HLS Implementation of a Flexible Resampling Filter Using Chirp-Z Transform With Overlap-Add

Master's Thesis in Embedded Electronic System Design

SEBASTIAN BENGTSSON
JOHAN NILSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's Thesis 2022

HLS Implementation of a Flexible Resampling Filter Using Chirp-Z Transform With Overlap-Add

SEBASTIAN BENGTSSON
JOHAN NILSSON

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

HLS Implementation of a Flexible Resampling Filter Using Chirp-Z Transform With Overlap-Add

SEBASTIAN BENGTSSON
JOHAN NILSSON

© SEBASTIAN BENGTSSON, JOHAN NILSSON, 2022.

Supervisor: Erik Börjeson, Department of Computer Science and Engineering
Company advisor: Christoffer Fougstedt, Ericsson Research, Ericsson AB
Examiner: Per Larsson-Edefors, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Teknolog Tryck
Gothenburg, Sweden 2022

HLS Implementation of a Flexible Resampling Filter Using Chirp-Z Transform With Overlap-Add

SEBASTIAN BENGTSSON
JOHAN NILSSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

A possible way of increasing the resolution with which the bandwidth of a system can be selected is by using a flexible resampling filter. Such a filter, using overlap-add (OA) and chirp-Z transform (CZT), is suggested and implemented with Vivado high-level synthesis (HLS) on a field-programmable gate array (FPGA). The resulting filter can handle flexible resampling by varying the input length from 37 to 87 samples while constantly outputing 32 samples. To evaluate this implementation, a reference case was also created, using OA with a regular fast Fourier transform (FFT). When comparing the two cases, it was found that the flexibility that was granted by the CZT implementation utilizes almost 4 times more digital signal processing (DSP) slices in the target FPGA. However, the throughput of the CZT implementation is 36 % higher. Finally, it is concluded that a flexible buffer will be needed to change the resampling ratio during runtime and a possible implementation in hardware is suggested.

Keywords: Bluestein, Chirp-Z, FFT, flexible resampling, overlap-add, radio communication, resampling filter, Vivado HLS

Acknowledgements

We would like to start by thanking Ericsson AB and in particular Henrik Forsman for giving us the opportunity and the means to do this thesis. We would also like to give a huge thanks to Christoffer Fougstedt, because the results in thesis would not have been achievable without his never-ending patience and knowledge. Our deepest gratitude also goes out to Erik Börjeson for his diligent proof reading and feedback that truly inspired this thesis to become what it is. We would also like to thank Johan Löfgren for his help with error-checking a part of our implementation based on an article that he wrote ten years ago. Finally, we would like to thank Per Larsson-Edefors for taking on the task as examiner and for all his feedback.

Sebastian Bengtsson & Johan Nilsson, Gothenburg, July 2022

Contents

List of Abbreviations **xi**

1 Introduction **1**
1.1 Aim . . . 2
1.2 Limitations . . . 2
1.3 Thesis Outline . . . 3

2 Method **5**

3 Theory **7**
3.1 Radio Communication . . . 7
3.1.1 Receivers . . . 8
3.1.2 Analog-to-Digital Conversion . . . 9
3.2 Discrete Fourier Transform . . . 10
3.3 Cooley-Tukey's Fast Fourier Transform Algorithm . . . 11
3.4 Bluestein's Chirp-Z Algorithm . . . 12
3.4.1 The Z-Transform . . . 13
3.4.2 The Chirp . . . 15
3.4.3 Algorithm . . . 16
3.5 Convolution . . . 17
3.5.1 Overlap-Add . . . 17
3.5.2 Overlap-Save . . . 18

4 Implementation **21**
4.1 Overlap-Add Top-Block . . . 21
4.2 160p FFT Sub-Block . . . 24
4.3 Chirp-Z Transform Sub-Block . . . 26
4.4 Implementation Approach . . . 27
4.4.1 Memory . . . 27
4.4.2 Ports . . . 28
4.5 Reference Designs . . . 28
4.5.1 OA-FFT . . . 28
4.5.2 65p and 55p CZT . . . 29
4.6 Verification . . . 30

5 Results **31**
5.1 Comparison with Floating Point . . . 31

5.2 Resource Use Comparison of Mixed-Radix FFT 33
5.3 OA Using FFT vs. Using CZT 34
5.4 Throughput 35
5.5 Power Usage 36

6 Discussion **39**
6.1 FFT Size and Complexity 39
6.2 Error Sources 39
6.3 HLS Optimization 40
6.4 DSP Slice Utilization 41
6.5 Downsampling Factor 41
6.6 Flexible Buffer 42

7 Conclusion **45**

Bibliography **47**

Abbreviations

ADC analog-to-digital converter

BB baseband

CLB configurable logic block
CZT chirp-Z transform

DFT discrete Fourier transform
DSP digital signal processing
DSP48E2 specialized digital signal processing slice in the FPGA

FF flip-flop registers
FFT fast Fourier transform
FPGA field-programmable gate array

HDL hardware description language
HLS high-level synthesis

iDFT inverse DFT
IF intermittent frequency
iFFT inverse FFT

LO local oscillator
LUT look-up tables

MEX MATLAB executable

OA overlap-add
OS overlap-save

P&R Place-and-Route

RF radio frequency
RRC root-raised-cosine

SNR signal-to-noise ratio
SRL shift-register logic

ZT z-transform

1

Introduction

An increasing number of the electronic devices manufactured today utilize the wireless domain to communicate, with 4.7 billion 4G devices estimated to be in use at the end of 2021 and 4.4 billion 5G devices are forecast by 2027 [1]. As an effect, the heavily contested radio spectrum [2] will need to accommodate more data transmission, from today's 80 exabytes per month to 370 exabytes per month by the end of 2027 [1]. Hence, the available spectrum will have to be more efficiently utilized [3], which can be done with optimized bandwidth usage.

Radio communication is commonly performed with a transmitter and receiver, and the latter often contains a resampling filter, which is the focus of this thesis. The resampling stage can be used to reduce the number of samples to be processed and this reduces the bandwidth, which reduces the computational power required in the following signal processing steps. The focus of this thesis is to explore the possibility of supporting different bandwidths without affecting the analog parts in a transmitter by using a flexible resampling filter. For a traditional receiver or transmitter to support different bandwidths, the sample rate and clock rate of the system must be varied. This variability increases the verification effort [4] as all rates must be tested to ensure that no clock spurs violate the specification.

In the 19th century, Joseph Fourier observed that any function can be described as a sum of sinusoidals [5] and although this is only true with conditions, it paved way for what would become the Fourier transform. The Fourier transform is a tool for decomposing a periodic function so that its frequency components, and their powers, can be found [6]. This tool is popular in radio communication where frequencies are commonly used to transmit information. However, digital systems will use the discrete Fourier transform (DFT), since these will operate on discrete data points rather than continuous curves. As the mathematical description of the DFT is of quadratic time complexity, the fast Fourier transform (FFT) algorithm is commonly used instead. As the DFT operates on digital values, the signal needs to be sampled at a fixed time interval, called the sampling rate, so that a sequence of discrete points is obtained. Then the FFT can be applied to these points, which is equivalent to performing a DFT. The output of the FFT is a new sequence of points that indicate what frequencies that carry the power in the original signal [6]. To achieve optimal performance of the FFT algorithm, it should be possible to factorize into suitable prime factors. For implementation purposes, powers of two or four are often selected.

Fractional resampling [7] can be performed by selecting the FFT and iFFT size appropriately. However, the commonly used static FFT would require separate

implementations of suitably sized transforms for all the sampling rates that need to be supported. To achieve non-integer fractional resampling ratios, a possible alternative is to employ Bluestein's FFT algorithm [8], also known as Chirp-Z, which can be used to transform an arbitrary number of inputs [9]. Ericsson has created a MATLAB script that uses Chirp-Z and overlap-add [10] as a proof of the latter concept, indicating that this could be a viable method. This thesis further explores the chirp-Z transform by implementing it in hardware for steady state.

1.1 Aim

With the purpose of achieving higher bandwidth selection resolution, the aim of this thesis is to use high-level synthesis (HLS) to implement a resampling filter with a flexible input length. This filter is going to utilize the chirp-Z transform (CZT) on the input samples as it allows for an arbitrary input length. Furthermore, the length is going to be adjustable by swapping the chirp parameters that the transform uses for the convolution. The output length should always be fixed so that the sampling ratio is decided by the length of the flexible input, which will also allow non-integer resampling ratios. This aim can be broken up into the following partial goals:

1. Create a model of a Chirp-Z–based overlap-add resampling filter in MATLAB.
2. Implement the MATLAB model into hardware using Vivado HLS [11] and run it on an field-programmable gate array (FPGA) evaluation board. The target board will be a Xilinx Zynq UltraScale+ RFSoC ZCU111 [12], provided by Ericsson.
3. Create a reference case using FFT and OA with a static resampling ratio.
4. Compare the hardware implementation of the chirp-Z based resampling with the OA-FFT reference case, regarding
    - achievable clock period,
    - resource utilization, and
    - power usage.

1.2 Limitations

As the design will be implemented with HLS, it could also be interesting to study if a custom VHDL or Verilog implementation would perform better. However, this would take too much time, which could be put to better use by developing a flexible buffer that would enable changing the resampling ratio during runtime. This thesis will only look at the possibility of implementing a design on the ZCU111 FPGA, even though an application specific integrated circuit (ASIC) design could retain the flexibility and would probably perform better since the routing can be more optimized for the design. As HLS has a lot of optimization options, it will not be possible to comprehensively test all of them and some aspects might not be completely optimized.

1.3 Thesis Outline

This thesis starts with the methodology in Chapter 2, where it is described how MATLAB and Vivado HLS were used to create the hardware implementation. Chapter 3 introduces contextualizing background to how radio communication works, followed by descriptions of the FFT and Chirp-Z algorithms together with convolution of streaming data. Based on that theory, Chapter 4 describes how the new filter, using overlap-add together with the CZT, was designed and implemented in hardware. The filter's size and timing is presented in Chapter 5 together with the results from a reference design that uses FFT instead of CZT in the filter. A comparison between the implemented filter and the reference design will be discussed in Chapter 6. The thesis then ends with its conclusion on trade-offs between Chirp-Z and FFT in Chapter 7.

2 Method

First of all, a pre-study was carried out to thoroughly understand the Chirp-Z algorithm and how it could be broken down into smaller, easier to implement, pieces. This was to gain the necessary understanding of the algorithm and how it could be used along with overlap-add (OA) to achieve convolution on streaming data, which in turn would allow flexible resampling to work. To be able to use Chirp-Z it was important to first understand how the FFT algorithm works, since it is used in the convolution that is the core of the Chirp-Z algorithm. Furthermore, it was important to understand FFT because it is used in the static-resampling reference design that the Chirp-Z implementation was compared to.

To gain some practical experience with the two different DFT algorithms, further development of Ericsson's OA-CZT model in MATLAB was conducted. The goal was to improve the proof-of-concept model to show that Chirp-Z and overlap-add could be used in a resampling block. This model was redesigned to handle any arbitrary number of inputs so that it could later be used as a blueprint for development in hardware. At that stage, the key was to write as verbose scripts as possible so that all steps abstracted away within the MATLAB functions became visible. The design for the CZT block was based on the implementation presented *The Chirp-Z Transform Algorithm and it's Applications* [9]. Once finished, it was compared to the output of MATLAB's own built-in CZT-function to check that the results were correct. The idea was that if the model-building step was conducted in enough detail, the hardware implementation step was likely to become easier to finish.

Once the MATLAB model had been completed, C++ programming was used with the HLS library to implement the transform and convolution functionality. To enable adjustment of the types that were used in the HLS file, C++'s `template` functionality needed to be used. This meant that instead of putting the functionality in a .cpp file and the declarations in a .h file, both the functionality and the declarations were put into the header file. The functionality could then be verified with a C++ wrapper file which was used to map the ports to the component (created by the header file) and set the data types. The verification was then conducted by generating MATLAB executable (MEX) files with bit equivalent functionality to the hardware description language (HDL) generated by Vivado HLS [11]. This generation was done by calling MEX [13] on the C++ wrapper with HLS as a compiler flag and the path to Vivado's HLS libraries as arguments. The generated MEX function could then be used as regular MATLAB function to compare the implemented functionality to that of a built in MATLAB function. To get familiar with the HLS workflow,

a FFT generator script was written, based on similar scripts already created by Ericsson, for a radix-5 FFT in MATLAB that produced the intended C++ file. The radix-5 FFT was later used together with a radix-2 and radix-3 FFT to construct composite mixed-radix FFTs. Particularly, a 160-point FFT was needed for the CZT to handle an adequate number of values, which was created as a 5×32-point FFT.

Finally, the Chirp-Z implementation was evaluated by comparing it to a reference case with a static resampling FFT implementation, which had also been implemented using HLS. The comparison included resource utilization, computation speed and power consumption of the implementations. The goal with this comparison was to gain a solid understanding of the hardware cost that comes with the flexibility that a CZT-based implementation offers.

3

Theory

This chapter will present the fundamentals of radio communication associated with the thesis and the inner workings of the FFT algorithm that is commonly used in the resampling filters. Finally, a mathematical description of the CZT algorithm is given, followed by an explanation of circular convolution.

3.1 Radio Communication

Most modern radio communication can be described by Shannon's communication model [14], which partitions the system into five pieces, as seen in Fig. 3.1. In the case of radio communication the data to be transferred is often represented by binary numbers and the channel is often through the air, between two antennas. The transmitter is made up of a digital and an analog stage that leads up to the antenna. In these stages the data is filtered and modulated so that it only takes up the necessary bandwidth. The inverse of these operations is done on the receiver side and the details of this will be the topic of the following sections.

Figure 3.1: Block scheme of Shannon's general communication system. The Receiver block is circled because that is where this thesis aims to make an improvement. Adapted from [14].

The first task of the transmitter is to make the bits transmittable. The bits are modulated to symbols that represent a perfect square wave, which is not suitable to transfer via radio frequencies as it has infinite bandwidth. Instead the bits are modulated to symbols that are upsampled and filtered with a limited bandwidth filter. Typically a root-raised-cosine (RRC) function is used for this limited bandwidth filter, since a transmit RRC filter matched with the receiver RRC filter ensures an inter-symbol interference (ISI) free transmission [15]. The band limited signal must then be converted to analog and filtered again, to remove any noise added from the analog conversion. Finally, it can be output to the channel via the antenna. The channel part of Shannon's model is needed to describe the signal degradation that is due to losing energy and being interfered by e.g. other radio transmissions. The channel is an important part as it helps determine which receiver type is best suited for the specific system.

The essence of the receiver's operation is to invert all the steps that the transmitter performed, which means that similar operations are performed in the reversed order. This thesis aims to improve the resampling step that takes place immediately after the analog-to-digital conversion. To describe how and where this is done, the remainder of this section will be dedicated to receiver architectures.

3.1.1 Receivers

The superheterodyne receiver was first suggested in the early 20th century [16] and is the foundation of modern radio receivers. It is included in this section to give a context to the problems that the other receivers solve. The superheterodyne receiver builds on the concept of using analog circuitry to mix down a high radio frequency to an intermediate frequency with the help of a local oscillator, which can bee seen in Fig. 3.2. Hence, the superheterodyne receiver consists of three bandwidth stages:

1. The radio frequency (RF) or the carrier frequency that is used between antennas, indicated with blue in the following figures.
2. The intermittent frequency (IF) resulting from mixing the local oscillator (LO) frequency and RF, indicated with red in the following figures.
3. The baseband (BB) where the signal will be demodulated and interpreted, indicated with green in the following figures.

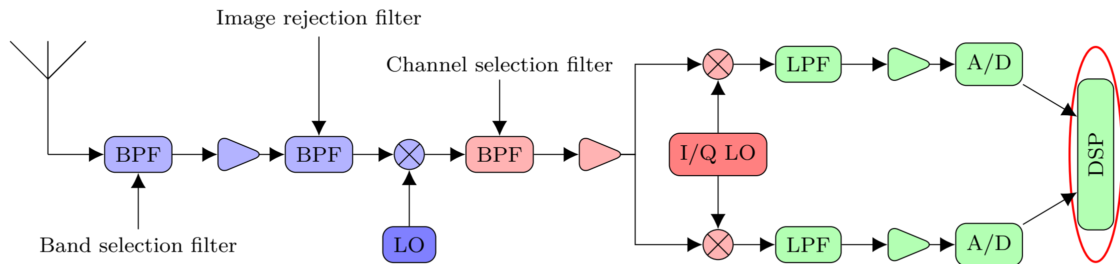


Figure 3.2: Block diagram of a superhetrodyne receiver. After being received by the antenna, the signal is passed through a band-pass filter, amplifier and a real mixer. The output from the mixer is the signal at IF, which is also filtered, amplified and mixed but with a quadrature mixer. Finally, the resulting in-phase and quadrature signals are low-pass filtered and amplified before being converted to digital. Adapted from [4].

After the first mixer, the signal will have a bandwidth of the difference between the carrier radio frequency and the local oscillator frequency. The reduced bandwidth makes it possible to have higher RF at the same time as good selectivity among the BB frequencies. The main drawback of this architecture is the image frequencies [4, 17], which are the frequencies going into the mixer that are mirrored around the LO frequency. If there is interference that is the same distance away from the LO frequency as the RF, then it will be mixed into the IF which causes distortion. To avoid distortion, these interference frequencies can be filtered away with an appropriate band-pass filter called an image rejection filter. The IF signal can then be filtered, quadrature mixed and demodulated. When choosing the IF, there is a

trade-off to be made: a higher frequency increases sensitivity and makes it easier to detect signals while a lower frequency facilitates channel selection.

Another approach, called direct conversion, is to not use an IF and instead mixing directly from carrier frequency to baseband, where the analog-to-digital conversion takes place [4]. Direct conversion introduces the problem of LO leakage that is due to mixers not being ideal and part of the LO frequency will be included in the output of the mixer. However, this design has the advantage of reducing the number of discrete analog components, making the receiver more flexible as there are less components that needs tuning. Furthermore, moving directly from RF to BB centers the signal around the zero frequency rather than IF enabling more work to be done at the lower frequencies, hence decreasing the power consumption. The drawback is higher noise requirements on the analog parts, as the receiver is very sensitive to low frequency noise [17].

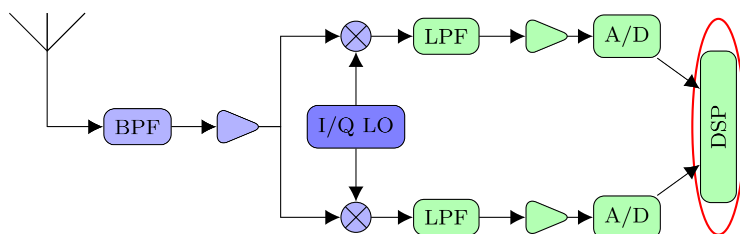


Figure 3.3: Block diagram of a direct-conversion receiver. The signal is passed through a band-pass filter and a low-noise amplifier before being quadrature mixed. The filtering and amplification are repeated and the signal is then converted to digital. Adapted from [4].

3.1.2 Analog-to-Digital Conversion

Independent of the architectures used, the resulting signal will have to be adjusted to retain as much information as possible. Signal restoration is done in the digital domain, just after the conversion from analog to digital, with tools like sampling frequency recovery and timing recovery.

To reduce the complexity of the DSP, the digital signal is often downsampled to a lower sampling frequency in order to reduce the number of samples to calculate. The downsampling is done just after the signal has been converted to the digital domain. Downsampling a signal x by a factor of K is performed by removing samples from the signal in the following way [18]:

$$x(Kn) = x_d(n), \tag{3.1}$$

essentially skipping over every $K-1$ sample. Discarding samples can only be done when the signal quality is high enough, so that it does not affect the frequencies of interest. The Nyquist criterion defines the lower limit as frequencies that are above half the sampling frequency, these will fold over into the spectrum of interest and become indistinguishable [15]. Once the samples have been removed it is important to filter the signal using a low-pass or a band-pass filter to remove any risks of noise from unwanted frequencies to fold into the wanted signal [19].

To perform the downsampling in the receiver, some form of resampling filter must be used. Traditionally, it is trivial to perform downsampling by integer factors with a resampling filter using (3.1). Achieving non-integer resampling factors, however, becomes more complex and requires the use of more advanced filters like polyphase or Farrow [19].

In later sections a method will be presented that might make it possible to achieve these non-integer ratios by doing resampling in the frequency plane. This alternative method, called the CZT, is performing a DFT but does so in a more flexible way than the FFT. What the DFT is and how the different implementations work is described in the following sections.

3.2 Discrete Fourier Transform

To fully understand how the FFT algorithm operates, it is important to first explore the principles of the discrete Fourier transform (DFT). DFT is used to find frequency components of a sampled continuous time signal, i.e. to find patterns in the time signal that repeats themselves.

The DFT maps a set of data points $(f[0], \ldots, f[N-1])$ from the time (or space) domain $f[j]$ to the frequency domain $F[k]$. These data points are usually discrete vectors of data sampled at a given interval from continuous functions or signals. The DFT creates a representation of the data in f, using sums of complex sinusoidals where the fundamental frequency is given by $\omega = e^{-i2\pi/N}$ [6, 20]. The DFT of f is then given by

$$F[k] = \sum_{j=0}^{N-1} f[j] \cdot e^{-i2\pi kj/N}, \quad k = 0, \ldots, N-1, \tag{3.2}$$

where k is the index of the harmonic frequency, j is the index of the sampled time signal and N is the number of samples of the data vector [6]. The last part ($e^{i2\pi kj/N}$) represents the roots of unity [21], or also known as the fundamental frequencies [20].

It is possible to return from the frequency to the time domain again, by using the inverse DFT (iDFT)

$$f[j] = \frac{1}{N} \sum_{k=0}^{N-1} F[k] \cdot e^{i2\pi kj/N}, \quad j = 0, \ldots, N-1, \tag{3.3}$$

which is quite similar to DFT, but the iDFT contains the normalizing factor $\frac{1}{N}$ and the exponent of the roots of unity are positive, not negative [6, 20].

By notating the roots of unity as

$$\omega_N^{kj} = e^{-i2\pi kj/N}, \tag{3.4}$$

the DFT matrix can describe the DFT sums in (3.3) as

$$\begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ \vdots \\ F_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_N^1 & \omega_N^2 & \dots & \omega_N^{(N-1)} \\ 1 & \omega_N^2 & \omega_N^4 & \dots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_N^{(N-1)} & \omega_N^{2(N-1)} & \dots & \omega_N^{(N-1)^2} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{bmatrix}$$

By looking at the DFT matrix, the complexity of the DFT calculation can be seen as $\mathcal{O}(N^2)$ since there will be N frequencies to analyze for N data points. For larger blocks of data, the computation will be very time consuming.

3.3 Cooley-Tukey's Fast Fourier Transform Algorithm

The fast Fourier transform (FFT) algorithm is a powerful tool for calculating the DFT efficiently. It has been observed several times throughout history but it was not until Cooley & Tukey published their paper in 1965 [21] that it gained traction and it has since become indispensable. Though Cooley & Tukey's FFT is the most common algorithm for calculating DFT, several others were established later, e.g. Bluestein's FFT algorithm in 1970 [8] and Winograd Fourier transform algorithm (WFTA) in 1976 [22].

Since DFT is a crucial step in many forms of signal processing, the need to reduce the computational power was absolutely necessary. Cooley and Tukey [21] realized that because of the symmetry in the DFT matrix, the DFT operation could be broken down into several smaller DFT problems, which were less time consuming to compute in parallel. This dissecting algorithm reduced the complexity of the calculation from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$ [6, 23].

The foundation to understand FFT is to first recognize that all odd-indexed twiddle factors, which the notation in (3.4) is called, can be broken down to even-indexed, i.e. $\omega_N^{k(2r+1)} = \omega_N^k \omega_N^{k(2r)}$. This breakdown means that all the odd-indexed terms in the data series can be separated from the even ones. The separation of odd- and even-indexed terms also makes it possible to divide the sum into two smaller summations, half the size of the original.

$$F[k] = \sum_{j=0}^{N-1} f[j] \cdot \omega_N^{kj} \tag{3.5}$$

$$F[k] = \sum_{r=0}^{N/2-1} f[2r] \cdot \omega_N^{k(2r)} + \sum_{r=0}^{N/2-1} f[2r+1] \cdot \omega_N^{k(2r+1)} \tag{3.6}$$

$$F[k] = \sum_{r=0}^{N/2-1} f[2r] \cdot \omega_N^{k(2r)} + \omega_N^k \sum_{r=0}^{N/2-1} f[2r+1] \cdot \omega_N^{k(2r)} \tag{3.7}$$

This separation will also affect the root of unity used in the summation, ultimately breaking down a N-point DFT to two N/2-point DFTs. Since the two N/2-point

DFTs are smaller and can be calculated in parallel, the calculation is much lighter and faster. Next thing to recognize is that the twiddle factors are periodic, which means that $\omega_{N/2}^{(m+N/2)} = \omega_{N/2}^{m}$, this will also reduce and simplify the calculations further, since there will be fewer unique twiddle factors to calculate [6, 23].

$$F[k] = \sum_{r=0}^{N/2-1} f[2r] \cdot \omega_{N/2}^{kr} + \omega_N^k \sum_{r=0}^{N/2-1} f[2r+1] \cdot \omega_{N/2}^{kr} \tag{3.8}$$

$$F[k] = x_{even}[k] + \omega_N^k \cdot x_{odd}[k] \tag{3.9}$$

It is common to describe the FFT algorithm in a butterfly diagram. The butterfly can display FFTs with different radix, i.e. the number of inputs on each partial DFT. For example, an 8 point FFT can be described using pure radix-2 FFT. The radix-2 FFT needs three stages with four 2-point DFT's at each stage to calculate 8-point DFT, as shown in Fig. 3.4. A mixed-radix FFT can be used to process a number of samples that is not limited by a power of two. For example, three radix-2 DFT's can be used in the first stage and two radix-3 DFT's in the second stage to create a 6-point FFT, as shown in Fig. 3.5.

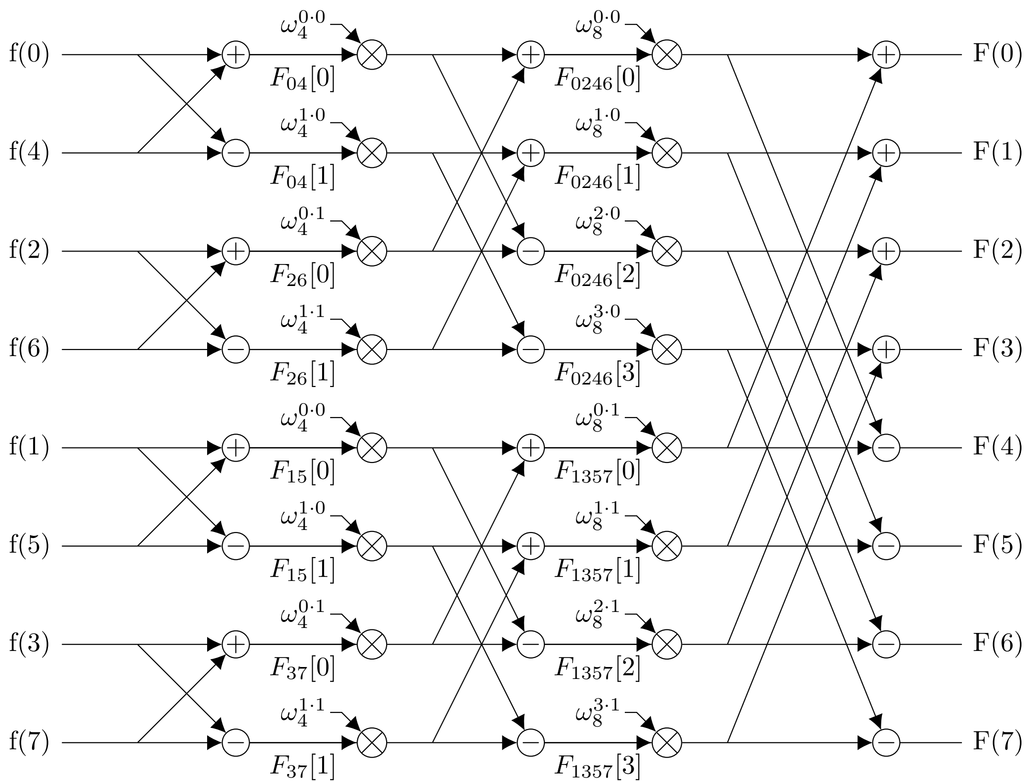


Figure 3.4: Butterfly diagram showing a radix-2 FFT for an 8-point input. Adapted from [6].

3.4 Bluestein's Chirp-Z Algorithm

The chirp-Z transform (CZT) was originally called Bluestein's fast Fourier transform [8], as it was his way of computing the DFT of an arbitrary set of data points.

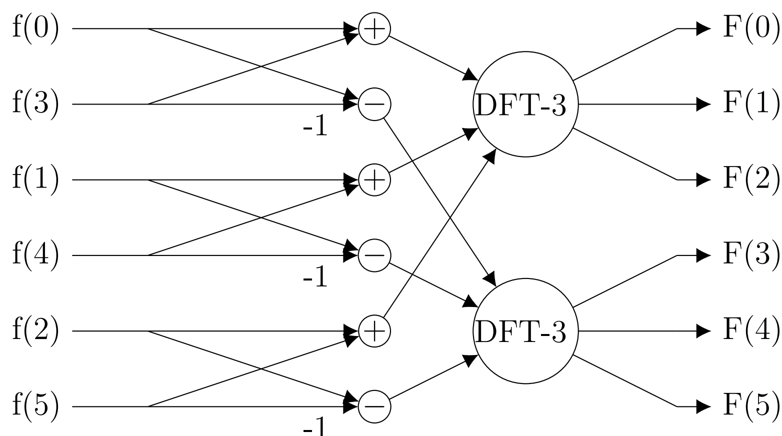


Figure 3.5: A mixed-radix FFT diagram using four radix-2 FFT's in the first stage and two radix-3 FFT's in the second stage to process a 6-point input.

The algorithm takes a linear filtering approach to perform the DFT that is based on the observation that chirp filtering some signal $f(t)$ is a good approximation of taking its Fourier transform. In the z–plane, this chirp filter takes the shape of an arc which can be adjusted so that it falls on the unit circle, effectively performing a "zoomed" FFT. This zooming has the benefit of just calculating the interesting points while simultaneously giving a higher resolution for the same number of points calculated. One of the most advantageous features of the algorithm is that the number of input points can be easily changed by changing the number of points constituting the chirp filter [9]. Furthermore, the filter can be precomputed and stored in memory for quick access, making it possible to change CZT length during runtime.

3.4.1 The Z-Transform

To better understand the properties of the CZT, the z-transform (ZT) must first be mentioned, beginning with its definition. The ZT for right-sided ($X(k) = 0, k < 0$) and finite signals can be expressed as

$$X_k = \sum_{n=0}^{N} x_n z^{-n}, \tag{3.10}$$

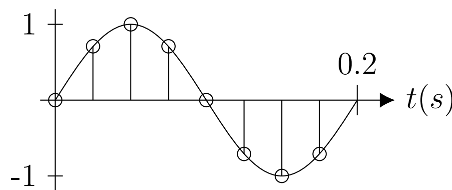
where X_k is the frequency-domain result, x_n is the time-domain samples and z is the complex number

$$z = re^{j\omega}, \tag{3.11}$$

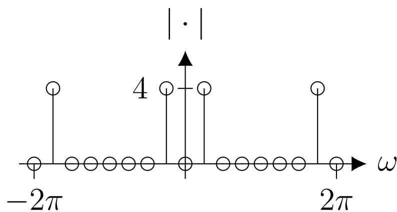
describing the scaling factor r and the fundamental frequency ω [18]. The scaling factor makes it possible to analyze exponentially increasing signals by setting it to the inverse of the exponent, extending the functionality of the DFT. The resulting points in X can be visualized in the z–plane as points around the unit circle with the angular spacing

$$\omega = 2\pi \frac{f}{f_s}, \tag{3.12}$$

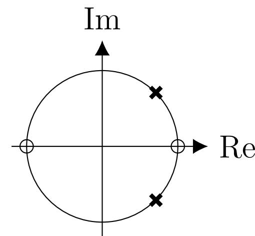
where f is the frequency of the signal being sampled and f_s is the sampling frequency. To visually present the information from the ZT, a pole-zero plot is often used, shown in Fig. 3.6c. A pole-zero plot shows where the zeros and poles are located in the z–plane, with the angle from the real axis indicating frequency relative to ω. Considering the poles, their distance $|X|$ from the unit circles origin indicate the system's stability, where points inside the circle indicate a stable system and points outside indicate an unstable system. Finding poles and zeros can be done by analyzing the frequencies in the ZT to see where it goes towards infinity and zero, respectively.



(a) Sine wave, with a frequency of 5 Hz, samples at 40 Hz.



(b) DFT of the function in Fig. 3.6a.



(c) Z–transform of the function from Fig. 3.6a. Poles marked with crosses and zeros with circles.

Figure 3.6: An example of signal analysis using the DFT and the ZT. It can be noted that the pole placement in (c) resembles that of the amplitude peaks in (b) if the range $0 \leq \omega < 2\pi$ from the ω axis would be wrapped around the unit circle.

In Fig. 3.6 an example is used to show the resemblance between a regular DFT and a ZT when analyzing a sine wave, without scaling. In Fig. 3.6a a regular sine wave is plotted, which is neither stable nor unstable as it oscillates with a constant amplitude. This wave shows up in Fig. 3.6c as a pole on the unit circle at ± 45 degrees from the real axis. The DFT in Fig. 3.6b indicates that there is energy in both the first and last point, but not the zeroth. To be noted here is that both these points appear at 1/8 of the way around the circle in both cases.

In the following section, an alternative z will be described that does not extend around the whole unit circle.

3.4.2 The Chirp

The z-plane chirp contour that is used to perform the chirp-Z transform can be described with the multiplication

$$z_k = AW^{-k}, \tag{3.13}$$

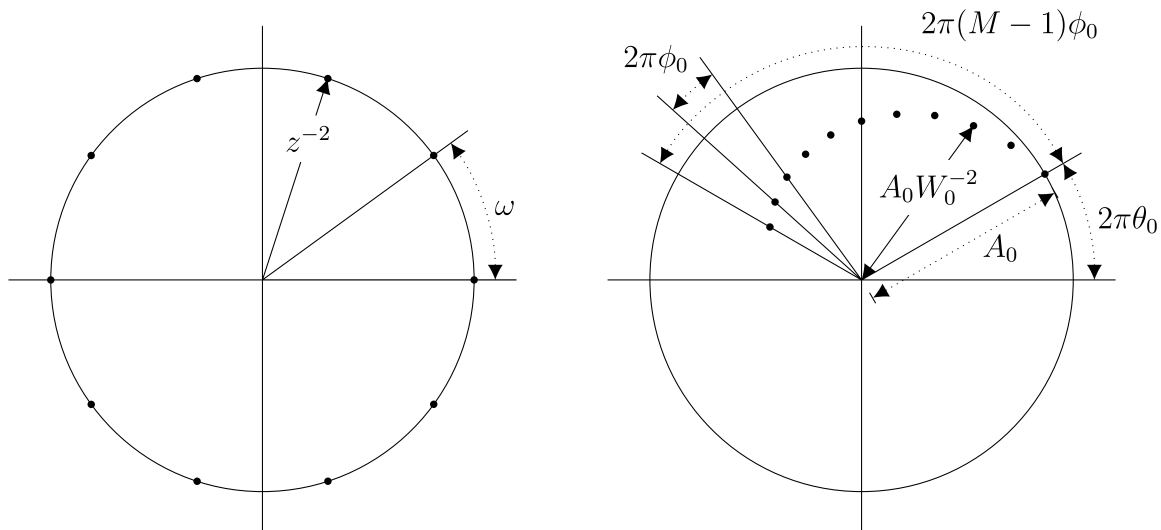
where A and W are two complex numbers defined as

$$A = A_0 e^{i2\pi\theta_0} \quad \text{and} \quad W = W_0 e^{i2\pi\phi_0}, \tag{3.14}$$

where θ_0 and ϕ_0 define the starting angle and the angular step, respectively. A_0 defines the starting point's distance from origin and W_0 determines if the contour will have a constant radius or if it moves towards or away from origin. These variables and their effect on z are shown in Fig. 3.7 together with the corresponding ZT contour for the same number of points. Using the arc as the path for the transform results in the following equation:

$$X_k = \sum_{n=0}^{N-1} x_n A^{-n} W^{nk}, \tag{3.15}$$

where $A^{-n}W^{nk}$ has replaced z^{-n} in the original z-transform (3.10). The effect of controlling z gives that only the part of interest from the frequency spectra is calculated.



(a) The angular spacing of the sampling points for a 10 point ZT.

(b) One example of an AW chirp contour with 10 points, plotted in the Z-plane. Adapted from [9].

Figure 3.7: Visual example of distribution of z values between the ZT (a) and the CZT (b).

It was by expanding the indexing of (3.15), in the following way, that Bluestein managed to extend the functionality of CZT beyond that of Cooley-Tukey's FFT:

$$nk = \frac{n^2 + k^2 - (k-n)^2}{2}. \tag{3.16}$$

Rewriting (3.15) with (3.16) makes it possible to break the CZT algorithm into smaller pieces that could be implemented as a 3 part process, giving the final expression

$$X_k = \sum_{n=0}^{N-1} x_n \underbrace{A^{-n} W^{n^2/2}}_{\text{pre-multiply}} \overbrace{W^{(k-n)^2/2}}^{\text{Convolution}} \underbrace{W^{k^2/2}}_{\text{post-multiply}} . \tag{3.17}$$

3.4.3 Algorithm

The chirp-Z transform achieves the same thing as the FFT but does so with an extended functionality at the cost of higher complexity $\mathcal{O}((N+M)\log(N+M))$, compared to FFT's $\mathcal{O}(N\log N)$. As opposed to the FFT mentioned in Section 3.3, the CZT can take an arbitrary number of points N of a sampled signal as input and output an arbitrary number of samples M. The key to this flexibility is that the CZT uses convolution to achieve the transform, in addition to sophisticated matrix operations. For the algorithm to be efficient, this convolution is done by moving to the frequency plane with FFT, multiplying the required numbers and then moving back with the iFFT. For the algorithm to work, the number of points L that the previously mentioned FFT can handle must be chosen according to

$$L \geq N + M - 1. \tag{3.18}$$

As a consequence, the second step to the algorithm is to pad the input with zeros. This zero padding is done after the signal has been premultiplied with the values defining the contour, the operations can be described as follows

$$y_n = \begin{cases} x_n A^{-n} W^{n^2/2} & n = 0 \leq n \leq N-1 \\ 0 & n = N \leq n \leq L-1, \end{cases} \tag{3.19}$$

in accordance with Fig. 3.8. To make the convolution circular, $W^{n^{-2}/2}$ must be sliced and then rearranged in the following way:

$$v_n = \begin{cases} W^{-n^2/2} & 0 \leq n \leq M-1 \\ W^{-(L-n)^2/2} & L-N+1 \leq n \leq L \\ \text{Arbitrary} & n = \text{other values}, \end{cases} \tag{3.20}$$

v_n and y_n are convoluted by multiplication in the frequency plane and then pointwise multiplied into a signal called G_r. This signal is transformed back to the time domain

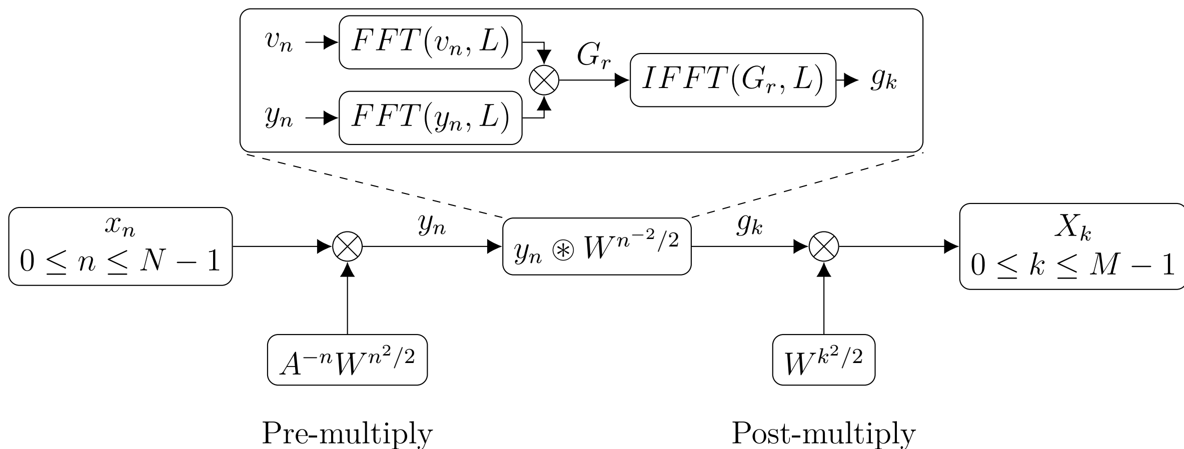


Figure 3.8: Block diagram of the CZT method. Adapted from [9]

with iFFT and multiplied with the final part of (3.17), $W^{k^2/2}$. The result from this multiplication is the final signal X_k. An overview of these steps can be seen in Fig. 3.8.

It is apparent that the CZT will require more hardware resources than a regular FFT but with two highly interesting new features:

- Both numbers of inputs and outputs can be arbitrary numbers which do not need to be composite.
- The contour supplied to the transform does not need to be a circle and the points along the contour can have arbitrary angular spacing.

3.5 Convolution

Non-cyclic data is a problem if any type of fast Fourier transform is going to be used, because such a transform on a finite number of points must be periodic to produce correct output. To make non-cyclic data, e.g. that received by a radio communication system, possible to convolute, there are two common methods: overlap-add (OA) and overlap-save (OS). Both methods work by breaking the long data up into smaller pieces of length N which are separately convoluted and then recombined, without loosing any information. These methods are usually based on a convolution using FFT in the frequency domain, which often assumes that the number of samples to handle is a power of 2, for simplicity [10, 24]. This is not necessarily the case when the CZT is used, as was explained in Section 3.4.

3.5.1 Overlap-Add

To decide the lengths N that the input x needs to be split into, the L samples that the transform can handle must first be decided. To do this, a filter kernel of length H that is going to be multiplied with the input signal must also be selected [10], which is typically a root-raised-cosine (RRC) filter. Both the filter and the input signal must be padded to the length $L \geq H + N - 1$ to make the vectors equal

length and to avoid edge effects [10]. The zero padded data block and filter kernel are transformed to the frequency domain and multiplied to achieve convolution. The product is then transformed back to the time domain, where it is concatenated so that H samples at the end of one convolution are overlapped and added to the start of the next convolution as shown in Fig. 3.9. According to [24], a 1-dimensional convolution is 80 % as complex to perform with OA compared to OS.

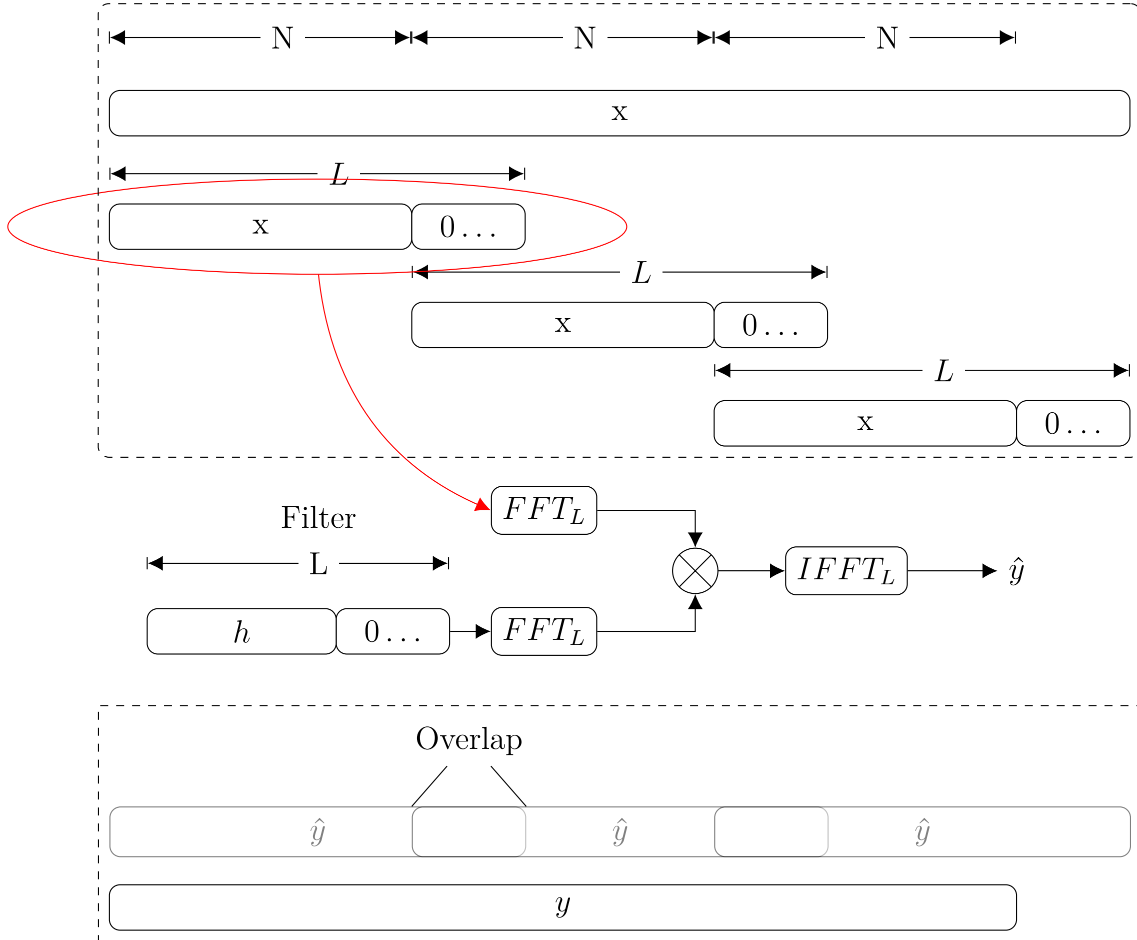


Figure 3.9: Block diagram of the overlap-add convolution. Adapted from [24].

3.5.2 Overlap-Save

Just like OA, overlap-save splits the data stream x into smaller N sized blocks, according to the definition in Section 3.5.1. This splitting means that edge effect are not avoided through zero padding. Instead the next data block overlaps the selection of the first [10] which can be described with the following subsets

$$\hat{x}_0 = [x_0, \dots, x_{L-1}] \tag{3.21}$$
$$\hat{x}_1 = [x_N, \dots, x_{L+N-1}] \tag{3.22}$$
$$\hat{x}_2 = [x_{2N}, \dots, x_{L+2N-1}] \tag{3.23}$$

The data block is then transformed, multiplied and transformed again the same way as for OA. However, instead of adding the overlapping samples they are discarded

and only the part of length N is saved, as shown in Fig. 3.10. As the overlapping samples from the output are discarded this means that OS requires less memory during runtime compared to OA, because OA needs to save the output in order to concatenate it with the previous block [24].

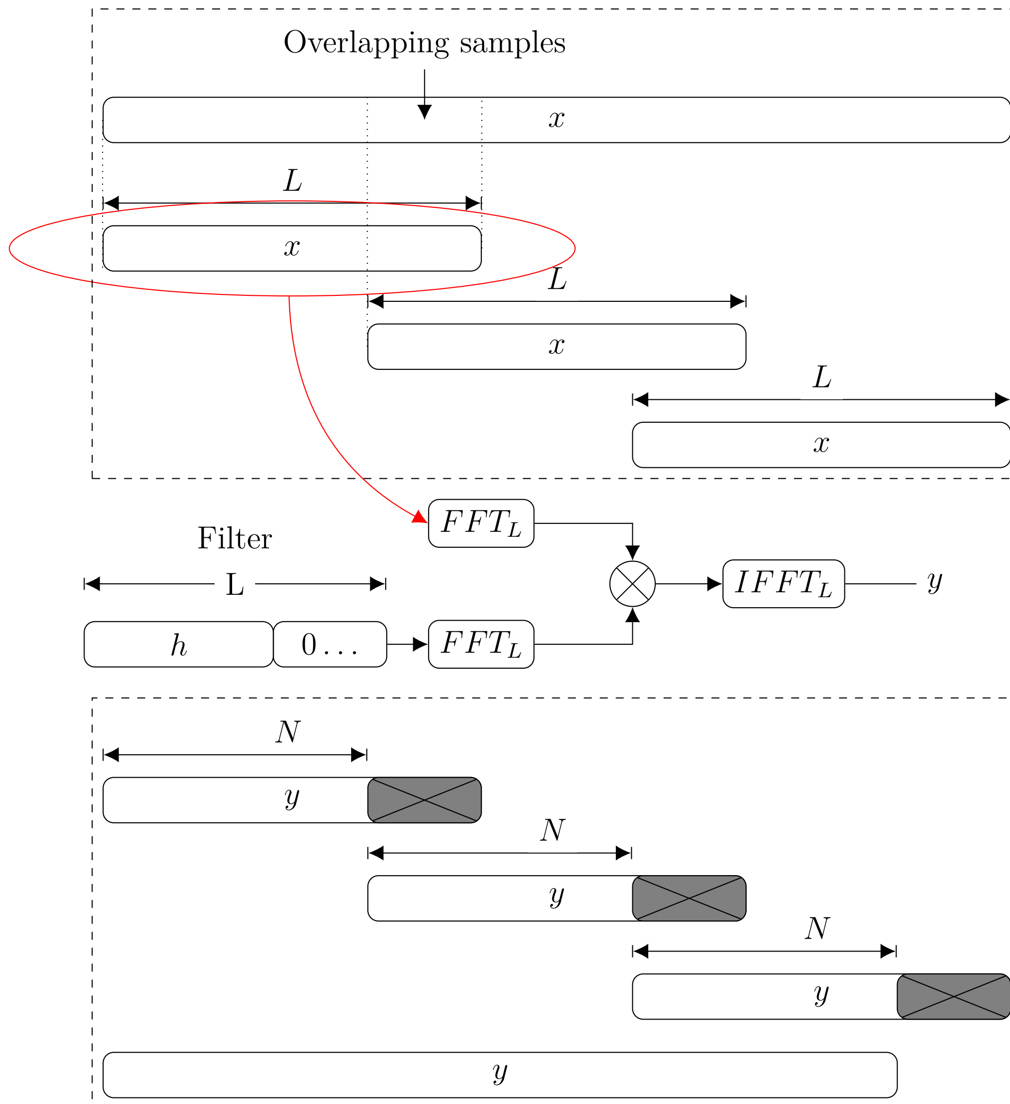


Figure 3.10: Block diagram of the overlap-save convolution method.

4

Implementation

This chapter describes the decisions taken for the hardware implementation of the resampling filter found next to the ADCs. Firstly, we will explain how the RRC filter affects the dimensions of overlap-add (OA). Then we will describe how OA affects the dimensions of CZT and its internal FFT. Then this chapter ends with an explanation of a few reference designs that were constructed to be able to evaluate the size and performance of the implemented flexible filter. The last section explains how the function of the implemented design was verified with a MATLAB model.

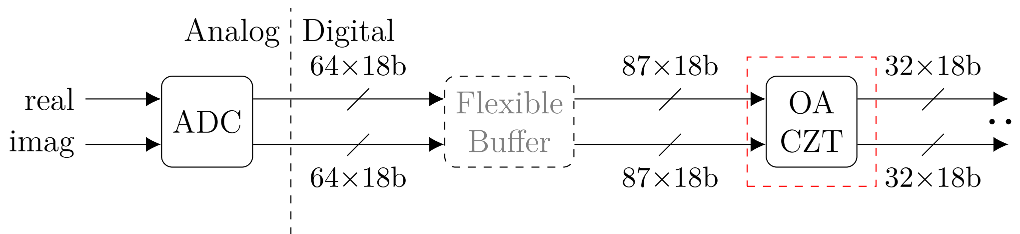


Figure 4.1: The implemented flexible resampling filter using overlap-add and the chirp-Z transform together with the needed flexible buffer.

The flexible resampling filter can be seen together with the suggested flexible buffer in Fig. 4.1. For this resampling filter to function with a completely flexible input length during runtime, it requires a flexible (gearbox) buffer between itself and the ADC that can change how many samples that are sent to the input of OA-CZT. However, the focus for this implementation has solely been the resampling filter. Further details on how the CZT and OA are internally connected will be shown in this chapter. A description of constraints for the flexible buffer is presented in Chapter 6 as a suggestion for future work.

There were two main constraints that the implementation had to be designed around. The first was that it must process at least 64 new complex samples every clock cycle, which means that it should be a flat design that computes every sample parallel to one another. The second constraint was that it should operate using the internal 250 MHz clock, which means that there are 64 new samples every 4 ns.

4.1 Overlap-Add Top-Block

Of the two convolution methods presented in Section 3.5, the only feasible alternative is OA. This is because of the CZT implicitly zero padding the input data, just like

in the OA algorithm, making them compatible. This is not case for the OS which instead uses the input signal to pad the transform, which is not possible with the CZT, unless half of the input width is sacrificed.

A distinct feature of the CZT is that it can "zoom in" on a specific part when transforming, instead of utilizing the entire unit circle as a regular FFT. The zoomed region is chosen by selecting a starting point and an angular step, as seen in Fig. 3.7, in such a way that the arc coincides with the unit circle. This feature can be utilized by the OA method after the RRC pulse shaping filter is transformed to the frequency domain, because then the CZT can avoid calculating the near-zero samples that a radix-2 FFT would need to perform.

The RRC filter that is used in the OA-block has the same parameters as the RRC filter in the transmitter of Ericsson's experimental communication system, to ensure correct data recreation. This filter is defined by 64 coefficients in the time domain. When transformed to the frequency domain, the RRC filter will be defined by 128 samples to be able to fit with a static FFT resampling filter using a 2-to-1 sampling ratio, which can be seen in Fig. 4.2. However, only 74 of these samples will be far-from-zero values, where 10 samples on each side will form the slopes, as seen in Fig. 4.3b.

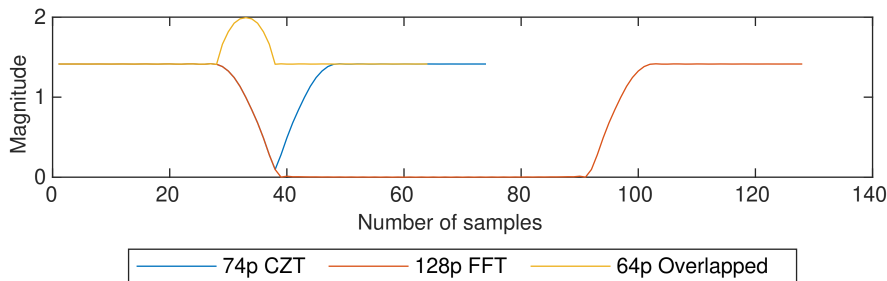
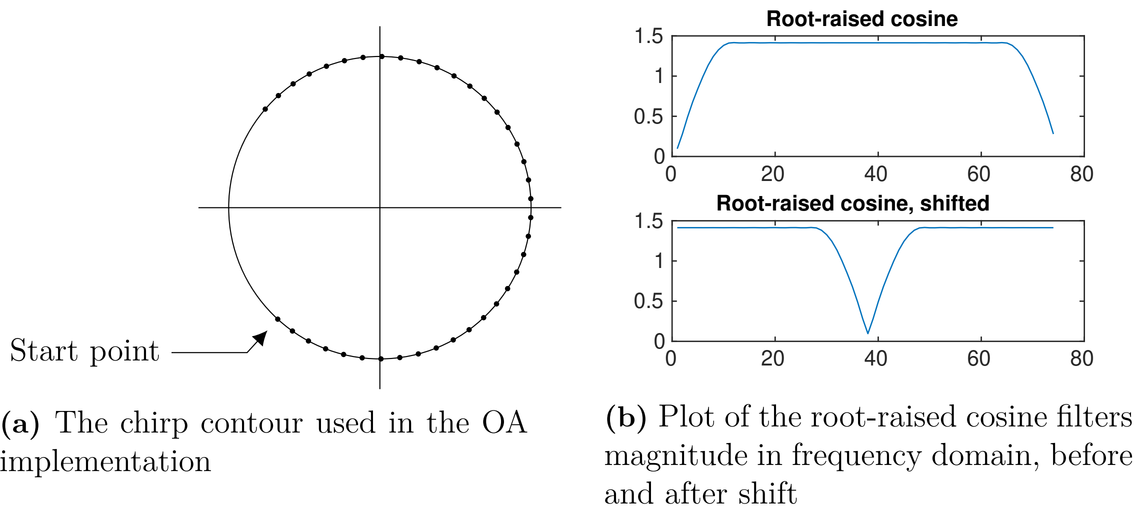


Figure 4.2: The root-raised-cosine filter, transformed to the frequency domain with a 128p FFT and a 74p CZT which then results in the 64p overlapped version

The CZT's zoom feature will avoid the calculation of $128 - 74 = 54$ near-zero samples, based on the size of the RRC filter, as previously stated. To achieve this zoom, the chirp is set to have the same angular spacing as a regular 128 point FFT but with the starting point set to -37 to make the 74 samples wrap symmetrically around the zero angle, see Fig. 4.3a. It should be noted that due to the position of the chirp contour's starting point, the transform needs to be shifted so that the lowest value is placed in the middle of the samples, see Fig. 4.3b. This shift is so that the samples will appear in the right order, as seen in the 128p FFT case. The effects of CZT zoom-feature on the RRC filter can be seen together with an 128p FFT in Fig. 4.2. It is apparent from this figure that the 74p CZT is just 128p FFT without the zero points, which is why they result in the same 64p overlapped output.

The profile of the transformed RRC filter can be be seen in Fig. 4.2, both before and after being overlapped. The reason for the overlap is to remove the slopes and flatten the filter, but it does create an undesirable bump in the middle of it. However, after



(a) The chirp contour used in the OA implementation

(b) Plot of the root-raised cosine filters magnitude in frequency domain, before and after shift

Figure 4.3: Plot of the chirp contour used with the CZT and the resulting transform of the root-raised cosine filter

the received signal is convoluted with the RRC filtered data the bump effect of the RRC filters will cancel out. The resulting data width after overlap is then a 64p transform, which is seen in Fig. 4.2, that enables the use of a 64p iFFT, as seen in Fig. 4.4.

The 64p iFFT takes the signal back to the time domain where both the overlap and addition takes place. As the filter and the signal are of equal length, the overlapping parts will also be of equal length, i.e. the lower 32 samples from one clock cycle are overlapped with the upper 32 samples from the previous cycle.

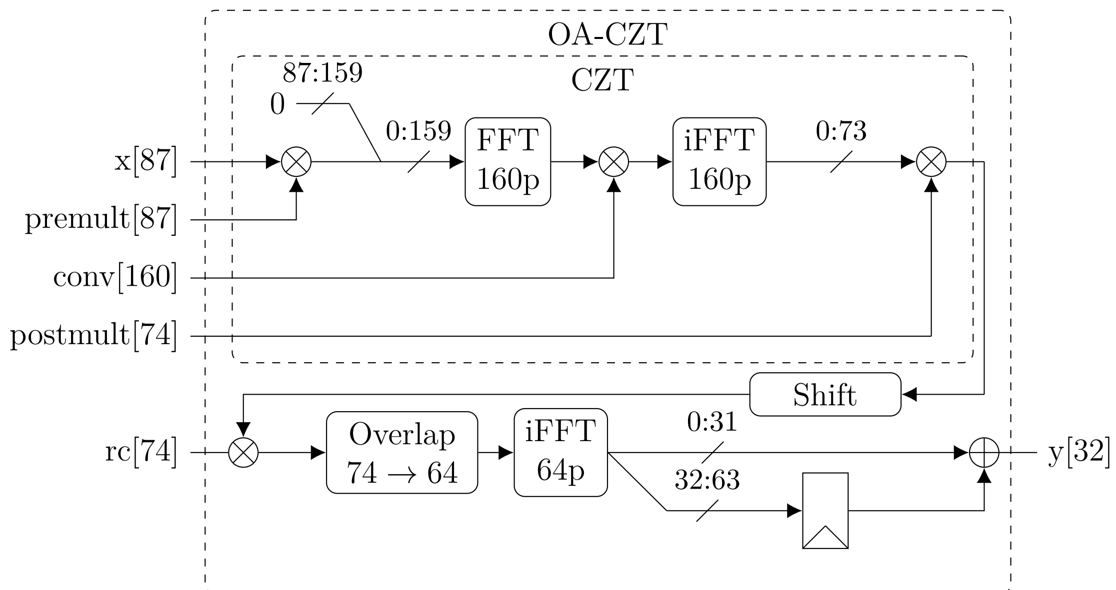


Figure 4.4: Block diagram of the OA-CZT method

4.2 160p FFT Sub-Block

To have comparable systems, both the OA-FFT reference case and the OA-CZT case should be able to handle 2:1 downsampling, meaning 64 input samples must be possible. The Chirp-Z algorithm uses an L point FFT in the convolution stage when y_n becomes g_k, which can be seen in Fig. 3.8. This ratio meant that, according to (3.18), the number of samples that the transform must be able to handle is

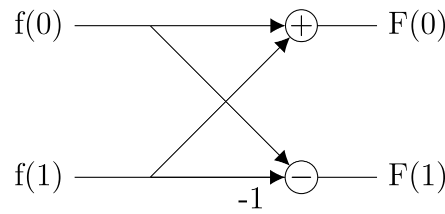
$$\begin{aligned} L &\geq N + M - 1 \\ L &\geq 64 + 74 - 1 \\ L &\geq 137 \end{aligned}$$

The closest radix-2 FFT that can perform this transform is 256 points, which would mean extensive zero padding and calculation of zero values that are never going to be used. To save area, a 160-point FFT was implemented instead by creating a mixed-radix of $2^5 \times 5 = 160$. The current implementation first constructs five sets of radix-2 FFT with $2^5 = 32$ number of inputs for each FFT. After that, a set of five radix-5 FFT's can be connected to the outputs of the radix-2 FFT's, as in Fig. 3.5, to create a new mixed-radix FFT with 160 input and output points.

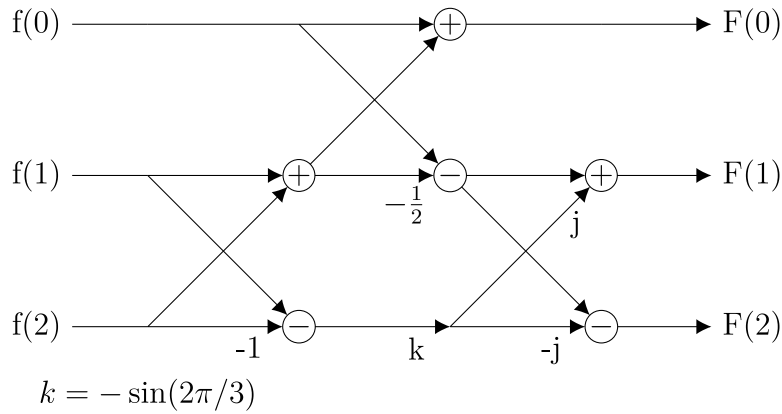
Table 4.1: Some of the possible radix combinations for the FFT in CZT created by the radix-3 and radix-5 implementations.

Mixed radix	Input points	Mixed radix	Input points
2^0	1	$2^3 3$	24
2^1	2	2^5	32
$2^0 3$	3	$2^3 5$	40
2^2	4	$2^4 3$	48
$2^0 5$	5	2^6	64
$2^1 3$	6	$2^4 5$	80
2^3	8	$2^5 3$	96
$2^1 5$	10	2^7	128
$2^2 3$	12	$2^5 5$	160
2^4	16	$2^6 3$	192
$2^2 5$	20	2^8	256

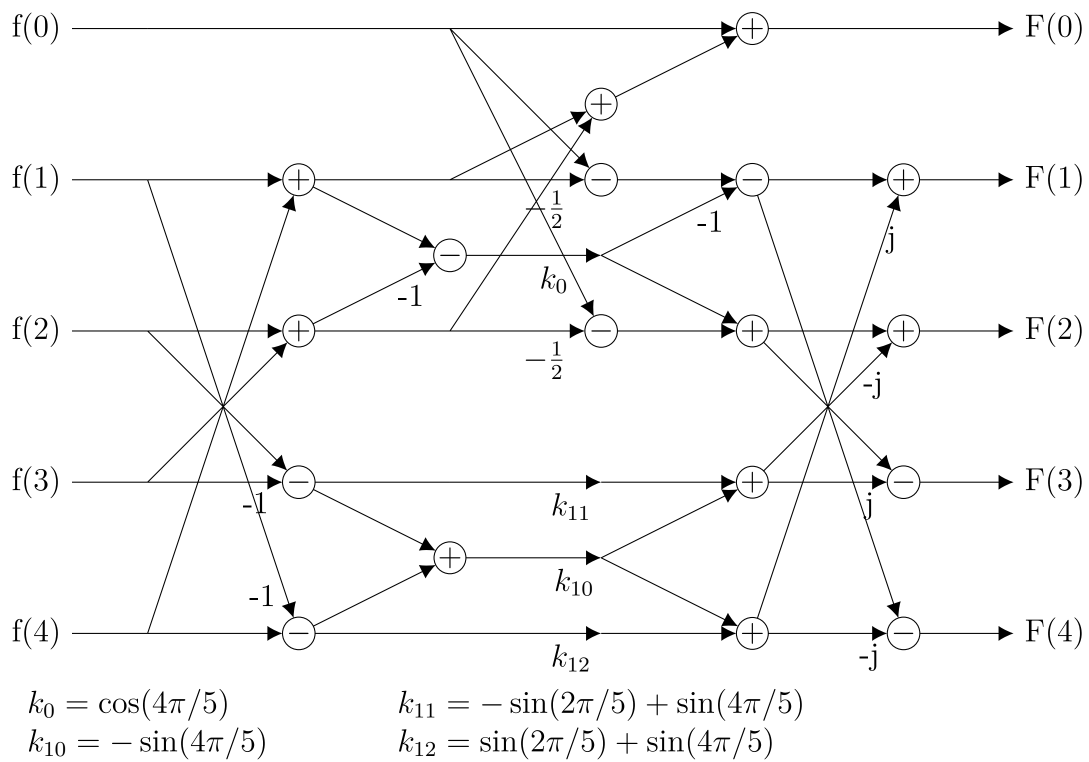
To create more possible variations of the FFT sizes, a radix-3 implementation was also created and can be connected after the radix-2 FFT's in a similar way as radix-5. This makes it possible to further optimize the CZT implementation if the number of inputs or outputs need to be changed in the future. Table 4.1 introduces some of the possible input/output sizes for the FFT created with the mixed-radix implementations using radix-3 and radix-5. The radix-3 and radix-5 FFT's were implemented using the results from Löfgren and Nilsson [25]. The diagrams for these FFT algorithms are shown in Figs. 4.5b–4.5c. To be able to compare their complexity, the diagram for radix-2 FFT can be seen in Fig. 4.5a. It must also be noted that the implementation of radix-2 FFT is optimized by utilizing split-radix 2/4, which is a design that reduces the number of multiplication needed [26].



(a) The implementation diagram for radix-2 FFT butterfly.



(b) The implementation diagram for radix-3 FFT butterfly.



(c) The implementation diagram for radix-5 FFT butterfly.

Figure 4.5: All radix designs were implemented using the results from [25] and all figure have been adapted from [25].

4.3 Chirp-Z Transform Sub-Block

As explained in Section 3.4, the CZT can take an arbitrary integer number of inputs within a certain span. This span can be determined by solving the inequality (3.18), with the given $L = 160$ and knowing that the output from the CZT block needs to be $N = 74$. This setup widens the CZT input M to a new maximum value of 87 points. When testing the system, the lower limit was found to be around 37 samples where the signal started to become distorted. Hence, it is possible for the chirp to select an arbitrary number of inputs between 37 and 87 samples by controlling the three different signals corresponding to the stages in Fig. 3.8. The `premult` signal only affects the input samples and hence has identical width, the `conv` signal affects the whole L point transformed signal and the `postmult` signal affects the M output samples.

The algorithm for performing the CZT was first implemented in MATLAB to facilitate the implementation in C++ by outlining the steps. The most significant change when moving from MATLAB to the hardware oriented C++ was that the data types needed to be changed from `double` to a built in type called `ap_fixed`. This type is, as the name suggest, a fixed point type and must be instantiated both with a word length, zero point placement and directives for how overflow should be handled. Using a fixed point representation places more stringent requirements on how numbers are handled as precision is lost when numbers become small or when numbers grow too large and there is overflow. To achieve the smallest error, the numbers inside the algorithm need to be scaled so that they are as close as possible to, but never exceed, '1' since that might cause overflow. The implementation is using a word length of 18 bits with the zero point after bit 2, making it possible to represent the span of numbers shown in Table 4.2. This number of bits is chosen because of DSP48E2 slices [27], that are part of the ZCU111 FPGA, have two ports and one of them has a limited word length of 18 bits.

Table 4.2: The span of numbers that are possible to be represented with 18 bits where the zero point comes after bit 2.

	Value
Max	1.99998474121
Min	-2.00000000000
Resolution	$2^{-16} = 1.52587890625 \cdot 10^{-5}$

To reduce the number of calculations performed in the hardware, the chirp parameters of the CZT are precomputed outside of the design and loaded as needed. In Fig. 4.4, these parameters are named `premult`, `conv`, `postmult` and correspond directly to the stages in (3.17). Additionally, the coefficients that represent the RRC-filter can also be precomputed outside of the hardware implementation and are seen as `rc` in Fig. 4.4. This removes a lot of complex arithmetic which otherwise would require large area for computations that very rarely will be performed. Furthermore, to change the resampling ratio it is only a matter of updating the precomputed values and changing the amount of zero-padding on the data.

An optimization was also implemented that combined the multipliers associated with `postmult` and `rc`, which reduced the amount of needed DSP48E2 slices. This was possible as the shift operation just as well can be performed after this compound multiplication. The optimized implementation can be seen in Fig. 4.6. As the precalculation of the `rc` parameter previously included a shift operation, it had to be updated so that the sample values were not shifted twice.

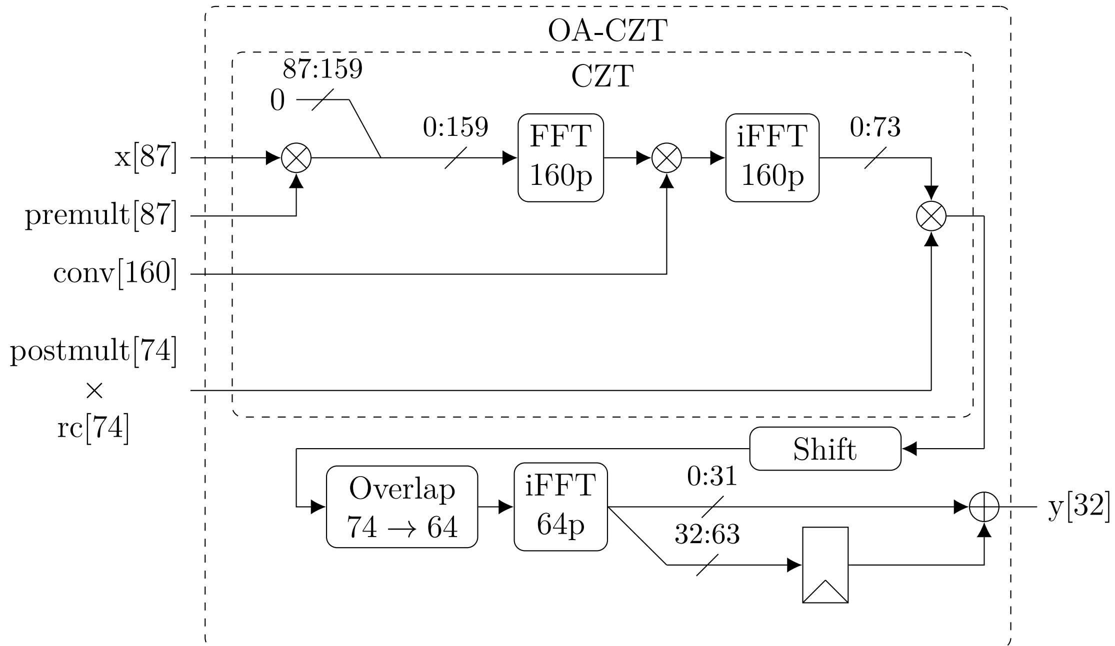


Figure 4.6: Block diagram of the optimized OA-CZT method.

4.4 Implementation Approach

HLS adds a level of abstraction between the HDL and the user, thus some of the fine grain control over how the hardware is created gets taken away. To regain this control, Vivado HLS offers directives, called pragmas, that can be passed to the preprocessor to realize code in different ways [11]. The most important pragma for the whole design was `#pragma HLS ii=1` which tells the tool that the design needs to be able to accept new data every clock cycle. Additionally, this pragma raises warnings when some part of the design does not meet this requirement, which was incredibly helpful when debugging. The next step was to achieve the desirable timing of 250 MHz, which was fulfilled by targeting the design at a lower clock period. It was necessary to lower the clock target, otherwise Vivado had problems to route the design effectively.

4.4.1 Memory

As the implementation is going to be used with streaming data, the most important design goal is throughput so that new data can be processed every clock cycle. For this throughput to be achieved, a couple of things need to be controlled:

1. The way that variables are stored
2. The way that arithmetic is handled

If no directive is given, the HLS compiler will store all data given as arrays in block RAM without any consideration for throughput. As all precomputed data is given as pointers to arrays it would require several clock cycles to fetch the data before computing anything with it. This problem was solved with the help of two directives:

- `#pragma HLS DATA_PACK port=<port>`
- `#pragma HLS ARRAY_RESHAPE variable=<variable> complete dim=1`

`DATA_PACK` stores the data structures side by side in one continuous chunk of memory instead of randomly spread by the memory controller. `ARRAY_RESHAPE` reorders this memory to be stored as column vectors rather than a row vectors for faster access. The `dim` switch decides the number of columns to reshape the variable into.

Once all the inputs could be read simultaneously, the arithmetic needed to be handled. The default behavior of HLS is to not unroll any loops i.e. if 10 numbers were going to be multiplied then it would take 10 clock cycles with one number each cycle. This approach is very area efficient but provides the worst possible throughput. However, this is easily solved by setting the initialization interval `#pragma HLS PIPELINE ii=1` at the top level of the design, which forces all loops to unroll.

4.4.2 Ports

As previously stated, the throughput requirements are the most strict, which also puts tight requirements on the ports that control the data flow. In general, the inputs and outputs from the block are controlled with the pragma `#pragma HLS INTERFACE` which decides how the interface is instantiated. The input and output ports both need to utilize `#pragma HLS INTERFACE ap_vld port=<port>` which creates both a port, `<port>`, and a valid signal called `<port>_vld` which indicate that the data currently on the buffer is valid.

The three ports handling the chirp will not change in a rate that is anywhere near the rate of the input/output ports. These chirp ports can instead be defined as `#pragma HLS INTERFACE ap_stable port=<port>`, which will not create any extra ports or signals to verify that the data is correct and hence saving some resources.

4.5 Reference Designs

To evaluate the newly implemented flexible filter using OA-CZT, it was necessary to also build a few different reference designs to fairly evaluate the cost of the flexibility.

4.5.1 OA-FFT

Two static resampling filters using OA-FFT were implemented as reference designs to compare resource utilization and timing. Both reference design uses OA for the convolution, but instead of CZT they use either a 128p or a 256p FFT. The 128p

FFT creates a static resampling filter with a 2:1 sampling ratio, while 256p FFT creates a 4:1 sampling ratio. The block diagram of the implemented resampling filter using 128p FFT can be seen in Fig. 4.7.

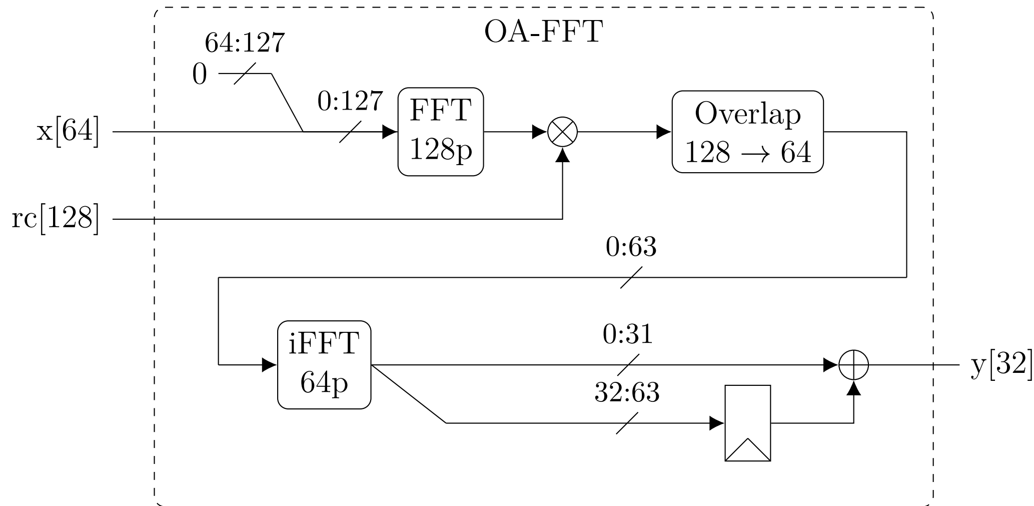


Figure 4.7: Block diagram of the reference design using OA-FFT.

Comparing OA-FFT to OA-CZT in Fig. 4.4, there are a few distinct changes, besides the obvious change in wordlengths. The whole CZT block has been replaced by a single 128p FFT. This also demanded that the input data now had to be zero-padded before entering the 128p FFT block, which is more similar to how it is described in Fig. 3.9. In CZT, however, the zero-padding is a required function built into the algorithm and therefore did not need any prior zero-padding on the input data. Another change that could be seen is that the RRC filter now uses all 128 points described in Fig. 4.2 and therefore no longer needed the shift operation.

The reason for building OA with 128p FFT was to compare the new flexible 2:1 resampling filter with a static one. The OA-design using 256p FFT was used to compare the performance for the maximum number of inputs (87 samples) possible for CZT to a pure radix-2 FFT design.

4.5.2 65p and 55p CZT

As stated in Section 4.2, the input width of the CZT makes it possible to resample in ranges exceeding the target of 2:1. This would make the comparison with a regular 2:1 FFT unfair and, hence, another implementation was made with an input port that was cut short to 64 samples. This truncation meant that $(87 - 64) \times 18 \times 2 = 828$ bits smaller input bus, making routing easier. The logic in the implementation is still similar as the removed ports are zero padded to give valid values to the 160p FFT

Another design was also created, with a 128p FFT inside the CZT instead of the 160p one. As the FFT block is the single largest block in the design and also used twice, for both a transform and an inverse transform, this made a real impact on the size. Changing the size of the internal FFT also reduced the usable resampling

ratio down to a maximum of 55:32, due to (3.18) giving $128 \geq 56 + 74 - 1$, but the resources saved is considerable.

4.6 Verification

Another feature that Vivado HLS offers is the use of testbenches written in C++, making the interface to both the design and external files simpler than with traditional HDL. This testbench can then be used both after the C++ code has been synthesized to HDL and, more importantly, a register transfer level (RTL) testbench is automatically generated and used. The ability to verify the generated hardware is especially important when working with HLS as there are risks for erroneous RTL generation if the pragmas have not been specified correctly.

The implemented C++ testbench was built around test vectors that are generated by a MATLAB testbench which performs verification by comparing a floating point model with the MEX binary. In the MATLAB testbench, a test vector containing around 100 000 randomly generated 64 quadrature amplitude modulated symbols is generated and upsampled with the same ratio that the OA-CZT system downsamples. The samples that make up these symbols are sent to both the floating point model, MEX binary and exported to text files. Furthermore, the chirp parameters mentioned in Section 4.2 are generated by the same MATLAB testbench, based on the resampling factor, and exported to text files. The C++ testbench imports the input data and parameters from these text files to then compare the testbench result with the results from the MEX version. Finally, the testbench returns '0' if the results were completely identical, allowing implementation to continue to the next step of the tool-chain.

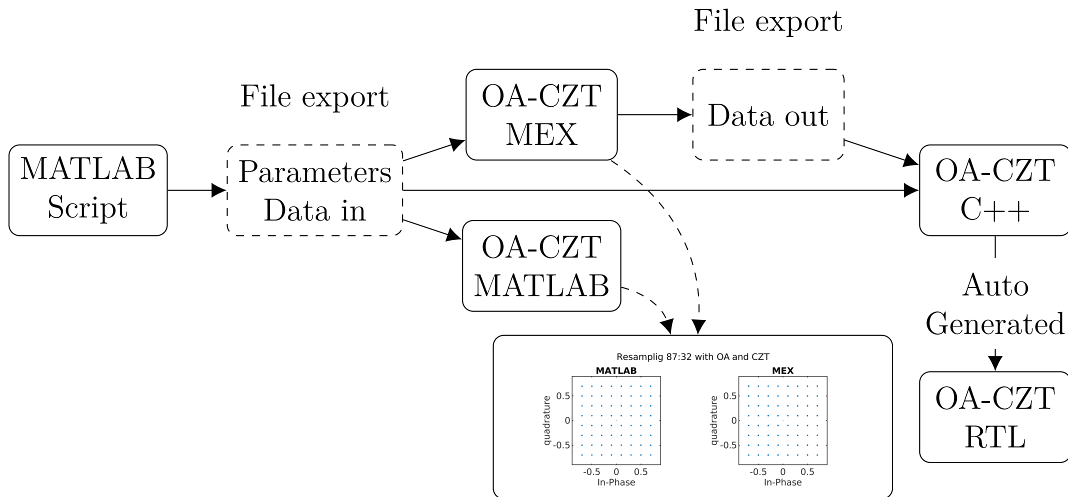


Figure 4.8: Block diagram of how the testbenches relate to each other.

5
Results

Presented in this chapter are the results from the implemented OA-CZT together with results from various reference cases with OA using both FFT or CZT. Firstly the results from the implemented design's fixed point representation is shown with the software model's floating point, followed by the resource utilization of different FFT sizes. Lastly, the resource utilization and throughput of the OA-CZT are presented with different reference cases, to estimate the cost of the flexibility.

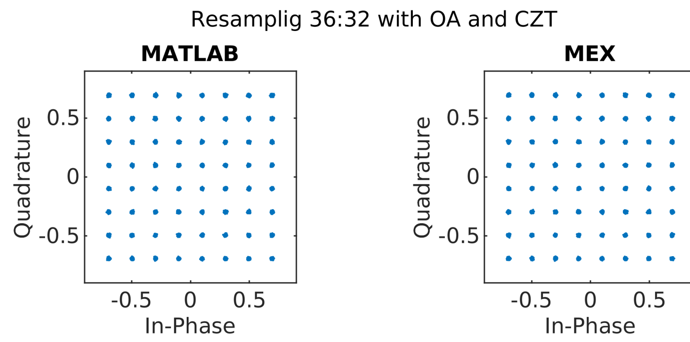
In the coming tables, the resource utilizations will be based on the number of DSP-slices and configurable logic block (CLB). Each CLB is built up by a number of look-up tables (LUT), flip-flop registers (FF) and shift-register logic (SRL) [28].

5.1 Comparison with Floating Point

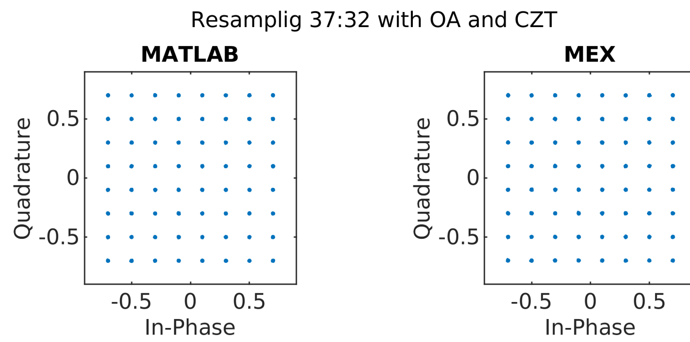
As stated in Chapter 4.6, the implemented hardware was based on, and verified with, floating-point models created in MATLAB. Functionality was ensured by comparing the binary MEX file to the MATLAB model in two ways:

1. Constellation plots were generated, giving a direct visual indication of how well the filter resampled the signal.
2. The signal-to-noise ratio (SNR) was calculated to verify that the error power of each point was acceptable.

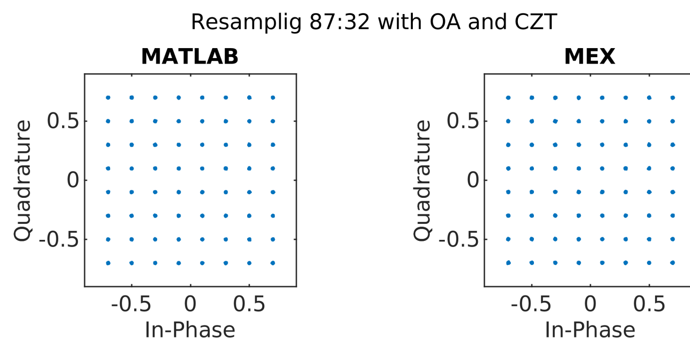
The constellation diagrams in Fig. 5.1 will present the edge cases of the resampling ratios and show what happens in the cases when the ratio is outside of the allowed range. For the comparison to be fair, the constellation diagrams were normalized, which was needed due to scaling inside the MEX file. The lower limit was found empirically to be at a resampling ratio of 37:32 as 36:32 started to introduce noise, see Figs. 5.1a–5.1b. This degradation is likely caused by the RRC filter that starts to fold in on itself, which in turn is an effect of violating the Nyquist criterion. Two cases around the upper limit can be seen in Figs. 5.1c–5.1d and when the resampling ratio goes above the specified limit it is severely degraded. The degradation, shown in Fig. 5.1d, appears because the upper limit and the resampling filters input width are the same number and everything above that will be zeros. The reason for zero padding everything above can be seen in (3.20), where the different values would start to overlap. To further compare the performance of these edge cases, the SNR was also calculated and is presented in Table 5.1.



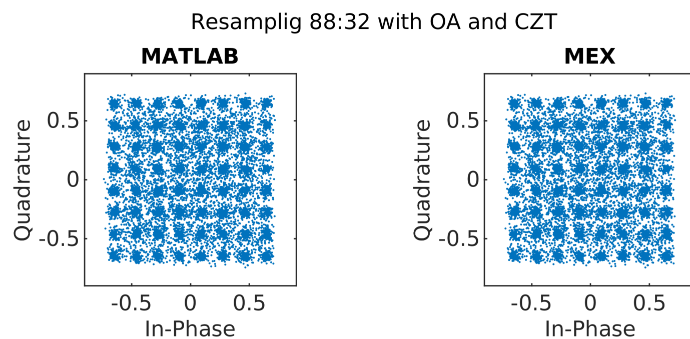
(a) Outside of the lower limit. Partial degradation can be seen.



(b) Inside the resampling limits with peak performance close to the lower limit.



(c) Inside the resampling limits close to the upper limit. Nearly identical to (b).



(d) Outside of the upper limit leading to input data being lost and severely degraded.

Figure 5.1: The upper and lower edge cases for the resampling ratio as defined by (3.18) and the Nyquist criterion, respectively

Table 5.1: Signal-to-noise ratio of the normalized edge cases in OA-CZT

Resampling		36:32	37:32	87:32	88:32
MATLAB	[dB]	33.980	39.128	38.153	20.378
MEX	[dB]	34.190	38.321	38.641	20.367

Table 5.2: Signal-to-noise ratio for OA-FFT

		128p FFT	256p FFT
Resampling		64:32	128:32
MATLAB	[dB]	42.605	44.535
MEX	[dB]	36.772	35.764

5.2 Resource Use Comparison of Mixed-Radix FFT

Using mixed-radix FFT, makes it possible to construct FFT's with a number of inputs that is not only limited to a power of two. However, the designs for radix-3 and radix-5 are far more complicated compared to radix-2, which can be seen in Fig. 4.5. To gain a greater understanding in how much bigger these mixed-radix implementations are, a 32p FFT was compared to its mixed-radix counterpart: 32x3p and 32x5p. The resource utilization for these implementations, post Place-and-Route (P&R), can been seen in Table 5.3. This table shows that mixed-radix-3 uses almost five times as many CLB's and more than six times as many DSP's, compared to the the 32p FFT reference design. The mixed-radix-5 implementation utilizes around nine times as many CLB's and 14 times as many DSP's.

Table 5.3: The available resources and how they were utilized after P&R for different mixed-radix (32p, 96p, 160p) constellations.

Board		FFT 2^5p		FFT $2^5$3p		FFT $2^5$5p	
Logic	Avail.	Util.	[%]	Util.	[%]	Util.	[%]
CLB	53 160	1 341	2.52	6 208	11.68	12 062	22.69
LUT	425 280	6 775	1.59	31 489	7.40	65 289	15.35
FF	850 560	4 901	0.58	23 517	2.76	40 813	4.80
SRL	213 600	160	0.07	480	0.22	1 010	0.47
DSP	4 272	68	1.59	435	10.18	952	22.28

To be certain that a 160p mixed-radix FFT truly is smaller than a 256p radix-2 FFT, the possible radix-3 and radix-5 mixed-radix FFT implementations between 128p and 256p radix-2 FFT's were implemented. The results from these resource utilization reports can be seen in Table 5.4. These results clearly shows that resource usage increases with the number of input points regardless of if the implementation is mixed-radix or pure radix-2, i.e. no radix-2 implementation with a larger number of input points utilize less resources than a mixed-radix implementation with fewer input points.

To further demonstrate how the internal FFT block grow in resource utilization with the number of input points, the graph seen in Fig. 5.2 shows how the number

Table 5.4: The available resources and how they were utilized for regular radix-2 FFT (128p and 256p) compared to split-radix (160p and 192p) constellations.

Board		FFT 128p		FFT $2^5$5p		FFT $2^6$3p		FFT 256p	
Logic	Avail.	Util.	[%]	Util.	[%]	Util.	[%]	Util.	[%]
CLB	53 160	8 239	15.50	12 062	22.69	14 934	28.09	18 871	35.50
LUT	425 280	42 168	9.92	65 289	15.35	73 380	17.25	96 743	22.70
FF	850 560	26 639	3.13	40 813	4.80	53 549	6.30	59 042	6.94
SRL	213 600	360	0.17	1 010	0.47	890	0.42	740	0.35
DSP	4 272	516	12.08	952	22.28	1 075	25.16	1 284	30.06

of DSP slices increase with the number of input points. The number of DSP slices used for the different implementations was deemed a good measure of complexity since the DSP-block is a far more complex resource block compared to e.g. LUT's and FF's. Based on the percentages shown in Table 5.4, it was also deemed that the number of DSP slices would likely be a limiting factor in the available sizes of FFT possible to implement on the target FPGA. The graph in Fig. 5.2 shows two steps where the number of DSP slices increase more drastically and these appear when the number of input points moves from a strict radix-2 implementation to a mixed-radix one, e.g. from 64p to 96p and from 128p to 160p.

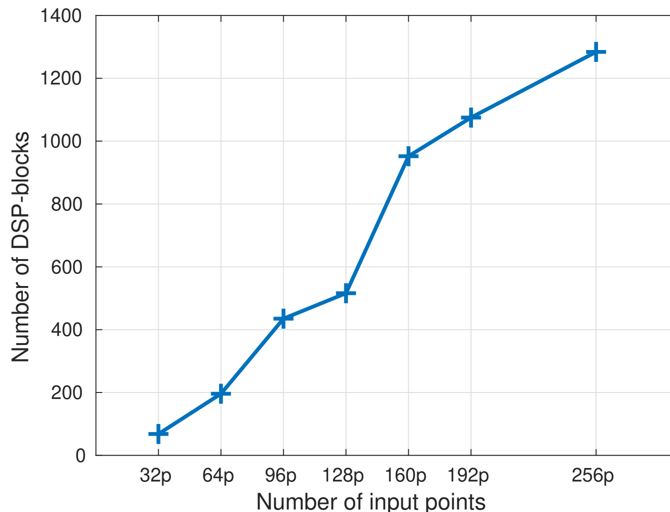


Figure 5.2: How the complexity (measured based on the number of needed DSP slices) scales with the number of inputs. There are a total of 4272 DSP slices available in the target FPGA.

5.3 OA Using FFT vs. Using CZT

The ADCs output 64 samples every clock cycle and to process these with an FFT using a 2:1 resampling ratio, the FFT needs to be 128p wide, according to the Nyquist criterion. Therefore, the implemented OA-CZT-resampling filter, using 64 points, is best compared to a 128p FFT regarding timing and resource utilization.

As previously stated, the implemented OA-CZT-resampling filter can input an arbitrary number of inputs between 37 to 87 and to compare this with FFT there are two implementations needed: 37-64 samples can be processed with a 128p FFT and 65-87 samples must be processed with a 256p FFT. Therefore, Tables 5.5–5.6 shows the resource utilization for both 128p and 256p FFT compared to CZT using 64 or 87 samples.

Table 5.5: The resource utilization for OA-CZT resampling filters

Board		55p CZT		64p CZT		87p CZT	
Logic	Avail.	Util.	[%]	Util.	[%]	Util.	[%]
CLB	53 160	35 251	66.31	40 475	82.88	41 698	78.44
LUT	425 280	150 464	35.38	168 883	41.03	172 504	40.56
FF	850 560	234 247	27.54	257 791	31.64	268 113	31.52
SRL	213 600	14 836	6.95	8 000	3.75	14 087	6.60
DSP	4 272	2 503	58.59	3 798	91.34	3 913	91.60

Table 5.6: The resource utilization for OA-FFT resampling filters

Board		128p FFT		256p FFT	
Logic	Avail.	Util.	[%]	Util.	[%]
CLB	53 160	22 529	42.38	35 944	67.61
LUT	425 280	78 748	18.52	143 055	30.77
FF	850 560	124 008	14.58	234 913	27.62
SRL	213 600	6 328	2.96	12 200	5.71
DSP	4 272	1 072	25.09	1 806	42.28

A OA-CZT resampling filter with a maximum input width of 55 points was also presented as a smaller alternative that might be able to compete with the OA-FFT in size. The smaller size is due to the two internal FFTs that were changed from 160p to 128p. The resource utilization for this implementation can also be seen in Table 5.5. To give a more comprehensive comparison for the size differences, all these resource utilizations are shown in Fig. 5.3.

5.4 Throughput

Another trade-off that is made when choosing between the 87p-input CZT version and the 64p-input version involves the peak throughput. The throughput is given by: *input/output samples* × *frequency*. In this implementation the target clock frequency is 250 MHz (4 ns clock period), which results in a maximum throughput of 21.75 GSPS when the input is 87 samples per clock cycle and 16 GSPS when the input is 64 samples. The achieved clock periods for the different implementations can be seen in Table 5.7 and the calculated throughput can be seen in Table 5.8.

There were no problems with achieving the designated clock period of 4 ns for the OA-FFT implementations. However, it was harder for the OA-CZT implementations to be routed effective enough for the timing constraint to be achieved. Therefore

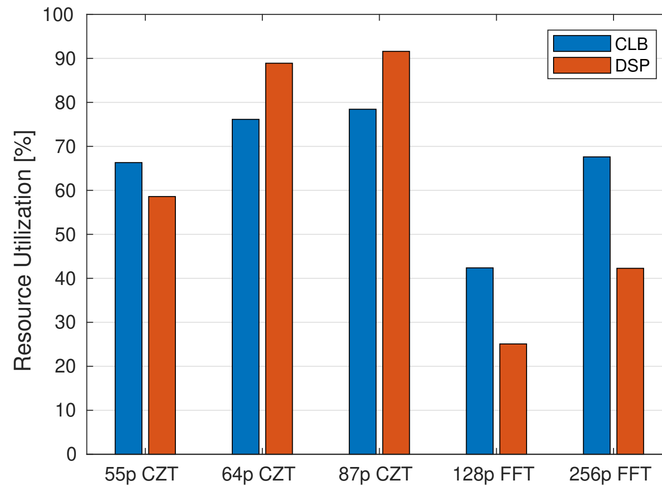


Figure 5.3: The resource utilization for different configurations of OA with either CZT or FFT

the target clock period had to be lowered to 3.25 ns for the implementation to successfully route an implementation that could operate within 250 MHz.

Table 5.7: The timing criteria and performance of the different OA resampling filter configurations.

		CZT			FFT	
		55p	64p	86p	128p	256p
Clock period target	[ns]	3.250	3.250	3.250	4.000	4.000
Clock period post-synth.	[ns]	3.091	3.906	3.906	3.091	3.091
Clock period post-impl.	[ns]	3.391	3.877	3.920	3.869	3.873

Table 5.8: The throughput on both input and output of the different OA resampling filter configurations.

		CZT			FFT	
Direction		55p	64p	86p	128p	256p
Input	[Gsps]	13.75	16.0	21.75	16.0	32.0
Output	[Gsps]	8.0	8.0	8.0	8.0	8.0

5.5 Power Usage

Another important aspect of a design is how much power it draws since this both contributes heat to the overall system as well as increasing energy usage. However, power is hard to estimate without running the design in a real-life setting and hence the presented reports only constitute an indication of what the power usage will be. The power reports, see Table 5.9, used for this comparison have been produced by Vivado after the designs were P&R.

Table 5.9: The power usage of the OA-CZT implementation and the reference case.

Implementation	Power [W]
OA-CZT 87p	35.662
OA-FFT 128p	9.878

The `report_power` tool had no information about the switching in the design so the built in *vectorless propagation engine* was used. This engine assigns activity to all inputs randomly and switches them until some signal propagates to the output ports. The reference case and the CZT both present values that are proportional to the number of DSP48E2 slices. This claim is backed up by the fact that OA-CZT uses close to four times as much power and DSP-slices as OA-FFT.

6
Discussion

Three variations of the OA-CZT implementation were implemented and compared to the two reference cases using OA-FFT. All of these were within the timing and resource constraints of the system and showed the weaknesses and strengths of both the reference case and the CZT implementation. This chapter will further discuss some of these strengths and weaknesses.

6.1 FFT Size and Complexity

The reason for settling at a 160p mixed-radix FFT is to keep the design to a minimum. It is possible to build a mixed-radix that is even closer to 137p by constructing, e.g. $2 \times 5^2 \times 3 = 150$. However, both implementations for radix-3 and radix-5 are far more complex compared to radix-2. So to save complexity and resources, it was preferred to utilize radix-2 as far as possible in the mixed-radix implementation.

In this implementation a mixed-radix FFT is composed of two parts: the first one is the 2^n radix-2 component and the second is either one set of radix-3 or of radix-5 components. In reality it is possible to build mixed-radix FFT with several powers of radix-3 and radix-5. Since radix-3 and radix-5 are larger implementations than radix-2, it is unknown if there is a mixed-radix implementation utilizing a majority of radix-3 or radix-5 FFT's that has fewer input points and still generates a larger overall implementation compared to a pure radix-2 implementation with a higher number of inputs. Therefore, it is important to note that the results shown in Fig. 5.2 may not be true for all cases, as the graph indicates that it is always the FFT implementation with the highest number of input points that is the largest implementation.

The reference designs are using pure radix-2 FFT's to keep the design as simple as possible and because it works with the sampling ratios, but they could might as well had used mixed-radix FFT too.

6.2 Error Sources

As stated in Section 4.1, the CZT transform does not compute all the points for the RRC filter, hence avoiding unnecessary calculations. However, it should be noted that the near-zero values that get discarded when transforming the RRC filter do contain information due to the finite impulse response. The fact that this part is

assumed to be perfectly zero creates a source of distortion as a matched RRC filter, without perfect zeros, is used when sending the signal. This distortion can be seen in the SNR for the different MATLAB models in Tables 5.1–5.2. Furthermore, it can be noted that the SNR between the MATLAB and MEX in Table 5.2 differs by 1-2 dB. The difference in SNR emerges because these designs were not optimized in regards to scaling, meaning they might be losing precision due to the fixed point quantization.

Another error source in the system is quantization noise, where most of the noise will come from the ADCs as they only have 14-bit resolution. Nevertheless, the finite resolution of the remaining system will also affect the final output as it sets the limits for the precomputed values. However, the ADCs are not modeled in this system and hence, this error will only be noticeable once the system is implemented on real hardware.

6.3 HLS Optimization

Even though the user guide for Xilinx HLS was consulted very often, it was not possible to test all the pragmas. The pragmas that seemed the most relevant for the design were, of course, used. Nevertheless, there might be directives that would have improved the performance in other aspects, such as timing, that were missed. One interesting command was the `#PRAGMA HLS INLINE`, which disassembled the whole design and implemented it without a hierarchy. This pragma made it possible for slices to be shared between functions, but impossible to understand which part of the design used which resources. It was also perceived that it took much longer for an `INLINE` design to synthesize than a hierarchical design. However, the low initiation interval made all blocks of the design work constantly, leaving no room for resource sharing. Hence, the `INLINE` pragma made a very marginal difference on the number of multipliers.

Another option for improving the implemented hardware is to rewrite parts of the C++ code so that it enables new ways of optimization. This rewrite approach was attempted when creating the radix-5 FFT as some of the multiplications in that file would always evaluate to 1 or 0. So these multiplications were removed and replaced with constants. Surprisingly, the optimized implementation showed very little difference in utilization compared to the original implementation, indicating that the HLS compiler already performed these changes.

The design could also be influenced with functions from the HLS library, where two were of particular interest for this design. `CmpyFourMult` and `CmpyThreeMult` made it possible to chose between two complex multiplications methods

- `CmpyFourMult` uses 4 multipliers and 2 adders
- `CmpyThreeMult` uses 3 multipliers and 5 adders

As the designs initiation interval needs to be low, it is required to be completely parallel, which becomes a problem due to the high number of multiplications in the algorithm. Hence, choosing the right multiplication can be the difference between the design fitting the board or not. `CmpyThreeMult` was used for the implementations

in this thesis as that resulted in the smallest number of multipliers, without notably degrading the signal-to-noise ratio.

Finally, Vivado HLS slowed down the compilation significantly when the designs started to take up close to the whole FPGA. This stagnation indicates that implementing the design in smaller pieces and putting these together might be an interesting option to try. In addition to being easier for the tool, it might also facilitate the optimization as smaller blocks are easier to review and compare to the design goals.

6.4 DSP Slice Utilization

As all of the multiplications that are instantiated in the design use `CmpyThreeMult`, it is expected that each complex multiplication would use three DSP48E2 slices. Looking at the design checkpoints generated by Vivado HLS, it is apparent that the multiplications are not instantiated correctly as many of them use more than three DSP48E2 slices. When constructing the system presented in Fig. 4.6, the expected number of multipliers can be seen in Table 6.1.

Block	Mults
Pre-mult.	**261**
- mult	261
Conv.	**2360**
- FFT	940
- mult	480
- iFFT	940
Post-mult.	**222**
- mult	222
OA	**192**
-iFFT	192
Sum	3035

Table 6.1: The expected number of DSP48E2 slices that the OA-CZT implementation should utilize

However, the OA-CZT implementation used 3913 DSP slices, as seen in Table 5.6. A possible cause for the high number of DSP slices is that the pragmas for the HLS synthesis had side effects which implemented the HDL design differently than what was assumed. Another explanation can be that the complex multiplications is performed by DSP slices that utilize adders only, and therefore needs five DSP slices to perform one complex multiplication.

6.5 Downsampling Factor

In the implementation presented in this thesis the CZT system is limited to values between 37 and 87 samples. These limits can, however, be moved by changing different parts of the design:

- The upper limit can be changed by switching the size of the FFT component.
- The lower limit can be adjusted with the RRC filter.

Nevertheless, this implementation is the smallest achieved OA-CZT resampling filter that is comparable to an OA-FFT(using 128p FFT) resampling filter and it still uses close to four times as many DSP48E2 slices. The design can, however, manage higher downsampling factors than a static OA-FFT and change resampling ratio. The higher resampling ratio also offers higher throughput and comparable timing. In addition to this, the flexibility makes it possible to use one clock to support multiple transmission rates, that would otherwise require the clock to change frequency. As the same clock frequency can cater several transmission rates, the hardware only needs to be verified for this frequency. In the current implementation, without a flexible buffer, this is possible to do by changing the number of samples that are forwarded to the block.

One example would be if a transmission system was designed around a frequency of 250 MHz with 64 symbols arriving every clock cycle that, for some reason, had the transmitter side frequency decreased by 1/8 to 218.75 MHz. With a fixed downsampling factor, this would require the clock rate on the receiver side to decrease as much, but with flexible downsampling the receiver can continue running at 250 MHz but resample with a factor of 56 instead. As the only thing that has changed in the system is the precomputed values, the analog parts are not affected, meaning they only need to be verified for a single clock rate. An important note about the previous example is that it assumes that there is a flexible buffer available that can serve any data width within the limits of the resampling system. Such a flexible buffer was investigated in this thesis, but unfortunately not finalized. The following section will describe the problems that make the implementation of such flexible buffer difficult.

6.6 Flexible Buffer

A constraint given by the project owner was that the initiation interval, i.e. the number of clock cycles between new data, should be 1. An inherent property of a radio communication system is that the data in the channel cannot be put on hold and hence the initiation interval will always need to be low. This requirement demands a lot of parallelism, which introduces resource problems on the FPGA since the data buses will be very wide. The ADC will be providing 64 complex samples, where every sample is represented by 36 bits (18 bits for the real and 18 bits for the imaginary part), with every clock cycle. The reason for having 18 bit numbers when the ADCs only delivers 14 bit samples are:

- to have some precision left when the numbers grow large and need to be scaled down, and
- the limiting factor is still the DSP blocks.

To handle this, a bus width of $64 \times 18 \times 2 = 2304$ needs to be written to memory on every rising clock edge.

The resampling filter, using CZT, will then input an arbitrary number between 37 and 87 samples every cycle, giving a maximum output bus width of $87 \times 18 \times 2 = 3132$. The flexible number of inputs means that the resampling filter will input either too few or too many samples most of the time. This demands some sort of buffer between the ADC and the resampling filter to ensure that no samples from the channel are lost.

The buffer needs to be dimensioned to hold at least three sets of 64 complex samples to accommodate the worst case. This is because when the CZT filter needs to be loaded with 87 samples but there are less than $87 - 64 = 23$ samples left from the calculation in the cycle before. This will total up to a buffer size of at least $22 + 64 + 64 = 86 + 64 = 150$ samples that needs to hold $150 \times 18 \times 2 = 5400$ bits. During this time when the buffer is loading up with enough samples, a valid signal will not be sent to the subsequent blocks making them hold and wait until there is available data again.

To always have the latest samples at the top of the buffer there are different approaches that can be taken. One is the first in, first out method that always shifts up the samples to be at the top of the buffer, and the other is to use a circular buffer. The circular buffer could have been a nice implementation if the demand for initialization interval was not as high, but because of it any position in the buffer must be possible to output at any time, requiring lots of multiplexing. The first in, first out, on the other hand will not suffer from the same multiplexing problem due to the upper samples always being output. However, the latter type of buffer must be able to move all the samples between 37 and 87 places every clock cycle. The first in, first out approach seemed the most promising and an attempt to implement it was made but never finished.

The flexible buffer will also affect the throughput of the total system. The calculated values of throughput, in chapter 5.4, are true for the OA-CZT block itself, but when the OA-CZT is incorporated in a larger system, the throughput for that system will be different. The ADCs will probably be a limiting factor. For the constraints in this thesis where the ADCs outputs 64 samples each clock cycle, it would require two clock cycles to load OA-CZT with 87 samples. Similarly, the 55p CZT design would not be viable either, as the buffer would never stop growing when it outputs less than it inputs.

7 Conclusion

This thesis set out to create a flexible resampling filter using overlap-add (OA) and chirp-Z transform (CZT), which was successfully achieved. We conclude that it is possible to implement a flexible resampling filter that can handle an arbitrary number of inputs within two limits. The limits are set based on two parameters that are controlled at hardware creation: the transform length and the output length. The upper limit is set by the transform that is used within the CZT, as that needs to be longer than the sum of the input and output lengths minus one. As a 160p mixed-radix FFT is used and 74 output points are required, the upper output limit becomes 87 samples. The bottom limit is set by the Nyquist criterion of the output length from the CZT, which results in 37 samples in the case demonstrated in this thesis. Within these limits, the number of inputs can be chosen arbitrarily and then be zero padded to the upper limit.

A reference case was also created, which used a static resampling ratio of 2:1 and was built using regular FFTs. When the flexible OA-CZT implementation is compared to the static OA-FFT reference implementation, it is concluded that the cost of this flexibility is that the design uses four times as many DSP48E2 blocks. It should, however, be noted that this increased resource usage also gives increased downsampling factor as the CZT version offers near three to one downsampling. The design is able to run at the same desired clock frequency of 250 MHz, while achieving comparable throughput as the 2:1 downsampling reference case. Furthermore, the power usage is observed to scale with the number of DSP blocks as the CZT implementation used 36 W, compared to the reference designs 10 W.

The limiting factor for this resampling filter is the buffering of the values to be handled, which is caused by the massively wide buffer that is needed. Some effort was put into creating such a buffer but due to shortage of time that work was discontinued and instead encouraged as future work as that would enable runtime flexible resampling. Furthermore, the design has never been tested on the target hardware as that was the activity with the lowest priority from Ericsson's perspective.

Bibliography

[1] R. Möller, "Ericsson mobility report 2021," Ericsson, Stockholm, Sweden, Tech. Rep. EAB-21:010887 Uen, 2021.

[2] R. C. Qiu, Z. Hu, H. Li, and M. C. Wicks, "Introduction," in *Cognitive Radio Communications and Networking.* John Wiley & Sons, Ltd, 2012, ch. 1, pp. 1–13. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118376270.ch1 [Accessed: 2022-06-01]

[3] M. Hunukumbure, J. P. Coon, B. Allen, and T. Vernon, "Spectrum – the life blood of radio communications," in *The Technology and Business of Mobile Communications: An Introduction.* Hoboken, NJ, USA: John Wiley & Sons Ltd., 2022, ch. 7, pp. 219–240.

[4] M. Renfors, M. Juntti, and M. Valkama, "Signal processing for wireless transceivers," in *Handbook of Signal Processing Systems*, S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds. Cham, Switzerland: Springer International Publishing, 2019, pp. 251–310. [Online]. Available: https://doi.org/10.1007/978-3-319-91734-4_8 [Accessed: 2022-01-27]

[5] J. B. J. Fourier, *The Analytical Theory of Heat*, ser. Cambridge Library Collection - Mathematics, A. Freeman, Ed. Cambridge, UK: Cambridge University Press, 2009.

[6] E. W. Hansen, "The discrete Fourier transform," in *Fourier Transforms: Principles and Applications*, 1st ed. John Wiley & Sons, Inc., 2014, ch. 3, pp. 109–164.

[7] M. Borgerding, "Turning overlap-save into a multiband mixing, downsampling filter bank," *IEEE Signal Processing Magazine*, vol. 23, no. 2, pp. 158–161, Mar 2006.

[8] L. Bluestein, "A linear filtering approach to the computation of discrete Fourier transform," *IEEE Transactions on Audio and Electroacoustics*, vol. 18, no. 4, pp. 451–455, Dec 1970.

[9] L. R. Rabiner, R. W. Schafer, and C. M. Rader, "The Chirp Z-transform algorithm and its application," *The Bell System Technical Journal*, vol. 48, no. 5, pp. 1249–1292, May 1969.

[10] C. E. Kim and M. G. Strintzis, "High-speed multidimensional convolution," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, no. 3, pp. 269–273, May 1980.

[11] Xilinx, Inc, “Vivado 2021.2 - High-Level Synthesis (C based),” 2022. [Online]. Available: https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0012-vivado-high-level-synthesis-hub.html [Accessed: 2022-01-10]

[12] ——, “Zynq UltraScale+ RFSoC ZCU111 evaluation kit,” 2022. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/zcu111.html [Accessed: 2022-01-08]

[13] MathWorks Inc., “”C++ with MATLAB”,” 2022. [Online]. Available: https://se.mathworks.com/help/matlab/cpp-language.html [Accessed: 2022-03-29]

[14] C. Shannon, “Communication in the presence of noise,” *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, Jan 1949.

[15] J. Anderson, *Digital Transmission Engineering*, ser. Digital and Mobile Communication Series. United States: IEEE - Institute of Electrical and Electronics Engineers Inc., 2005.

[16] E. Armstrong, “A new system of short wave amplification,” *Proceedings of the Institute of Radio Engineers*, vol. 9, no. 1, pp. 3–11, Feb 1921.

[17] P.-I. Mak, S.-P. U, and R. P. Martins, “Transceiver architecture selection: Review, state-of-the-art survey and case study,” *IEEE Circuits and Systems Magazine*, vol. 7, no. 2, pp. 6–25, Sep 2007.

[18] D. Sundararajan, *Digital Signal Processing: An Introduction*. Cham, Switzerland: Springer, 2021.

[19] F. J. Harris, “Resampling filters,” in *Multirate Signal Processing for Communication Systems (2nd Edition)*. River Publishers, 2021, ch. 7. [Online]. Available: https://app.knovel.com/hotlink/khtml/id:kt012KW6UC/multirate-signal-processing/multirate--references [Accessed: 2022-06-11]

[20] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge, UK: Cambridge University Press, 2017.

[21] J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series,” *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, Apr 1965.

[22] S. Winograd, “On computing the discrete Fourier transform,” *Proceedings of the National Academy of Sciences*, vol. 73, no. 4, pp. 1005–1006, Jan 1976.

[23] L. F. Chaparro and A. Akan, “Chapter 11 - discrete Fourier analysis,” in *Signals and Systems Using MATLAB (Third Edition)*, 3rd ed., L. F. Chaparro and A. Akan, Eds. Academic Press, 2019, ch. 11, pp. 637–720. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128142042000223 [Accessed: 2022-03-03]

[24] J. A. Fernandez and B. V. K. V. Kumar, “Multidimensional overlap-add and overlap-save for correlation and convolution,” in *2013 IEEE International Conference on Image Processing*, Melbourne, VIC, Australia, 2013, pp.

509–513. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6738105 [Accessed: 2022-02-08]

[25] J. Löfgren and P. Nilsson, "On hardware implementation of radix 3 and radix 5 FFT kernels for LTE systems," in *2011 NORCHIP*, Lund, Sweden, 2011, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/6126703?reload=true [Accessed: 2022-03-14]

[26] W. Li, M. Vesterbacka, and L. Wanhammar, "An FFT processor based on 16-point module," in *Proc. of NorChip Conf*, Linköping, Sweden, Nov 2001, pp. 125–130. [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A272329&dswid=2494 [Accessed: 2022-06-02]

[27] Xilinx, Inc, "UltraScale Architecture DSP Slice," 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1 [Accessed: 2022-04-12]

[28] ——, "UltraScale Architecture Configurable Logic Block," 2017. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug574-ultrascale-clb [Accessed: 2022-06-04]