Hardware-Efficient Root Value Estimation for Zadoff-Chu Sequences of Known Length

Hampus Eriksson Rygaard
ha0518er-s@student.lu.se
Marcus Forsberg
ma6735fo-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Johan Thunberg, EIT, LTH

Co-Supervisor: Per Andersson, EIT, LTH

Examiner: Pietro Andreani

June 11, 2026

© 2026
Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

In wireless communication, Zadoff-Chu (ZC) sequences are often used for synchronisation between transmitters and receivers due to their advantageous mathematical properties. They are extensively used in LTE and 5G systems, as well as for UAV communications and various other applications. Blind detection of ZC sequences, meaning that their structural parameters are unknown, has applications for security and privacy. However, blind ZC sequence detection is computationally expensive, especially for longer sequences, restricting deployment to centralised processing platforms.

This thesis investigates hardware efficient algorithms for the detection of ZC sequences for deployment on resource-constrained edge devices. We simulated and evaluated three algorithms: time-based cross-correlation, fast cross-correlation, and phase difference estimation. The results demonstrated that both cross-correlation algorithms achieved acceptable performance down to a Signal-to-Noise Ratio (SNR) of -14 decibels. However, the time-based cross-correlation consumed 10 times the energy, rendering it impractical. In contrast, the phase difference estimation algorithm consumed 10 times less energy than the fast cross-correlation algorithm, but the acceptable performance was limited to SNRs above -6 dB.

To further reduce energy consumption, these algorithms were also modified. This yielded significant power savings proportional to the modification degree; however, performance decreased accordingly. Consequently, these trade-offs demonstrate that the optimal algorithm and modification configuration depend on the intended SNR environment, providing a dynamic low-power solution.

Keywords: Zadoff-Chu sequences, Signal detection, Energy efficiency.

Popular Science Summary

When communicating wirelessly, a fundamental challenge is determining exactly when a message starts and stops. If the receiver is off by even a fraction of a second, the entire message can be corrupted. To prevent this, systems use a digital handshake called a synchronisation sequence. This sequence tells the receiver exactly when to start listening, synchronising it to the transmitter.

While many types of synchronisation sequences exist, Zadoff-Chu sequences are widely used. For example, in drone communications, they are used to synchronise and stabilise live video footage. These sequences can both vary in length and shape, providing a large set of unique sequences. Because an operator and device share a known sequence, they can easily find each other. However, this synchronisation sequence also acts as a digital flag. Surveillance and security systems can use this flag to detect unauthorised devices and potentially intercept the data transmitted.

The catch is that these security systems don't know what specific sequence the devices use. So, to detect a rogue device, they need to test a large number of possibilities. This process is so computationally expensive that the detection systems are bound to heavy stationary computer systems such as desktop PCs or servers. Bringing these security systems onto lightweight portable devices like microcontrollers is difficult.

Therefore, this thesis investigates alternative algorithms and modifications to reduce computational requirements. Through simulations, this study maps out a range of flexible options in which detection performance is traded for reduced computational requirements. Ultimately, this framework allows the user to select the exact performance required for their device, minimising excess compute.

Acknowledgments

We wish to express our deepest gratitude to our supervisor, Johan Thunberg, and co-supervisor, Per Andersson, at EIT, LTH, for their continuous guidance and insights. Furthermore, we would like to thank our industrial supervisors for sharing their time and expertise. They have been a tremendous help throughout the course of this project. Lastly, we would like to thank family and friends for their support.

Table of Contents

List of Figures

Chapter 1
Introduction

1.1 Background

In wireless communication, synchronisation between transmitter and receiver is crucial. The receiver needs to detect when signal transmission begins for communication to be possible. This is commonly achieved by using a synchronisation sequence before or during transmission. For example, in Orthogonal Frequency Division Multiplexing (OFDM), Zadoff-Chu (ZC) sequences can be utilised as a synchronisation sequence. ZC sequences are widely adopted due to their ideal correlation properties, allowing precise timing acquisition even in dynamic conditions [1]. The two defining parameters of ZC sequences, sequence length and root value, determine their auto- and cross-correlation properties. For a given sequence length N, the root value can take the values $q = 1, 2, ..., N-1$. Each root value constructs a unique ZC sequence of length N.

This thesis addresses the problem of identifying a ZC sequence and its root value, as well as finding its position within a longer noisy signal sequence. For short ZC sequences, this is relatively simple, but the complexity quickly increases with increasing sequence length. Not only does the possible number of root values increase with sequence length, but so does the computational cost required to evaluate for each such. In standardised OFDM-based systems like LTE and 5G, where the possible root values are predefined, the overall problem is reduced to identifying the ZC sequence location. However, for non-standard OFDM-based systems, the possible sequence lengths and root values are unknown at identification, necessitating novel approaches to overcome the high computational cost of conventional search methods.

1.2 Motivation

These non-standard OFDM systems are, for example, used by Unmanned Aerial Vehicle (UAV) vendors [2]. Unauthorised UAV activity poses significant risks to critical infrastructure, public safety, and privacy, making real-time detection and monitoring essential for airspace security and regulatory compliance. However, the high computational cost of detecting non-standard OFDM signals limits real-time detection to centralised processing platforms such as high-end desktop PCs or dedicated servers.

Enabling the capability of performing ZC detection and root estimation on edge devices is an important step toward providing more flexible and responsive risk management. However, for this to be feasible, the constraints on power-restricted edge devices must be addressed.

While UAV detection serves as a motivating example, it reflects a broader challenge in making non-standard OFDM signal detection possible on resource-constrained edge devices. To address this challenge, hardware-accelerated algorithms that meet strict compute and power constraints while maintaining acceptable detection performance are needed.

1.3 Previous works

Several studies have addressed ZC sequence detection, and the detection method outlined in [3] serve as a basis for this thesis. Although they use a similar detection method to one of those explored in this thesis, they do not consider the energy-related computational cost required for hardware-level algorithm execution. Furthermore, the study assumes that a ZC sequence of known root appears with a set time offset from the ZC sequence with an unknown root. This simplifies the problem since the known-root ZC sequence can be used to acquire timing before performing root estimation of the unknown-root ZC sequence.

The method outlined in [4] shows that ZC sequence detection and, more importantly, root estimation can be performed at a very low computational complexity. This is done by differentiating the signal twice. Doing this to a ZC sequence transforms it into a constant, where the phase depends on the root. Finding the average phase of a signal containing a ZC sequence, therefore, approximates the root. This method is incredibly efficient computationally, avoiding the need to compare the signal with ZC sequences of different roots entirely. It does have its drawbacks, however. The main issue is that the method only works for low-noise scenarios. In noisy environments, it cannot reliably estimate the root of the ZC sequence.

1.4 Project Aims

This thesis aims to develop and simulate hardware-efficient methods for detecting ZC sequences of known length by selecting prominent base algorithms and improving those with power-saving modifications. These modifications reduce energy use by either decreasing the number of operations or by lowering the cost per operation.

In this thesis, the proposed modifications are primarily made with Application Specific Integrated Circuit (ASIC) implementation in mind. Nonetheless, some of the proposed modifications are generic and can be applied to a variety of realisation platforms, such as Field Programmable Gate Arrays.

1.5 Thesis structure

This thesis is structured as follows:

- **Chapter 2: Theoretical Background -** Contains theoretical background to follow the thesis.
- **Chapter 3: Problem Formulation -** Presents the detection task and assumptions made.
- **Chapter 4: Methodology -** Explains the algorithms and modifications as well as how they were tested and evaluated.
- **Chapter 5: Results -** Presents the results in figures and tables.
- **Chapter 6: Discussion -** Analyses the results, highlights promising algorithms and modifications and discusses limitations of the study.
- **Chapter 7: Conclusion and Future Works -** Summarises the findings and covers possible future works.

Chapter 2
Theoretical Background

This section introduces the background information necessary to understand the later sections of the thesis. It covers Zadoff-Chu sequences, the basics behind signal detection, dynamic power, methods to reduce power consumption, and performance metrics used for the evaluation.

2.1 Zadoff-Chu Sequences

ZC sequences are sequences of complex values that have specific properties which make them useful for wireless communications protocols. They are widely adopted for many purposes in LTE and 5G communications [1]. ZC sequences belong to a family of sequences called Constant Amplitude Zero Autocorrelation (CAZAC) sequences.

The general formula for an odd-length ZC sequence $s_q[n]$ is

$$s_q[n] = \exp\left(-j\pi q \frac{n(n-1)}{N_{zc}}\right) \tag{2.1}$$

, where the odd number $N_{zc} \in \mathbb{N}$ is the length of the sequence,$n = 0, 1, ..., N_{zc} - 1$ is the sequence index, and $q = 1, 2, ..., N - 1$ is the root value. Notation may vary throughout literature (q is commonly replaced with either u or r and q may occasionally be used to explicitly describe a cyclic shift), but this is the notation that will be used throughout the thesis. The first of the two properties that CAZAC sequences share, constant amplitude, means that $|s_q[n]| = 1$ for all values n. The second, zero autocorrelation, means the correlation between two identical sequences will be non-zero only when the cyclic shift is 0 [1]. A cyclic shift of m means the indexing changes, with n being replaced by $(n + m) \mod N_{zc}$ in the formula.

Another useful property for ZC sequences is their ideal cross-correlation property. Two ZC sequences with equal length N_{zc} and different root values q_1 and q_2 will have a constant cyclic cross-correlation of $\sqrt{N_{zc}}$. Note that this property only holds when $|q_1 - q_2|$ and N_{zc} are coprime, meaning the only natural number evenly dividing both numbers is 1 [1]. For this reason, prime length ZC sequences are commonly used. This way, all root value pairs will be coprime to N_{zc}.

2.1.1 Mathematical Observations of Zadoff-Chu Sequences

Taking the phase difference of a ZC sequence $s_q[n]$ will result in a complex sinusoid with linear phase. This is shown in the equation below, where $\omega = \frac{\pi q}{N}$.

$$d_q[n] = s_q[n]\overline{s_q[n-1]} = e^{-j\omega n(n-1)+j\omega(n-1)(n-2)} = e^{-2j\omega n} \tag{2.2}$$

From the expression, it can be seen that the complex sinusoid has a frequency defined by $2\omega = 2\frac{\pi q}{N}$. This frequency scales linearly with the root q and is inversely proportional to the sequence length N. Consequently, if the sequence length is known, the set of all possible frequencies can be pre-calculated.

Note that this property holds for other signals with quadratic phase $z[n] = e^{j(an^2+bn)}$ and is thus not unique for ZC sequences.

$$e^{j(an^2+bn)} * e^{-j(a(n-1)^2+b(n-1))} = e^{j(a(2n-1)+b)} \tag{2.3}$$

2.2 Cross-Correlation

Cross-correlation is a mathematical operation that provides a measure of similarity between two time signals at different time offset τ. Cross-correlation between two continuous signals f and g is defined as

$$(f \star g)(\tau) = \int_{-\infty}^{\infty} \overline{f[t]}g[t+\tau]dt \tag{2.4}$$

, where t is the time and $\overline{f[t]}$ is the complex conjugate of $f[t]$. Larger cross-correlation values mean a higher degree of similarity between the two signals.

When cross-correlating finite, discrete functions, t and τ are replaced with discrete indices m and n respectively, modifying the equation to a non-cyclic correlation

$$(f \star g)[n] = \sum_{m=0}^{M-1} \overline{f[m]}g[m+n] \tag{2.5}$$

which is valid for values of n such that the indices $m+n$ remain within the valid range of g. Here, M is the length of f, which is assumed to be the shorter sequence. This equation can be interpreted as a dot product between the complex conjugate sequence $\overline{f[m]}$ and a shifted window of g of length M, where n is the offset shift. The offset can be seen as an index for where a window on g should be placed, which determines where the dot product is computed. By sweeping over all offsets $n = 0, 1, ..., L-1$ (sliding the window), a vector is created where each index represents an offset. The index with the highest value indicates which offset yields the maximum overlap. Correlating all possible offsets becomes computationally expensive since a dot product of length N is computed for each possible offset, resulting in a computational complexity of $O(NL)$.

When working with complex values, such as ZC sequences, the cross-correlation for each offset is a complex number. If either f or g is phase-shifted, as is often the case for real signals, the resulting cross-correlation will also be phase-shifted by

the same amount. However, the magnitude will be unaffected. It is therefore advantageous to take the magnitude of the cross-correlation as it yields the similarity independent of phase differences.

2.3 Fourier Transform

The Fourier transform is a mathematical operation, denoted as $\mathcal{F}$, which converts a time domain signal into its frequency domain representation [5]. It is based on the fact that any signal which is well-behaved in the sense that a Fourier transform exists can be transformed using the formula

$$\mathcal{F}(f(t)) = F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt \tag{2.6}$$

, where $f(t)$ is the signal value at time t and ω is the angular frequency. The resulting output $F(\omega)$ is a complex-valued function that describes the amplitude and phase for each frequency wave component [5].

According to Euler's formula, $e^{-j\omega t}$ can be expressed as $cos(\omega t) - jsin(\omega t)$. Therefore, $f(t)e^{-j\omega t}$ can be seen as an operation that measures how well each point in the time domain matches its corresponding time point on the cosine and sine wave. If the points match well, $F(\omega)$ will have a large magnitude, and it means there is a high similarity between the input signal and the complex exponential basis. Essentially, the magnitude of $F(\omega)$ shows how much each frequency contributes to the signal.

2.3.1 Inverse Fourier Transform

There is also a mathematical operation called the inverse Fourier transform, denoted as $\mathcal{F}^{-1}$, which converts a frequency domain representation back into its time domain signal. It is given by

$$\mathcal{F}^{-1}(F(\omega)) = f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}d\omega \tag{2.7}$$

, see [5].

2.3.2 Discrete Fourier Transforms

In real-life applications, it is not possible to sample a signal from $-\infty$ to ∞, nor is it possible to sample with infinitely small time steps, i.e., infinite sampling frequency. To still be able to use the Fourier transform and inverse Fourier transform on sampled signals, different formulas suited for discrete and finite signals have to be used. The discrete Fourier transform is defined, see [6], as

$$F_k = \sum_{n=0}^{N-1} f_n e^{-j2\pi \frac{k}{N} n} \tag{2.8}$$

, where N is the number of samples, f_n is the value of sample n, and k is the number of cycles per N samples.

Similarly, the discrete inverse Fourier transform is defined as

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k e^{j2\pi \frac{k}{N} n} \tag{2.9}$$

Implementing Equation (2.8) and Equation (2.9) as stated would result in a time complexity of $O(n^2)$. However, thanks to the symmetry in the formulas, it is possible to calculate the discrete transforms with a time complexity of $O(n \log(n))$ [7]. Algorithms utilising this have been given the fitting names of fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT). An excellent example of this is the radix-2-Cooley-Tukey algorithm, which uses a divide-and-conquer approach to compute the DFT and is used in this thesis. A discrete-time signal of length N (where N is a power of two) is first divided into groups of two. The algorithm then processes data through a series of stages. In each stage, the odd and even indices within the groups are paired. For each pair, the odd-indexed element is multiplied by a twiddle factor, which is a complex phase constant with an amplitude of one. This modified odd-index element is then both added to and subtracted from the paired even-indexed element. The resulting sum becomes the new even-indexed element, while the difference becomes the new odd-indexed element. These elements are then merged with the neighbouring group, doubling the group size to 4, passing as inputs to the next stage, where the process repeats. This recursion stops once there are no more groups to combine. Each stage requires N additions and N/2 multiplications. Because the group size doubles each stage, the total number of stages equals $\log_2(N)$ [8]. If the original sequence length is not a power of two, the signal can be padded with zeros to the nearest power of two

2.3.3 Fast Cross-Correlation

The cross-correlation for all offsets of two time domain signals can be calculated by multiplying the corresponding Fourier transforms, where one has been conjugated, and taking the inverse Fourier transform [9]. The formula for this is

$$f(t) \star g(t) = \mathcal{F}^{-1}(\overline{F(\omega)} \cdot G(\omega)) \tag{2.10}$$

It shows how a complex mathematical operation in the time domain can be simplified to multiplication in the frequency domain. In the time domain, the time complexity would be $O(n^2)$, but by using the relationship above in combination with an FFT and IFFT for the transforms, the time complexity can be reduced to $O(n \log(n))$.

2.4 Power Consumption

Power consumption in digital electronics consists of two components, static power and dynamic power. Static power is constant and is primarily due to transistors not fully turning off, causing a leaking current. Dynamic power is the power dissipated when the capacitive load switches between two voltage states, high to

low or vice versa. It depends on the clock frequency f, the supply voltage V_{dd}, the capacitance C, and the activity ratio a. Accordingly, dynamic power can be expressed as

$$P_{dynamic} = CV_{dd}^2 af \tag{2.11}$$

, see [10].

2.4.1 Power Consumption During Arithmetic Operations

The mathematical functions relevant for this thesis, such as Fourier transforms and cross-correlations, are constructed of simple additions and multiplications. Looking at the energy consumption of performing these operations, it becomes clear that multiplications are much more expensive. The average energy cost of performing an addition of bit length L using a standard ripple carry adder scales as $O(L)$ [11]. Meanwhile, if one builds a multiplier using the same components, the energy cost would scale as $O(L^2)$. It would therefore be advantageous to perform additions instead of multiplications if at all possible, especially if working with large bit lengths. Similarly, divisions are even more complex and impose even higher energy costs than multiplications [12]. On hardware, integer divisions or multiplications by a power of 2 are equivalent to a bit shift, which is free, as it simply involves the rewiring of cables. However, this can only be implemented where the multiplier or denominator is predetermined.

These energy cost scales assume integers are used for the operations. It's also possible to use floating point numbers, which divide the bits into value (called mantissa) and exponent. This enables the representation of values over a very large dynamic range and the representation of values smaller than 1. However, it comes with a decreased number of significant digits (since some bits are dedicated to the exponent). Another drawback is the increased complexity and resulting higher energy cost [13].

It's difficult to compare the absolute energy consumption of different operations since it's dependent on numerous factors such as process node, supply voltage, et cetera. However, if those are controlled, comparisons can be made. For example, in a 45 nm process node with a 0.9 supply voltage, the rough energy cost for an integer addition of 8 bits was 0.03 pJ, whereas for 32 bits it was 0.1 pJ. Similarly, a multiplication costs 0.2 pJ for 8 bits and 3.1 pJ for 32 bits [14]. It should be stressed that these ratios are not universal, but they show the trend.

Although floats are commonly used, it is possible to represent decimal numbers with integers by utilising fixed-point arithmetic, which scales up the values by a factor. The most convenient scaling factors to use are powers of two, since, as mentioned, this multiplication is computationally free.

2.4.2 Power Consumption During Memory Access

It isn't just computation that requires power. Reading from or writing to memory also consumes energy. The amount of energy will, as with arithmetic operations, depend heavily on the technology. Generally, accessing external memory will be much more expensive than internal memory [15]. Additionally, reading or writing

to memory and transporting information between processing and memory blocks carries a large overhead. Therefore, storing data in memory should be minimised wherever possible. This is especially true if reading or writing to external memory [15].

For on-chip memory, Static Random Access Memory (SRAM) is the natural choice of technology. This is mainly because of its low latency for memory access. It also has the potential for lower power draw since it, unlike its competitor, Dynamic Random Access Memory (DRAM), does not require refreshing to keep values stored, assuming it is powered. Its main source of energy consumption is during memory access, when the right memory cells need to be selected, read, and potentially rewritten.

2.5 Modern Design Concerns for Digital Integrated Circuits

Power consumption is a central concern in modern digital circuit design. For a long time, the advancement of digital integrated circuits (ICs) was largely due to Dennard scaling. Dennard scaling states that the power consumption of a transistor decreases in proportion to its decrease in area [16]. This means power density remains constant, so more computation can be performed on a given area without worrying about excessive power density. In later years, however, Dennard scaling has broken down. Supply voltages can no longer be reduced at the same rate, resulting in higher power densities. When power density reaches a certain point, it is no longer feasible to transport the excess heat generated by performing calculations at a fast enough rate. This leads to the phenomenon commonly referred to as "dark silicon". The entire integrated circuit can't run simultaneously, or it would overheat. Dark silicon refers to the amount of circuitry that must remain inactive for the chip to avoid overheating.

The limitation of not being able to run the entirety of digital ICs simultaneously makes application-specific circuits more viable. Application-specific circuits are circuits designed to perform one specific function. They are not as versatile as general-purpose processing units, but they have the potential to require much less energy to perform their function than a general processor would. Since area is no longer the bottleneck of digital ICs, this has become a more viable strategy. Power draw is an important metric, especially in power-restricted environments such as Power over Ethernet solutions.

When designing algorithms on application-specific circuits, it is important to consider the implications of every function. The flow of data, degree of parallelisation, and other such factors will affect not only power draw but also power density and area usage. These factors will have to be tailored to the specific use-case of the circuit.

2.6 Methods for Reducing Power Consumption

To decrease an integrated circuit's operational power consumption, one must either lessen the number of computations or reduce the energy cost per individual computation.

Reducing the number of computations can be done by modifying how a function is performed. The FFT is a perfect example of this. Instead of doing the intuitive and straightforward DFT, some clever mathematics allows the computational complexity to go from $O(n^2)$ to $O(n\log(n))$. Even without sacrificing any accuracy, efficiency gains in the realm of several orders of magnitude could be made. This is especially true for larger systems. It would be very ambitious to presume that gains of a similar magnitude could be made for the problems presented in this thesis, but different algorithms and modifications will still be explored with the aim of reducing computational complexity.

To reduce the cost of computations, the input size of numbers to the operators can be reduced so that fewer bits are used. However, this also decreases the number of significant digits, and thus the accuracy of the result. This can be done by limiting the largest value, scaling down the starting values or scaling down the values after certain operations.

2.7 Performance Metrics

When evaluating the performance of binary or multi-class classification algorithms, precision and recall can be used as metrics, which range from 0 to 1, with 1 indicating a perfect score. Simply put, precision measures the quality of predictions, while recall measures the completeness of the detection process. The detection algorithms in this thesis will be evaluated as binary classifiers. As such, a True Positive (TP) is a detection where the algorithm identifies an embedded ZC sequence and returns both the correct root and the correct offset. A False Positive (FP) occurs when the algorithm detects a signal with an incorrect root or offset, or if the algorithm detects a signal when none is present. A false Negative (FN) occurs if a ZC sequence is present in the signal, but the algorithm fails to predict anything.

The following equations describe these metrics:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.12}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.13}$$

The specificity metric will also be used in this thesis. It measures the algorithm's ability to remain silent when no ZC sequence is in the signal. A True Negative (TN) occurs when the signal doesn't contain any ZC sequence, and the algorithm makes no prediction. The equation for specificity is:

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{2.14}$$

Chapter 3
Problem Formulation

The detection problem addressed in this thesis is to identify a ZC sequence embedded at an unknown interval within a time-discrete noisy data stream, which is obtained by sampling at a frequency f_s. This stream contains at least one entire ZC sequence of known length N_{zc}. The stream is divided into overlapping subsegments of length $L = 2^n$ for $n \in \mathbb{N}$, where $L > N_{zc}$. To ensure that any embedded ZC sequence is fully contained within at least one subsegment, the subsegments overlap by $N_{zc} - 1$ samples. The task is then to identify potential ZC sequences within these subsegments.

These subsegments are represented by a signal sequence $x[n] = \overline{x}[n] + \xi[n]$ of length L, where $\overline{x}[n]$ contains zeroes that may be interrupted by a ZC sequence s_q of length N_{zc} at a random index. $\xi[n]$ represents noise, which is modelled as Additive White Gaussian Noise (AWGN). SNR is the ratio between signal and noise power, commonly measured in decibels. Because the subsegments overlap, only complete ZC sequences will be considered when performing sequence detection.

An example of how a segment with an embedded ZC sequence may look for varying SNRs is shown in Figure 3.1.

The method most commonly used for ZC sequence detection is cross-correlation. An example of how the cross-correlation between the middle signal in Figure 3.1 and two different ZC sequences is shown in Figure 3.2. The upper graph shows the cross-correlation result when the ZC sequence does not have the same root value as the one embedded in the signal, and the lower shows the cross-correlation result when the ZC sequence does have the same root value as the one embedded in the signal.

The sampled data is modelled with no frequency or phase offset. This is not a realistic scenario, which is why the absolute value of the cross-correlation will be used, as discussed in Section 2.2. This removes the need to model a constant phase offset, since the absolute value of the cross-correlation is not affected by it.

The subsegments described in this chapter are what will be used to evaluate the detection algorithms outlined in Chapter 4. Note that in a real scenario, these would be part of a larger data segment, which is not taken into account in the evaluation.

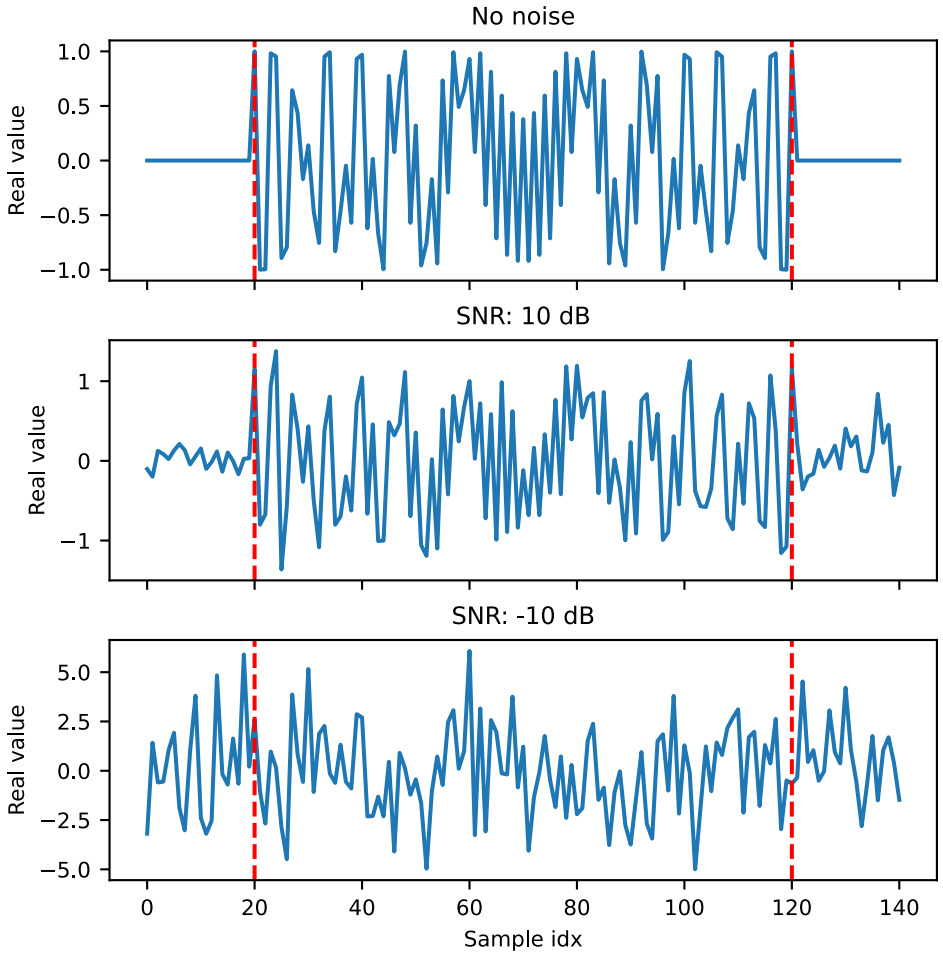


Figure 3.1: Real value component of a signal with an embedded ZC sequence of length 101 and root value 50 for three different SNR levels.

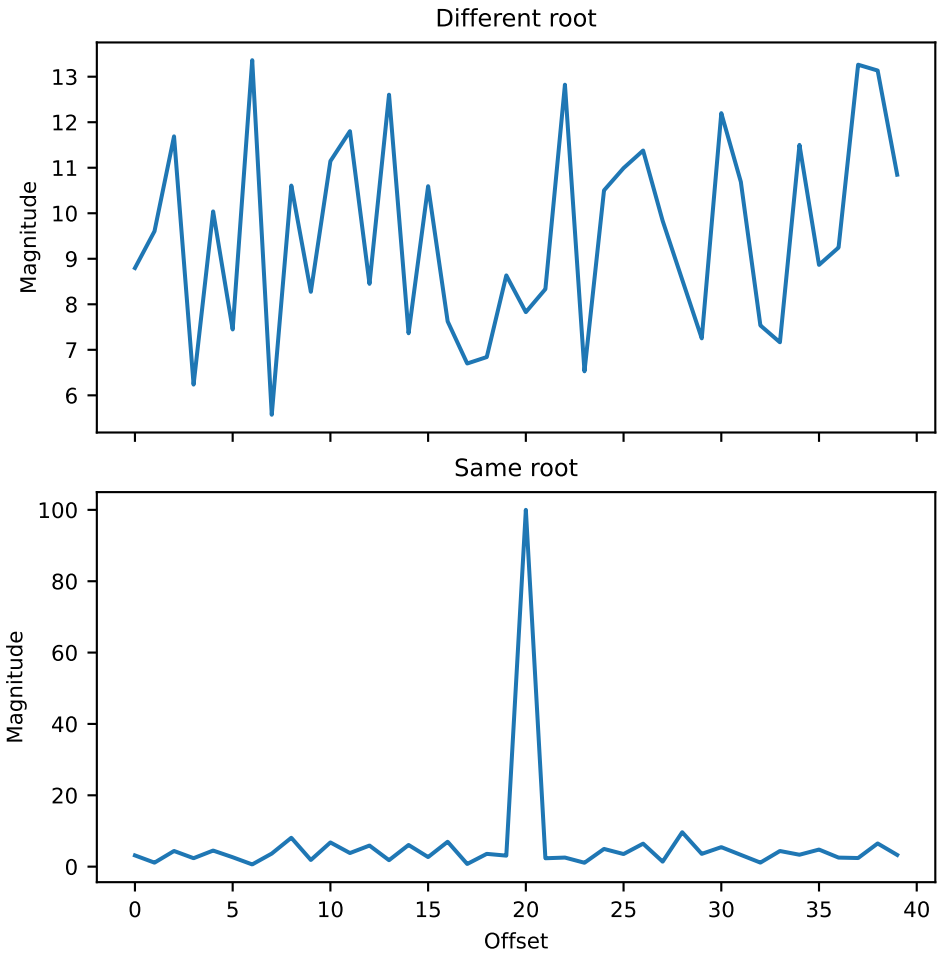


Figure 3.2: How the magnitude of the cross-correlation between the signal in Figure 3.1 at 10 dB SNR and a perfect ZC sequence depends on its root value.

Chapter 4
Methodology

This chapter explains the investigated algorithms, which are split into three categories: time-based cross-correlation, fast cross-correlation, and phase difference estimation. The explanations for the working principles of each algorithm will be followed by a block diagram illustrating them. A legend for symbols used in these block diagrams can be seen in Figure 4.1. Additionally, the chapter covers modifications such as downsampling, stacking and lowering bit precision. Lastly, the structure of the test and the evaluation of the results are explained.

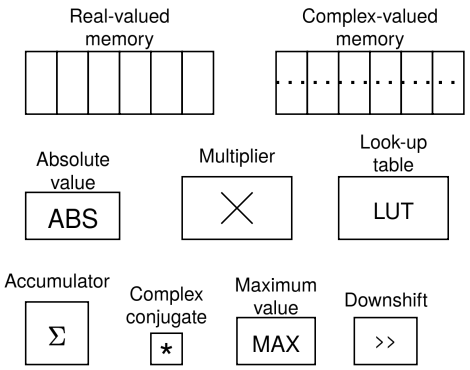


Figure 4.1: Symbols used to illustrate functions/elements used in the detection algorithms.

4.1 Cross-Correlation Algorithm

This algorithm uses cross-correlation to predict the root and offset of a ZC sequence embedded in a larger signal sequence. The algorithm will be referred to as the CC detection algorithm. A block diagram illustrating the working principles of this algorithm is shown in Figure 4.2. It generates ZC sequences of a given length, one root value at a time. For each root value, the ZC sequence is cross-correlated with the signal across all possible offsets. Due to the correlation properties of ZC sequences as described in Section 2.1, cross-correlating with the correct root value should result in a peak at the offset when the sequences overlap completely. Each

iteration is finalised with a peak detection to determine if any offset is significant. If there is a significant peak at an offset, the search is stopped, and the offset and the root of the ZC sequence are returned as the prediction. If there is no significant peak at any offset, the iteration continues.

A possible modification to reduce the amount of computation is to downsample the cross-correlation. Modifying Equation (2.5) to

$$(s_q \star x)[n] = \sum_{m=0}^{\widetilde{N}} \overline{s_q[m * k_{ds}]} x[m * k_{ds} + n] \tag{4.1}$$

, where k_{ds} is the downsampling factor and $\widetilde{N} = \lfloor \frac{N-1}{k_{ds}} \rfloor$ will scale the number of sample points and computation required to perform the cross-correlation by approximately $1/k_{ds}$. Additionally, downsampling the ZC sequences will lower the number of computations required to generate them in the first place. This modification will be referred to as downsampling.

Another possible modification explored was element-wise addition of multiple ZC sequences with different root values, forming stacked ZC sequences. Consequently, this modification was labelled stacking. The stacked ZC sequences $\widetilde{s_i}[n]$ are given by

$$\widetilde{s_i}[n] = \sum_{q=(k_s \cdot i)+1}^{k_s \cdot (i+1)} s_q[n] \tag{4.2}$$

, where k_s is the stack size determining how many ZC sequences are stacked before cross-correlation, and $i = 0, 1, 2, ..., \lceil \frac{N-1}{k_s} \rceil - 1$. Note that in cases where $\frac{N-1}{k_s}$ is not an integer value, the indexing in the sum might exceed the allowed range of root values q. In these cases, the sum with the maximum value of i will only index until $q = N-1$. These stacked ZC sequences are cross-correlated with the signal instead of the individual ZC sequences, reducing the total number of iterations. Since it is not possible to determine an exact root value from the stacked cross-correlation, ZC sequences in a stack are iterated through as specified in the unmodified algorithm if a stack yields a significant offset. Moreover, the predicted offset from the stack can be reused when doing the fine search, so that the cross-correlation window is only placed at the offset. This will further reduce the amount of computation and is investigated as an additional option when utilising the stacking modification.

These modifications introduce stack size, downsample factor, and a precise switch as adjustable parameters to be tested. A potential drawback of these modifications is that the detection might lose performance at lower SNR. Also, if an offset found during stacked cross-correlation is wrong and it is reused in the fine search, any potential matching ZC sequence in the stack will most likely not be found.

The number of computational and memory operations performed in this algorithm was calculated at a block level, illustrated in Figure 4.2 and summarised for each stage in Table 4.1. The total computational and memory operations are also shown. q represents the root value found by the algorithm and the number of iterations performed. If no ZC sequence is found, $q = N_{zc} - 1$ because all possible roots were tested. L is the signal length, and N_{zc} is the ZC sequence length.

Lastly, L' is the number of offsets, which is equal to $L - N_{zc} + 1$. It should be noted that complex-valued additions consist of 2 real-valued additions, complex-valued multiplications consist of 4 real-valued multiplications and 2 real-valued additions, and complex-valued memory operations count as 2 real-valued memory operations, as motivated in Section 4.7.1. The counts displayed in the table are the number of real-valued operations, where complex operations have been converted to their real-valued equivalents.

Table 4.1: Computational and memory operations performed in each stage and the total operations performed during the CC detection algorithm.

Stage	Additions	Multiplications	Memory
Gen ZC	0	$q \cdot N_{zc}$	$q \cdot 5N_{zc}$
CC	$q \cdot L' \cdot 4N_{zc}$	$q \cdot L' \cdot 4N_{zc}$	$q \cdot L' \cdot 4N_{zc}$
CC Result	0	0	$q \cdot L'$
Total	$q \cdot L' \cdot 4N_{zc}$	$q \cdot N_{zc} \cdot (1 + 4L')$	$q \cdot (N_{zc} \cdot (5 + 4L') + L')$

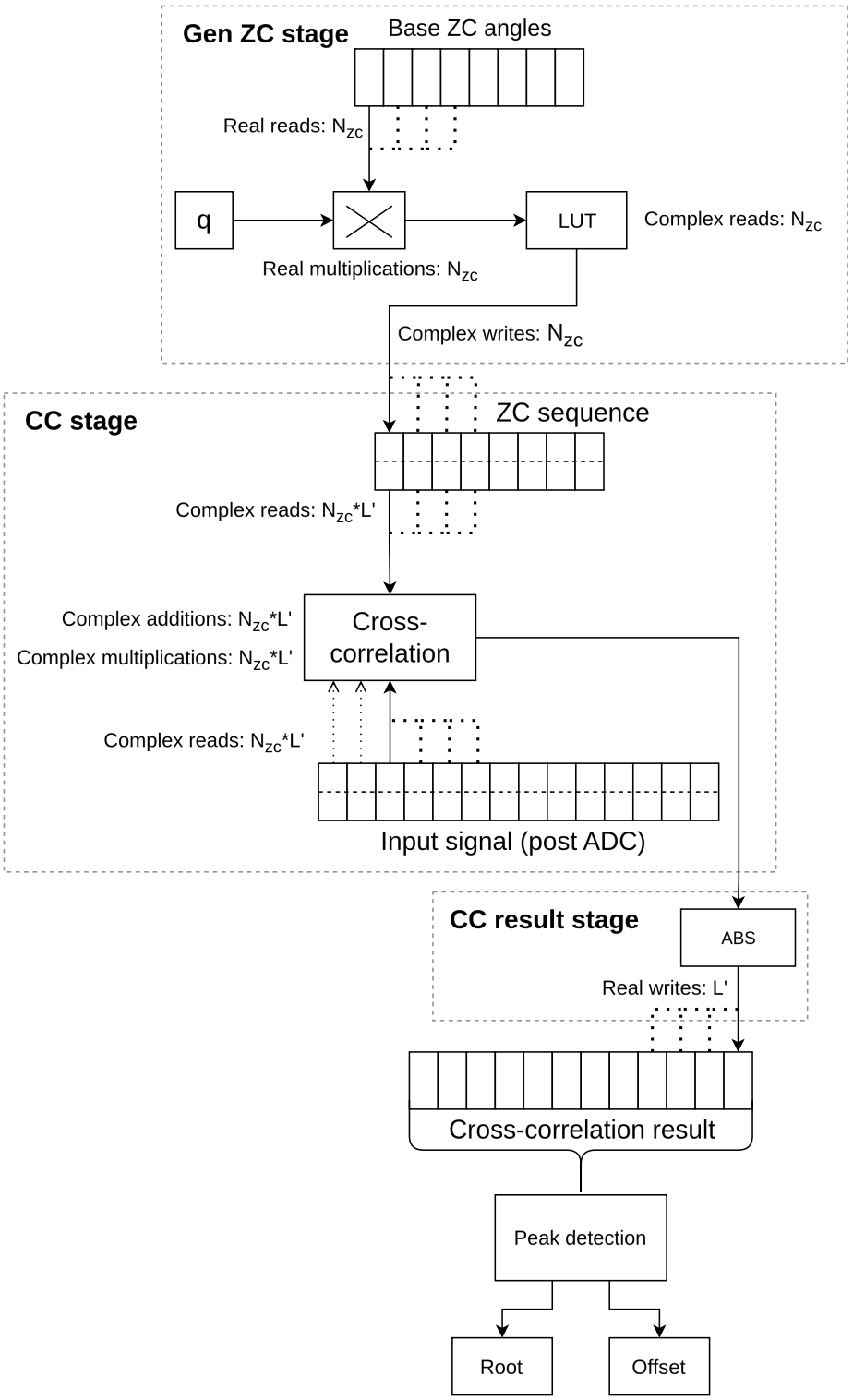


Figure 4.2: Block-level representation of the time-domain cross-correlation algorithm. All stages of this algorithm are repeated for every root investigated.

4.2 Fast Cross-Correlation Algorithm

The main difference between this algorithm and the one previously introduced in Section 4.1 is the method for calculating the cross-correlation between the ZC sequence and the signal. It uses the FFT and IFFT as described in Equation (2.10) to perform the cross-correlation, resulting in fewer computations. The algorithm will be referred to as the FFT detection algorithm. A block diagram illustrating the working principles of this algorithm is shown in Figure 4.3. Due to how the FFT and IFFT algorithms are implemented, it is required that the signal length is a power of two. Using FFT and IFFT also introduces a requirement that the sequence must be the same length as the signal; therefore, the ZC sequences have to be zero-padded prior to FFT.

To reduce the number of computations, it is still possible to stack ZC sequences and have a precise mode for in-place cross-correlation at a found offset. However, it is no longer possible to utilise downsampling, because the cross-correlation is performed in the frequency domain. If the downsampling was performed before the FFT, there is a risk that the remaining indices of the generated ZC sequence and embedded ZC sequence are different and no detection can be made.

The number of computational and memory operations performed in this algorithm was calculated at a block level, illustrated in Figure 4.3 and summarised for each stage in Table 4.2. The total computational and memory operations are shown in Table 4.3. q represents the root value found by the algorithm and the number of iterations performed. If no ZC sequence is found, $q = N_{zc} - 1$ since all possible roots were iterated through. L is the signal length, and N_{zc} is the ZC sequence length. Lastly, L' is the number of offsets, which is equal to $L - N_{zc} + 1$. The counts displayed in the table are the number of real-valued operations, where complex operations have been converted to their real-valued equivalents as described for Table 4.1.

Table 4.2: Computational and memory operations performed in each stage of the FFT detection algorithm.

Stage	Additions	Multiplications	Memory
Gen ZC	0	$q \cdot N_{zc}$	$q \cdot 5N_{zc}$
FFT	$3L \cdot \log_2(L) \cdot (q+1)$	$2L \cdot \log_2(L) \cdot (q+1)$	$5L \cdot \log_2(L) \cdot (q+1)$
FFT Combine	$2qL$	$4qL$	$6qL$
IFFT	$3L \cdot \log_2(L) \cdot q$	$2L \cdot \log_2(L) \cdot q$	$5L \cdot \log_2(L) \cdot q$
CC Result	0	0	$3qL'$

Table 4.3: Total computational and memory operations performed during the FFT detection algorithm.

Operation	Equation
Additions	$3L \cdot \log_2(L) \cdot (2q+1) + 2qL$
Multiplication	$q \cdot N_{zc} + 2L \cdot \log_2(L) \cdot (2q+1) + 4qL$
Memory	$5q \cdot N_{zc} + 5L \cdot \log_2(L) \cdot (2q+1) + 6qL + 3qL'$

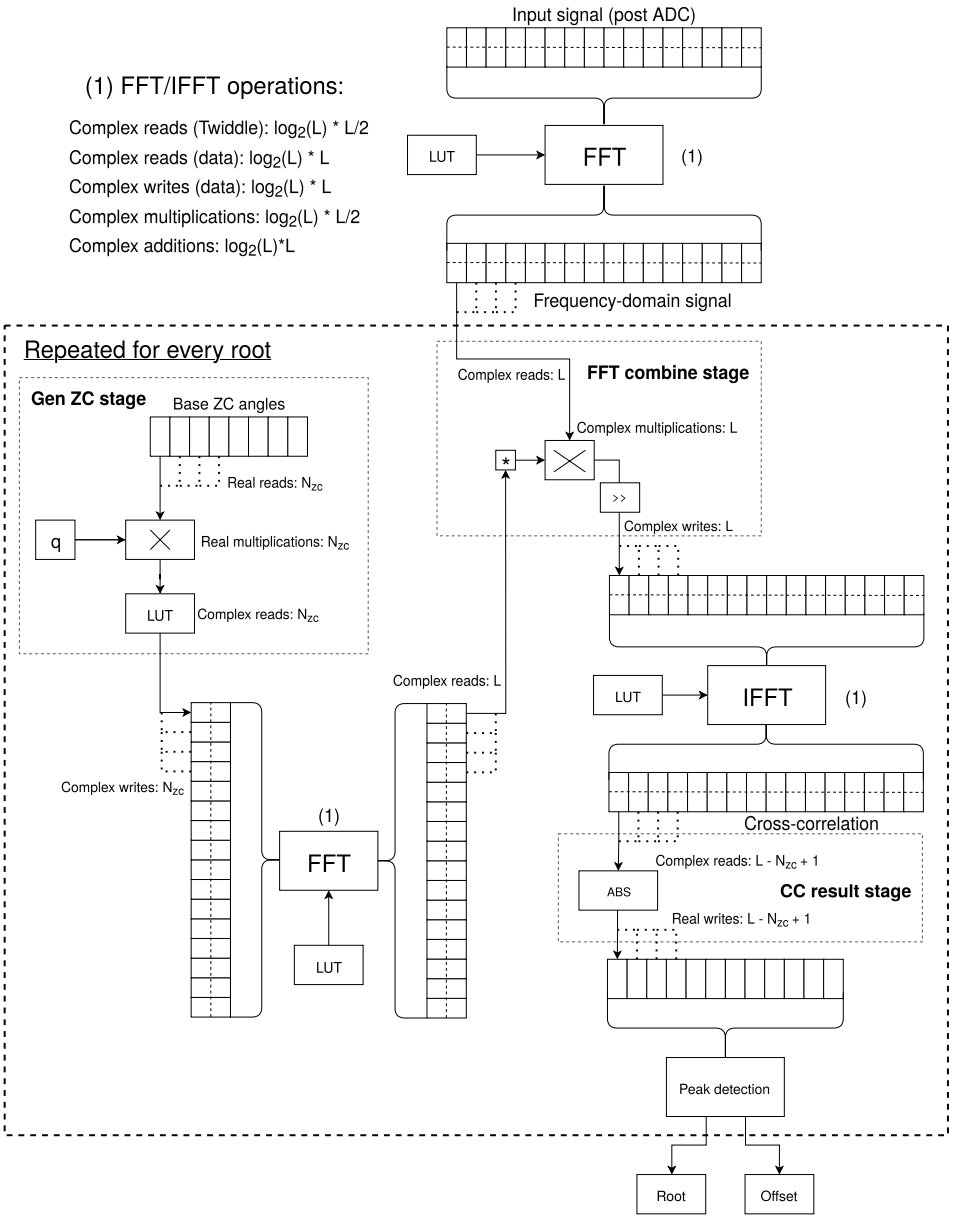


Figure 4.3: Block-level representation of the FFT-based cross-correlation algorithm.The initial FFT of the signal is only repeated once, with the remaining stages being repeated for every root investigated.

4.3 Peak Detection

Both cross-correlation-based algorithms require a way of detecting a peak in the cross-correlation result. The chosen method for peak detection locates the largest peak in the cross-correlation and compares it with the surrounding indices. The threshold for a peak to be classified as significant is computed as the mean of the surrounding indices plus 7 times their standard deviation, which helps filter out noise and reduce false positives. As motivated in Section 2.2, the absolute values of the cross-correlation are taken before peak detection is performed.

Since root values are evaluated one at a time and detection halts after a significant peak is found, peak detection must not produce false positives. Thus, the threshold was chosen high enough to return false positives very rarely. A consequence of this is that precision will be very high, whereas recall will most likely be lower.

4.4 Phase Difference Estimation Algorithm

This algorithm utilises the property described in Section 2.1.1 by computing the phase difference of the signal, transforming the embedded ZC sequence into a complex sinusoid. However, differentiating the signal also affects the characteristics of the noise and it can no longer be considered white noise, which might affect detection accuracy. The algorithm will be referred to as the Phase Difference Estimation (PDE) algorithm. A block diagram illustrating the working principles of this algorithm is shown in Figure 4.4. Each possible root value will have a unique frequency when differentiated. These frequencies are stored and fetched to generate complex sinusoids ("Gen waves stage" in Figure 4.4). This is done by multiplying the frequency by discrete time steps and placing the result in a look-up table to convert from phase ϕ to complex exponential $e^{j\phi}$. The frequency of these complex sinusoids will then match all of the possible frequencies of the differentiated ZC sequence of the given length. Because the sinusoids are periodic, cross-correlation for every offset is unnecessary. Instead, a single dot product of the differentiated signal and generated sinusoid can be performed (CC stage in Figure 4.4), drastically reducing the number of computations. The absolute value of the dot product is then used to determine a "cost" for each frequency. Unlike in the other algorithms, all roots are tested before a prediction is made. Since no offsets are tested, only the root can be determined from these costs. The root corresponding to the frequency with the highest cost is flagged as the most likely root value present in the signal. To verify the root value and determine where in the signal the ZC sequence is embedded, a single iteration of the FFT algorithm with the predicted root is performed.

This algorithm also allows for downsampling and stacking sinusoids to further reduce the number of computations. These follow the same explanation as the modifications in Section 4.1 and provide the same advantages. A slight difference is that the downsampling reduces the number of samples in the dot product instead of cross-correlation. Also, as this algorithm does not directly predict the position of the ZC sequence in the signal, there is no precise detection mode for stacking.

The number of computational and memory operations performed in this algorithm was calculated at a block level, illustrated in Figure 4.4 and summarised for each stage in Table 4.4. The total computational and memory operations are also shown. However, the verification stage is excluded from the total and is presented separately in Table 4.5. The number of computational and memory operations in the verification stage is the same as the total in the FFT detection algorithm when $q = 1$. Q is the total number of roots, which is $N_{zc} - 1$, L is the signal length, $L' = L - 1$, and N_{zc} is the ZC sequence length. The counts displayed in the table are the number of real-valued operations, where complex operations have been converted to their real-valued equivalents as described for Table 4.1.

Table 4.4: Computational and memory operations performed in the PDE detection algorithm stages and the total operations performed over those stages.

Stage	Additions	Multiplications	Memory
Diff	$2L'$	$4L'$	$2L + 2L'$
Gen Waves	0	QL'	$Q(1 + 2L')$
CC	$4QL'$	$4QL'$	$Q(1 + 2L')$
Total	$L'(4Q + 2)$	$L'(5Q + 4)$	$Q(4L' + 2) + 2(L + L')$

Table 4.5: Computational and memory operations performed in the verification stage of the PDE detection algorithm.

Operation	Equation
Additions	$9L \log_2(L) + 2L$
Multiplication	$N_{zc} + 6L \log_2(L) + 4L$
Memory	$5N_{zc} + 15 \log_2(L) + 6L + 3L'$

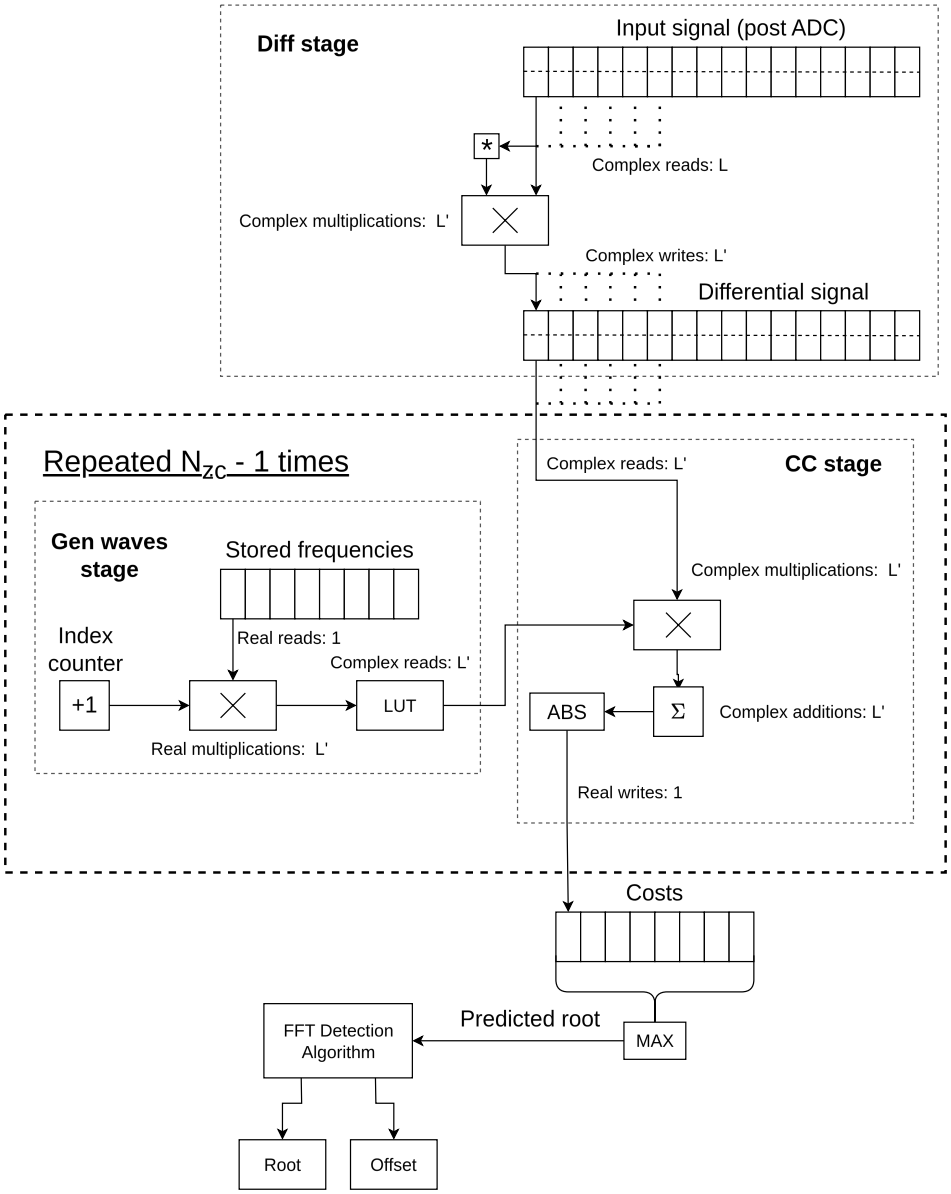


Figure 4.4: Block-level representation of the phase difference estimation algorithm. The CC and gen waves stage was repeated $N_{zc} - 1$ times, and the rest were only repeated once.

4.5 General Modifications

To minimise computational costs, the algorithms were modified to utilise fixed-point arithmetic instead of floating-point. This approach redefines the mapping of bit values by using a power-of-two scaling factor. For example, with a scaling factor of 2^2 (4), the least significant bit will represent a value of $\frac{1}{4} = 0.25$ instead of 1. This means the number 1 would be represented by a binary value of 0100. The scaling factor determines the minimum distance between two numbers, and thus dictates the resolution. While a higher factor yields a higher resolution, it also increases the bit size of numbers. Consequently, minimising the scaling factor whilst maintaining acceptable resolution is essential.

Additionally, the algorithms contain a few steps where division is used, but they have been modified to only use divisions by a factor of two in these steps, so hardware-wise, it is a bit shift. This avoids the high computational cost of divisions.

To further reduce costs, the formula for absolute values is replaced by the Alpha Max Plus Beta Min algorithm [17]. This method approximates the magnitude by replacing the multiplications and square root with bit shifts and additions, maintaining a maximum approximative error of 2.8%.

During the search step, sinusoids and ZC sequences are generated using a Look-Up Table (LUT) to avoid trigonometric functions, which are computationally expensive. The constant components of these sequences are pre-stored and multiplied by an index or root to determine the LUT input. The LUT outputs are also pre-scaled by the scaling factor to remain consistent with the fixed-point environment. Therefore, the LUT output resolution is also governed by the scaling factor. Similarly, a LUT is used for generating the twiddle factors used in the FFT and IFFT.

4.6 Test structure

To evaluate the performance of the algorithms and modifications, a controlled simulation environment was developed. The algorithms themselves were developed from scratch in Python, as were the evaluation and computational tracking functions. In an evaluation, a detection algorithm, with or without modifications, is tested against signals with varying SNR values ranging from 0 to -15 dB in 1 dB increments. The algorithms had a default scaling factor of 64, as preliminary studies showed a higher factor would not meaningfully improve the detection probability regardless of SNR.

4.6.1 Signal Generation

For each test iteration, a ZC sequence of length 601 is embedded within a longer carrier signal of 2048 samples. Each test signal has a randomised root and offset to get a robust evaluation. The test signal is then passed through an ideal 13-bit signed analogue-to-digital converter (ADC). The digitised signal is then scaled to match the fixed-point scaling factor specified in the test. Notably, if the chosen

scaling factor is lower than the ADC precision (2^{13}), a reduction in the signal resolution occurs.

4.6.2 Test Execution and Data Collection

Each evaluation consists of 100 iterations per SNR level. To ensure statistical significance, every combination of algorithm and modification was evaluated at least five times. During an evaluation, the following data points were collected for each SNR:

- **Computational Metrics:** The average number of arithmetic operations and memory operations (read/writes).
- **Bit sizes:** Average number of times each bit size is needed to represent the inputs to an arithmetic operation.
- **Detection Metrics:** The counts of True Positives (TP), False Positives (FP), and False Negatives (FN).

All arithmetic and memory operations were tracked from the point at which the test signal was received and terminated before peak detection. Peak detection was excluded from the computational tracking to ensure that the detection algorithms were evaluated fairly based on the quality of their resulting correlation values. Attempting to reduce the peak detection cost would introduce additional variables and potentially worsen the performance, making it difficult to isolate the impact of the primary modifications. Additionally, including the peak detection would not significantly influence the overall computational cost. Therefore, a reliable and accurate peak detection algorithm was used in all tests.

The bit size tracking is based on the minimum number of bits required to be able to represent both inputs to an operation. This means that each operation will have a connected bit size, and a distribution will be formed. For example, 4 + 8 will be tracked as an addition with the bit size 5. This is because 5 bits are needed to represent the signed integer 8, and 4 bits are needed for the signed integer 4. Since the larger of the two bit sizes is chosen to represent the operation, the tracked bit size will be 5.

A separate evaluation was conducted to determine specificity. This involved testing the algorithm on 100 noise-only signals containing no embedded ZC sequence. This was done to accurately capture True Negatives (TN) and False Positives (FP) without creating a bias in the primary detection test. From these results, the specificity was calculated using 2.14.

The modifications and values assessed are summarised in the table below.

Table 4.6: Modifications and values assessed.

Parameter	**Value**
Scaling factor	4, 8, 16, 64
Stack size	1, 2, 4, 6
Precise mode	on/off
Downsample factor	1, 2, 3, 4

4.7 Evaluation

The performance of the algorithms and modifications was based on the energy consumption in arbitrary units, precision, recall, and specificity across a span of SNR values.

4.7.1 Energy Consumption Model

To estimate energy consumption, the computations were divided into stages representing hardware blocks. Within each block, the largest bit size input that occurred for addition and multiplication determined the bit size used by the respective operator. These bit sizes, N_{add} and N_{mult}, also represent the relative energy cost of performing an operation. The cost of an addition is approximated as N_{add}, while the cost of a multiplication is approximated as N_{mult}^2. Consequently, complex operations will be converted to their corresponding real components, as it makes it easier to translate an operation to an energy cost. To determine the largest bit sizes that occurred in each stage, the tracked bit sizes were used. However, only the results from the tests with the highest SNR were used, as these represent where the largest digits occur. Lastly, the stages were summed into a total energy consumption.

Generally, energy consumption during memory operations scales with the number of word lines and bit lines. In this thesis, all memory blocks contained a similar number of word lines. Thus, the word line amount was assumed to be constant. With this, the scaling only depends on the number of bit lines, which is equivalent to the maximum stored bit size. To relate memory operations to additions, a conversion factor was used to estimate the cost of a single memory access as 5 additions. The bit size for these additions is the largest bit size present during the majority of memory operations. For the FFT algorithm, this corresponds to the maximum bit size in the IFFT stage. In the CC and PDE algorithm, it is equal to $\log_2(scaling\ factor) + 1$, which is 7 for the base scaling factor case. This energy model assumes a small-scale SRAM architecture and is based on previous experiences. For reading or writing complex values, the energy consumption is approximated to that of two real memory operations, since a complex number is twice the number of bits. Therefore, a complex memory operation will be counted as two real memory operations before converting the number of operations to energy.

The energy cost for the different operation types is summarised in Table 4.7 where N is a dynamic bit size that depends on the operation. It should be stressed that these costs are in arbitrary units and do not represent real energy consumption, but they are usable for relative comparisons. The arbitrary unit is the number of additions of bit size 1, and will interchangeably be expressed as energy consumption or energy cost

Table 4.7: Energy cost for the different operation types.

Operation	Energy Cost
Addition	N
Multiplication	N^2
Memory access	$5N$

4.7.2 Performance Metrics and Statistical Analysis

Detection performance metrics, precision and recall, were calculated using the formulas defined in Section 2.7, Performance Metrics. To ensure statistical significance, results from identical evaluations across distinct SNR levels were collected to calculate an average and a 95th percentile range for each metric and SNR. Additionally, precision scores for SNR values with fewer than five total detections were excluded to maintain statistical significance. As energy consumption was already averaged over 100 test iterations, its percentile range represents the spread of these averages.

Chapter 5
Results

This chapter presents the performance of the algorithms and the impact of the proposed computation cost reduction methods. First, the baseline detection performance of the unmodified algorithms is presented, followed by the modified variants.

Three types of figures are used throughout the chapter. The first illustrates the bit-size distribution for addition and multiplication in each computing stage. On the x-axis, the minimum bit size required to represent both inputs to an operation is shown, and on the y-axis, the average number of occurrences is shown. The second and third present detection performance metrics (recall and precision) and energy consumption (operation and memory), respectively, as functions of SNR. In these latter two figure types, values are shown as averages with a shaded 95th percentile range. Each figure may include multiple algorithms or modification values for comparison. Specificity was found to be very high for most cases, so it will only be presented for cases where it was below 0.95.

5.1 Algorithms

The detection performance of the unmodified algorithms is presented in Figure 5.1. The CC and FFT detection algorithms had identical detection performance, which began diminishing after -12 dB as the recall score started to decline. Their identical performance is because the mathematical operations in the FFT and CC algorithms are equivalent. In contrast, the PDE detection algorithm's recall score started to diminish below -5 dB and more steeply, reaching approximately zero as precision performance also began to decline. Additionally, the precision score for the PDE algorithm was only statistically significant for SNR equal to or above -10 dB.

The energy consumption of the unmodified algorithms is presented in Figure 5.2. The CC detection algorithm had the highest energy consumption, approximately an order of magnitude higher than the FFT detection algorithm and two orders of magnitude higher than the PDE detection algorithm. Unlike the PDE algorithm, whose power consumption was independent of the recall score, both the CC and FFT algorithms displayed an inverse correlation between recall score and power consumption. Additionally, the PDE algorithm has an operations cost almost 8 times higher than the memory cost, making the total energy cost

predominantly operational, unlike the other algorithms, where the energy cost is split almost equally.

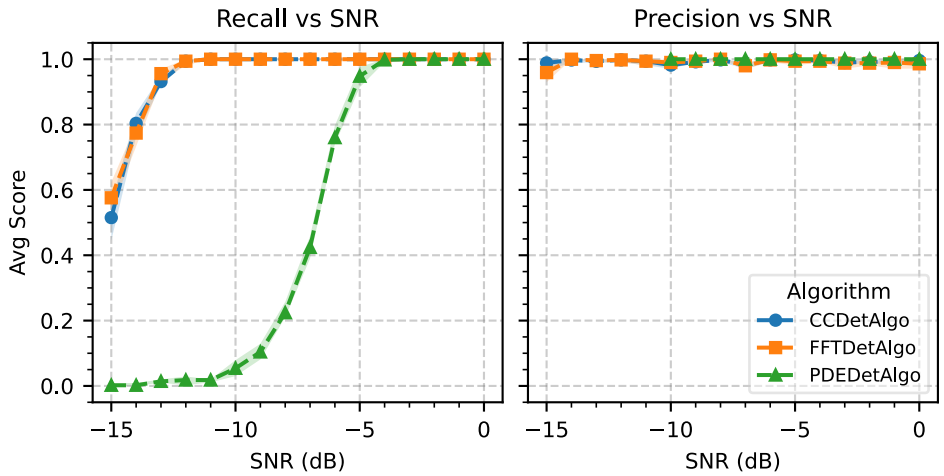


Figure 5.1: Detection performance of the unmodified algorithms.

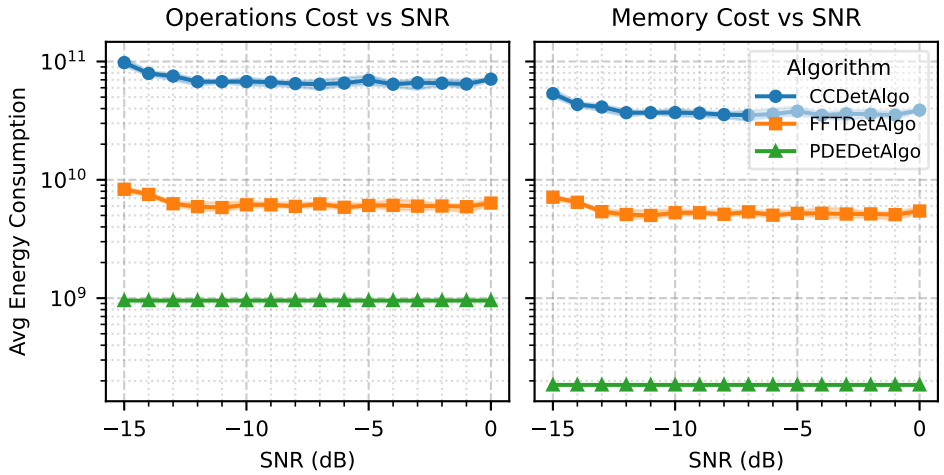
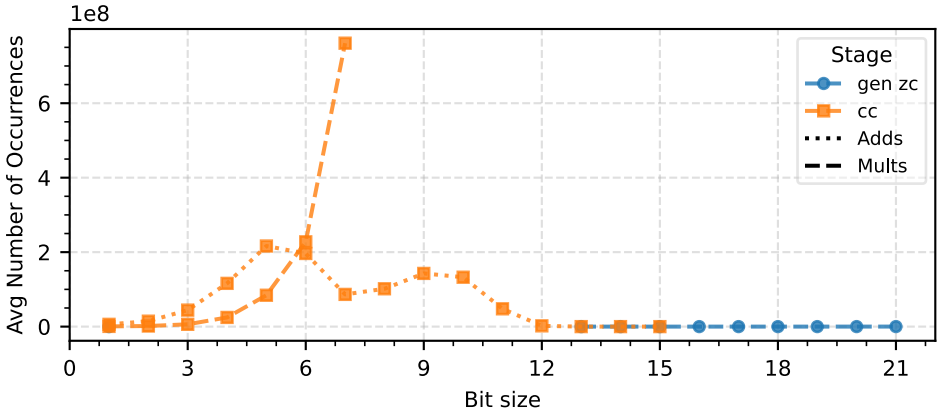
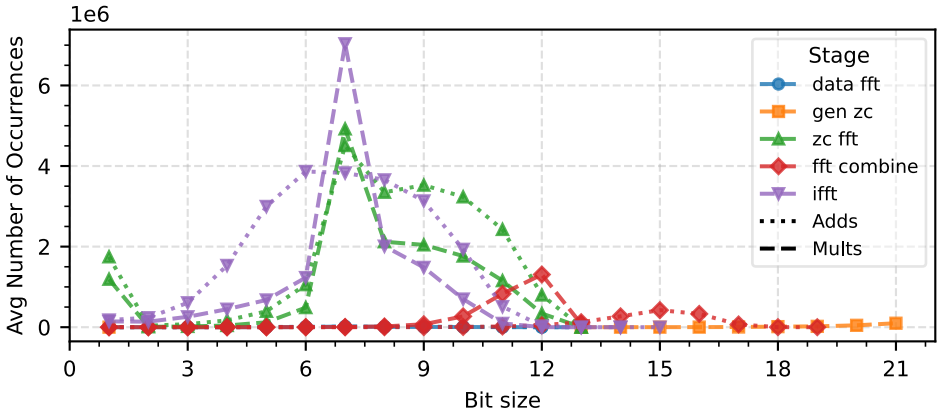


Figure 5.2: Energy consumption in the unmodified algorithms.

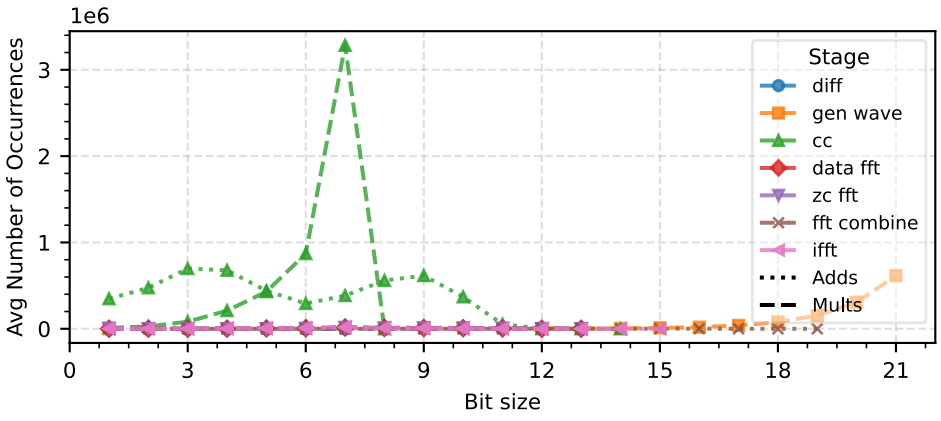
The bit size distributions for the CC, FFT, and PDE algorithms are presented in Figure 5.3. None of the algorithms used a bit size above 21. In the CC algorithm (Figure 5.3a), the majority of computations occurred in the cross-correlation stage labelled cc. For the FFT algorithm (Figure 5.3b), computation was concentrated in the FFT/IFFT and spectral multiplication stages. Lastly, in the PDE algorithm (Figure 5.3c), computation was dominated by the cross-correlation stage and the ZC wave generation.



(a) CC detection algorithm.



(b) FFT detection algorithm.



(c) PDE detection algorithm.

Figure 5.3: Bit size distribution of additions and multiplications for the three unmodified algorithms.

The compute-heavy stages are further presented in Table 5.1, where the CC algorithm had the most operations and the highest cost. For the PDE algorithm, half the cost was due to generating the waves, and in the FFT algorithm, the ifft stage and the fft of the zc sequence contributed the most to the total cost. Notably, the FFT algorithm had the overall biggest bit sizes, the PDE algorithm had the biggest bit size, and the CC algorithm had the smallest.

Table 5.1: Stages exceeding 10% of total compute in the algorithms.

(a) CCDetAlgo

Stage	Adds	Bit (A)	Cost (A)	Mults	Bit (M)	Cost (M)
cc	1.1×10^9	15	1.7×10^{10}	1.1×10^9	7	5.4×10^{10}

(b) FFTDetAlgo

Stage	Adds	Bit (A)	Cost (A)	Mults	Bit (M)	Cost (M)
fft combine	1.3×10^6	19	2.5×10^7	2.6×10^6	13	4.4×10^8
ifft	2.3×10^7	15	3.4×10^8	1.4×10^7	14	2.8×10^9
zc fft	2.1×10^7	13	2.8×10^8	1.4×10^7	13	2.4×10^9

(c) PDEDetAlgo

Stage	Adds	Bit (A)	Cost (A)	Mults	Bit (M)	Cost (M)
cc	4.9×10^6	14	6.9×10^7	4.9×10^6	8	3.1×10^8
gen wave	0	0	0	1.2×10^6	21	5.4×10^8

5.2 Modifications

5.2.1 Cross-Correlation Algorithm

When the CC algorithm utilised downsampling, the recall score diminished several dBs earlier, and it scaled with the downsampling amount. However, the steepness in the loss of recall score appeared to be independent of the downsampling value. Similarly, the precision also worsened earlier as downsampling increased. This is shown in Figure 5.4.

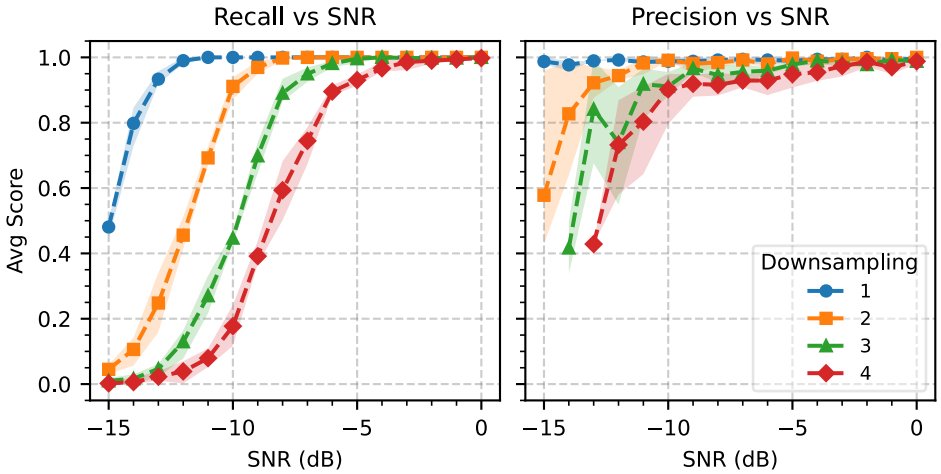
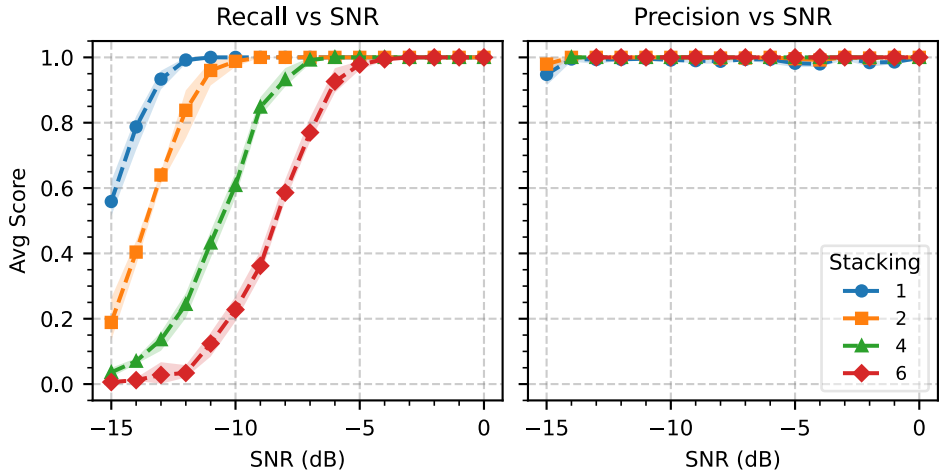
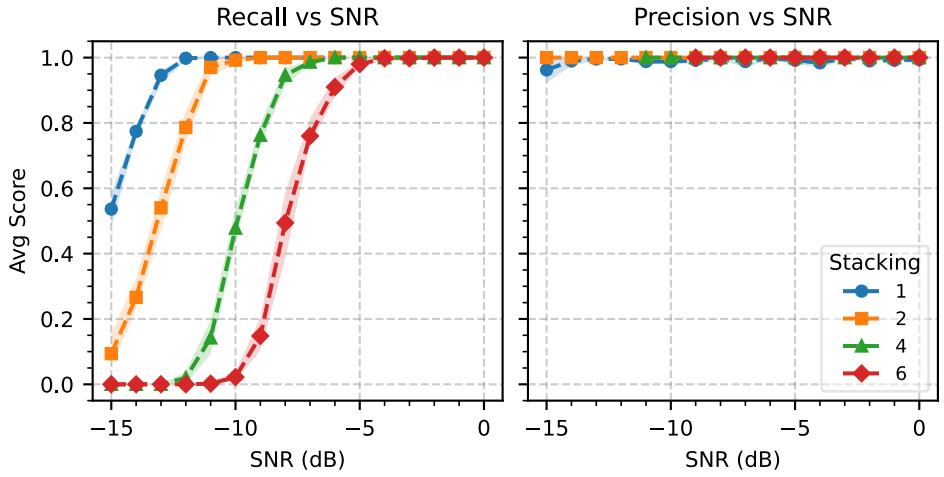


Figure 5.4: Detection performance of the CC detection algorithm utilising downsampling.

The detection score of the CC algorithm utilising different stacking sizes, with the precise mode on and off, is shown in Figure 5.5. Both stacking cases showed a decrease in recall score as the stacking size increased, but it was slightly more evident when the precise mode was active, as the recall score loss was steeper. Additionally, the precision score for stack sizes 4 and 6 in Figure 5.5b became statistically insignificant earlier than the corresponding stack sizes in Figure 5.5a.



(a) Precise mode inactive.



(b) Precise mode active.

Figure 5.5: Detection performance of the CC detection algorithm utilising stacking, with and without precise mode active.

The energy consumption for the downsampling and stacking modification, with and without precise mode, is shown in Figures 5.6 and 5.7. All modification types showed a near-identical negative correlation between memory and operation cost as a function of the modification value. Notably, the energy consumption increased as the recall score decreased.

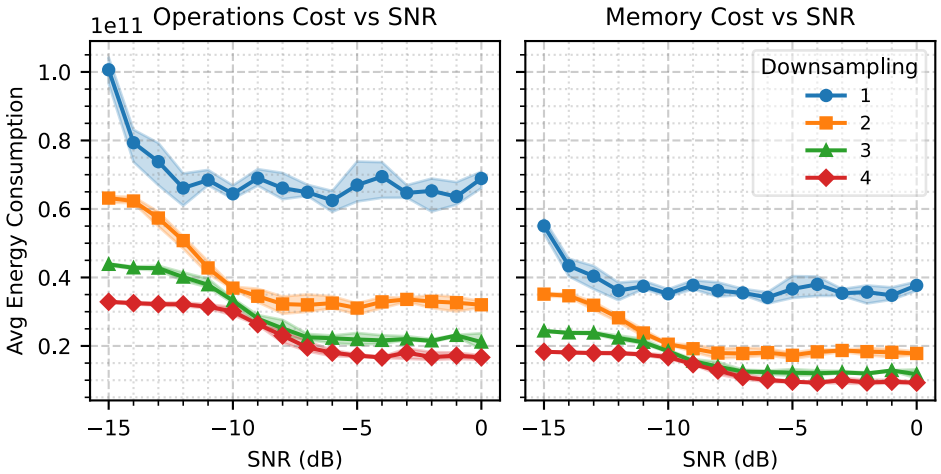
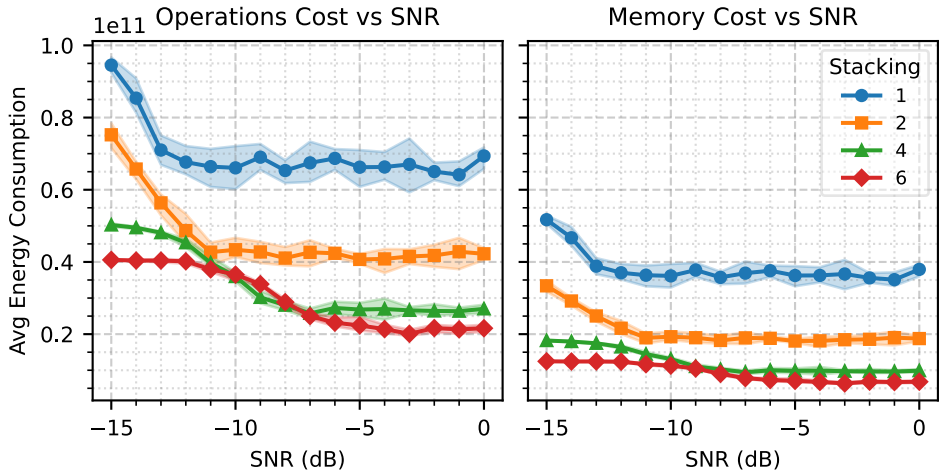
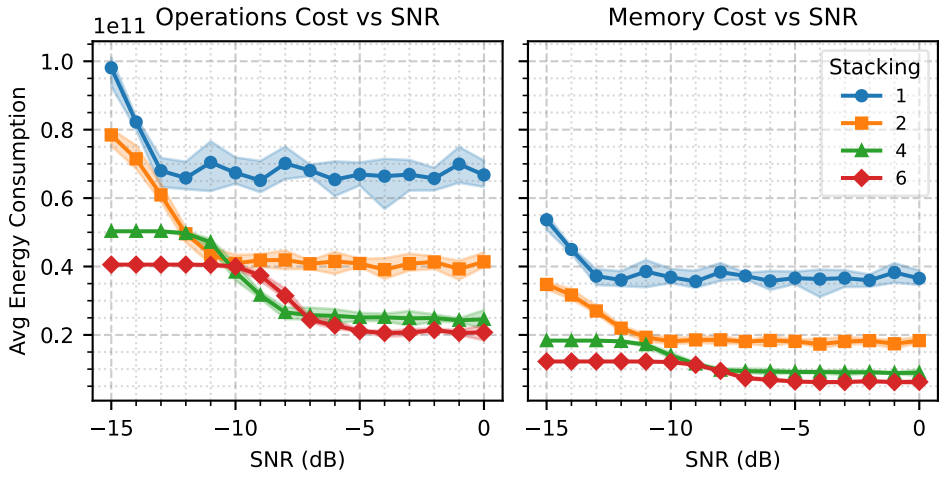


Figure 5.6: Energy consumption of the CC detection algorithm utilising downsampling.



(a) Precise mode inactive.

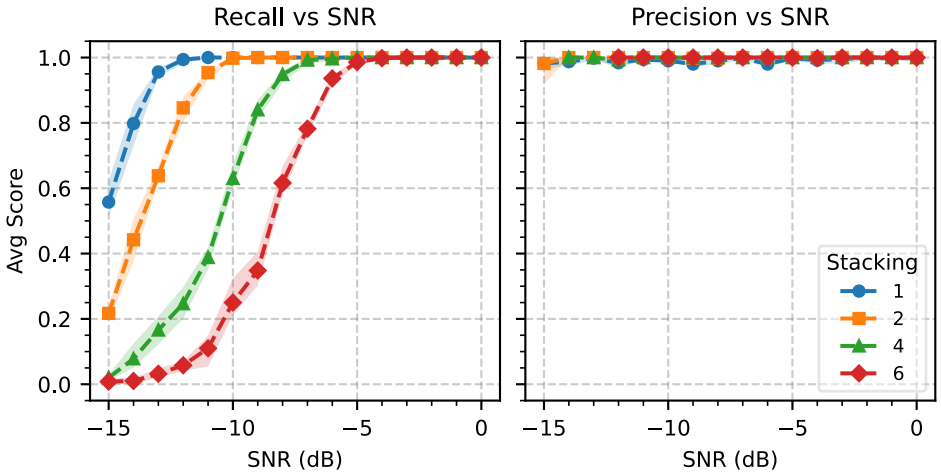


(b) Precise mode active.

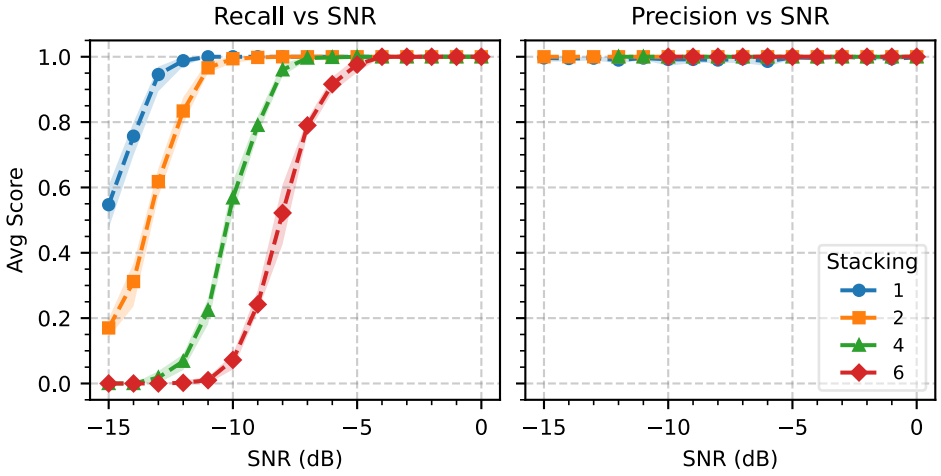
Figure 5.7: Energy consumption of the CC detection algorithm utilising stacking, with and without precise mode active.

5.2.2 Fast Cross-Correlation Algorithm

In Figure 5.8, the detection performance of the FFT algorithm with stacking is shown. Since the mathematical operations in the FFT and CC algorithm are equivalent, see Section 2.3.3, the results were identical to those obtained with the CC algorithm when using the same modifications (Figure 5.5).



(a) Precise mode inactive.



(b) Precise mode active.

Figure 5.8: Detection performance of the FFT detection algorithm utilising stacking, with and without precise mode active.

When the scaling factor was lowered, the detection performance worsened, as shown in Figure 5.9. For factor 16, the loss in recall and precision score was minuscule. However, for factors 8 and 4, the loss became more noticeable. Most notably, for a scaling factor of 4, the performance was very poor, as the precision was only 0.25 at 0 dB SNR and declined at higher SNR values. Additionally, the average specificity for factor 4 was 0.01. In comparison, the specificity for factor 8 was 0.93, and the other factors had a score above 0.95.

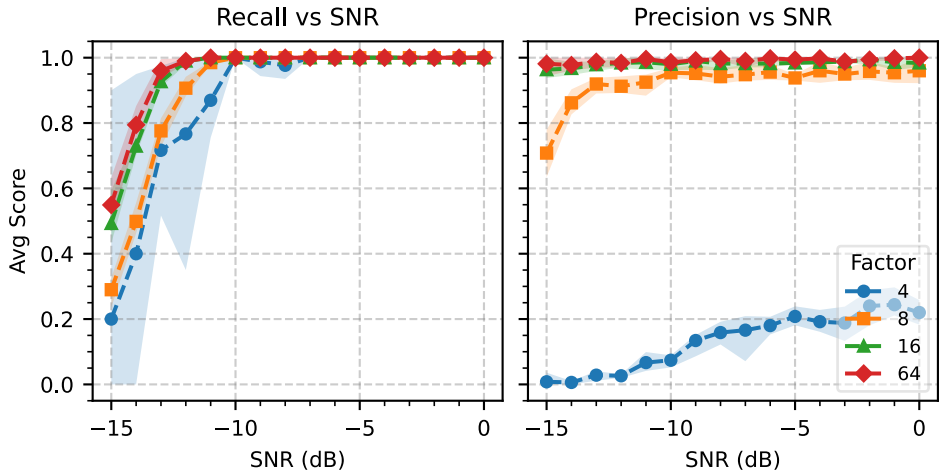
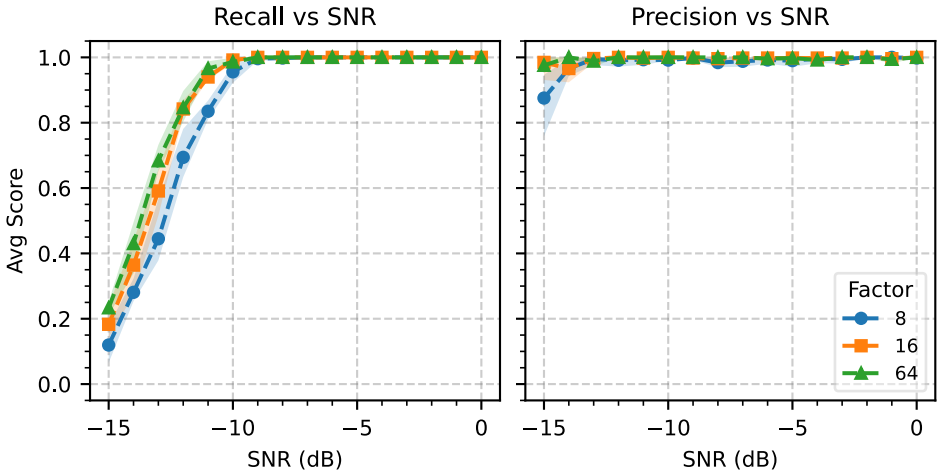
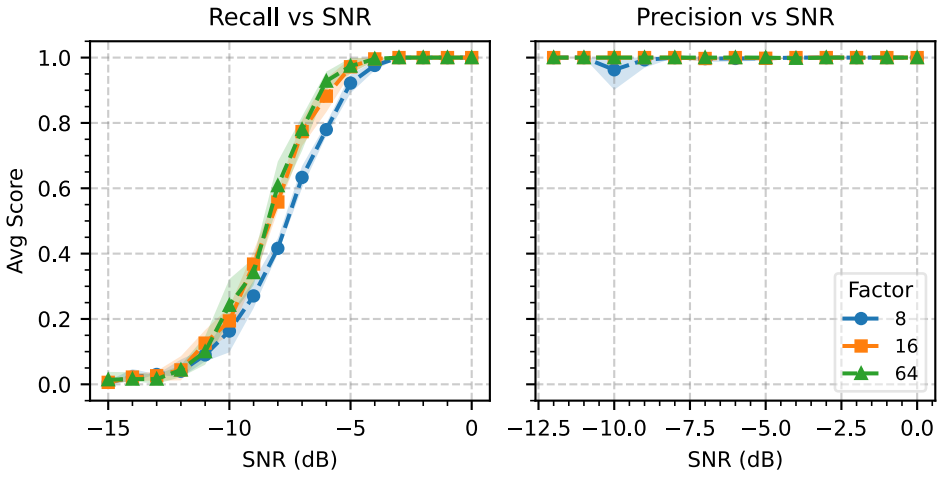


Figure 5.9: Detection performance of the FFT detection algorithm utilising lower scaling factors.

In Figure 5.10, the decrease in scaling factor was combined with a stack size of 2 and 6, and the precise mode was inactive. It appeared as if the performance losses in Figures 5.8a and 5.9 were superimposed on each other. Interestingly, the precision performance for factor 8 seemed to improve with a stack size of 2. For stack size 6, this could not be said as the precision score was statistically insignificant at -15 dB SNR.



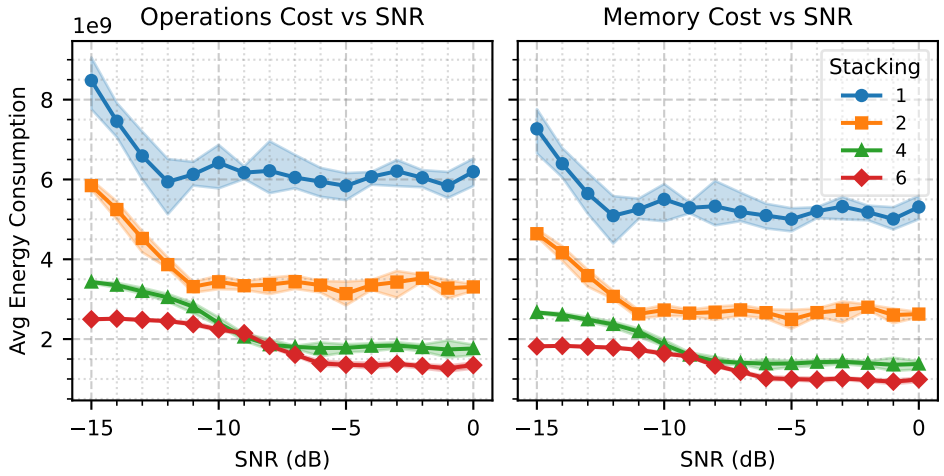
(a) With stack size 2.



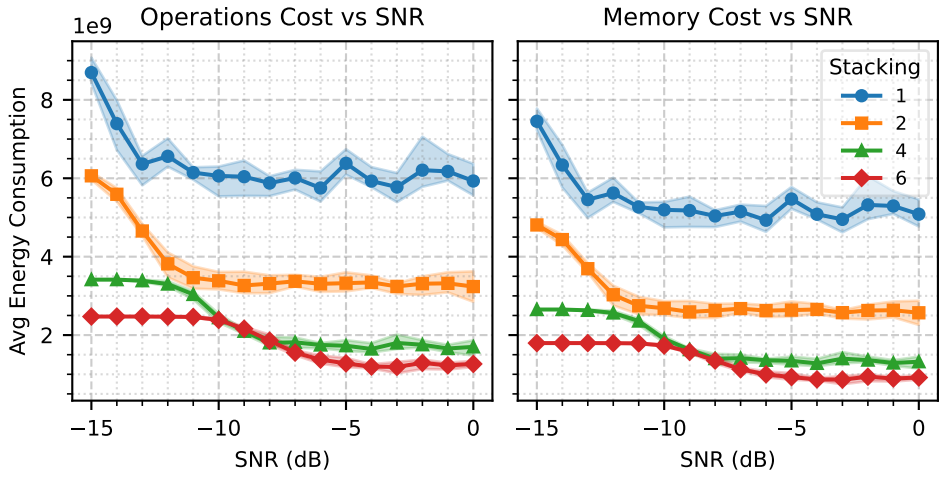
(b) With stack size 6.

Figure 5.10: Detection performance of the FFT detection algorithm utilising lower scaling factors in combination with stacking, where the precise mode is inactive.

The energy consumption of the FFT algorithm with stacking (Figure 5.11) displayed the same negative correlation with stack size as was observed for the CC algorithm using identical stacking parameters (Figure 5.7). The key difference was that the FFT algorithm consumed less energy than the CC algorithm for the same stack size. It can also be observed in Figure 5.11 that the energy consumption increased as the recall score decreased.



(a) Precise mode inactive.



(b) Precise mode active.

Figure 5.11: Energy consumption of the FFT detection algorithm utilising stacking, with and without precise mode active.

Figure 5.12 displays the reduction in energy consumption when the scaling factor was lowered. The energy reduction appeared to diminish when going from a scaling factor of 64 to 8. The operation cost limit appeared to be equal to the energy consumption when stacking two ZC sequences. However, scaling factor 4 was an outlier, which reduced the energy consumption significantly.

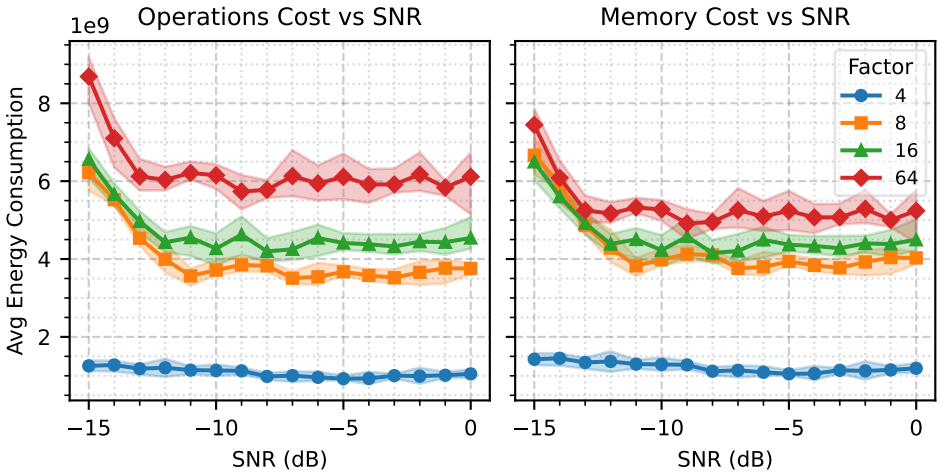
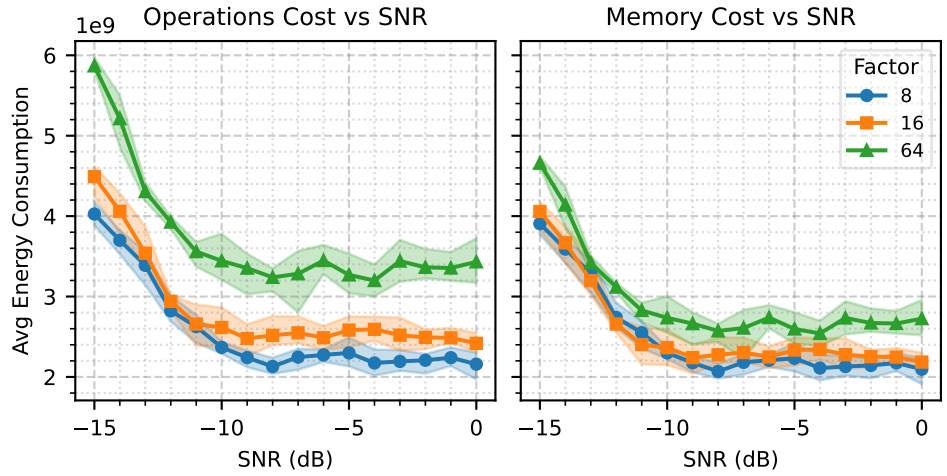
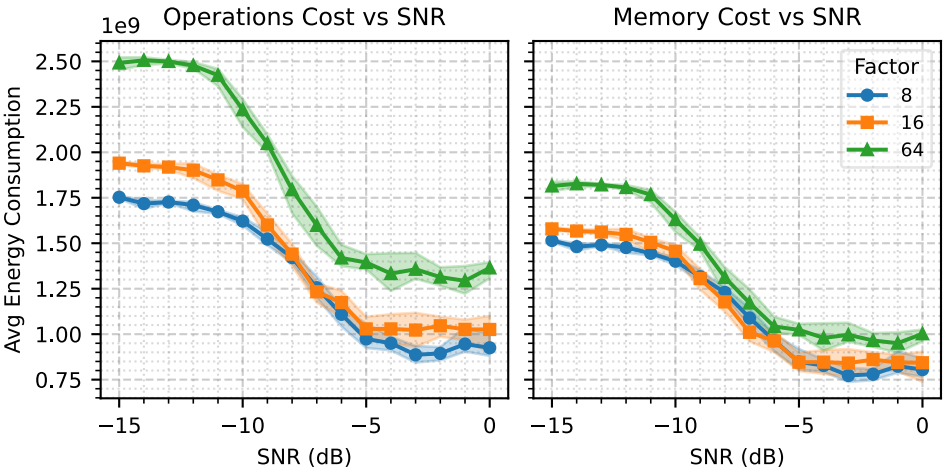


Figure 5.12: Energy consumption of the FFT detection algorithm utilising lower scaling factors.

When combining stacking with a lower scaling factor, the FFT algorithm's energy consumption decreased noticeably. Similar to the detection performance, the energy reduction seemed to be the two individual energy reductions in Figures 5.11 and 5.12 superimposed on each other.



(a) With stack size 2.

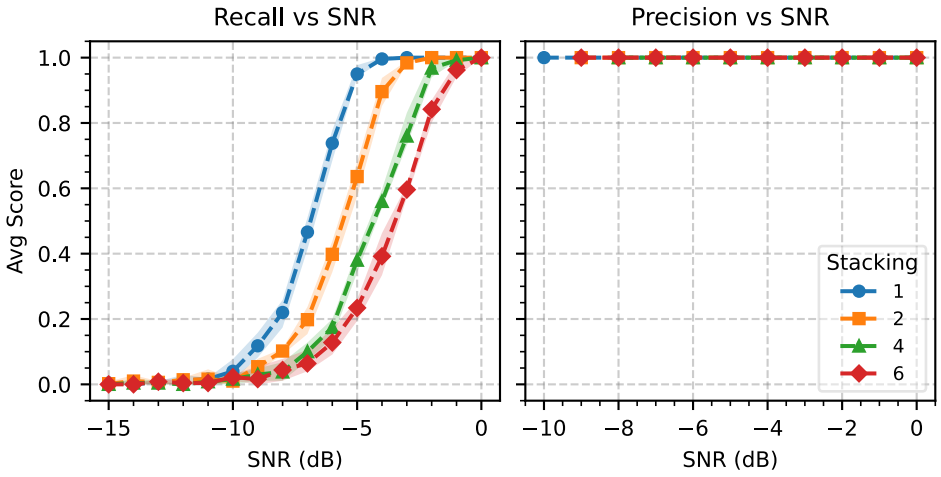


(b) With stack size 6.

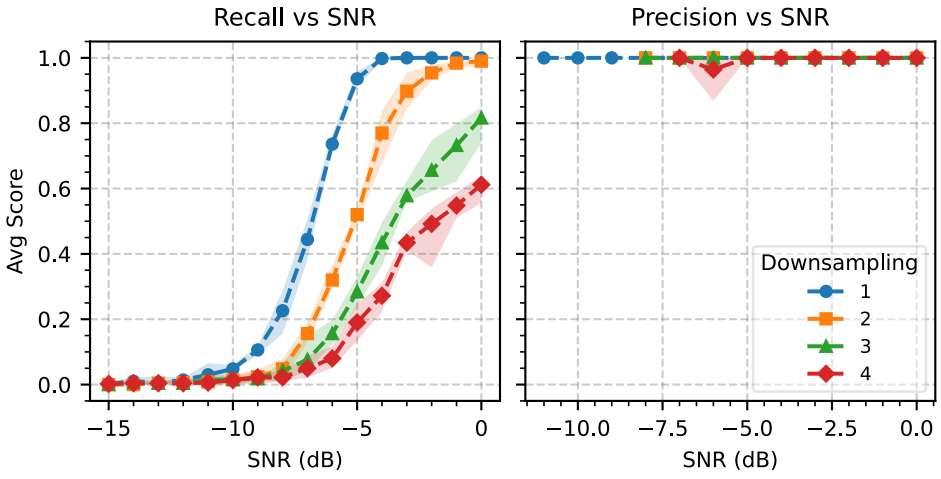
Figure 5.13: Energy consumption of the FFT detection algorithm utilising lower scaling factors in combination with stacking, where the precise mode is inactive.

5.2.3 Phase Difference Estimation Algorithm

Figure 5.14 shows the impact on detection performance when the PDE algorithm is modified to use stacking or downsampling. The impact of using a stack size of 2 and downsampling by 2 is very similar, with downsampling performing slightly better. However, for values above this, the downsampling has a much worse impact on the recall score. Equally, the precision becomes statistically insignificant earlier as downsampling increases. Stacking is much less affected by this, as only stack size 6 has lost precision score points in Figure 5.14a. Notably, stacking sizes 4 and 6 shows very similar recall scores.



(a) Utilising stacking.

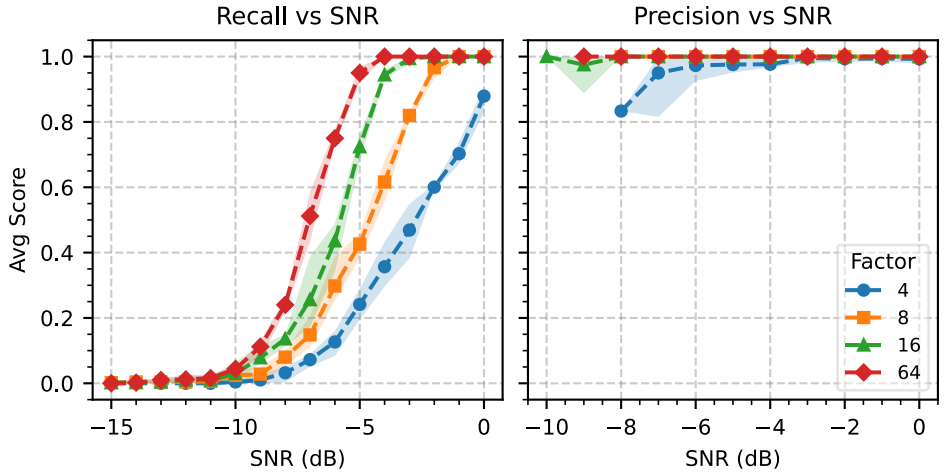


(b) Utilising downsampling.

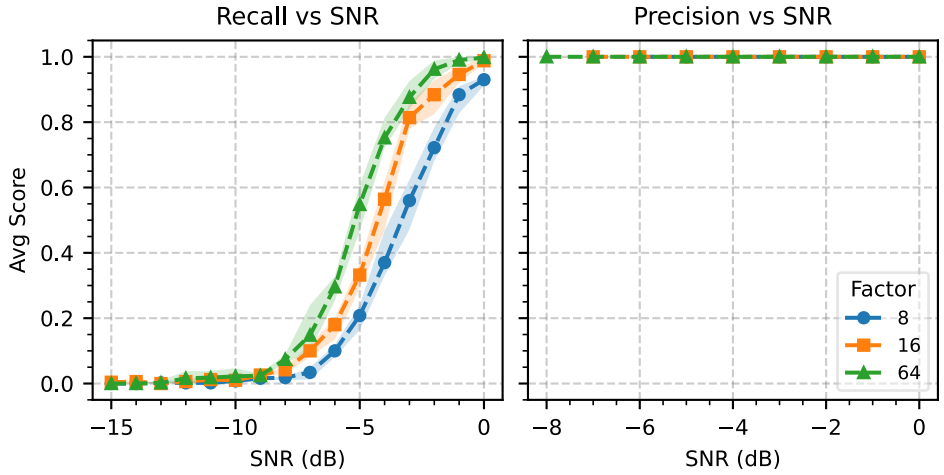
Figure 5.14: Detection performance of the PDE detection algorithm utilising different modifications.

Figure 5.15 shows the detection performance when the PDE algorithm was

modified to use a lower scaling factor, and in Figure 5.15b, the algorithm was further modified to also use downsampling 2. In Figure 5.15a, the performance started to deteriorate substantially once the factor was 4. Not only did the recall score deteriorate, but the precision score also decreased to 0.8 at -8 dB SNR. For factors above 4, the precision score was unaffected, but the recall score was reduced. When downsampling was also utilised, the performance worsened even further across all scaling factors.



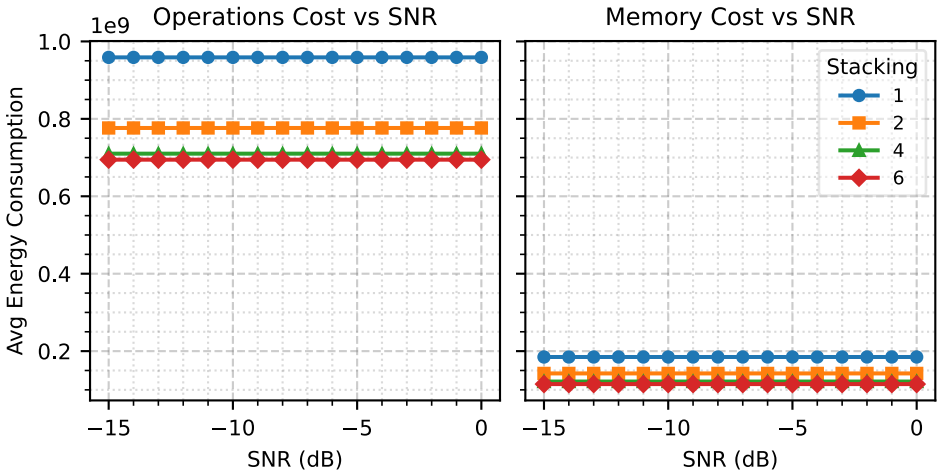
(a) Without downsampling.



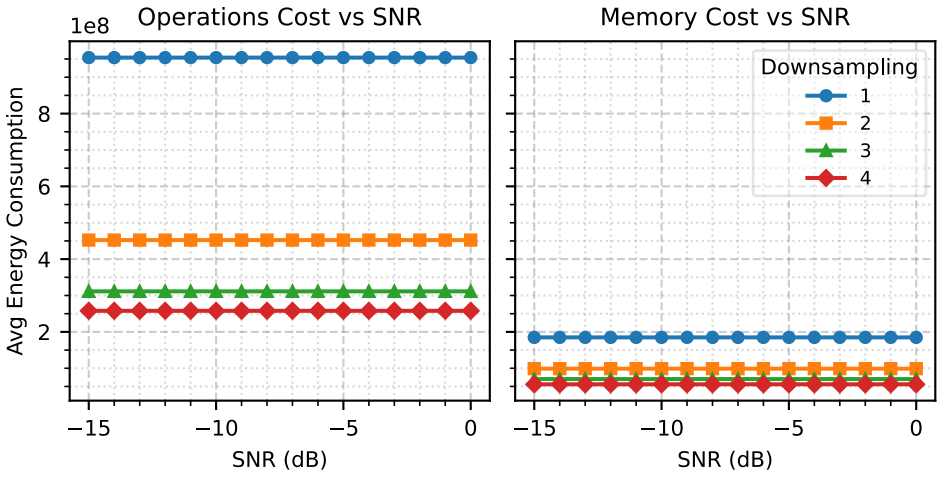
(b) With downsampling 2.

Figure 5.15: Detection performance of the PDE detection algorithm utilising different scaling factors.

Figure 5.16 illustrates the impact of downsampling or stacking on energy consumption. Downsampling offered a more substantial reduction in both operational and memory cost. Additionally, energy savings from stacking ZC waves diminished as the stack size increased.



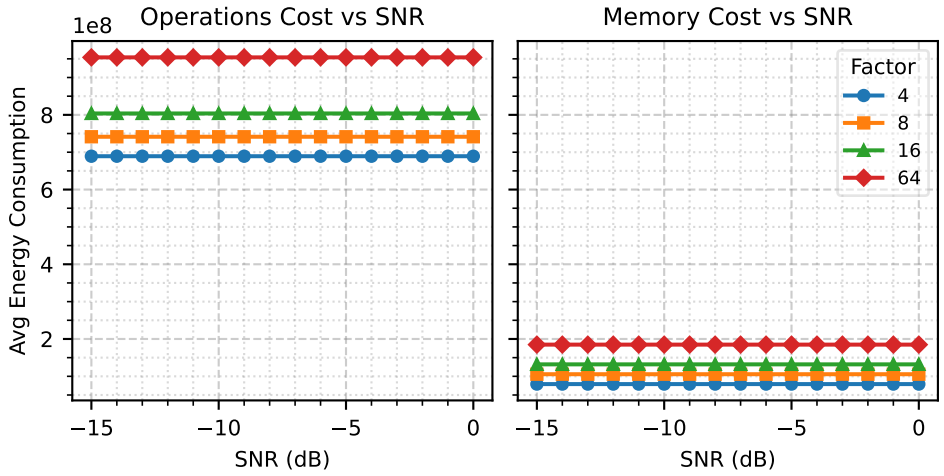
(a) Utilising stacking.



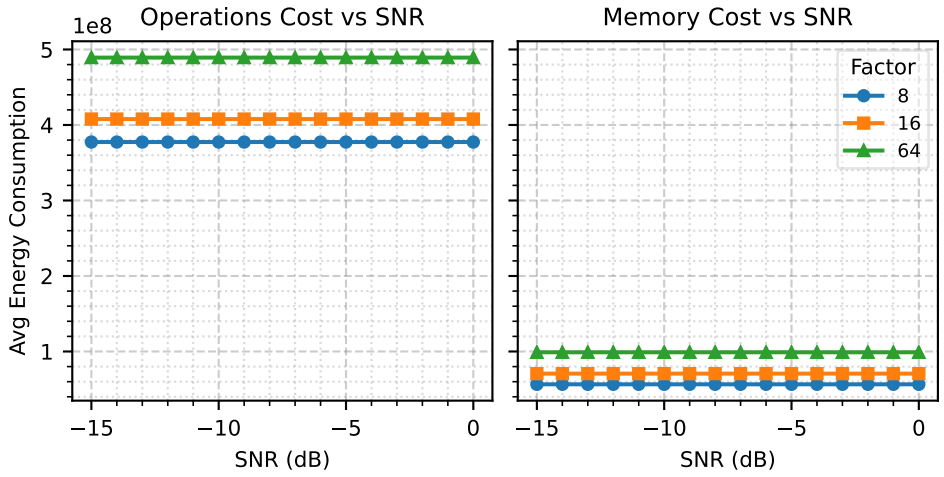
(b) Utilising downsampling.

Figure 5.16: Energy consumption of the PDE detection algorithm utilising different modifications.

In Figure 5.17, the energy consumption of the PDE algorithm modified to use a lower scaling factor is shown and in Figure 5.17b, the algorithm was further modified to use downsampling 2. Regarding operation cost reduction, lowering the scaling factor was less effective than downsampling, but it performed similarly to stacking. Notably, lowering the scaling factor yielded diminishing returns for reducing the operation cost. However, unlike stacking, memory cost reduction did not diminish and remained comparable to the yields from downsampling. Combining a lower scaling factor with downsampling lowered the memory cost even further. As observed with the FFT algorithm, the results in Figure 5.17b appeared to be the result of superimposing the two individual modifications onto one another.



(a) Without downsampling.



(b) With downsampling 2.

Figure 5.17: Energy consumption of the PDE detection algorithm utilising different scaling factors.

Chapter 6
Discussion

This chapter analyses the results gathered from the algorithms and modifications. The analysis will combine the energy consumption and detection performance to determine which algorithms and metrics show the most promise. Additionally, the analysis will also explain the features seen in the figures. Lastly, the limitations of the study are discussed.

6.1 Algorithms

When comparing the base algorithms' results shown in Figures 5.1 and 5.2, the CC algorithm is clearly inferior to the FFT algorithm. It consumes 10 times more energy whilst having the same detection performance. Consequently, the CC algorithm is excluded from the recommendations. In contrast, there is a trade-off between the FFT and PDE algorithms. Using a 0.8 recall score as the threshold for acceptable performance, the FFT algorithm drops below this limit at -14 dB SNR, whereas the PDE algorithm reaches this at -6 dB SNR. However, the superior performance of the FFT algorithm also comes with a tenfold increase in the energy consumption. From this, it can be argued that the FFT algorithm is best suited for low SNR environments, while the PDE algorithm is more appropriate for high SNR environments or applications where energy efficiency is the primary constraint.

6.1.1 Operation and Memory Costs

The computational energy consumption of these algorithms is determined by two primary factors: the total number of operations and the maximum bit sizes across individual stages (Figure 5.3). These maximum bit sizes are critical because they dictate the baseline energy cost for all operations within a specific stage. However, the total cost is heavily dominated by the computationally intensive stages detailed in Table 5.1, which completely overshadow the low-compute stages. Consequently, the high computational cost of the CC algorithm is due to the number of multiplications and additions performed in the cross-correlation stage, which is two orders of magnitude higher than that in the other algorithms' high compute stages. Although the CC algorithm features smaller maximum bit sizes than the FFT algorithm, the sheer volume of operations outweighs this advantage. Similarly, the PDE algorithm achieves a lower computational cost than the FFT

algorithm because it requires fewer total operations and mostly utilises smaller bit sizes.

As established in the energy model (Section 4.7.1), the memory cost scales with the number of memory accesses, represented as five additions each. The bit size and thereby cost of these additions depend on the scaling factor for the CC and PDE algorithms, whereas it equals the bit size of the IFFT stage for the FFT algorithm. Since the scaling factor was 64 in the base case, the bit size is 7 for both the CC and PDE algorithm, compared to 15 for the FFT algorithm (Table 5.1). Based on this known conversion between memory cost and memory operations in combination with the memory costs in Figure 5.2, it can be concluded that the CC algorithm requires the most memory operations, while the PDE algorithm requires the fewest. This finding agrees with the theoretical calculations presented in Chapter 4.

A drawback with the PDE algorithm is that all roots are tested in each detection attempt, unlike for the other algorithms, where the search is stopped once a root is found. Consequently, the PDE algorithm wastes a lot of energy, as waves are generated and cross-correlated unnecessarily. Since the search indexes over root values from lowest to highest, this is especially true for low root values. However, this energy waste will also occur in the other algorithms if no detection is made, since all roots will be tested. This could, for example, occur if the signal subsegment doesn't contain a ZC sequence or if the SNR is too low. It is the reason why the CC and FFT algorithms' energy costs increase when their recall score falls. So for the FFT and CC algorithm, the energy costs are divided into two sections: no detection and detection. The severity of this energy increase depends on how often a signal contains a ZC sequence. If they are frequent, the FFT and CC algorithm energy cost won't be noticeably affected. On the other hand, if they are uncommon, the energy cost will in reality be higher. It should be noted that the PDE algorithm does not introduce additional energy costs when no detection is made.

6.1.2 Bit Sizes

In Table 5.1a, the reason why the CC algorithm has a low multiplication bit size is due to the scaling factor, which was 64. It makes it so that the maximum bit size of the input data and generated ZC sequence is 7. Since the CC multiplication only uses these as inputs, the bit size of the multiplication block never needs to exceed $\log_2(scaling\ factor) + 1$. The same follows for the PDE algorithm, but it is one bit size bigger due to the differentiation stage, where the input can grow by one bit before it is used in the cross-correlation. For the FFT algorithm, the multiplication bit sizes are bigger; this is because during the FFT, numbers grow stepwise, up to a factor of $\sqrt{N}$, where N is the number of input elements. Consequently, the last multiplications in the FFT will have larger inputs, which define the bit size used by the stage. So, using fewer input elements would lower the necessary stage bit size. This is also why a bit shift after the FFT combination stage is necessary. Without it, the bit size would grow even larger in the IFFT stage, requiring a high bit size and significantly increasing the compute cost.

It can also be observed in Table 5.1 that most add bit sizes are bigger than

the corresponding multiplication bit size. This is because the addition steps are always performed after the multiplication steps. For example, in cross-correlation, two vectors are multiplied element-wise and then summed. Since additions are performed after multiplications, the input values can grow significantly, causing a necessity for larger bit sizes in adders. Another reason is due to the fact that the algorithms are working with complex numbers. This makes it so that a multiplication results in two additions on numbers that have already been multiplied, causing a similar scenario as mentioned earlier.

Complex numbers also increase the number of computations significantly, as one complex multiplication is 4 real multiplications and 2 real additions, and a complex addition is 2 real additions. The complex multiplications are the reason why there are considerable addition counts, even when few complex additions are made.

In the FFT and IFFT stages of the FFT algorithm, the bit size of the adder and multiplier does not differ significantly because the transform is implemented with the radix-2-Cooley-Tukey algorithm. As described in Section 2.3.2, the transform is performed in stages where multiplications are followed by additions and the output from a stage is used as input in the next stage. Consequently, the bit sizes for the operators stay connected. Notably, values only grow during additions because multiplications are only performed with a twiddle factor, which has a magnitude of 1.

6.2 Modifications

The unmodified algorithms serve as a baseline for the analysis of the modifications. They impose requirements for a modification to be deemed useful. For example, a modification shouldn't result in lower performance than another base algorithm whilst consuming more power. With this criterion, it can be determined that the CC algorithm, even with modifications, will never be applicable. This is because when the CC algorithm reaches the energy consumption equivalent to what the base FFT algorithm uses, its performance has been reduced to that of the base PDE algorithm. Simply put, the FFT algorithm was shown to be a better starting point for modifications, which is the reason why the CC algorithm wasn't tested with combined modifications. Another requirement for modifications to be deemed useful is that the recall should not drop below 0.8 at 0 dB SNR.

6.2.1 Stacking

As explained in Section 4.1, stacking sequences lowers the total number of ZC sequences that have to be tested. Consequently, the number of computations and memory operations is lowered, which in turn lowers the energy consumption. This can be observed in Figures 5.7, 5.11 and 5.16a where the algorithms utilise stacking. For the CC and FFT algorithms, the energy cost when using stacking appears to be the base energy cost divided by the stack size. However, the reduction comes at the cost of lowering the recall score. According to Figures 5.5 and 5.8, the 0.8 recall score threshold starting at -14 dB appears to move by approximately +3

dB when doubling the stack size. Notably, at a stack size of 6, the modified algorithms achieve an acceptable detection performance 1 dB lower than that of the unmodified PDE algorithm. Additionally, the FFT algorithm has an operation cost only slightly higher than that of the unmodified PDE algorithm when detections are made. However, when recall is low (no detection is made), the cost is approximately twice as high.

Due to the linearity of the cross-correlation operation, the result of cross-correlating the signal subsegment with a stack of ZC sequences is equivalent to the sum of the cross-correlations with the individual ZC sequences in the stack. Since only one ZC sequence in the stack could be the correct root value, the peak will not be magnified. The noise present in the signal, however, will propagate through each individual cross-correlation and thus be magnified in the stacked cross-correlation. Thus, noise sensitivity is increased by stacking ZC sequences before cross-correlation. The ZC sequences in the stack also have a non-zero cross-correlation with the ZC sequence in the signal subsegment, adding additional variation in the cross-correlation result.

Unfortunately, the precise mode had an insignificant effect on the total energy cost. This is because a majority of the computation and memory access is done during the stack creation and correlation. The stage where the predicted stack is checked accounts for only a minority of the cost; therefore, even if this is reduced with a precise mode, it won't be noticeable. However, if larger stacks were used or many false stacks were predicted, this balance would shift, and the stack checking stage would be more important. Unfortunately, as the recall score would be worse than the unmodified PDE algorithm, such large stack sizes are not worth it. Additionally, with a larger stack size, more ZC sequences need to be checked, and the risk of a stack being wrongfully predicted is higher. If the total number of ZC sequences that need to be checked from wrong stacks is too big, the energy costs would eventually start to increase instead.

For the detection score, the precise mode slightly degraded performance, Figures 5.5 and 5.8 shows that the recall score loss is steeper when precise mode is active. This occurs because the correct stack is detected, but the offset is wrong. As discussed in Section 4.1, this causes the precise search to fail in detecting the correct ZC sequence within the stack. Therefore, the search continues, and no detection attempt is made, producing a false negative, lowering the recall score. So, for our low stack size case, the precise mode makes detection performance worse without any extra energy reduction. Therefore, it was not used in further simulations and is not recommended as a modification.

Unlike the CC and FFT algorithms, the PDE algorithm's performance loss is not as severe as the 0.8 recall threshold moves by +1.5 dB when doubling the stack size. However, the base threshold is at -6 dB. This less severe performance loss could be because differentials $d_q[n]$ (Equation (2.2)) created from ZC sequences of length N with different roots q will be orthogonal to each other over the range of n. Stacking the differential waves will still amplify noise, but unlike with ZC sequences, the waves themselves will have a cross-correlation of 0 if their corresponding root values aren't identical. This means that, unlike stacking ZC sequences as in the FFT algorithm, the cross-correlation of the waves will not degrade the result.

It can be seen in Figure 5.16a that the energy cost reduction is less effective and that it saturates at stack size 4. Consequently, stack size 4 is the recommended maximum for the PDE algorithm. Due to how the algorithm works, all stacks will be tested in each detection; therefore, the wave generation cost in Table 5.1c will still be the same no matter the stack size. The CC stage cost will, on the other hand, be reduced as the stack size increases. Originally, these stages are similar in cost, but as the stack size increases, the wave generation will be the only dominating cost, setting a limit on the energy cost reduction. The reason why the cost in Figure 5.16a is not saturated closer to the wave generation cost of 5.4×10^8 is due to the cost from all the other minor stages and the cost from adding all the waves into their stacks.

This is not apparent in the other algorithms because their non-generation stages are far more expensive than their generation counterparts, thereby dominating the total cost. Consequently, much higher stack sizes are required for saturation to occur.

6.2.2 Downsampling

When downsampling, the number of sample points scales proportionally with $\frac{1}{k_{ds}}$, as described in Section 4.1. Since this determines the number of memory operations and the computations without changing the bit sizes in the high-cost stages of the PDE and CC algorithm (Table 5.1), the energy cost reduction should follow the same pattern. This is confirmed when looking at Figures 5.6 and 5.16b. The recall threshold in the CC algorithm appears to deteriorate the most for downsampling 2, where it moves by +3.5 dB, then lessens to +2 dB for the other downsampling levels.

The white noise present in the signal follows a normal distribution. The fact that ZC sequences have a constant amplitude of one means the cross-correlation of the noise and a ZC sequence at each offset will be a sum of normal distributions (if phase is ignored). Since the absolute value is taken after cross-correlating, phase can be ignored. Summing n normal distributions with standard deviation σ_1 will lead to a new normal distribution with standard deviation $\sigma_2 = \sqrt{n}\sigma_1$.

In the cross-correlation case, this would mean the standard deviation of the resulting noise scales as the square root of the sliding window size. The peak height would instead scale linearly with the sliding window size, since all ZC sequence indices have the same magnitude. This means the noise scales down slower than the peak does with the downsampling factor, leading to worse performance in high noise environments. From some basic tests, it was observed that the standard deviation of the noise in the cross-correlation indeed decreased by a factor of approximately $\frac{1}{\sqrt{2}}$ when the downsampling factor was changed from 1 to 2. The peak height, however, decreased by a factor of 2.

Another possible reason for the decrease in performance is the fact that downsampling deteriorates the ZC sequences themselves. The cross- and auto-correlation properties described in Section 2.1 will no longer hold, which might give rise to additional "false" cross-correlation peaks at the wrong offsets and root, which get flagged by the peak detection. These "false" peaks could explain the decrease in precision seen in Figure 5.4. It is worth noting that the decrease in precision hap-

pens when the recall is already very low. This means few true positives are found, which also means very few false positives are needed to meaningfully degrade the precision. The precision values also vary greatly between tests, as can be seen in the intervals in Figure 5.4. Nevertheless, there is a significant difference, meaning the precision does get worse. For auto-correlation, downsampling could lead to detection failure since "false" peaks make the real peak seem less statistically significant.

For the PDE algorithm, downsampling by 2 moves the threshold by +2 dB, and downsampling by 3 moves it an additional +4 dB, placing it at 0 dB with a recall score of 0.8, meaning it's almost classified as unusable. Downsampling by 4 does place the recall score below 0.8 at 0 dB; therefore, downsampling by 3 is the limit before the PDE algorithm becomes unusable. Similarly to downsampling the CC detection algorithm, the peak seems to scale linearly with the number of indices, while the noise scales as its square root. Since the noise of the differentiated signal no longer follows a normal distribution, however, it would be presumptuous to assume it would scale in the same manner. Nevertheless, the data suggest it does.

Downsampling also has interesting effects on the differentials themselves. For a downsampling factor of 2, Equation (2.2) essentially changes to

$$\widetilde{d_q}[\widetilde{n}] = e^{j4\pi q\widetilde{n}/N} \tag{6.1}$$

, where $\widetilde{n} = 1, 2, ..., \lfloor N/2 \rfloor$. This function no longer has an integer amount of periods over the span of $\widetilde{n}$, meaning the orthogonality of trigonometric functions, which helped differentiate different root values starts to break down. Aliasing also becomes an issue when downsampling. For a length $N = 601$ and root value $q_1 \geq 301$ the differentiated signal becomes

$$\widetilde{d_{q_1}}[\widetilde{n}] = e^{j4\pi\widetilde{n}(301+x)/601} = e^{j4\pi\widetilde{n}(0.5+x)/601} * e^{j2\pi\widetilde{n}} = e^{j4\pi\widetilde{n}(0.5+x)/601} \tag{6.2}$$

, where $x = q_1 - 301$.

$\widetilde{d_{q_2}}[\widetilde{n}]$ where $q_2 = x$ or $q_2 = x + 1$ will considerably interfere with $\widetilde{d_{q_1}}[\widetilde{n}]$, creating a peak at those root values as well.

6.2.3 Scaling Factor

When lowering the scaling factor, the number of computations and memory operations remains; however, as seen in Figure 5.17 and Figure 5.17a, the operation cost decreases. When going from a scaling factor of 64 to 16, the distributions in Figure 5.3 are shifted two steps to the left with bit size 1 being the minimum, as this is needed to represent the number 0. Consequently, the bit sizes in the compute-heavy stages shown in Table 5.1 are reduced by the step size, thereby lowering the cost per addition and multiplication.

This reduction is more significant for multiplications as the cost scales quadratically with the scaling factor. The reduction also depends on the starting bit size and the step size of the shift. For example, shifting 2 bits when the maximum bit size is 15 reduces the operational cost of addition to $\frac{15-2}{15} \approx 0.86$ of the original

cost and multiplication to $\left(\frac{15-2}{15}\right)^2 \approx 0.75$. For a maximum bit size of 7, these relative costs drop to 0.71 and 0.51, respectively.

When going from a scaling factor of 8 to 4 in the PDE algorithm (Figure 5.17a), the operation cost reduction can be seen saturating. This is due to the same reason as discussed in Section 6.2.1, the factor only reduces the cross-correlation stage and not the generation stage, which is why we see a similar operation cost threshold (7×10^8) as seen for the stacking modification. Unlike for stacking, the memory cost reduction never saturates because all memory costs in the algorithms are linearly proportional to the scaling factor.

From Figures 5.10 and 5.15, it can be deduced that the FFT algorithm has a higher tolerance for lower scaling factors than the PDE algorithm when it comes to recall score. A scaling factor of 16 shows minimal loss in recall score for the FFT algorithm, with a scaling factor as low as 8 showing only a difference of approximately +1 dB for the detection threshold compared to a scaling factor of 64.

When using a scaling factor lower than 8, however, precision seems to collapse for the FFT algorithm, as is visible in Figure 5.9. The drastic decrease in precision for a scaling factor of 4 while recall stays high indicates that the peak detection falsely flags the wrong roots but that correct roots are still not discarded. Since both the signal and the ZC sequences used for cross-correlation are highly quantised for low scaling factors, the mathematical properties of the ZC sequences will start to break down. This might, like in the case of downsampling, lead to "false peaks" that trigger the peak detection. Additionally, it could also lead to degradation of the correct peak, so that it is not detected. It is also possible that the distribution of values in the result for low scaling factors changes. If that's the case, the exact values for the peak detection, see Section 4.3, could be tuned to increase precision for low scaling factors.

The PDE algorithm is more sensitive to the scaling factor when it comes to recall. The higher sensitivity indicates that a higher scaling factor is required to maintain mathematical properties to be able to properly predict the root. Unlike the FFT algorithm, however, precision stays high for low scaling factors. This is due to the differences in process steps. For the PDE algorithm, the most likely root is found and then tested using one iteration of the FFT algorithm. To get a false positive and lower the precision score, the wrong root would have to be found by the PDE algorithm and also be flagged as a positive by the FFT algorithm step. For the FFT algorithm, an average of approximately $N_{zc}/2$ roots would have to be discarded before getting the chance to even detect the correct root. One positive out of these roots is enough to yield a false positive result, lowering the precision.

Changing the FFT algorithm scaling factor to 16 yielded a 20% reduction in energy costs compared to a scaling factor of 64 with negligible detection performance degradation. Therefore, a scaling factor of 16 should be implemented as a baseline configuration for the FFT algorithm. A scaling factor of 8 shows minor power savings compared to 16, with some degradation of detection performance. It is a possible option, but not good enough to be recommended. For the PDE algorithm, a scaling factor of 16 shows approximately 15% reduction in energy costs compared to a scaling factor of 64 while reducing the detection threshold by about +1 dB. It is not recommended to go lower than 16 in the scaling factor for

this algorithm, since further reductions in energy costs are limited with significant detection performance losses.

6.3 Combining Modifications

When combining modifications, the CC algorithm was excluded due to the FFT algorithm being a much better starting point. As seen in Figures 5.10 and 5.13, the FFT algorithm was modified to utilise stacking and a lower scaling factor. It was decided to exclude the precise mode, as it only introduced drawbacks. Similarly, the scaling factor 4 was excluded because it made the algorithm unusable. As deduced, the scaling factor 16 is almost a free energy reduction, and when combined with a stack size of 6, the operation's cost comes very close to that of the unmodified PDE algorithm, whilst performing better, as the recall threshold is 1 dB lower than for the unmodified PDE algorithm. Additionally, the recall score loss is not as steep, maintaining some usability for dBs beyond the threshold. However, once recall drops, the operations cost increases beyond that of the PDE algorithm. Also, even when detections are made, the total energy cost is almost 50% higher because the memory cost in the FFT algorithm is 4 times higher than that in the unmodified PDE algorithm. Higher stacking values were tested, but not presented, because the detection performance was worse than in the unmodified PDE algorithm, whilst consuming more energy. Therefore, the PDE algorithm should generally be chosen for SNR values above -6 dB.

Using lower stack sizes in combination with a scaling factor of 16 is still beneficial if the detection performance doesn't have to be the best, but still needs to perform better than the PDE algorithm. Combining a scaling factor of 8 instead of 16 with any stack size is not recommended, as the detection performance loss outweighs the energy cost reduction.

For the PDE algorithm, stacking was tested with downsampling 2, but due to the low starting recall score, no stacking could be made without having the recall score drop below 0.8 at 0 dB. Therefore, the results were not shown, and instead, we focused on combining the scaling factor with downsampling 2. It was chosen over stacking 2 because it lowers the energy cost more, whilst providing a similar detection performance. Additionally, higher stacking sizes had limited energy reduction capabilities. Similarly, higher downsampling values were not explored as they reduced the detection performance too much to be able to be combined with a lower scaling factor. Lastly, scaling factor 4 was excluded from the combination because the recall score was too low. From Figures 5.15b and 5.17b it can be deduced that factor 16 is the most optimal combination with downsampling 2, as it barely reduces the recall score whilst still providing a noticeable energy reduction.

6.3.1 Other Modifications Explored

There were two other modifications explored, but due to the suboptimal results, they were not included in the results. The first of these was a modification to the FFT algorithm where ZC sequences are placed in series instead of being stacked.

This modification makes use of the otherwise zero-padded space described in Section 4.2, removing the cost of adding sequences whilst still testing multiple sequences per correlation. Not having to add the sequences in a stack does decrease the cost, but it is not noticeable since the correlation is the costly step. From tests, it was seen that it performed slightly worse than the stacking variant. Additionally, the maximum number of ZC sequences per series is limited by the length of the zero-padded space. Therefore, we chose to prioritise the stacking algorithm.

The second modification was for the CC algorithm. It changes the cross-correlation to compute the imaginary and real parts separately and then sums them, removing 1 addition and 2 multiplications per otherwise complex multiplication. Essentially, this ignores the imaginary part of the cross-correlation result and is equivalent to taking the real part of the unmodified result. However, it was found that this only works when there is no phase difference. Because when only looking at the real part and not the magnitude of the correlation, matching signals with a phase difference becomes increasingly difficult to detect as the difference grows. Therefore, this modification would not work in realistic scenarios where phase differences are impossible to avoid and can be caused by multiple factors, such as a frequency offset between the transmitter and receiver.

6.4 Recommendation

We do not believe that there is a one-size-fits-all algorithm; instead, we believe that the best algorithm depends on the SNR environment and use case. The algorithm should be chosen to best fit the required SNR performance, so as not to waste energy on unnecessary performance. To provide a smooth transition between performance and energy cost levels, modifications to the base algorithms can be used. Starting with the FFT detection algorithm yields the highest performance and energy consumption, but by using the stacking modification (without the precise mode active), multiple detection performance levels and energy levels are introduced. Once the modified FFT algorithm comes close to the performance of the PDE algorithm (stack size 6), it is best to switch over to the PDE algorithm and provide further options by modifying it with stacking, downsampling or scaling factor (with and without downsampling 2). The modifications discussed above serve as a guide for viable modification choices alongside their trade-offs and limitations.

For scenarios where ZC sequences occur rarely in a signal, the FFT detection algorithm will consume between 1.5 and 2 times the energy it would when detecting sequences. This scenario favours the PDE algorithm, which does not increase energy consumption when no detection is made. Therefore, when ZC sequences are uncommon, and the required SNR is at most -1 dB below the recall threshold of the PDE algorithm (-6 dB), it can be argued that it would be more worthwhile to use the PDE algorithm instead of the FFT algorithm with stacking 6. Although it has a recall score of 0.8 at -7 dB, in contrast to the recall score of 0.4, which the PDE would have, the energy consumption would be too high to justify the increase in performance.

An advantage of the FFT detection algorithm is that the recall degradation

slope is less steep than in the PDE algorithm, so even if the SNR falls below the 0.8 recall threshold, the algorithm will still work for another -2 dB, but with a recall score as low as 0.4 instead. For the PDE algorithm, it only takes a -1 dB step. In scenarios where the SNR sometimes falls below the algorithm's recall threshold, and not missing the ZC sequence is crucial, the FFT algorithm is favoured. For such cases, it can be argued that the extra energy cost is worth the less SNR sensitive detection performance.

6.5 Limitations

One limitation encountered during the writing of this thesis is the difficulty of estimating the power draw of an integrated circuit when simulating its function in software. It will depend not only on exact hardware implementation and chosen technology node, but also on the exact inputs to the computational elements of the circuit and in which order these inputs appear. For the calculations made in this thesis, it was assumed that all additions/multiplications will depend only on the bit size of the adder/multiplier itself, and not the input values. In reality, the binary representations of the values that occur during operations matter a lot. For an addition, adding the numbers 16 (signed binary 010000) and 2 (signed binary 000010) won't result in any carries between adder blocks (result 010010), but addition between the numbers 15 (signed binary 001111) and 1 (signed binary 000001) will result in a ripple carry over essentially the entire adder (result 010000). The amount of bits required is identical, but the amount of switching varies greatly. However, the data being processed in this thesis is largely noise. Therefore, it was assumed that, on average, the number of bits switched in the inputs would scale linearly with the number of bits required by a stage, and the energy for arithmetic operations would scale as described in Section 2.4.1.

Calculating the energy consumption of memory accesses is especially difficult. The comparison between additions and multiplications is simple, since the latter is constructed using the former. Memory is an entirely different function from arithmetic, meaning the implementation of it could differ even if the arithmetic functions stay constant. It is also difficult to find sources with power consumption figures for modern process nodes, making it difficult to create an energy model for memory access based on literature. This is why the energy costs of the memory operations are also shown separately from the arithmetic operations. The conversion factor between the two, as described in Section 4.7.1, gives the possibility of comparing total energy usage, but should be considered a rough estimate. It's more useful to keep arithmetic and memory separate while comparing the different algorithms' energy costs.

Since the absolute power draw of a circuit depends on many factors, the energy costs were calculated in arbitrary units in this thesis. The numbers were calculated by using the scaling of energy consumption described in Section 2.4.1. This allows for comparison between the algorithms, but provides no information about absolute energy usage.

This thesis also did not consider many of the non-idealities present when actually transmitting and receiving signals. Frequency offsets, other signals with

quadratic phase, and other factors could affect the performance of the detection algorithms.

Chapter 7

Conclusion and Future Works

7.1 Conclusion

In conclusion, the unmodified CC detection algorithm had no applicable scenario due to its inferior energy-to-detection-performance ratio when compared to the unmodified FFT detection algorithm. Furthermore, it was proven that there were no possible modifications to the CC algorithm that could make it applicable. In contrast, the unmodified PDE algorithm was found to offer an energy-cheap alternative, best suited for high SNR environments.

Across most algorithm modifications, a direct correlation between detection performance loss and energy reduction was found. The only exception to this was lowering the scaling factor to 16 in the FFT algorithm, decreasing energy consumption by 20% without affecting the detection performance. As such, it should be the new baseline FFT detection model.

When modifying the PDE algorithm, it was observed that the energy reduction from stacking waves quickly saturated at a stack size of 4. The same could be seen when lowering the scaling factor; however, it only resulted in a saturated reduction when the factor was as low as 4, and it only affected the operation cost. When downsampling in the PDE algorithm, the recall quickly dropped below 0.8 at 0 dB, limiting the downsampling to a maximum of 3. So, for these reasons, the viable modification values in the PDE algorithm are rather limited.

For stacking in the FFT algorithm, it was found that the precise mode did not noticeably decrease energy consumption and slightly reduced detection performance; therefore, it was discarded. It was also found that the energy cost when stacking was equal to the base cost divided by the stack size. When comparing it to the unmodified PDE algorithm, it could be concluded that stacking ZC sequences in the FFT algorithm could never reduce the energy consumption further, whilst maintaining a better detection performance. Therefore, instead of increasing the stack size beyond 6 in the FFT algorithm, the PDE algorithm should be chosen and modified if further energy reduction is desired.

Ultimately, the optimal selection of base algorithms and modifications depends on the intended SNR environment. The modifications provide a transition between the high-performance FFT algorithm and the low-energy consumption PDE algorithm, effectively minimising energy consumption without sacrificing necessary detection performance.

The biggest limitation of this thesis is the simplified estimation of power consumption for arithmetic and memory operations. First, the energy cost for multiplications and additions was assumed to depend only on the operator bit size. This approach neglects that energy consumption is dynamic, depending on the binary representation of the inputs. Second, the energy consumption of memory operations was assumed to scale linearly with the bit size of elements stored, and the conversion from a memory operation to energy cost was a rough estimate. Third, power consumption was only modelled in arbitrary units. While this allows for relative comparison, it provides no information about absolute power usage.

7.2 Future Work

One area which was not investigated during the course of this thesis was non-idealities in the transmission and reception of the signal. In reality, things like relative velocity between transmitter and receiver will introduce a frequency offset. Further studies could explore how these non-idealities affect the different detection algorithms presented in this thesis.

Another potential improvement to the methods outlined in this thesis would be to have more frequent downshifting of values, ensuring they don't grow too far from their original values. It's visible in Figure 5.3 that the values grow substantially in some process steps, which increases energy consumption. It's entirely possible that limiting the growth of values by downshifting more frequently would result in worse performance, but it would also save a lot in terms of energy.

Another interesting alternative for reducing the power consumption of arithmetic operations is the use of approximate circuits. Approximate circuits reduce the cost of operations by reducing the amount of logic in the circuit, which will introduce fixed-bias errors in the calculations. There are many ways to construct approximate computing circuits, but the general idea is to reduce the number of gates inside the multiplier/adder to reduce the amount of energy used to perform an operation. The fixed-bias errors in the calculations will depend on the exact circuit implementation. Therefore, the decrease in accuracy will be deterministic; a particular set of inputs will always give the same output. For all cases, to use approximate computing circuits, one has to trade energy consumption for accuracy. A library of approximate adders and multipliers which could be used for this purpose can be found in [18].

This thesis also consisted exclusively of simulations. Implementing a version of the detection algorithms on a field-programmable gate array to run them on actual hardware would give insights as to how they perform in more realistic scenarios. This will, in all likelihood, be more energy-intensive than for a dedicated ASIC solution, but it could be more realistic than the energy consumption estimations made in this thesis without the expense of developing and producing an ASIC.

References

[1] Andrews JG. A primer on Zadoff-Chu sequences. Available from: https://arxiv.org/pdf/2211.05702.

[2] Zhou F, Wang P, Ozger M, Cavdar C. Blind Detection of Drones using OFDM-Based Zadoff-Chu Sequences with Field Tests; 2025. p. 1482-7.

[3] Zhou F. Blind Detection of Unmanned Aerial Vehicles using OFDM-Based Zadoff-Chu Sequences. 2024. Available from: https://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A1945262&dswid=-8652.

[4] Au Yeung CK, Lo BF, Torborg S. Drone Detection via Low Complexity Zadoff-Chu Sequence Root Estimation. In: 2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC); 2020. p. 1-4. Available from: https://ieeexplore.ieee.org/document/9045243.

[5] Blackledget JM. Chapter 4 - The Fourier Transform. In: Blackledget JM, editor. Digital Signal Processing (Second Edition). second edition ed. Woodhead Publishing Series in Electronic and Optical Materials. Woodhead Publishing; 2006. p. 75-113. Available from: https://www.sciencedirect.com/science/article/pii/B9781904275268500051.

[6] Anand AV. A Brief Study of Discrete and Fast Fourier Transforms; 2010. Participant paper, University of Chicago Mathematics REU. Available from: https://math.uchicago.edu/~may/VIGRE/VIGRE2010/REUPapers/Anand.pdf.

[7] Cooley JW, Tukey JW. An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics of Computation. 1965;19(90):297-301. Available from: http://www.jstor.org/stable/2003354.

[8] McFee B. Radix-2 Cooley-Tukey. In: Digital Signals Theory; 2022. Accessed: 2026-05-20. Available from: https://brianmcfee.net/dstbook-site/content/ch08-fft/FFT.html.

[9] Lyon D. The Discrete Fourier Transform, Part 6: Cross-Correlation. Journal of Object Technology. 2010 03;9:17-22.

[10] Jacob B, Ng SW, Wang DT. OVERVIEW: On Memory Systems and Their Design. In: Jacob B, Ng SW, Wang DT, editors. Memory Systems. San Francisco: Morgan Kaufmann; 2008. p. 1-54. Available from: https://www.sciencedirect.com/science/article/pii/B9780123797513500023.

[11] Montalvo LA, Parhi KK, Satyanarayana JH. Estimation of average energy consumption of ripple-carry adder based on average length carry chains. In: VLSI Signal Processing, IX; 1996. p. 189-98.

[12] Ellaithy D. Approximate Multiplication and Division Calculation for DSP Applications. Applied and Computational Mathematics. 2024 09;13:178-85.

[13] Hettiarachchi DLN, Davuluru VSP, Balster E. Integer vs. Floating-Point Processing on Modern FPGA Technology; 2020. p. 0606-12.

[14] Horowitz M. 1.1 Computing's energy problem (and what we can do about it). In: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC); 2014. p. 10-4.

[15] Gauthier L, Ishihara T. Implementation of Stack Data Placement and Run Time Management Using a Scratch-Pad Memory for Energy Consumption Reduction of Embedded Applications. IEICE Transactions. 2011 12;94-A:2597-608.

[16] Dennard RH, Gaensslen FH, Yu HN, Rideout VL, Bassous E, LeBlanc AR. Design of ion-implanted MOSFET's with very small physical dimensions. IEEE Journal of Solid-State Circuits. 1974;9(5):256-68.

[17] Burt RW. An Improved Digital Algorithm for Fast Amplitude Approximations of Quadrature Pairs. The Deep Space Network Progress Report. 1974;42-20:114-8.

[18] Mrazek V, Hrbacek R, Vasicek Z, Sekanina L. EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2017; 2017. p. 258-61.