block. The beamformer is generated using Xilinx System Generator and MATLAB Simulink; the RF data is saved in Matlab workspace and used by the Xilinx block digital beamforming is implemented in Virtex-5 FPGA. In this thesis it was identified that the Ultrascale Virtex FPGA series was a high end FPGA and was more expensive. To reconstruct the receive delay for a 64 channel beamformer, this FPGA can be chosen.

Other hardware FPGA designs employing a larger number of channels for highly scalable applications like sonar also exist that use the same concept of advance beamforming and FPGA computation. One such example is the design in *H. J. Hewener et al.* [10] that uses 128 channel

beamformer using MicroBlaze in the front end and stores time delays in external DDR3 memory with a throughput of 30fps. Although this method is ultrafast, it is extremely bulky and not intended for the application presented in this thesis. A similar concept is presented in A. A. Assef et al. [4] that uses a simple delay and sum algorithm. During the course of the thesis, it was identified that for high quality applications that involve the use of all 128 elements of the ultrasound array (i.e 128 receive channels), an external memory can be used. This external memory can serve as a storage for pre calculated square root values in the form of LUTs, this way the square root operation used in the receive delay calculation can be eliminated.

Chapter 4

Problem Statement

Breast Ultrasound is superior to other forms of breast cancer diagnostic tools due to its ease of use, cost and rapid advancement in its technology. One of the major challenges that occur in the diagnosis of a breast cyst is the circular shape of the cyst leading to speckle artefact and reverberations within the cyst. To solve the issues due to the speckle artefact, the coherent wave compounding is adapted.

The technological challenge that persists to this day is the computation of time delays during beamforming. The most critical part of beamforming is the time delay calculations as the delay profiles of each element of the ultrasound array along with the echo samples are to be summed up to provide a single signal that represents a pixel of the entire ultrasound image. The delay is calculated as the propagation path from the transmit element to the focus point and back to the receiving element, this delay calculation involves arithmetic logic such as square root, divisions, and summations. Computation of transmit time delay and receive delay requires trillions of calculations which is highly computationally intensive for the Ultrasound probe and system. Thus computing beamforming time delays for each element of the array is the bottleneck of the ultrasound image reconstruction process. To analyse hardware design, pipelining and exploit parallelism, the use of an FPGA to perform this time delay computation is proposed. By this, an understanding of resources, memory and area used, are explored.

Principal Research Question: How to optimize the time delay calculations of ultrasound beamforming and what FPGAs can be proposed to perform time delay calculations?

The reconstruction of ultrasound signals starts with a basic algorithm called the delay and sum algorithm which is to be reconstructed on hardware via HDL coder for Matlab Simulink. By creating the beamformer with the help of HDL Coder, one can understand the underlying requirements of hardware that are needed to perform arithmetic operations that compute the time delay. Further, trigonometric and square root operations are replaced by its CORDIC variants and its effects on the ultrasound image, and FPGA resource utilization is determined. This can further be mapped using performance measuring tools to see how resources are utilized and react when scaling up the application.

SUB Research Question 1: How to optimize the time delay calculations of ultrasound beamforming using the CORDIC algorithm ?

CORDIC variants of cos, sin and the square root is used in the calculation of transmit delay and receive delay, the effects of this on the SNR and Contrast ratio of the resulting ultrasound

image are observed. Further, the hardware design is reconstructed and FPGA synthesis is performed, this gives an idea on resource utilization and timing analysis.

SUB Research Question 2: What FPGAs can be proposed to compute transmit and receive time delays for ultrasound beamforming?

Trade-offs observed during hardware design are discussed and a suggestion on optimal hardware design is given. Further, suggestions on Xilinx FPGAs that are best suited to calculate the receive and transmit delay are provided.

Chapter 5

Beamforming and CORDIC Implementation

5.1 Complexity of beamforming

Several articles mentioned in chapter 3 discuss the complexity of beamforming in detail, many of which focus on the computation of time delays which is the bottleneck of Ultrasound beamforming. Complexity is due to the large number of calculations that have to be performed to compute receive delay, transmit delay and beamforming in general. This section explores the number of mathematical operations required to compute receive delay, transmit delay and beamforming for the data set considered. As mentioned in chapter 2, the dataset is of a uff format; The dataset used is that of a hypoechoic cyst i.e. the cyst is brighter than that of the background. The dataset used is recorded on an Alpinion scanner with a 128 element L3-8 probe. The dataset consists of 21 plane waves that are being transmitted from the ultrasound probe and the echo is received by all 128 elements of the probe. Based on the equations discussed in chapter 2 the number of additions, subtractions, multiplications, division and the square root is calculated. The resulting image is a 512×512 image that is obtained from compounding 21 images. The first image is obtained when the first element is transmitting a planar wave and all 128 elements are receiving the echo, similarly, the second, third ... up to twenty-one. These images are compounded to give one resulting 512×512 image that is used in further chapters.

Receive delay is calculated as the square root of the distance between the element to the source (one pixel at a time) and back to the element, as represented in equation 2.6. Therefore the receive delay is a 2D matrix of size 262144×128 (the output image has 512×512 pixels therefore 262144 point sources). Transmit delay (represented in equation 2.4) is calculated for 21 transmitting elements therefore, the size of the 2D matrix is 262144×21.

Beamformed signal is a vector containing 262144×1 elements of complex-valued numbers where each element represents the beamformed signal of one pixel in the output ultrasound image (represented in equation 2.1). The figure 5.1 below describes the number of operations it takes to calculate the transmit delay, receive delay and beamformed signal for the given dataset.

Name	Size	Bytes	Data type	Add	Sub	Mul	Sq-root	Div
Receive Delay	262144x128	134217728	single	67.108.864	100.663.296	100.663.296	33.554.432	33.554.432
Transmit Delay	262144x128	134217728	single	524.288	5.505.024	27.525.120	0	5.505.024
Beamformed signal	262144x128	134217728	single	2.688	5.505.024	1.409.286.144	0	0

Figure 5.1: Number of operations it takes to compute the receive delay, transmit delay and beamformed signal matrices

5.2 Minor Optimizations

On observing the algorithm closely there are a few redundancies that can be eliminated. First of which is that there is no y coordinate as the ultrasound probe has only two degrees of freedom, the x-axis and z-axis. The receive delay equation originally calculates the receive delay for all three coordinates assuming the elements on the y-axis to be zero, as represented in the equation below,

$$\tau_{rx} = \frac{\sqrt{x_m^2 + z_m^2 + y_m^2}}{c} \tag{5.1}$$

This can be replaced with the following equation, thereby eliminating the need to compute receive delay along y axis

$$\tau_{rx} = \frac{\sqrt{x_m^2 + z_m^2}}{c} \tag{5.2}$$

Secondly, division by a constant can be eliminated by converting it to a multiplication of the inverse of the element. Speed of sound is the divisor in several computations, this is converted to multiplication with the inverse of the value of the speed of sound.
Thirdly, In this application involving breast tissues, transmit apodization is a 262144 X 21 matrix containing all 1's. It is observed that each element in this transmit apodization matrix is multiplied to the 262144×128 receive apodization to give the final apodization matrix. This is redundancy since any number multiplied with 1 gives the number itself. Another reason behind setting the default value of the transmit apodization as 1 is that for the application of breast cysts, since they are superficial, a transmit apodization is not required and can be eliminated [19]. With these minor changes in computation, the number of operations can be reduced drastically. As represented in figure 5.2, the bottleneck is the multiplication operation and the square root operation. As addition and subtraction take fewer clock cycles (around 1-2 clock cycles) to compute and multiplications (typically 6 clock cycles) and square root take many more (around 25 clock cycles), depending on the hardware. Now that the bottleneck of beamforming is realized which is the number of multiplications and the number of square root operations, the aim is to try and find a solution.

Trigonometric operations sin and cos are involved in the calculation of transmit delay, these add to latency and utilize many multipliers. The sine and cosine of a number is calculated using taylor series. Taylor series is an approximation based function where the sin or cosine is calculated using an infinite series of polynomials involving a lot of multiplications and divisions. On the whole, computing the sin or cos of a number takes hundreds of clock cycles (depending on the precision). For compound imaging, the angle of incidence constantly changes every scan and this requires a lot of calculation of trigonometric functions. Therefore, trigonometric functions are also to be optimized.

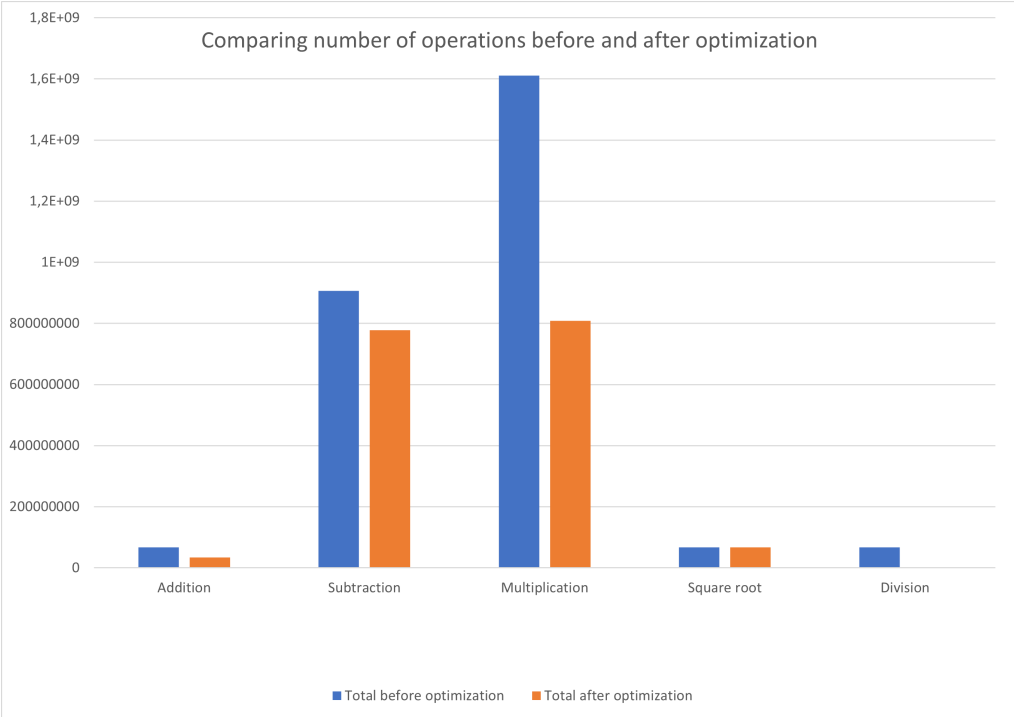

Figure 5.2: Number of operations

5.3 Operations and their effect on FPGA

FPGA's come in different sizes and speeds, dedicating parts of the logic to specific tasks is up to the user. FPGA's have DSP slices that each have a multiplier, accumulator, pre-adder, SIMD, pattern-detector and logic unit. The multiplier, accumulator each take in different word lengths, depending on the FPGA, for example, the Virtex-7 series has a 25*18 bit multiplier and 48-bit accumulator and the number of DSP slices vary between different boards. Larger devices (example: Xilinx spartan-6 series) have would have 1000 or more of these multiplier blocks that are used to perform mathematical computations with speed. It is also possible to decompose large multiplication into smaller operations using several DSP's adders to sum the partial product. In general, it would take 3 or 4 clock cycles to get the output for a single-precision multiplication (this is the latency). In a fully pipelined version, the throughput increases and would be 1 result per clock cycle. Similarly, for division, the latency is 8 cycles but throughput for a fully pipelined architecture is 1 result per cycle (for Xilinx spartan-6 series) In general, there would always be trade-offs between size, speed and cost. For example for speed up, adding multipliers does increase the number of outputs per cycle but at the cost of the area since the logic size increases.

Therefore, It is hard to estimate the effect of these mathematical operations on the hardware at this stage, which is why a hardware design using HDL coder is explored to gain a better understanding on the trade-offs. The table below gives aims to understand the relationship between FPGA resources, FPGA device package dimension and it's cost. It can be observed that when the number of resources increases, the packaging area and its cost also increase. Therefore after performing FPGA synthesis and appropriate FPGA can be chosen for the application of Ultrasound beamforming. Many ultrasound applications for ultrasound beamforming for breast cysts use Xilinx Kintex Ultrascale or Xilinx Virtex Ultrascale series, in this thesis, the Ultrascale series is chosen to perform FPGA synthesis.

Kintex Ultrascale	Logic cells	LUTs	DSPs	Size (mm)	Cost (USD)
KU025	318,150	145,440	1,152	35 x 35	1,052
KU040	530,250	242,400	1,920	35 x 35	2,193
KU085	1,088,325	497,520	4,100	40 x 40	4,876
KU115	1,451,100	663,360	5,520	40 x 40	7,601

5.4 CORDIC algorithm

The previous section identifies multiplications and square root operations as the bottleneck of the computing transmit delay, receive delay and beamforming. CORDIC algorithm is used to reduce the multiplications while computing the cos, sin and square root. It has the potential for efficient and low-cost implementation towards multiple DSP applications. Many operations can be computed using CORDIC, including several trigonometric, hyperbolic, division and square root operations and hence this proof of concept can be used to further improve the whole ultrasound beamforming algorithm. Using CORDIC has a few disadvantages; the first is its speed; since it takes many iterations to converge to the required precision. Second is the area consumption since it requires many variable shifters and ROM's to store angle constants. However, there have been many improvements in the past years to overcome these two disadvantages [8]. A common way to overcome latency is pipelining which produces one output every clock cycle, but this comes with the cost of area as pipelined stages require more resources.

Based on the description of the CORDIC algorithm given in chapter 2, it is observed that multiplication by 2^i is replaced by a bit shift. Shift operations are quicker than multiplication. This section describes the use of CORDIC to compute sin, cos and square root operations. Since multiplication operation is mapped to DSP slices of an FPGA, a reduction in multipliers would give a reduction in the number of DSP slices that are used for computation, this can be observed in the section on hardware design using HDL Coder.

5.4.1 Transmit Delay

The equation to compute transmit delay is replaced by the following,

$$\tau_{tx}[x,z] = \sum_{z=0}^{T-1} scan_{zaxis}.cordiccos(azimuth_z).cordiccos(elevation_z) \\ +scan_{xaxis}.cordicsin(azimuth_z).cordiccos(elevation_z)$$

Results

The table 5.4.1 represents a list of experiments performed to observe the effect of `cordicos` and `cordicsin` on the output image. The quality of the signal and the beamformed image was measured using SNR and Contrast Ratio, as described in chapter 2.

Experiment 1 is the floating-point reference value which is considered to be the ideal and the highest quality of the beamformed signal and output image. Setting the number of iterations to 14 and the fixed point notation of word length to 14 and fraction length to 12 is a good choice for hardware implementation. This is because increasing the number of iterations and fixed-point precision greater than these values did not result in much of an improvement in SNR and Contrast Ratio. Therefore it would be a logical choice to choose the parameters in experiment 6 for hardware design using HDL coder.

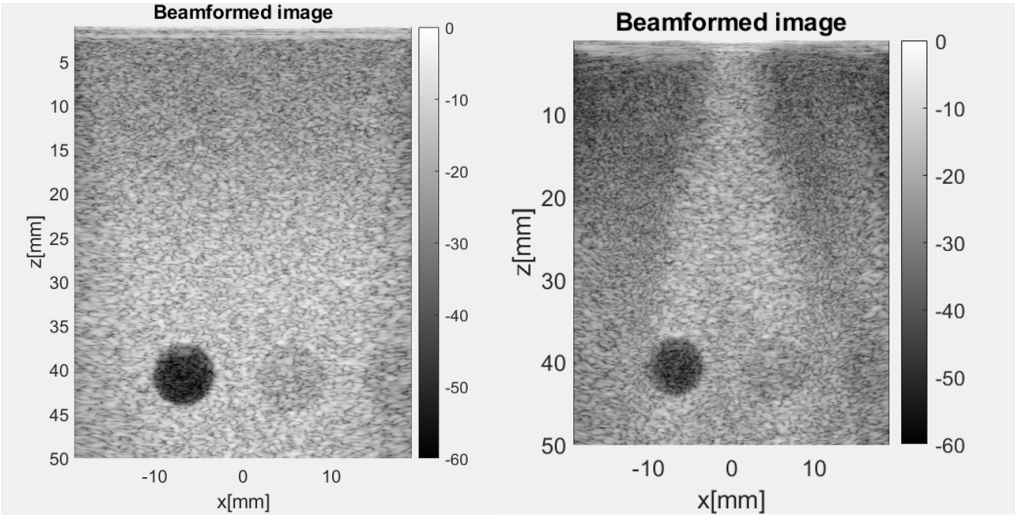

Figure 5.3: CORDIC trigonometric operation. on the left is an image produced with fixed point notation WL-14 and FL-12 and niters-14; On the right is a poorer quality image produced with fixed point notation WL-20 and niters-6

heightExp No.	Data type	niters	SNR	Contrast Ratio
1	floating point	0	1.3176	28.7
2	sfi(v,20)	6	1.0794	18
3	sfi(v,20)	14	1.3166	28
4	sfi(v,14,12)	12	1.3175	27.81
5	sfi(v,14,12)	14	1.3169	27.75
6	sfi(v,18,16)	12	1.3176	28.52
7	sfi(v,20,18)	16	1.3165	28.55

It was observed that using `cordicos` and `cordicsin` decreased the quality of the output image. Figure 5.3 shows the result of using a fixed point representation for experiment 2 and figure 5.3 shows the output image for experiment 4. As the precision decreases and the number of iterations decrease, the image quality decreases like that of the image on the right.

5.4.2 Receive Delay

CORDIC square root is used in the place of the square root operation. The equation is replaced by the equation below,

$$\tau_{rx} = \frac{cordicsqrt(x_m^2 + z_m^2)}{c} \tag{5.3}$$

As expected the quality of the image decreases as and when the fixed point precision and niters decrease. This can be observed in the figure 5.4 where the left is the output image with the fixed-point notation of WL-18, FL-16, niters-12 and the right represents an image with fixed-point notation WL-14, FL-12 and niters-2; it can be seen that the image on the right has a poorly defined boundary of the cyst which is why the contrast ratio is below 20. It can also be observed that CORDIC square root requires a higher precision (as compared to CORDIC cos and sin) to converge to the square root result.

Results

The table below contains a set of experiments performed to observe the effect of `cordicsqrt` on the output of the beamformed image. Experiment 1 is the floating-point reference which is considered

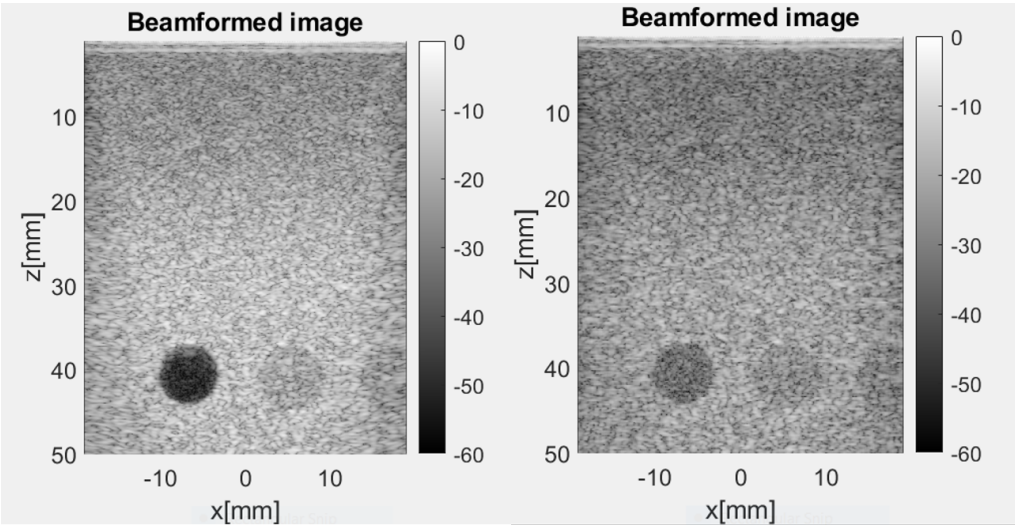

Figure 5.4: CORDIC square root operation, on the left is an image produced with fixed point notation WL-18 and FL-16 and niters-12; On the right is a poorer quality image produced with fixed point notation WL-14, FL-12 and niters-2

as the ideal and highest quality SNR and beamformed image. After fixed-point notation of WL-18 AND FL-16 and niters 12, there is not much of a difference in the quality of the image, hence these parameters can be chosen to work with to ensure a high-quality ultrasound image.

Exp No.	Data type	niters	SNR	Contrast Ratio
1	floating point	0	1.3176	28.7
2	sfi(v,14,12)	2	0.9860	11.8
3	sfi(v,16,14)	4	1.3006	22.5
4	sfi(v,16,14)	12	1.3022	22.63
5	sfi(v,16,14)	14	1.3022	22.7
6	sfi(v,18,16)	12	1.3171	27.8
7	sfi(v,20,18)	12	1.3163	28.48

5.5 HDL Coder : Hardware Design

5.5.1 Beamforming hardware design

MATLAB Simulink is used to create a hardware design to compute the transmit delay, receive delay and beamforming. A high-level block diagram of the ultrasound beamforming process in Simulink is represented in figure 5.5 There are several ways to create the hardware design, based on the user's experience, knowledge and interpretation. Many hardware optimizations can be applied to each of the blocks to increase the speed, reduce memory, area and reduce hardware resources used. Tackling each block in the figure independently and then creating a final model that takes in data and computes the final ultrasound image is the approach taken here. Transmit delay and receive delay are the first hardware blocks that this thesis focuses on. In the future, there is potential to implement the hardware design for beamforming and explore the optimizations HDL coder presents. The block diagram in figure 5.5 represents the beamforming process for a single element of the ultrasound array transmitting a signal and 8 elements receiving the signal. It was not possible to use all 128 elements as receivers to reconstruct beamforming since Simulink takes a lot of time to compute the result and to reconstruct this form of a parallel implementation. In the block diagram, transmit delay is calculated for 1 element and receive delay is calculated for

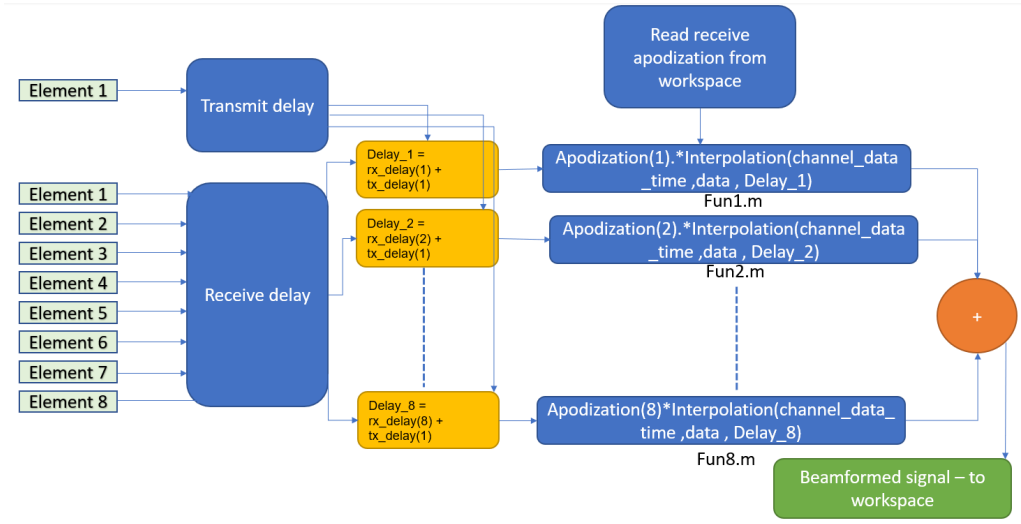

Figure 5.5: Simulink high level block diagram of Ultraosund beamforming

8 elements of the array, these are summed and fed to the block that is a MATLAB function that performs Delay and Sum beamforming, each of the 8 beamformed signals is added to produce a single beamformed signal.

5.5.2 HDL Coder optimizations

Hardware optimizations are offered by HDL coder, the hardware blocks, subsystems and the model all can be subjected to optimizations such as pipelining, resource sharing, increasing speed, reduction of area etc.

1. Block level optimization

   - Pipelining parameters such as output pipelining and input pipelining can be set to a constant value to specify the pipeline depth and allow pipelining to happen.

2. Subsystem level optimization A subsystem creates a hierarchy in the HDL, an entire subsystem can be optimized based on this.

   - Area Optimizations: Hardware can be reused by specifying sharing and streaming factor. Sharing factor is a resource sharing optimization that tries to share up to N resources. Streaming factor allows for the sharing of an atomic part of the design across multiple channels. It is set to the size of the input vector. An atomic part is a part of the hardware design that is independent of other parts. A channel in this thesis represents the hardware design required to calculate the time delay for a single element of an ultrasound array.
   - Speed optimizations: Distributed Pipelining reduces the critical path of a subsystem by re-timing the registers in the input and output pipelining as well as other delays inserted.

3. Model-level optimizations

   - Automatic delay balancing: optimization techniques like streaming and sharing factor give rise to delays in other paths of the model, this is balanced by setting automatic delay balancing. Delay balancing ensures that functional integrity is preserved with reference to the original model

- Adaptive Pipelining: Some Simulink blocks like the delay block and pipeline registers will be automatically inserted with this setting. This way, manually inserting pipelining can be avoided.

Chapter 6

Transmit Delay Implementation

6.1 Transmit Delay

Each block in figure 5.5 is to be considered separately. The first step is to use hardware blocks Simulink offers to compute the transmit delay for all 21 transmitting elements. The second step is to use the fixed point designer tool that collects ranges and plots a histogram on the dynamic ranges of the hardware blocks. The fixed-point tool also suggests fixed point values that one can use in their design. Once the fixed-point design is ready, the steps mentioned in chapter 2 under HDL coder are performed, and a report is generated. The report gives a detailed description of the resources required to compute the result along with optimization and timing summaries. The fixed point design is created based on the equation of transmit delay, in equation 2.4. The unoptimized version is represented in figure 6.1. To check for correctness, the transmit delay values are compared to the floating-point design. Both these designs are unoptimized versions and since HDL coder allows for many optimizations the next sections describe the optimizations used.

6.1.1 Optimized Transmit Delay

Transmit delay is a 262144×21 matrix containing the time delays transmitted by 21 transmitting elements of the array for each of the 512×512 pixels (point sources). As discussed in the literature survey we know that these calculations are computationally intensive. The design in figure 6.2 represents the hardware design computing the transmit delay for 4 out of 512×512 pixels and all 21 transmitting elements of the ultrasound array. MATLAB CORDIC blocks are used to compute the CORDIC equivalent value for cos and sin azimuth and elevation angles. In the figure 6.1 the first input is a single pixel along z axis scan line. The second input is the azimuth angle. The third input is the angle of elevation. There are 21 angles of elevation and 21 azimuth angles that are

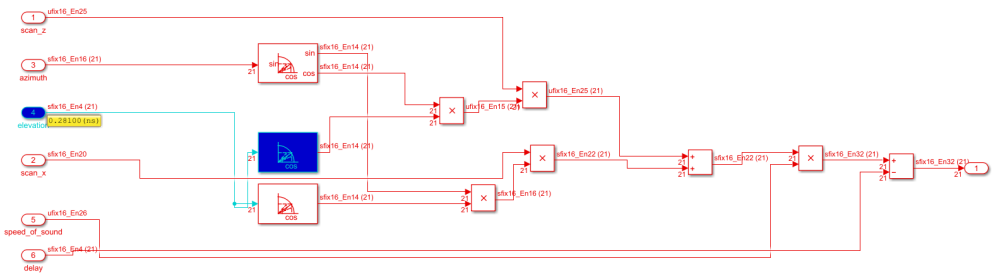

Figure 6.1: Hardware design to compute transmit delay of 21 elements using CORDIC trigonometric function

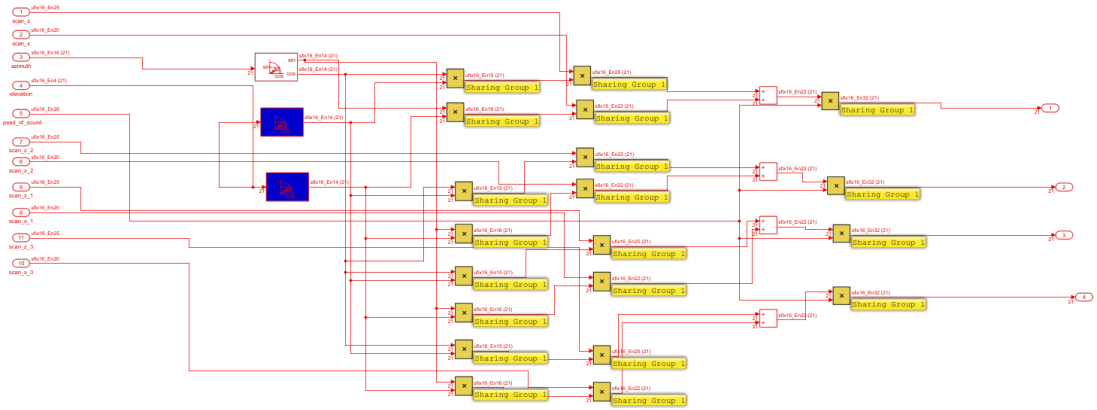

Figure 6.2: Shared resources in the optimized transmit delay design

streamed in. The fourth input is the pixel along the x axis scan line. The sixth input is the inverse of speed of sound. The output is the transmit delay for the pixel in position [1,1]. Similarly, in figure 6.2 there are four pixels that are inputs. Again, there are 21 elevation and azimuth angles being streamed in. The output for this design is the transmit delay for 4 pixels of the 512×512 ultrasound image.

Some optimizations can be used to reduce the number of resources and increase throughput. These are discussed here along with their effect on the design:

- Area optimization : Streaming factor is set to the length of the input vector for the CORDIC blocks. Since there are 21 transmitting elements of the ultrasound array, there are 21 azimuth and elevation angles, therefore the size of the input vector is 21. The streaming factor is set to 21. By doing this, each element of the input vector utilizes the same set of resources i.e the design streams the input values one after the other. Figure 6.3 shows the streaming groups. Each of the streaming groups are represented in different colours. Looking at the figure from right to left, the blocks in yellow belong to one streaming group, similarly with the red, blue, green blocks.

- Area optimization : Sharing factor is set as 5, so five product blocks are involved in calculating the transmit delay for a single pixel. Figure 6.2 represents the sharing groups. There are 4 sharing groups, each computing the transmit delay for a single element. There are 5 multiplication operations required to compute the transmit delay for a single pixel, this requires 5 product blocks, by setting the sharing factor as 5, a single product block is shared among the computations. This optimization drastically decreases the number of multipliers used for computation.

- Delay balancing is set. Delay blocks with delay values matching the pipelining latency are set in paths that do not introduce delays. For example, if the pipeline latency introduces is 28, 28 delay blocks are used in the design to compensate for this.

- Pipelining : Adaptive pipeline is set. There are 2 stages of pipeline introduced for every output. There are 4 output lines, each of which has 2 pipeline stages.

- Speed Optimization : Distributed pipelining is set, with this 1-bit registers are re-timed and the speed is increased.

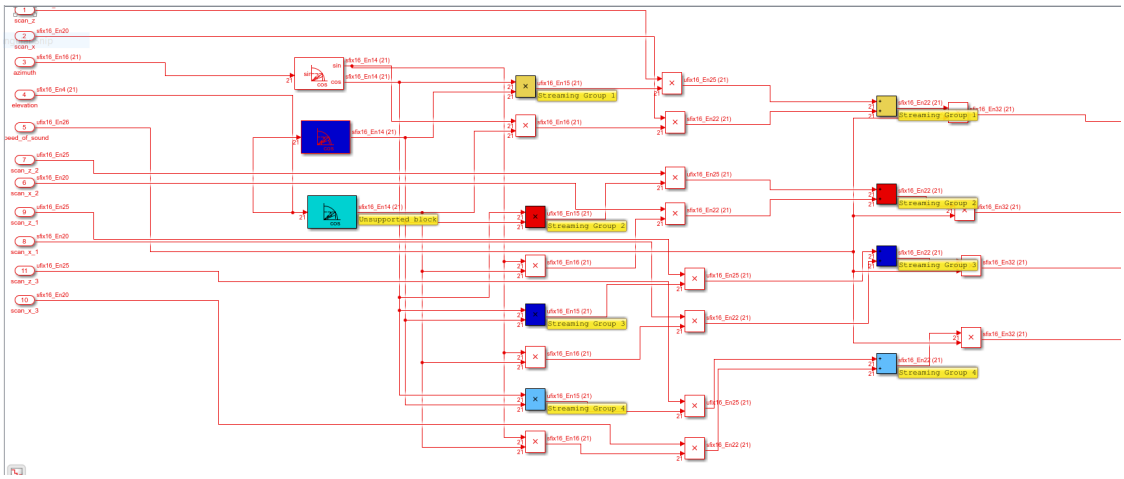

Figure 6.3: Streaming group in the optimized transmit delay design

Experiments and Results

This sections explores in detail three designs that show significant change in resource utilization. The first design is the floating point design, this is a reference design to which other two designs are compared to. The second is the the CORDIC equivalent design, the third is the hardware design with area, speed and resource optimizations and with CORDIC equivalent cos and sin, this is the most optimized version of the three. A detailed description is below :

1. The first design is that of floating point design that calculates the transmit delay for 21 elements of the ultrasound array, this design utilizes the native floating point library by HDL coder to perform FPGA synthesis. This design has the streaming factor as 21 and the sharing factor as 5 and the target device is set to Zynq xc7z035 .

2. The second design is that of a fixed point design, using CORDIC trigonometric functions without any HDL coder optimizations and is targeted to Kintex Ultrascale+ xcku13p-ffve900-1-e.

3. The third design is that of a fixed point design using CORDIC trigonometric functions with the use of HDL coder optimizations as mentioned in the previous section. The target device is the Kintex Ultrascale+ xcku13p-ffve900-1-e.

The table below represents resource utilization for the three hardware designs. On comparing the three designs, it is observed that the optimized version of the fixed point CORDIC cos and sin yields the best results for resources. Traditional cos and sin use 3 more multipliers (also, 3 more DSPs on a FPGA) as compared to that of CORDIC equivalent cos and sin. As expected, using CORDIC reduces the multipliers required to compute the cos and sin values. CORDIC replaces multiplication with addition/subtraction and bit shifts which can be observed with the increase in adder/subtractor blocks and static shift operators. With an increase in the number of iterations, there is an increase in the number of 1-bit registers, keeping the sharing and streaming optimizations constant. CORDIC also introduces more multiplexers since CORDIC is computed in iterations. When these iterations are unrolled into multiple pipeline stages, each pipeline stage requires its own set of multiplexers.

Resource	Floating point	Fixed-p Cordic	Fixed-p Optimized Cordic
Multipliers	45	42	4
Adders/Subtractors	552	4582	4550
Registers	3434	4092	5307
Total 1-bit registers	58362	64277	88164
RAMs	0	0	0
Multiplexers	1671	2363	2347
I/O Bits	4324	1396	2164
Static Shift Operators	18	1260	2160
Dynamic Shift Operators	12	0	0

The table above represents the result after FPGA synthesis for the three designs. As expected, with CORDIC the number of multipliers decrease, therefore the number of DSP's also reduce. However, the number of LUTs and Registers increase. This is expected since CORDIC does use only LUTs, shift registers and addition/subtraction operations to compute values.

Fixed point can store a value in memory as an integer and can adhere to optimizations that have been set, unlike that of floating-point. Which is why the floating-point library consumes more memory as observed in the resource of block ram tile. The floating-point library uses single-precision (32 bits of storage) and double-precision(64 bits of storage) whereas fixed-point integer using an 8-bit or 16-bit representation. Timing requirements are set to 20 ns (clock running at 50 Mhz). This introduces data path delay which is the delay of the data path from the source to the destination. As for slack(the difference between required time and data path delay), negative slack can be solved by increasing the clock or adding register stages to break up combinatorial logic and allow for longer routes. Timing optimization is not performed here. However, in future work, it can be done to help match the speed at which ultrasound data must be processed. Slack indicates that the intended clock frequency could not be met. Here, the positive slack indicates that the data arrives faster than it is expected. The frequency can be increased. It was observed that this design was able to perform with clock frequency of 150-170MHz.

Resource	Floating point	Fixed-p Cordic	Fixed-p Optimized Cordic
Slice LUTs/ CLBs	22941	36801	32783
Slice registers/ CLB Registers	23012	32877	31773
DSPs	32	42	18
Block Ram Tile	4.5	0	0
URAM	0	0	0
Timing requirement	20 (ns)	20 (ns)	20 (ns)
Data Path Delay	4.333 (ns)	3.629	2.834
Slack	15.671(ns)	16.351	17.158

6.1.2 Effect of number of iterations on hardware resources

Figure 6.4 and figure 6.5 represent the effect of increasing the number of iterations on the hardware resources, signal to noise ratio and contrast ratio.

The table below represents the cost of increasing one CORDIC iteration for the fixed point notation of FL-16, WL-25. It can be observed that by increasing the iterations resources like adder/subtractor, registers, shift operators, CLBs, LUTs, pipeline latency increase. However, the number of multipliers and DSPs stay constant for a particular design. This is because the number of DSPs depend on the number of channels and also because CORDIC does not require multipliers for computation. Increasing the number of iterations beyond 11 does not have much of an effect on SNR and Contrast Ratio as seen in 6.5. However, the number of resources required to compute the CORDIC sin and cos do increase. Thus, for a fixed point notation of FL-16 and WL-25, the number of iterations can be set to 11 since there is no change in quality of the ultrasound image

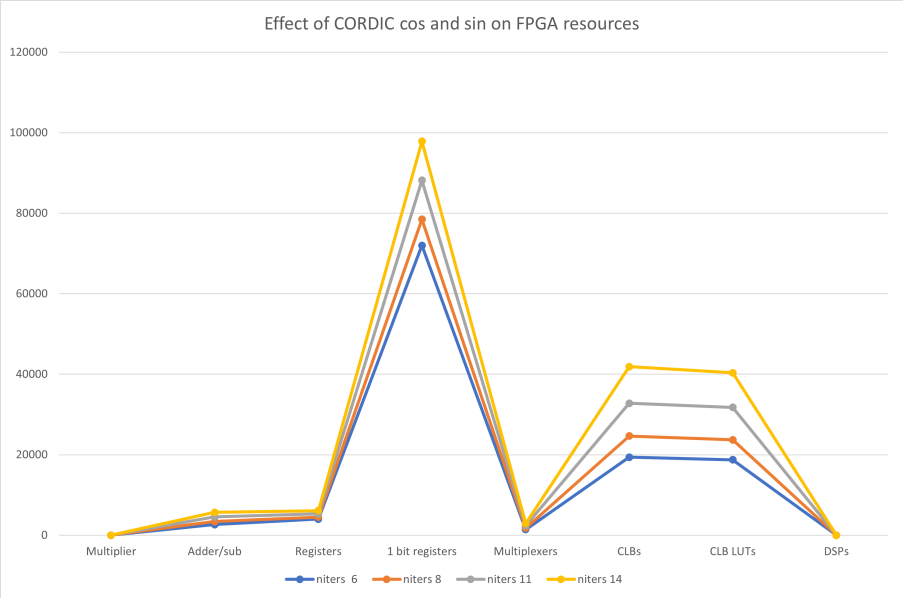

Figure 6.4: Effect of CORDIC sin and cos on hardware resources

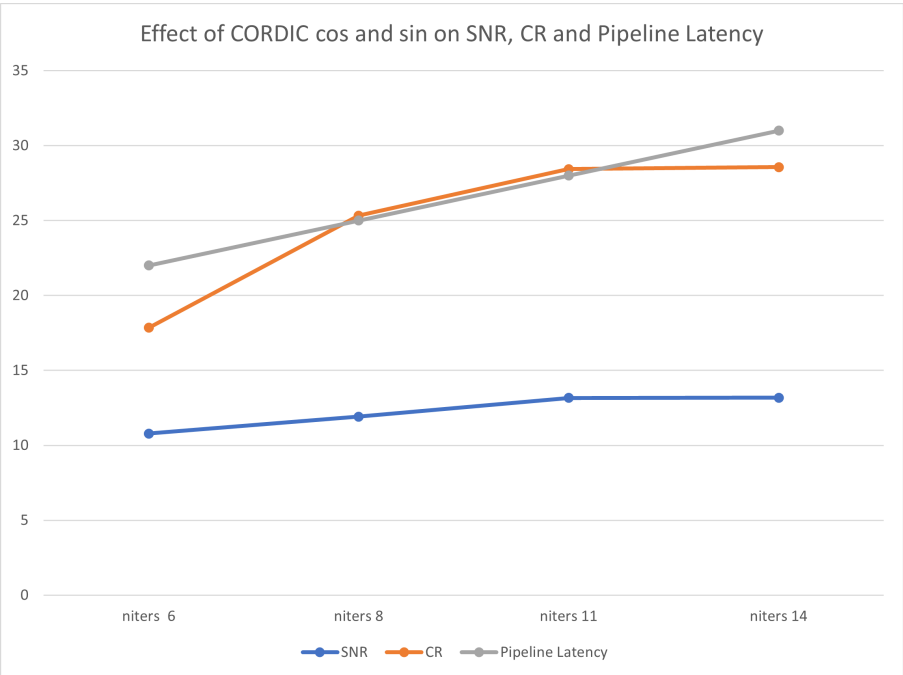

Figure 6.5: Effect of CORDIC sin and cos on SNR, CR and pipeline latency

No. Pixels	Xilinx FPGA	CLB LUTs utilization	CLB register utilization	DSP utilization	Cost
32	Kintex Ultrascale+ KU11P	1.38 %	1.18 %	0.37 %	€ 2.794,60
64	Kintex Ultrascale+ KU11P	2.39 %	1.49 %	0.37 %	€ 2.794,60
128	Kintex Ultrascale+ KU11P	3.01 %	0.020 %	0.37 %	€ 2.794,60
128 x 128	Kintex Ultrascale+ KU13P	93.81 %	67.61 %	0.33 %	€ 3.435,43
256 x 256	Kintex Ultrascale+ KU19P	75.23 %	54.2 %	1.01 %	€ 6.045,04

Figure 6.6: Resource utilization and cost when the resolution of output image increases

with number of iterations higher than 11. The cost of each iteration on hardware area and price is further discussed in chapter 8

Resource	Number
Multipliers	0
Adders/Subtractors	378
Registers	261
Total 1-bit registers	3231
Shift operators	1026
Multiplexers	189
CLBs	2889
CLB LUTs	2724
DSPs	0
Pipeline latency	1

6.1.3 Scaling up the hardware design

Scaling up the hardware design is done by increasing the number of transmitting channels and increasing the resolution of the ultrasound image. The current hardware design in previous sections supports 4 channels of the ultrasound array (i.e 4 transmitting elements). Increasing the number of transmitting elements increases the number of channels used. Similarly, increasing the number of point sources increases the resolution of the ultrasound image. By scaling up, the number of resources increase. The table in figure 6.6 represents the resource utilization on a Xilinx FPGA as the ultrasound image resolution increases. Correspondingly, figure 6.7 represents the drastic increase in FPGA resources as the number of pixels in the ultrasound image increase. It is clear that increasing the resolution and number of channels increase the resources required for computation. Both figure 6.7 and figure 6.6 represent the relationship between resources and the resolution of the ultrasound image for a **single channel**. It is observed (although not presented here) that the number of DSPs increase as the number of channels increases. With an increase in the resolution, there is no increase in DSPs as the resources are shared among the different elements of the input vector (pixels).

Figure 6.8 is a high level block diagram indicating the delay calculation for 21 transmitting elements. Since there are 21 transmitting elements, there are 21 transmitting channels. Each channel is responsible for calculating the transmit delay for all pixels of the ultrasound image. If the ultrasound image has a resolution of 256 x 256, it has the same amount of point sources. Each channel must calculate the CORDIC cos and sin of the azimuth and elevation angle i.e each channel

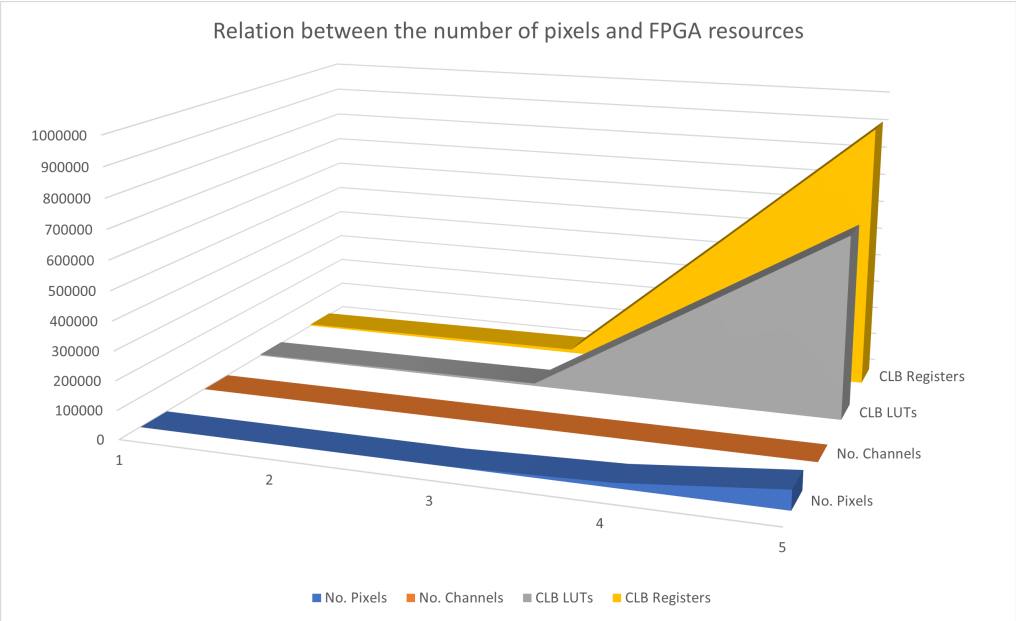

Figure 6.7: Increase in FPGA resources as the ultrasound image's resolution increases

has a different azimuth and elevation angle to it. Transmit delay logic has the hardware blocks responsible for the calculation of the transmit delay. This hardware design can be implemented by considering hardware in the loop simulation (HIL). HIL simulation is real-time and utilizes the connected hardware's resources and provides results in real-time, this eases the load on Simulink as the computation is offloaded on the Xilinx platform in use. Hardware in the loop can be used to figure out if the design fits on the FPGA and further HDL coder optimizations can be performed to improve timing and resource utilization. Performing synthesis with HIL is faster and less time consuming as compared to running the simulation on the pc.

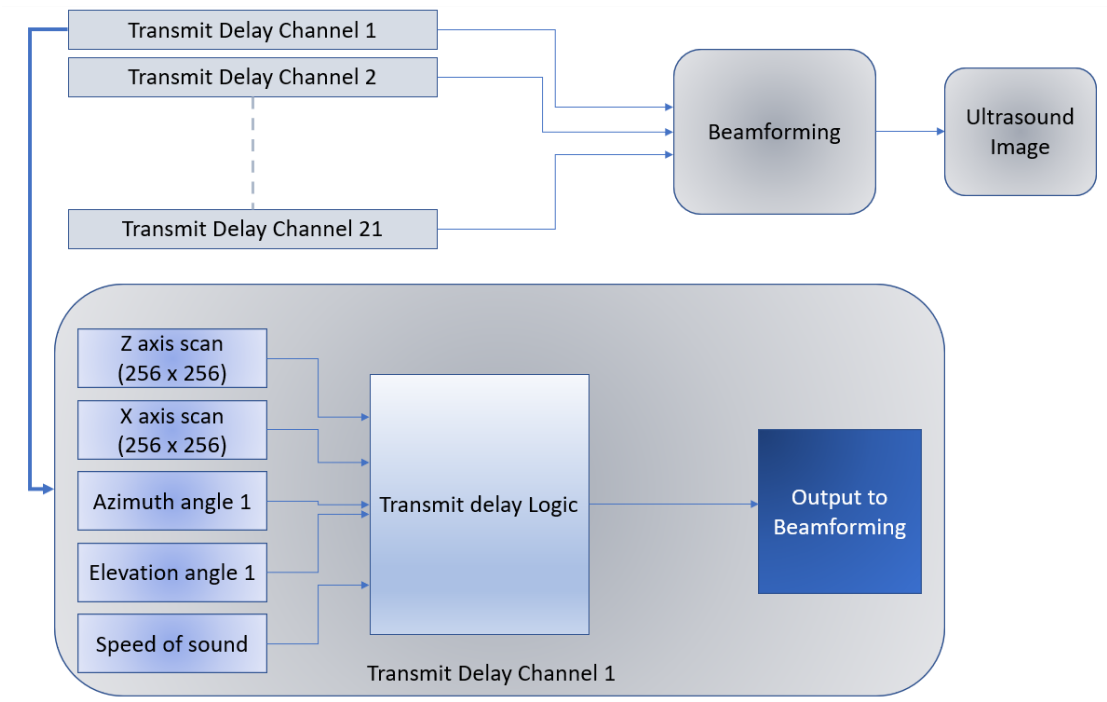

Figure 6.8: Scaling up the Transmit delay calculation based on the number of transmitting channels

Chapter 7

Receive Delay Implementation

7.1 Receive Delay

Receive delay is reconstructed in Simulink for a single-pixel (single point source) and 128 elements of the array receiving the signal. This design seemed to be most ideal to start with since all 128 elements of the ultrasound array are active and required for the reconstruction of the whole image. The hardware design is reconstructed using equation 5.2. Figure 7.1 represents the hardware design to compute the receive delay. The HDL coder optimizations used are mentioned below :

- Area optimization : Streaming factor is set to the length of the input vector. Since there are 128 receiving elements of the ultrasound array, the size of the input vector is 128. The streaming factor is also set to 128. By doing this, each element of the input vector utilizes the same set of resources i.e the design streams the input values one after the other.
- Area optimization : Sharing factor is set as 3 since three product blocks are involved in calculating the receive delay for a single pixel. Figure 7.1 represents three product blocks and one adder block in blue that share one multiplier. The product block that is not in blue uses another a multiplier for itself. Therefore this design requires two multipliers.
- Delay balancing is set. Delay blocks with delay values matching the pipelining latency are set in paths that do not introduce delays. For example, in this design, the pipeline latency introduced is nine, therefore 9 delay blocks are used in the design to compensate for this.
- Pipelining : Adaptive pipeline is set. The product block at the output and the product block computing z_m^2 have 2 pipeline stages each.
- Speed optimization : Distributed pipelining is set, with this 1-bit registers are re-timed and the speed is increased.

7.1.1 Results

The table below represents the resources required to compute the receive delay for a single point source (1 pixel) for 128 elements of the array i.e. the results of the hardware design in figure 7.1. Unfortunately, MATLAB does not offer a CORDIC optimized square root hardware block. Two designs are considered here, the first is the hardware design with the MATLAB square root block and the second is the same hardware design as the previous one except there is no square root block. From the table below it is clear that the square root operation is computationally intensive. The reason behind these two designs is to show how computationally intensive the square root operation is. The second design (without square root) is only represented here for a comparison. The number of multipliers, adder/subtractor, multiplexers and shift operators increase drastically by adding the square root block. The same optimization settings as above are used here. However,

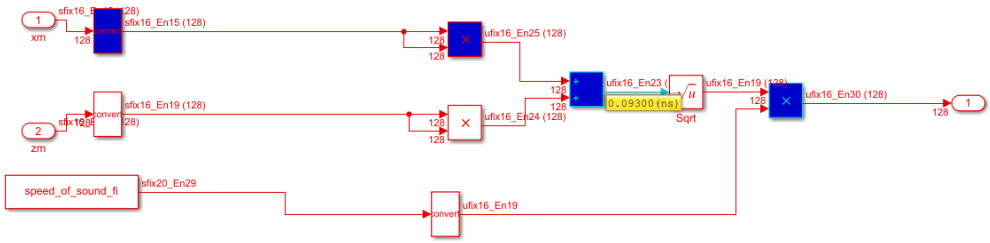

Figure 7.1: Receive delay using MATLAB sqrt function

the 3 product operations all share a single product block therefore the number of 1-bit registers increase to achieve this level of sharing.

Resource	With MATLABsqrt function	Without MATLABsqrt function
Multipliers	2	1
Adders/Subtractors	3848	9
Registers	1429	2837
Total 1-bit registers	35628	80271
RAMs	1	1
Multiplexers	9878	17
I/O Bits	6148	6148
Static Shift Operators	1920	0
Dynamic Shift Operators	0	0
Pipeline latency	9	8

The table below represents the resource and timing summary of FPGA synthesis of the design in figure 7.1. The target device is set as Kintex Ultrascale+ xcku13p-ffve900-1-e. The results are compared with the same hardware design with a slight variation by removing the MATLAB sqrt function. It can be seen that the number of look-up tables increases drastically with the square root operation. This is expected because complex operations that do not have a dedicated hardware pre-compute the result and store the results in a LUT, these values are then used during run time.

An interesting observation is the critical path delay. The critical path is a combinational path between an input and output that has the maximum delay. In the design with the square root operation, critical paths are broken into smaller paths and additional delays are added to meet the timing requirement of 20Mhz, therefore the total latency and register usage on the target FPGA increases. The critical path delay when using the square root function is almost 17 times more than without the square root function. Another observation would be the negative slack; slack is the required time minus the data path time, a negative slack indicates that the signal arrives later than the time it needs to be there i.e it is delayed. Here, the frequency needs to be reduced from 50Mhz to a frequency around 2-5Mhz to meet the timing constraint. The bottleneck in the receive delay lies with the square root operation since there is a limit in the size of lookup tables in FPGAs. Also since timing is always of concern when it comes to an ultrasound scan since the output ultrasound image needs to be viewed in real-time.

Resource	With MATLABsqrt function	Without MATLABsqrt function
CLBs	1.286e+05	3371
CLB Registers	9297	5802
DSPs	2	1
Block Ram Tile	0	0
Critical Path Delay	85.645 (ns)	4.332 (ns)
Timing requirement	20 (ns)	20 (ns)
Data Path Delay	33.209	0.984
Slack	-13.584	18.789

7.2 Solving the bottleneck of square root operation

The Xilinx system generator offers a CORDIC optimized square root operation block. Figure 7.3 represents the square root block by MATLAB Simulink and figure 7.2 represents the CORDIC 6.0 block by Xilinx system generator.

A simple experiment is performed to compare the resource utilization between these two blocks. FPGA synthesis for the target device Kintex Ultrascale+ xcku13p-ffve900-1-e is performed and the results are presented in the table below. The results are for input with a fixed point notation of WL-10 and FL-8. The Xilinx system generator block uses a lesser number of lookup tables to perform the square root of a number with the same fixed-point notation. In future implementations, the CORDIC square root block provided by the Xilinx system generator can be used to reconstruct the hardware design of the receive delay and FPGA synthesis can be performed. With this CORDIC square root block, using a lower precision input and output fixed-point notation and computing CORDIC for lesser number of iterations, it is possible to reduce the number of LUTs/CLBs used. This would, of course, mean that the quality of the ultrasound image will decrease. The trade-offs can be observed with experimentation just as it was done in previous sections. The Xilinx system generator square root block has the potential to provide optimized results for resource utilization (LUTs/CLBs). Working with CORDIC 6.0 would have been the next step in optimizing the receive delay calculation. This was not completed due to time constraints. The proof of concept is presented here and can be implemented in the future.

Resource	Xilinx system generator CORDIC 6.0	MATLAB sqrt block
CLBs	126	212
CLB Registers	137	0
DSPs	0	0

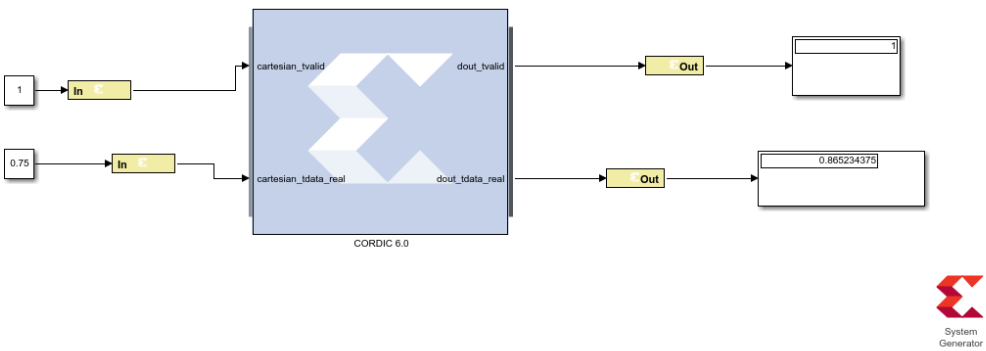

Figure 7.2: Xilinx System Generator CORDIC sqrt block

Figure 7.3: MATLAB sqrt function

No. Pixels	No. Channels	CLB LUTs	CLB Registers	DSPs
64	1	63722	4772	2
128	1	124980	9276	2
128 x 128	1	15682048	1153024	2
256 x 256	1	62728192	4612962	2

Figure 7.4: Increasing the resolution of the ultrasound image and its effect on FPGAs resources

7.2.1 Scaling up the hardware design

Performing synthesis with the existing MATLAB square root function yielded in results that were not feasible. This is shown in figure 7.4. As the number of pixels increased, the size of CLB LUTs and CLB Registers increased drastically to such an extent that they could not fit in the chosen Xilinx device. The data represented in the table is for a single channel, as this increases, the number of DSPs will increase to support the computation. Here we see the true problem of the square root operation spoken in literature. Design changes need to be made such that the design can fit on an FPGA. Firstly, decreasing the number of receiving channels to 32 would have benefits for a decrease in CLB utilization and DSPs. Secondly, the CORDIC 6.0 square root block will also reduce CLB utilization. Thirdly, reducing the precision of fixed-point also results in a decrease in CLB utilization. Once the CORDIC 6.0 IP block of Xilinx system generator is used instead of the square root, the design can be scaled up. To start with, a 128 receiving element beamformer can be reconstructed and this can be scaled down to 64 and 32 element beamformer, as shown in 7.5. For future work on the receive delay, this design can be used to perform synthesis on FPGA (utilizing Hardware in the loop simulation).

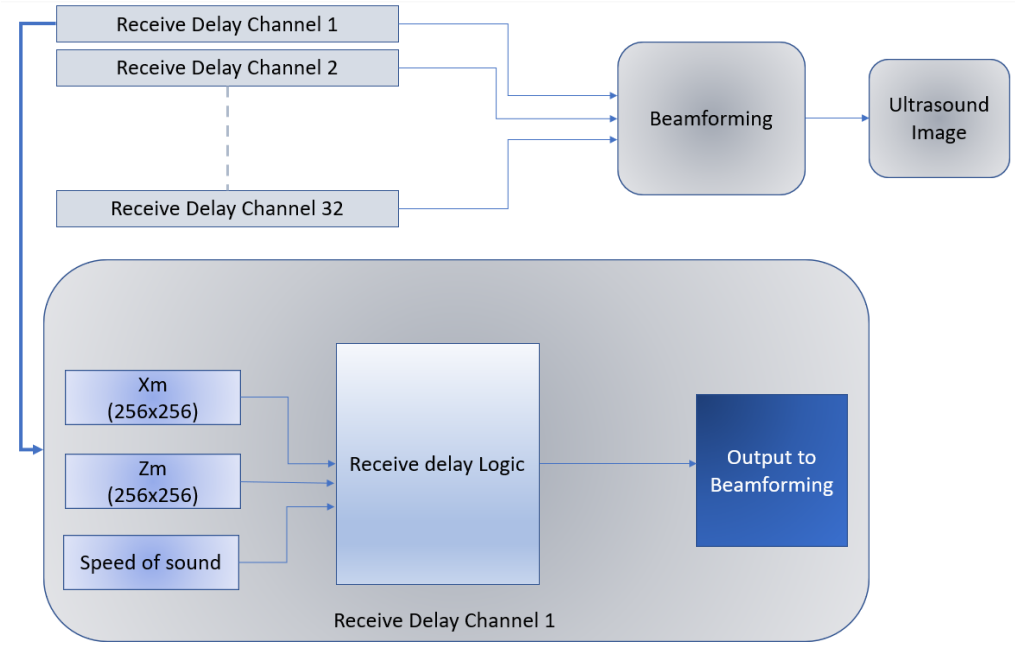

Figure 7.5: Scaling up the receive delay calculation

Chapter 8

Analysis

Chapter 3 presents a literature study on the complexity of Ultrasound beamforming, this complexity is explored in this thesis, both on an algorithmic level and a hardware level. This section discusses the trade-offs observed and documented during the thesis.

8.1 FPGA design trade offs

FPGAs are being used for the most demanding applications like AI, machine learning, signal processing etc, in some cases used with other processors to develop an entire system. As a result, the requirements for effective power, performance and area are import and the trade-offs are equally complicated to analyse. With complex designs such as beamforming, some choices need to be made upfront. An ultrasound scan that has to be performed in real-time would have its priority as speed whereas an ultrasound application that requires a portable system would have reduced power and area as its priority. Similarly, a system that requires a higher quality image would not prioritize a reduction in FPGA area/ number of resources over the quality of its result. Therefore design choices are made based on the requirement of the application. This section aims to briefly discuss some of the trade-offs observed and understood during the thesis.

8.1.1 Fixed point design

Choosing a fixed point design is more efficient for hardware since the value stored in memory is an integer (in this case of 16 bits) and many optimizations can be performed on the FPGAs without requiring additional hardware like a Floating Point Unit (for floating-point computations). It can be seen from previous chapters that resource sharing and pipelining was not effectively adapted by the floating-point design (by HDL coder) but it was adapted for a fixed point design thus enabling a reduction in resources. Without a dedicated FPU, floating-point calculations require significant logic resources on hardware and an increased number of clock-cycles per operation, thereby reducing the efficiency of the device. Floating-point operations require dynamic shift or float of the exponent used to scale the significand during run time, however in the case of fixed-point, the exponent value is defined by the fixed-point data type and is not computed during run time. By specifying the word length and fractional length for inputs and operators (product/division/adder/subtractor), the performance of the device and system improves, in terms of memory, speed, resource utilization.
The disadvantage of using fixed-point is of course it's limited dynamic range, however, fixed-point numbers eliminate the need for the hardware logic to calculate and perform dynamic shifts. Another overhead with fixed-point for an FPGA is that unless double precision is applied, the accuracy will be limited due to rounding errors. In cases of overflow and underflows, fixed point designer tool can be used to prevent this. As for precision, the fixed point notation implemented in the hardware design can be reused in the MATLAB script and the resulting image's SNR and

contrast ratio can be observed. The fixed point is more beneficial than floating-point and hence fixed-point is acceptable for most practical applications such as ultrasound beamforming.

8.2 Relationship between Logic and area of FPGA

The table below represents FPGA resources for Xilinx Ultrascale+ Kintex series. Five different boards are chosen and the change in area and cost as the number of resources increase is observed. The area of the FPGA does not necessarily increase as the number of logic blocks increase, like that of KU5P-FFVD, KU11P-FFVD and KU13P-FFVE. However, the cost does increase despite no change in the area.

Resource	KU5P-FFVD	KU11P-FFVD	KU13P-FFVE	KU15P-FFVA	KU19P-FFVJ
System Logic Cells	474,600	653,100	1,143,450	746,550	1,842,750
CLB Flip-Flops	433,920	597,120	682,560	1,045,440	1,684,800
CLB LUTs	216,960	298,560	341,280	522,720	842,400
Block RAM Blocks	480	600	744	984	1,728
UltraRAM Blocks	64	80	112	128	288
DSP Slices	1,824	2,928	3,528	1,968	1,080
Area (mm)	31x31	31x31	31x31	35x35	42.5x42.5
Cost in €	2.060,16	2.794,60	3.435,43	5.373,37	6.045,04

8.3 Relationship between CORDIC iterations and area, the latency of an FPGA

The CORDIC equivalent for cos, sin and square root operation is used to compute the transmit delay and receive delay. CORDIC square root utilizes a lot of CLB LUTs, CLB registers, this when scaled up to compute the receive delay for all 128 receiving elements of the array and for all the pixels of the ultrasound image exhaust resources and will not fit on the Kintex Ultrascale+ series. Using the CORDIC square root increases the cost of the FPGA because it utilizes too many CLB LUTs. Therefore, computing receive delay is expensive and a mid-high (Xilinx Ultrascale Kintex/ Virtex family) end FPGA with very large (1.4 million and above) CLB LUTs is a requirement.

Two types of CORDIC circuits can be implemented on hardware, the first being a sequential structure (iterative) based on three n-bit adders/subtractors and sign extending shifters, a LUT which performs one iteration per clock cycle. The second being a parallel implementation (cascaded) which is similar to an array multiplier structure consisting of adders/subtractors, constants that can be implemented as a combinational logic for small designs and can be pipelined. Here, consecutive iterations can be executed within one clock cycle.
It is observed that CORDIC increases latency in the application however this is not a problem in practical applications since pipelining is used. In the iterative CORDIC implementation of cos and sin i.e the fixed-point design in chapter 6 yields a critical path delay of 2.834 ns, this means the design can run at a clock speed of 325.8 Mhz. Similarly, the receive delay fixed-point design in chapter 7 has a data path delay of 33.209 ns, this means that the design can run at a clock speed of 30 Mhz. Increasing the number of iterations does increase the accuracy of the result, thereby increasing the SNR and contrast ratio. However, it adds to a lot of latency.

There are a few solutions to deal with the problem of area and latency :

1. Mapping the CLB LUTs to Block RAMs such that the coefficients can be held by the RAM. Not only does this reduce the logic utilization, but it also decreases latency and the speed

increases. [1]

2. Implement a parallel 'cascaded' CORDIC hardware design, consisting of rows of adders/subtractors with hardwired shifts and constants. The amount of parallelism depends on the number of iterations, if the number of CORDIC iterations is 10, there can be 10 parallel stages of this therefore in a cascaded version there is an output at every clock cycle. Latency can be improved here however the CLB consumption increases. In this case, a high end FPGA (Xilinx Virtex Ultrascale) must be used to meet the resource requirements. [29].

8.4 Ultrasound scan and frequency requirements

It is important to be familiar with the frequencies of the sound waves used in medical ultrasound since the selection of proper frequencies influences the image resolution and indirectly the depth of the field visualised. The transducer frequency is inversely proportional to the depth of penetration of the ultrasound signal and directly proportional to the image resolution. The most common frequency used for breast ultrasound is 7.5 Mhz and can reach up to 10 Mhz. Increasing frequency from 7.5Mhz to 10Mhz increases the resolution of the ultrasound signal. However, it reduces the penetration of the beam. The ultrasound beamformer consists of two parts, the transmit beamformer and the receive beamformer.

The transmit beamformer is responsible for initiating scan lines and generating the time pulse string that gets externally converted into high-voltage pulses for the transducer and also set the desired focal point of the cyst. Typically ultrasound elements are capable of transmitting signals at 20Mhz and require high voltages up to 100 Vpp. Thus it would be the task of the FPGA to drive the ultrasound transducers to produce an ultrasound signal of 7.5 Mhz to 10 Mhz.

The receive beamformer is responsible for receiving echo from the analogue front end and collecting data into representative scan lines. The two beamformers are time-synchronized and continuously pass timing, position and control data to each other. The delay is calculated in real-time based on the required instantaneous location of the ultrasound beam for a given scan line. The receiver front end must oversample the incoming data to enable better accuracy and meet the timing requirements of the back end FPGA.

On the whole, it is a difficult task for designers to match the timing across all channels without compromising to signal integrity. The FPGAs are connected to the ultrasound probes, and the DACs and ADCs at the interfaces are responsible to match each block/ channel with timing analysis. This leads to a discussion that is out of the scope of this thesis. Documents provided by Xilinx on this would be of better help [1] [6] [32].

8.5 Trade-off between quality and number of resources

There can be considerable variation in logic capacity for a given FPGA device in use. This depends on factors like, how well the application's logic functions match the architecture of the FPGA device, the efficiency of the synthesis tools, the skill and experience of the designer. There are two ways to measure the capacity of a device. The first is Gate Counting, and the second is Resource utilization. Gate counting involves measuring logic capacity in terms of 2 input NAND gates that are required to implement the same number of logic functions the resulting capacity estimates the users to compare capacities of different Xilinx devices. For a particular application, after synthesis, the CLBs and DSPs are mapped to gates and the number of logic gates used for computation is calculated. This is used to determine the capacity of the device and how much of it is used by the application. Resource utilization is a convenient way of calculating the percentage of resources that are utilized in the application/ computation and synthesis of the hardware design.

[1] https://zipcpu.com/blog/2017/06/12/minimizing-luts.html

This metric indicates the device capacity and how much of the resources are being used.
Scaling the hardware design or increasing the precision both cause the number of resources to increase, which in turn increases the cost of the FPGA since more logic is required. Increasing the number of iterations in CORDIC equivalent cos, sin and square root cause does increase the quality but at the cost of increased resources and price of hardware. Therefore, depending on the required quality of the ultrasound image, the fixed point precision and number of CORDIC iterations can be increased, provided the resource utilization does not exceed the maximum specified utilization for the application. A Suggestion would be to not go beyond 85% utilization is not considered as an ideal case since there exists some level of logic required in moving data along CLBs, interconnects, memory (which is used by the rest 15%)etc. These metrics represent typical utilization levels and devices can be chosen based on the application needs.

8.6 Trade-off between Ultrasound Image resolution and number of channels

An Ultrasound image of lower resolution utilizes lesser resources. This can be seen in the results of transmit-delay in chapter 6. The advantage of using FPGAs is that they are equipped with parallel processing and executing complex computations like square root and trigonometric functions have been feasible. The transmit and receive delay are to be calculated for all pixels of the ultrasound image and all active transmitting and receiving elements, this process utilizes a lot of logic. A high-level block diagram on how to compute these delay values for beamforming is represented in chapter 6 and 7, a compromise would have to be made between the resolution of the ultrasound image and number of transmit and receive channels. On revisiting the literature study, it is understood that the quality of the image increases having more receiving elements. Since the application that the thesis aims for is ultrasound imaging for a superficial breast cyst that has already been detected, a reduced resolution is acceptable and the number of transmit and receive channels can be prioritized. The number of DSPs increase with an increase in the number of channels. However, the number of DSPs do not increase with an increase in the number of pixels. Therefore, it can be expected that to reconstruct the transmit and receive delay for 256 x 256 pixel, 32 element ultrasound receive array a high-end FPGA should be used, Kintex Ultrascale+ KU15 and higher or any of Virtex Ultrascale+ series.

Chapter 9

Research Question and Answers

Principal Research Question: How to optimize the time delay calculations of ultrasound beamforming and what FPGAs can be proposed to perform time delay calculations?

The principal research question is split into two sub research questions and answered separately.

SUB Research Question 1: How to optimize the time delay calculations of ultrasound beamforming using the CORDIC algorithm ?

Ultrasound beamforming is computationally intensive. The number of multiplications and the square root operation is the bottleneck of Ultrasound beamforming. The CORDIC equivalent of cos, sin and square root operation is used in the calculation of transmit delay and receive delay. Using CORDIC has the benefits in reducing the multipliers required to perform operations since this iterative algorithm uses LUTs, adder/subtractors and shift registers to converge to the result. Using CORDIC reduces the number of CLBs LUTs, CLB registers and DSPs used but increases the pipeline latency of the design. CORDIC along with fixed-point design reduces the SNR and Contrast Ratio when compared to the floating-point equivalent. However, a good quality image can still be achieved by choosing an optimal fixed-point notation and the number of CORDIC iterations.

SUB Research Question 2: What FPGAs can be proposed to compute transmit and receive time delays for ultrasound beamforming?

Scaling up transmit and receive delay calculations lead to more resource utilization. The hardware design proposed for transmit delay and receive delay are discussed in previous sections, there is a trade-off between the number of pixels and number of channels such that the design would fit on an FPGA. A suggestion would be to use 21 transmitting elements and 32 receiving elements, this gives rise to 21 transmitting channels and 32 receiving channels. The resolution of the ultrasound image can be set to 256 x 256. For a higher-quality application, these parameters can be increased, but this gives rise to an increase in resources like CLB LUTs, CLB registers and DSPs thus increasing the area of the FPGA and the cost of the FPGA. There are two categories that FPGAs proposed can be put into. The first is a mid-range FPGA and the second is a high-end FPGA. The mid-range FPGA supports 21 transmitting channels and 32 receiving channels for a 256 x 256 ultrasound image and the high-end FPGA can be used for the ultrasound application larger than this. Hardware in the loop simulation and synthesis can be performed before narrowing down on the board one wants to use. It is suggested that transmit delay and receive delay should be calculated on different boards, boards with a very large (1.4 million) CLB LUTs are

most preferable to receive delays.

Mid Range FPGAs	High End FPGAs
Xilinx Kintex Ultrascale+ KU13P	Xilinx Virtex Ultrascale VU160
Xilinx Kintex Ultrascale+ KU15P	Xilinx Virtex Ultrascale VU190
Xilinx Kintex Ultrascale+ KU19P	Xilinx Virtex Ultrascale VU440

Chapter 10

Conclusion

During the course of this thesis, Ultrasound beamforming has been explored, in particularly the transmit and receive delay computation and it's FPGA synthesis has been performed. The complexity of beamforming has been identified, these are the multiplication and square root operations. As a solution to the complexity in computation, the CORDIC equivalent for trigonometric and square root operations has been implemented and the effects of it on signal to noise ratio and contrast ratio have been observed. Several trade offs have been discussed and suggestions on hardware design for future implementations have been indicated. The CORDIC square root equivalent does decrease the number of CLB LUTs and the CORDIC trigonometric operation does decrease the number of DSPs. For future implementation, to increase the speed and throughput of the application, a cascaded CORDIC hardware design can be utilized. There is also a trade off between the resolution of the image and the number of channels of transmitting and receiving elements of the ultrasound array. The number of receive channels can be scaled down to 64 or 32 to reduce resource utilization. Research questions are answered and recommended FPGAs for the computation of transmit and receive delays are discussed. In future implementations these devices can be used and hardware in the loop simulated design can be performed.

Bibliography

[1] Jon Alexander. Xilinx devices in portable ultrasound systems. *WP378 (v1. 2) May*, 13, 2013. 44

[2] M. Almekkawy, J. Xu, and M. Chirala. An optimized ultrasound digital beamformer with dynamic focusing implemented on fpga. In *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 3296–3299, Aug 2014. 17

[3] F. Angiolini, A. Ibrahim, W. Simon, A. C. Yüzügüler, M. Arditi, J. . Thiran, and G. De Micheli. 1024-channel 3d ultrasound digital beamformer in a single 5w fpga. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1225–1228, March 2017. 1, 17

[4] A. A. Assef, J. M. Maia, and E. T. Costa. Initial experiments of a 128-channel fpga and pc-based ultrasound imaging system for teaching and research activities. In *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 5172–5175, Aug 2016. 18

[5] J. Chen, A. C. H. Yu, and H. K. . So. Design considerations of real-time adaptive beamformer for medical ultrasound research using fpga and gpu. In *2012 International Conference on Field-Programmable Technology*, pages 198–205, Dec 2012. 17

[6] Marc Defossez. Connecting virtex-6 fpgas to adcs with serial lvds interfaces and dacs with parallel lvds interfaces. *Application Note: Virtex-6 FPGAs*, 2010. 44

[7] Simulink Documentation. Simulation and model-based design. 14

[8] M. D. Ercegovac and T. Lang. Redundant and on-line cordic: application to matrix triangularization and svd. *IEEE Transactions on Computers*, 39(6):725–740, 1990. 24

[9] M. A. Hassan and Y. M. Kadah. Digital signal processing methodologies for conventional digital medical ultrasound imaging system. In *American Journal of Biomedical Engineering*, pages 14–30, 2013. 17

[10] H. J. Hewener, H. Welsch, H. Fonfara, F. Motzki, and S. H. Tretbar. Highly scalable and flexible fpga based platform for advanced ultrasound research. In *2012 IEEE International Ultrasonics Symposium*, pages 2075–2080, Oct 2012. 17

[11] O. M. Hoel Rindal, A. R. Molares, and A. Austeng. A simple, artifact - free, virtual source model. In *2018 IEEE International Ultrasonics Symposium (IUS)*, pages 1–4, 2018. 3

[12] I. K. Holfort, F. Gran, and J. A. Jensen. Broadband minimum variance beamforming for ultrasound imaging. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 56(2):314–325, February 2009. 17

[13] T. Hopp, N. Ruiter, J.C. Bamber, N. Duric, and K.W.A. van Dongen. *Proceedings of the International Workshop on Medical Ultrasound Tomography: 1.- 3. Nov. 2017, Speyer, Germany.* KIT Scientific Publishing, 2018. 1

[14] A. Ibrahim, P. Hager, A. Bartolini, F. Angiolini, M. Arditi, L. Benini, and G. De Micheli. Tackling the bottleneck of delay tables in 3d ultrasound imaging. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1683–1688, March 2015. 16

[15] A. Ibrahim, W. Simon, D. Doy, E. Pignat, F. Angiolini, M. Arditi, J. . Thiran, and G. De Micheli. Single-fpga complete 3d and 2d medical ultrasound imager. In *2017 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–6, Sep. 2017. 1, 16

[16] G. Kim, C. Yoon, S. Kye, Y. Lee, J. Kang, Y. Yoo, and T. Song. A single fpga-based portable ultrasound imaging system for point-of-care applications. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 59(7):1386–1394, July 2012. vi, 2

[17] MATLAB. *version 9.3 (R2017b)*. The MathWorks Inc., Natick, Massachusetts, 2017. 7

[18] 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*, 34(4):940–949, Apr 2015. 17

[19] 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*, 56(3):489–506, 2009. 22

[20] F. Prieur, O. M. H. Rindal, and A. Austeng. Signal coherence and image amplitude with the filtered delay multiply and sum beamformer. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 65(7):1133–1140, 2018. 3

[21] F. Prieur, O. M. H. Rindal, and A. Austeng. Signal coherence and image amplitude with the filtered delay multiply and sum beamformer. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 65(7):1133–1140, July 2018. 7

[22] S. Ravichandran and V. Asari. Implementation of unidirectional cordic algorithm using precomputed rotation bits. In *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.*, volume 3, pages III–III, 2002. 10

[23] O. M. H. Rindal, A. Rodriguez-Molares, and A. Austeng. The dark region artifact in adaptive ultrasound beamforming. In *2017 IEEE International Ultrasonics Symposium (IUS)*, pages 1–4, 2017. 6

[24] Frederic Rivoallon. Measuring Device Performance and Utilization: A Competitive Overview (WP496). `https://www.xilinx.com/support/documentation/white_papers/wp496-comp-perf-util.pdf`, 2007. Last accessed: February, 2020. 17

[25] A. Rodriguez-Molares, O. M. H. Rindal, O. Bernard, A. Nair, M. A. L. Bell, H. Liebgott, A. Austeng, and L. Lvstakken. The ultrasound toolbox. In *2017 IEEE International Ultrasonics Symposium (IUS)*, pages 1–4, Sep. 2017. 7

[26] R. Sampson, M. Yang, S. Wei, R. Jintamethasawat, B. Fowlkes, O. Kripfgans, C. Chakrabarti, and T. F. Wenisch. Fpga implementation of low-power 3d ultrasound beamformer. In *2015 IEEE International Ultrasonics Symposium (IUS)*, pages 1–4, Oct 2015. 16

[27] L.W. Schmerr. *Fundamentals of Ultrasonic Phased Arrays*. Solid Mechanics and Its Applications. Springer International Publishing, 2014. vi, vi, 4, 5

[28] K. E. Thomenius. Evolution of ultrasound beamformers. In *1996 IEEE Ultrasonics Symposium. Proceedings*, volume 2, pages 1615–1622 vol.2, Nov 1996. 1

[29] Tanya Vladimirova and Hans Tiggeler. Fpga implementation of sine and cosine generators using the cordic algorithm. *Military and Aerospace Applications of Programmable Devices and Technologies Conference*, 01 1998. 44

[30] Jack E. Volder. The cordic trigonometric computing technique. *IEEE Trans. Electronic Computers*, pages 330–334, 1959. 10

[31] Tony Whittingham and Kevin Martin. *Transducers and beam-forming*, page 23–46. Cambridge University Press, 2 edition, 2010. 1

[32] X. Wu, J. Sanders, X. Zhang, F. Y. Yamaner, and Ö. Oralkan. A high-frequency and high-frame-rate ultrasound imaging system design on an fpga evaluation board for capacitive micromachined ultrasonic transducer arrays. In *2017 IEEE International Ultrasonics Symposium (IUS)*, pages 1–4, 2017. 44