Hardware Architecture for Order Statistic CFAR Algorithms

by

José Roberto Pérez Andrade

A dissertation

submitted to the program in Computer Science,

Computer Science Department

in partial fulfillment of the requirement for degree of

MASTER IN COMPUTER SCIENCE

at the

Instituto Nacional de Astrofísica, Óptica y Electrónica

October 2008

Tonantzintla, Puebla

Advisor:

Dr. René Armando Cumplido Parra

Research Scientist

INAOE

Abstract

The radar signal processing for sea navigation environments is useful for target detection and target localization, because a response in short time is requiered. The research on this area has been focused on variants of Constant False Alarm Rate (CFAR) detectors and sea clutter modeling. CFAR detectors are used in digital signal processing applications to extract targets from background in noisy environments. Others examples of applications are: image processing, medical engineering, power quality analysis, features detection in satellite images, Pseudo-Noise (PN) code detectors, among others. This thesis presents a hardware architecture that implements six variants of the CFAR detector based on linear and non-linear operations for radar applications and, it details the selection of these six variants and its parameter selection. Since some implemented CFAR detectors require sorting, a linear sorter based on a First In First Out (FIFO) schema is used. This sorter is capable of discarding the oldest datum and inserting the incoming data while keeping the rest of the data sorted in a single clock cycle. The sorter is composed of identical processing elements, thus it can be easily adapted to any data lengths, according to the specific application needs. This FIFO sorting process is described by four different parallel functions that exploit the natural hardware parallelism.

The proposed CFAR hardware architecture can be used as a specialized module or as a co-processor for Software Defined Radar (SDR) applications. The linear sorter can be used as a coprocessor or as a module in specialized architectures that continuously require to process data for non-linear filters based on order statistics.

The results of implementing the CFAR hardware architecture on a Field Programmable Gate Array (FPGA) are presented, discussed and compared against other works. Also, results of implementing the linear sorter on a FPGA are presented and compared against other reported hardware based sorters. Scalability results for several sorted elements with different bit widths are also presented.

Resumen

El procesamiento de señales de radar para entornos de navegación marítima, es útil para la detección y localización de blancos, donde se requiere un tiempo de respuesta en un lapso corto de tiempo. La investigación en el área de detección de blancos en señales de radar, se ha enfocado a la búsqueda de variantes del detector CFAR (Constant False Alarm Rate) y al modelado del ruido marítimo, obteniendo buenos resultados teóricos. Los detectores CFAR son usados en el procesamiento digital de señales, con el fin de detectar blancos en entornos marítimos, donde las condiciones ambientales existentes representan ruido añadido a la señal de interés. Sin embargo, los detectores CFAR también son usados en aplicaciones de procesamiento de imágenes, ingeniería médica, análisis de calidad de potencia, detección de características en imágenes satelitales, detectores de códigos PN (Pseudo-Noise), entre otras aplicaciones. Esta tesis presenta una arquitectura hardware que implementa seis variantes del detector CFAR basados en operaciones lineales y no lineales, además, detalla la selección de éstas seis variantes y la selección de sus parámetros. Ya que algunos detectores CFAR requieren ordenamiento, es usado un ordenador lineal basado en un esquema FIFO (First In First Out). Este ordenador es capaz de mantener una serie de datos ordenados, descartando el dato más antiguo e insertando un nuevo valor en su lugar correspondiente; realizando todas estas operaciones en un solo ciclo de reloj. El ordenador puede ser adaptado a cualquier longitud de datos, de acuerdo a las necesidades de la aplicación. Este ordenador está compuesto de elementos procesadores idénticos, cuyo funcionamiento es descrito por cuatro funciones concurrentes.

La arquitectura hardware CFAR propuesta puede ser usada como un modulo especializado o como un coprocesador para aplicaciones de Software Defined Radar (SDR). Por otro lado, el ordenador lineal puede ser usado como un coprocesador o como un modulo especializado en diseños que requieran procesar datos de manera continua o para filtros no lineales basados en estadísticas de orden.

Los resultados de la implementación de la arquitectura hardware CFAR propuesta y del ordenador lineal, ambos funcionando en un FPGA (Field Programmable Gate Array) son reportados, discutidos y comparados contra otros trabajos similares. También, resultados de escalabilidad para el ordenador para diferentes tamaños de palabras y cantidad de elementos ordenados son presentados.

Acknowledgments

Thanks to my parents, Pascual and Amelia, to my brothers Isaac and Saraí, for their support since I took the decision to leave my hometown, looking for enhance my skills. I know that it was not easy for all of us, but I am pretty sure it was worth it.

I am especially grateful to Dr. Arturo Sánchez Carmona, for his guidance before embarking on my master's studies, for encourage me to look far away and for believing in me to start larger projects.

I would like to express my deepest gratitude and appreciation to my thesis advisor: Dr. René Cumplido Parra, for its excellent guidance conducting this research, by the time he devoted to me since the beginning of this work, for his patience and specially for the opportunity that he gave me of knowing other places and other cultures. Also, I would like to express my gratitude for the comments of my thesis reviewers: Dra. Claudia Feregrino Uribe, Dr. Miguel Octavio Arias Estrada and Dr. Jesús Ariel Ochoa Carrasco. Their invaluable suggestions have enriched this work.

Thanks a lot to CONACyT, the National Council for Science and Technology from Mexico, for having sponsored my master studies, and the development of this research, through the scholarship number 204500.

I am extremely greatful with Adriana, my girlfriend, for your moral support and for your patience. Thanks for encourage me with your words, and overall thanks for illuminating my life this last year.

Thanks to my friends than made easy my life during my passage through the INAOE, without your support and friendship, I hardly had survived this two years: Caro, Hugo, Artemio, Alberto, Fernando, Atlántida and Alejandro, I will never forget you.

Finally, I am eternally thankful to God. Thanks for all the situations that he gave me, the good and bad times, because without these situations I would not have learned significant lessons. Thanks God for give me my parents, my brothers, my girlfriend, my friends and my teachers through my studies at the INAOE.

I want to dedicate this work to my parents, because they have been my support, my model to follow and my guidance throughout my life.

This thesis is only one goal accomplished by us, one of countless goals that wait us to be fulfilled.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation presents the design, implementation and validation on a Field Programmable Gate Array (FPGA) of a versatile architecture for some linear and nonlinear Constant False Alarm Rate (CFAR) detectors. These CFAR detectors are used in digital signal processing applications to extract targets from background in noisy environments, e.g. target detection in radar environments, image processing, medical engineering, power quality analysis, features detection in satellite images, Pseudo-Noise (PN) code detectors and Code Division Multiple Access (CDMA) wireless networks. Although there are multiple CFAR detector applications, the investigation and this dissertation are focused on radar topics.

1.1 Problem Approach

Navigation radars simplify the navigation in maritime environments. Unfortunately the weather conditions (clutter), thermal noise from radar devices, pulse jamming or other undesired echo received by the radar represent a problem because they add noise to the target signal. The problem of detecting these target signals in background noise of unknown statistics is also a common one in other sensor systems such as air radars and sonars.

Adaptive digital signal processing techniques are often used to remove noise and to enhance the detectability of targets in many situations. It is important for signal processing systems to operate in non-stationary background noise environments with a predetermined constant level of performance *i.e.*, in signal processing terms, the objective is to maintain a low constant false alarm rate. A possible solution that can be used to overcome the problem of noise added to the target signal is the CFAR detector, which sets a threshold adaptively, based on local information of total noise power. The threshold in a CFAR detector is set on a cell by cell basis, using estimated noise power by processing a group of reference cells surrounding the cell under investigation [1, 2].

There are various CFAR techniques proposed in the radar literature to deal with different problems present in radar applications. These techniques require linear operations (such as getting the average, the major or minor of a set of values) or nonlinear operations like sorting and selecting a value before performing a linear operation. These different techniques have been developed in order to deal with two situations that had required careful investigation. These problems are presented in regions of clutter transitions and multiple target situations. The first situation occurs when the total noise power received within a single reference window changes abruptly, leading to excessive false alarm or target masking. The second situation is encountered when there are two or more closely spaced targets in the reference cells, leading the CFAR detector to report only the strongest of the targets [2].

These situations mentioned in the literature shown that although the theoretical aspects of CFAR detectors are very advanced [2, 3, 4, 5, 6], there are few practical digital implementations. Traditionally, analog implementations have been used in radar systems for several years. In recent times, developments in programmable logic have made practical to explore digital implementations of CFAR and other algorithms to support the Software Defined Radar (SDR) paradigm. SDR systems can be implemented using programmable logic to accommodate various radar sensors for different detection conditions. This means they can be changed in run-time either by control of stored software or by downloading new functions [7, 8]. Using a configurable architecture implemented on an FPGA, is an alternative to avoid the functional-fixed hardware since it allows the modification of certain parameters of the CFAR detectors.

Using a configurable architecture helps to meet the real-time target detection requirement and provides SDR support. This real-time performance of adaptive digital signal processing algorithms is not achieved by general purpose processors or digital signal processors (DSPs) due to the high computational load of these digital processing algorithms. Both, general purpose processors and DSP do not meet the high computational requirements involved in radar signal processing chain because they are exceeded by the intrinsic requirements of the CFAR detectors. The intensive computational requirements due to the high data rate in radar signal processing cannot be met only by technology advances *i.e.*, smaller transistor area and thus speed improvements; but also by designing specialized architectures based on parallel computing [9]. However, when selecting some of these detectors to be implemented comes up the question: What detector should be selected?. Until now, there is not an optimum detector capable of dealing with the different situations explained above. A CFAR detector has a better performance for certain situation, meanwhile for other different situation a distinct CFAR detector has a better performance that the previous one. The CFAR performance is also affected by changing some of its characteristics like the false alarm desired. For these reasons, having a flexible CFAR detector implemented on a configurable architecture, capable of changing some of its characteristic or even changing the algorithm in run-time is an alternative to deal with different situations, providing real-time target detection and SDR support.

1.2 Objective

1.2.1 Main Objective

As a CFAR detector that can be considered optimal under any environmental circumstances has not been proposed yet, the objective of this research work is to design and implement a hardware architecture for different CFAR detectors, able of modify some of its parameters providing robustness to the target detection process, and being suitable to be implemented in SDR systems. The performance of the resulting architecture must meet the high data rate needed in the radar signal processing chain.

1.2.2 Secondary Objectives

- Design an architecture that uses few FPGA area resources for the CFAR detectors implemented.
- Select or design a sorting scheme for nonlinear detectors.
- Analyze the trade-off of the CFAR detectors, in order to help in the selection for being implemented on a hardware architecture.

1.3 Methodology

The following is an enumeration of the steps needed to accomplish the goals and to solve the problem:

1. For the developing of this investigation it was needed to understand radar basics.
2. Search and select linear and nonlinear CFAR detectors for being implemented as well as to understand the principal CFAR terms.
3. Select the most commonly used CFAR detectors and those whose characteristics were similar for being implemented.
4. Search in the literature a sorting hardware schema for the nonlinear detectors, providing some ideas for the design of a custom sorting schema.
5. Design and test the CFAR detector architecture in basis on the custom sorter schema designed.
6. Test the CFAR detector architecture for several configurations used in radar applications and test the sorting schema for various scalability configurations.

1.4 Thesis Overview

The rest of the dissertation is organized as follows:

- **Chapter 2** provides a theoretical fundament introduction to radar concepts as well as its functionality principles. The target detection process, CFAR linear and CFAR nonlinear detectors are explained.

- **Chapter 3** gives an overview of CFAR theoretical works and their implementations on systolic and specialized architectures. Also, the hardware sorters available in the literature are reviewed.

- **Chapter 4** explains the proposed CFAR hardware architecture and its specifications according to the radar requirements. Also, the sorter schema designed as an algorithm with some functionality examples is explained.

- **Chapter 5** presents the sorter scalability synthesis and performance results. The CFAR detector synthesis results and its general performance are presented too.

- **Chapter 6** summarizes the thesis and the conclusion and future work are presented.

Chapter 2

Theoretical Aspects

New radar techniques and applications have fueled the continuous growth of radar since its conception in 1930. During the World War II, countries like United States, United Kingdom, Germany, Soviet Union, France, Italy, Japan and Netherlands led the radar development. In the following years radar development was mainly concentrated on aspects that were not completed during the war. Among these developments are high-power stable amplifiers, Moving Target Indicator (MTI), pulse compression, Synthetic Aperture Radar (SAR), cognitive radar and SDR systems.

2.1 Radar General Aspects

The radar (acronym for **Ra**dio **D**etection **A**nd **R**anging) systems use electromagnetic waves for detection and location of reflecting objects such as aircrafts, ships, spacecraft, vehicles, people, weather formation, and terrain. In the case of moving objects it helps to identify their distance, altitude, direction and/or speed. The radar operates radiating energy into space and detecting the echo signal reflected from an object. Although the echo is usually very weak, it can be amplified. The reflected energy that is returned to the radar not only indicates the presence of a target, but by comparing the received echo signal, with the signal that was transmitted, its location can be determined. Radar can perform its function at long or short distances and under conditions impervious to optical and infrared sensors [1].

The basic principle of radar is illustrated in figure 2.1. A transmitter generates an electromagnetic signal that is radiated to the environment by an antenna. A portion of the energy of this signal is intercepted by a target and reradiated in many directions. Some of the reflected energy, that is directed toward the radar, is received by the same antenna and it is sent to the receiver. At the receptor, the electromagnetic signal is digitized and raw data are obtained which are ready to be processed. These data are processed to detect the presence of a target and determine its location. A single antenna is usually used on the time-shared basis for both transmitting and receiving when the radar waveform is a repetitive series of pulses.

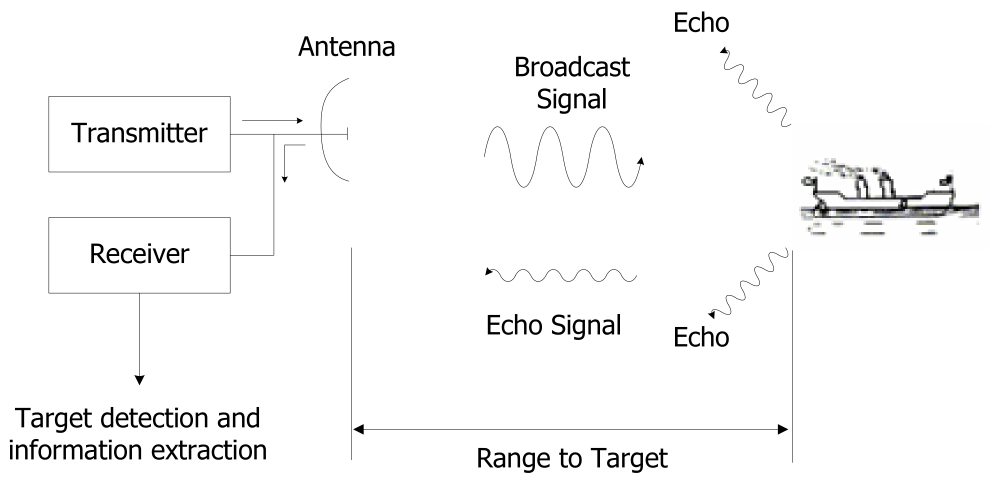


Figure 2.1: Radar signal processing system.

In order to understand the main radar characteristics, a brief introduction to these concepts is needed.

Radar Waveforms

Radar systems are normally divided into operational categories based on energy transmission methods: pulse method, continuous wave method, and frequency modulation method. The pulse method is the most common for transmitting radar energy. The other two methods are used in special applications.

Continuous wave radar uses the Doppler effect to detect the presence or speed of an object moving toward or away from the radar. It is used by fire control systems to track fast moving targets at close range. In frequency modulated radar, the energy is transmitted as radio frequency waves that continuously vary, increasing and decreasing from a fixed reference frequency. It is used in aircraft altimeters that gives a continuous reading of the altitude above the earth. Depending on the pulsed radar, the energy transmitted might vary from less than 1 microsecond to 200 microseconds. The time interval between transmission and reception is computed and converted into a visual indication of range in distance. This kind of radars is widely used in the navy [1].

Carrier Frequency

The carrier frequency is the frequency at which the radio-frequency energy is generated. The main factors influencing the selection of the carrier frequency are the desired directivity and the generation and reception of the necessary microwave energy. The higher the carrier frequency, the shorter the wavelength and hence the smaller antenna required. The problem of generating and amplifying reasonable amounts of radio-frequency energy at extremely high frequencies is complicated by the physical elements.

During the World War II, letter codes such as *S*, *X*, and *L*, were used to designate the distinct frequency bands used. The original purpose was to maintain military secrecy; but the letter designations were continued after the war as a convenient means to readily denote the region of the spectrum at which the radar operates. For example, letters *S*, *X*, and *L* denote the frequency ranges: 2-4 GHz, 8-12 GHz and 1-2 GHz respectively [10].

Range to a Target

The range, or distance, to a target is found by measuring the time the radar signal takes to travel to the target and turn back to the radar, as shown in figure 2.1. Although target location in angle, the target's velocity and the target nature can also be found by the radar signal, the measurement of range is still one of its most important functions. There are no competitive techniques or systems that can accurately measure long ranges in both clear and adverse weather as well as can radar systems [1].

The range to a target can be determined by the time T_R it takes the radar signal to travel to the target and goes back. Electromagnetic energy in free space travels with the speed of light, c = 3 x 10^8 *m/s*. Thus the time for the signal to travel to a target located at a range *R* and return back to the radar is 2*R/c*. The range to a target is then:

$$R = \frac{cT_R}{2} \tag{2.1}$$

Maximum Unambiguous Range

Once the signal is radiated into space by a radar, sufficient time must elapse to allow all echo signals to return to the radar before the next pulse is transmitted. The rate at which pulses must be transmitted, is determined by the longest range at which targets are expected. If the time between pulses T_p is too short, an echo signal from a long-range target might arrive after the transmission of the next pulse and be mistakenly associated with that pulse rather than the current pulse transmitted earlier. This can result in an incorrect and ambiguous measurement of the range. Echoes that arrive after the transmission of the second pulse are called second-time-around echoes (or multiple-time-around echoes). Such an echo would appear to be at a closer range than the actual one, and its range measurement could be misleading if it were not known to be a second-time-around echo [1]. The range beyond which targets appear as second-time-around echo is the maximum range, R_{un}, and is given by:

$$R_{un} = \frac{cT_p}{2} = \frac{c}{2f_p} \tag{2.2}$$

where T_p is the pulse repetition period = 1/f_p and f_p is pulse repetition frequency (PRF), usually given in Hertz or pulses per second.

Pulse Repetition Frequency

The rotation of a radar antenna through 360^o lasts some seconds. During this rotation time several hundred pulses are emitted. The pulse repetition frequency is defined as the number of pulses transmitted per second. It is necessary that the transmitted pulse must be separated by long non-transmitting time periods, allowing the pulse to travel to the target and return as a reflected echo, observing the maximum unambiguous range. Otherwise, transmission would occur during reception of the reflected echo of the preceding pulse, meaning multiple-time-around-echoes. With the antenna being rotated, the beam of energy strikes a target for a relatively short time. During this time, a sufficient number of pulses must be transmitted in order to receive sufficient echoes to produce the necessary indication on the radarscope. With the antenna rotating at 15 revolutions per minute (RPM), a radar set having PRF of 800 Hz will produce approximately 9 pulses for each degree of the antenna rotation [1, 10].

Pulse Length

Pulse length is defined as the duration of the transmitted radar pulse and it is usually measured in microseconds and denoted by τ. The minimum range at which a target can be detected is determined largely by the pulse length. If the target is so close to the transmitter that the echo is returned to the receiver before the transmission stops, the reception of the echo will be masked by the transmitted pulse. In this case, a short pulse is attractive since strong transmitter signal is not radiating when the weak pulse echo signal is being received [1]. Many radar sets are designed for operation with both short and long pulse length. Many of these radar sets are shifted automatically to the shorter pulse length on selecting the shorter range scales. On other radar sets, an operator can select the radar pulse length in accordance with the operating conditions. Radar sets have greater range capabilities while functioning with the longer pulse length because a greater amount of energy is transmitted in each pulse [10]. Figure 2.2 shows a pulse waveform example of a pulsed radar that exemplifies the PRF, pulse length, the radio-frequency energy generated to form the pulse and a target echo.

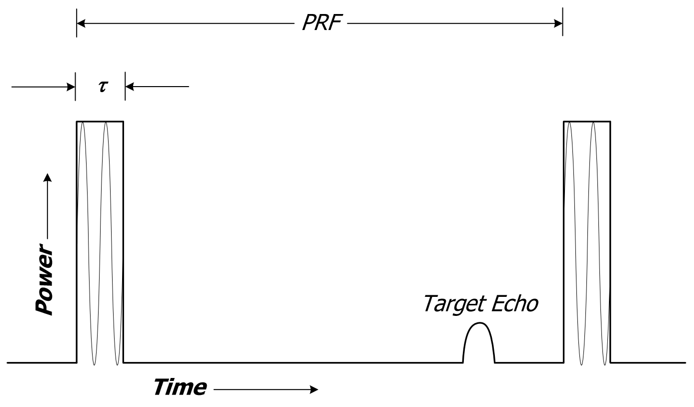


Figure 2.2: Radar pulse waveform example.

Range Resolution

Range resolution is a measure of capability of a radar set to detect the separation between those targets on the same bearing but having small difference in range. This range resolution R_0 is given by:

$$R_0 = \frac{c\tau}{2} \tag{2.3}$$

If the leading edge of a pulse strikes the target A at a slightly greater range while the trailing part of the pulse is still striking a closer target B, the reflected echoes of both targets will appear as a single elongated image on the radarscope.

There is a relationship among the pulse length, range resolution and detection range. The maximum detection range is sacrificed by using a shorter pulse length and a better range accuracy and range resolution are obtained. A long pulse length results in a poor range accuracy and range resolution, but the detection range is increased [1].

Radar Envelope

The amplitude of the reflected signal echoes is obtained from the target or other objects as function of the time. This amplitude is proportional to the range to the target. The mentioned function is known as the radar echoes envelope. This envelope is generated after a pulse emitted by the radar is sampled. Each sample represents a discrete range to a target and it is called range cell while the set of these samples is called a range silhouette. The minimum amount of range cells depends on the radar range resolution desired [11]. Figure 2.3 exemplifies a range silhouette.

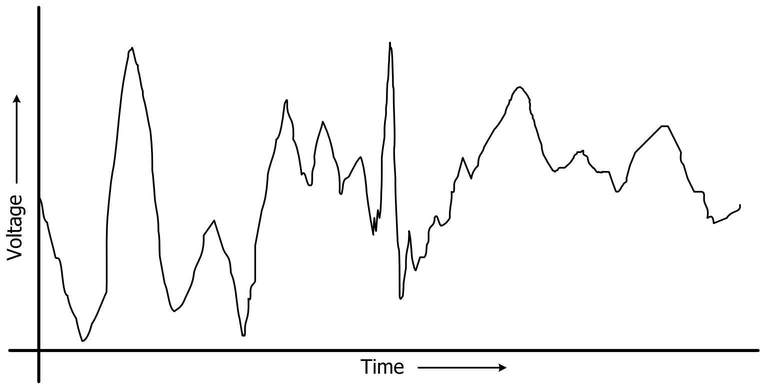


Figure 2.3: Range Silhouette composed by the radar echoes envelopes

2.1.1 The Simple Form of the Radar Equation

The radar equation relates the range to the characteristics of the transmitter, receiver, antenna, target, and the environment. It is useful not only for determining the maximum range at which a particular radar can detect a target, but it can serve as a way for understanding the factors affecting radar performance. The simple form of the radar equation expresses the maximum radar range R_{max}, it is given by:

$$R_{max} = \left[\frac{P_t G A_e \sigma}{(4\pi)^2 S_{min}} \right]^{\frac{1}{4}} \tag{2.4}$$

where :

P_t = Transmitted power, W

G = Antenna gain

A_e = Antenna effective aperture, m^2

σ = Radar cross section of the target, m^2

S_{min} = Minimum detectable signal, W

The radar cross section is the property of a scattering object, or target, that measures how detectable an object is with a radar. It represents the magnitude of the echo signal returned to the radar by a target. Except for the target's radar cross section, the other parameters of the simple form of the radar equation are under the control of the radar designer. This equation states that if long ranges are desired, the transmitted power should be large, the radiated energy should be concentrated into a narrow beam (large transmitting gain), the echo energy should be received by a large antenna aperture (also synonymous with large gain), and the receiver should be sensitive to weak signals [1].

Since there are several components that perform different actions in the signal processing radar chain, it is necessary to focus only in the target detection process link. In order to do this, a general block diagram for radar systems is explained.

2.1.2 General Radar Block Diagram

The operation of a pulsed radar may be described with the aid of the block diagram in figure 2.4, which exemplifies a set of radar components. The radar carrier signal is produced at low power by the *waveform generator*, which is the input to the power *amplifier*. A *pulse modulator* turns the transmitter on and off in synchronism with the input pulse in order to generate the pulse waveform. The amplifier amplifies the original signal produced by the waveform generator and this amplified signal is delivered by the *antenna* in order to be radiated into space. The *duplexer* allows a single antenna to be used on a time-share basis for both transmitting and receiving. The input echo is received by the *low-noise amplifier*. The *mixer* and the *local oscillator* convert the radio frequency (RF) signal of the echo to an intermediate frequency (IF) where it is amplified by the *IF amplifier*. The IF amplifier is followed by the *second detector*, whose purpose is to assist in extracting the signal modulation from the carrier. The combination of the IF amplifier, second detector and *video amplifier* act as an *envelope detector* to pass the pulse modulation (envelope) and reject the carrier frequency. The combination of IF amplifier and video amplifier is designed to provide sufficient amplification, or gain, to raise the level of the input signal to a magnitude where it can be seen on a display, such as cathode-ray tube (CRT), or to be input to a digital computer for further processing. At the output of the video amplifier, a decision is made whether or not a target is present. The decision is based on the magnitude of the output of the video amplifier. This process decision for target detection is made by the *CFAR detector*. The output of the CFAR detector is displayed in a *PPI*, or plan position indicator. The PPI is a presentation that maps in polar coordinates the location of the target in azimuth and range [1].

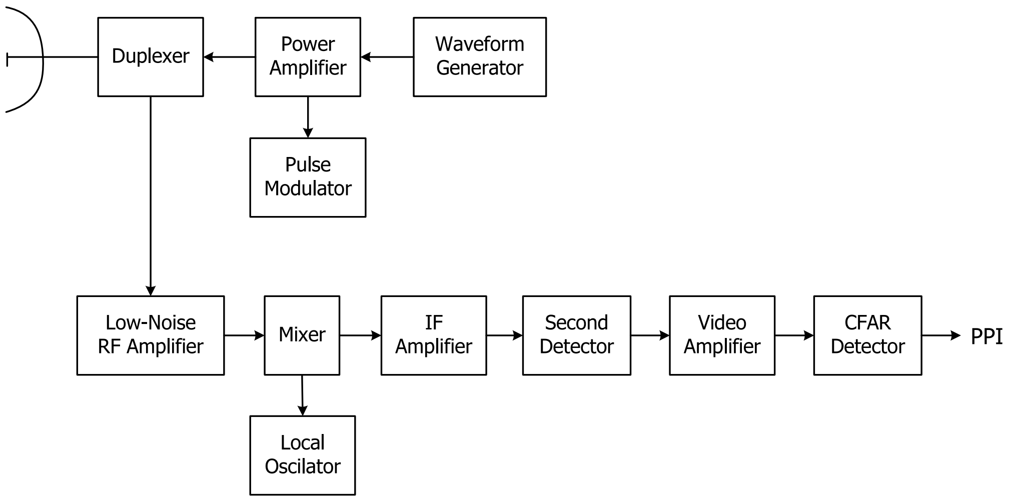


Figure 2.4: Radar block diagram

Although it is not shown in figure 2.4, the pulse integration is one important part in radar operation. The pulse integration is the process of adding the echo pulses from the target in order to obtain a greater signal-noise-ratio (SNR) before the detection decision is made. Many techniques have been considered in the past to provide integration of pulses. Integration that is performed in the radar receiver before the second detector is called predetection integration or coherent integration. On the other side, integration after the second detector is known as postdetection integration of noncoherent integration.

Once the link where the target detection is performed inside the processing radar chain (CFAR detector block) has been located, we can focus on the target detection problem and the CFAR detector.

2.2 Target Signals Detection

In practice, the simple radar equation form does not adequately predict the range performance of modern radars. It is not unusual to find that when equation 2.4 is used, the actual range might be only one half that predicted. The failure of the simple form of the radar equation is due to:

1. The statistical nature of the minimum detectable signal (usually determined by receiver noise).

2. Fluctuations and uncertainties in the target radar cross section.

3. The losses experienced throughout a radar system.

4. Propagation effects caused by the earth's surface and atmosphere.

The statistical nature of the receiver noise and the target cross section requires that the maximum radar range be described probabilistically rather than by a single number. Thus the specification of range must include the probability that the radar will detect a specified target at a particular range, and with a specified probability of making a false detection when no target echo is present. The range of a radar, therefore, will be a function of the *probability of detection*, P_d, and the *probability of false alarm*, P_{fa} [11].

The prediction of the radar range cannot be performed with arbitrarily high accuracy because of the uncertainties in many of the parameters that determine the range. Even if the factor affecting the range could be predicted, the statistical nature of the radar detection and the variability of the target's radar cross section and other effects make it difficult to accurately verify the predicted range. The Neyman-Pearson criterion applied to the radar envelope with a fixed or adaptive threshold (CFAR detector) are used for performing the detection of targets in a noisy background [1].

2.2.1 Neyman-Pearson Criterion

The Neyman-Pearson criterion is used in binary detection problems, also known as binary hypothesis test, like the radar detection. On this case, there are only two cases or states: target present and target absent:

- H_0 Null hypothesis that refers to only noise echo.
- H_1 Alternative hypothesis that refers to noise echo with target echo.

The null and the alternative hypothesis must be tested against the two possible states: δ_0 and δ_1 that represent the target absent and target presence respectively. The combinations of the two hypothesis and the two possible states make four possible situations:

1. H_0 is true having selected δ_0. This combination indicates that there are only noisy echoes and it has been selected target absent. This means a **successful no detection**.

2. H_1 is true having selected δ_1. This combination indicates that there are noisy echoes plus a target and it has been selected target presence. This means a **successful detection**.

3. H_0 is true having selected δ_1. This combination indicates that there are only noisy echoes and it has been selected target presence. This means a **false alarm**.

4. H_1 is true having selected δ_0. This combination indicates that there are noisy echoes plus a target and it has been selected target absent. This means a **missed detection**.

The false alarm and missed detection are known as type I error and type II error respectively. In a radar system, it is desirable to avoid type I errors or to keep false alarms in an acceptable level while maximizing correct detection.

The set of samples from a range silhouette can be grouped in two sample sets: the data set representing noise echo (H_0) and the data set representing the noise echo with target echo (*H_1*). From these two data sets, it can be obtained their pdf (probability density function): *f_0(x)* and *f_1(x)*. The pdf *f_0(x)* represents the probability of only noise echo presence and the pdf *f_1(x)* the probability of noise echo plus target echo (*P_d*). Graphically (figure 2.5), these two pdf have an intersection area that represents the false alarm probability (*P_{fa}*). A detection threshold level helps to delimit the false alarm area. In general, the threshold is set a function of the interference level to control the number of false alarms [11].

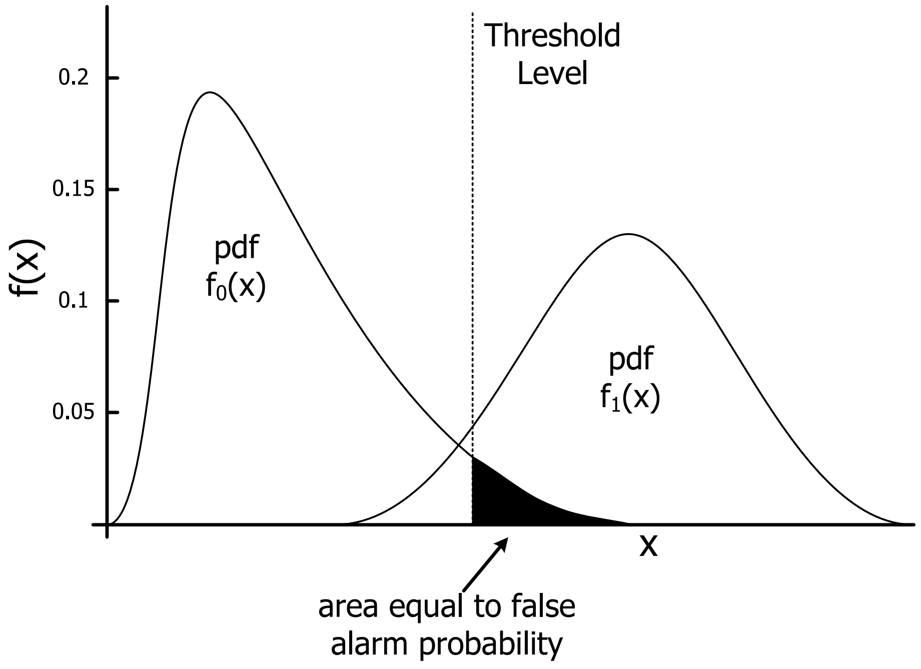


Figure 2.5: Pdf representing the Neyman-Person criterion.

2.2.2 Radar Signals Detection in Noise Presence

The radar detection of echo signal from targets is performed by the Neyman-Pearson criterion and establishes a threshold at the output of the receiver. If the receiver output is greater than the established threshold, it is declared a *target presence* (state δ_1); otherwise it is declared a *target absence* (state δ_0). The noise presence can be caused by weather conditions (clutter), thermal noise from radar devices, pulse jamming or interference. If the fixed threshold level is set properly, the receiver output would not exceed the threshold if only noise is present, but the receiver output would exceed the threshold if along with noise, a target is present. If the threshold level were fixed too low, noise alone might exceed it, situation called false alarm. If the threshold is fixed too high, only strong target echoes would be able to exceed it and weak target echoes might be not detected, situation called missed detection. In early radars, the threshold level was set based on radar operator judgment.

In figure 2.6 there is a range silhouette example of the output of a radar receiver where the fixed threshold line is represented by the horizontal dashed line and the adaptive threshold is represented by the dashed-dotted line. For this example, suppose that the value of the signal at points A and B are targets plus noise and C and D are just noise. The signal point A is detected correctly by the fixed threshold, while B is not strong enough to be detected and therefore it is missed. The noise from signal point B is weak because of the negative noise that was added to the original signal strength. Signal points C and D are false alarms, and they are increased because of the presence of positive noise. Signal point B could be detected if the fixed threshold was lower, but this might increase the false alarms. The selection of a proper fixed threshold is a compromise that depends upon how important it is to avoid the mistake of failing to declare a target presence (missed detection) or falsely indicating the presence of a target when none exists (false alarm). On the other hand, with an adaptive threshold, the signal points A and B, which exceed the threshold level, can be detected correctly, while C and D are assumed to be noise because they do not exceed the threshold [1].

In order to maintain the false alarm rate at a constant value, the threshold has to be varied adaptively. This is achieved when the CFAR detector automatically adapts (raise or reduce) the threshold level, thus avoiding overload of the automatic detection system. A constant false alarm rate is achieved at the expense of a lower probability detection of desired targets. Also CFAR produces false echoes when clutter there is nonuniform, suppresses nearby targets and decreases the range resolution. The detection probability and false alarm probability are specified by the system requirements.

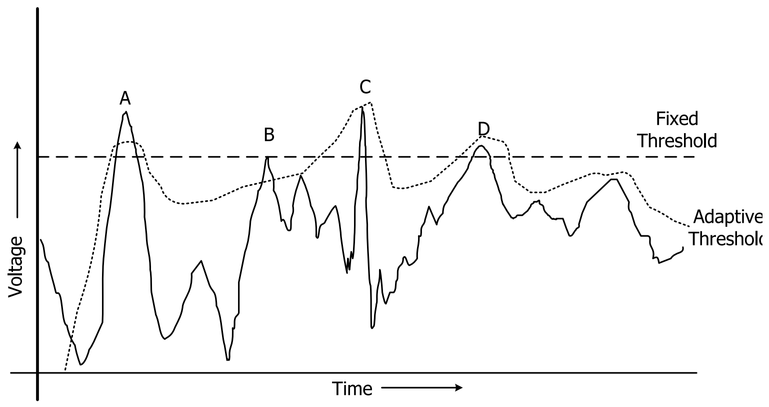


Figure 2.6: Example of a radar receiver range silhouette with fixed and adaptive threshold.

2.3 CFAR Detector

The need for a CFAR detector was recognized when the early automatic detection and tracking systems were installed as add-ons to existing radar with not moving target indicators (MTI) and poor MTI did not have a good clutter rejection. CFAR is needed for maintaining operation for automatic detection and tracking systems. If CFAR were not used in radar, it would cause excessive false alarm due to noise or clutter, decreasing the automatic detection and tracking system performance.

Several algorithms have been proposed for the CFAR detection module. The most commonly used CFAR algorithms are cell averaging (CA-CFAR) [12], greatest of (GO-CFAR) [13], smallest of (SO-CFAR) [14], generalized order statistics cell averaging (GOSCA-CFAR), generalized order statistics greatest of (GOSGO-CFAR) and generalized order statistics smallest of (GOSSO-CFAR) [5]. Figure 2.7 shows a general block diagram of a CFAR detector.

The CFAR detector consists of two sliding windows that surround the cell under test. Both, lagging and leading windows have $N = 2n$ reference cells, $M = 2m$ guard cells and a cell under test. Each cell stores an input sample and such values are right shifted when a new sample arrives. The CFAR detector is applied over the sliding window

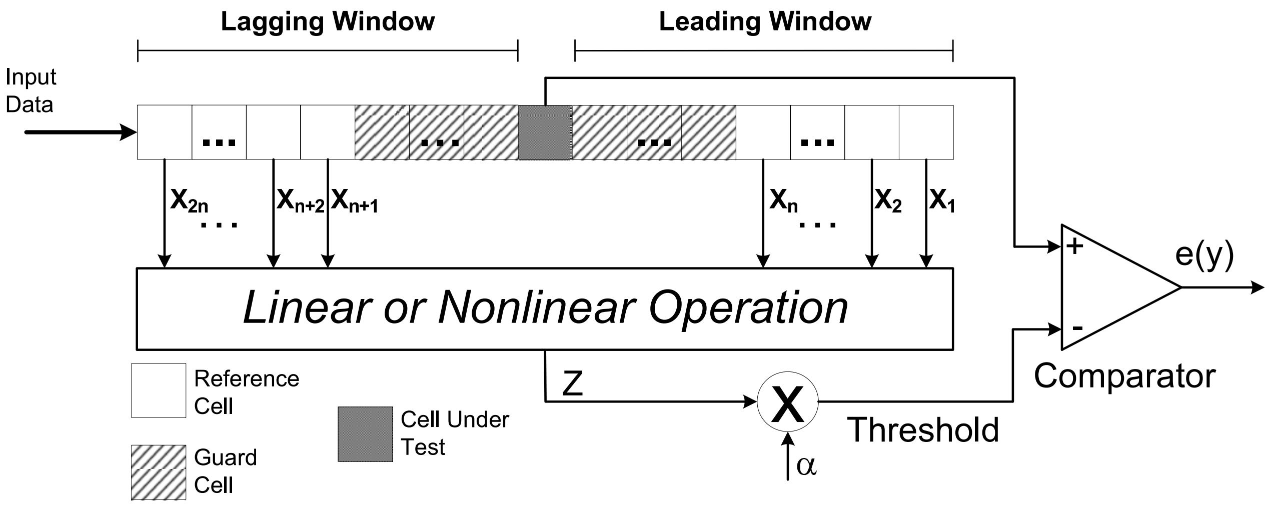


Figure 2.7: Generic CFAR Detector.

of *P* = *N*+*M*+*1* cells, which represent the sampled data of the discrete ranges, *i.e.* the range cells. The spacing between the cells is equal to the radar range resolution (usually the pulse width). The reference cells are used to compute the *Z* statistic meanwhile the guard cells are incorporated in order to avoid interference problems in the noise estimation. Depending on the technique, the *Z* statistic computing can be linear or nonlinear operation. A scaling factor α and the *Z* statistic are used to obtain the threshold. The scaling factor depends on the estimation method applied and the false alarm required according to the application. It is also related to the noise distribution in the radar environment. The resulting product α*Z* is directly used as the threshold value that is compared with the cell under test (CUT), to determine if the CUT is declared a target. The Neyman-Pearson hypothesis can be modeled by the following equation:

$$\begin{aligned} H_1 &: y = d + g \\ H_0 &: y = g \end{aligned} \tag{2.5}$$

where H_1 and H_0 are the target present and target absent of the Neyman-Pearson hypothesis; *d* represents the target signal and *g* the environmental noise component. The target decision is made according to the criterion represented by:

$$e(y) = \begin{cases} H_1, & CUT \geq \alpha Z \\ H_0, & CUT < \alpha Z \end{cases} \tag{2.6}$$

If the values of the CUT exceed the αZ statistic, then the target present is declared, *i.e.* the CFAR detector outputs 1 if a target is present, otherwise outputs 0:

$$e(y) = \begin{cases} 1, & CUT \geq \alpha Z \\ 0, & CUT < \alpha Z \end{cases} \tag{2.7}$$

The method to obtain the Z statistic from the reference window might be based on linear or nonlinear operations. When the method is based on linear operations it can be called a CFAR linear detector, and when it is based on nonlinear operations it can be called a CFAR nonlinear detector [15]. These different detectors allow to modify the detection performance according to different environmental situations present in radars.

2.3.1 Linear Detector

The most common linear detectors are the CA-CFAR, GO-CFAR and SO-CFAR. These detectors calculate the arithmetic mean of the amplitude contained in the Y_1 lagging cells and Y_2 leading cells from the CUT. The CA detector estimates the arithmetic mean, the GO and SO take the major and minor values of Y_1 and Y_2, respectively. Equations 2.8, 2.9, 2.10 summarize these three linear operations for the Z statistic:

$$Z = \frac{1}{2}(Y_1 + Y_2) \tag{2.8}$$

$$Z = \max(Y_1, Y_2) \tag{2.9}$$

$$Z = \min(Y_1, Y_2) \tag{2.10}$$

These three linear operations are represented in the figure 2.8, which shown how the Y_1 and Y_2 obtain the arithmetic mean from the lagging and leading cells.

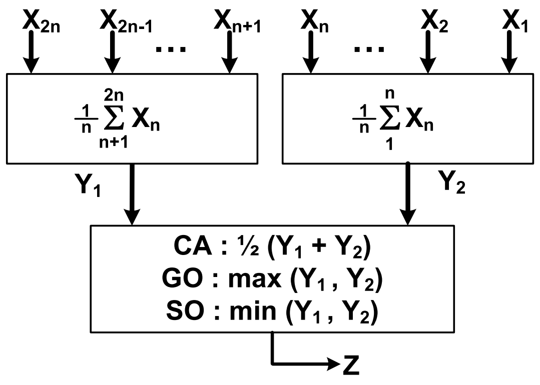


Figure 2.8: Linear CFAR detectors.

2.3.2 Nonlinear Detector

Several nonlinear detectors are based on order statistics, which consist in arranging in ascending order the random variables X_1, X_2, X_3, ..., X_{n-1}, X_n. Therefore the sequence achieved is:

$$X_{(1)} \leq X_{(2)} \leq X_{(3)} \leq .... \leq X_{(n-1)} \leq X_{(n)} \tag{2.11}$$

The indexes between parentheses indicate the rank-order number for the $X_{(i)}$. The idea on the rank-order filters is to select a value $X_{(k)}$, where k ϵ {1, 2, 3, ... ,*n-1*, *n*} from the sequence in equation 2.11 and then to use this $X_{(k)}$ value as a sample of the sorted data.

Among the nonlinear detectors that use the rank-order operation are: OSCA-CFAR, OSGO-CFAR and OSSO-CFAR [4]; and their generalized form called GOSCA-CFAR, GOSGO-CFAR and GOSSO-CFAR detectors [5]. These order statistic detectors need to perform a rank-order operation over the leading and lagging reference cells, *i.e.* sort the reference cells values and then select the *k-th* sorted value. The rank-order parameter *k* can be deliberately selected among the sorted values. The GOSCA-

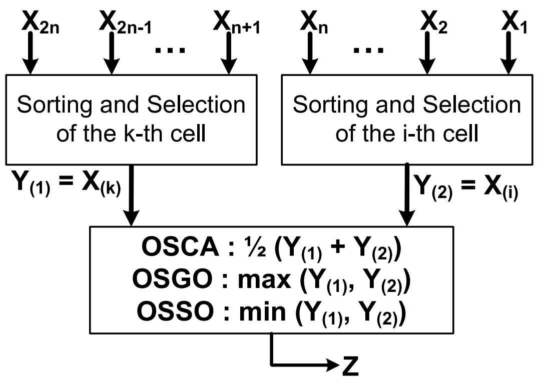
CFAR, GOSGO-CFAR and GOSSO-CFAR detectors, perform the selection of the *k-th* ($Y_{(1)}$) and *i-th* ($Y_{(2)}$) sorted value from the leading and lagging cells, respectively. Once these two values have been selected, the Z statistic is calculated in a similar way as the linear detectors, as shown in the three following equations:

$$Z = \frac{1}{2}(Y_{(1)} + Y_{(2)}) \tag{2.12}$$

$$Z = \max(Y_{(1)}, Y_{(2)}) \tag{2.13}$$

$$Z = \min(Y_{(1)}, Y_{(2)}) \tag{2.14}$$

These nonlinear operations are represented in the figure 2.10, which shows how the $Y_{(1)}$ and $Y_{(2)}$ obtain the *k-th* and the *i-th* value from the lagging and leading cells and then performs the linear operation.



X_{2n}
X_{2n-1}
X_{n+1}
X_n
X_2
X_1
Sorting and Selection of the k-th cell
Sorting and Selection of the i-th cell
$Y_{(1)} = X_{(k)}$
$Y_{(2)} = X_{(i)}$
OSCA : ½ ($Y_{(1)}$ + $Y_{(2)}$)
OSGO : max ($Y_{(1)}$, $Y_{(2)}$)
OSSO : min ($Y_{(1)}$, $Y_{(2)}$)
Z

Figure 2.9: Nonlinear CFAR detectors.

The difference between the generalized detectors and the OSCA-CFAR, OSGO-CFAR and OSSO-CFAR detectors is that laters only perform the selection of the *k-th* sorted value from both the leading and lagging cells. Therefore, OSCA-CFAR, OSGO-CFAR and OSSO-CFAR can be considered a special case of their generalized counterpart when $k = i$.

Other nonlinear detectors found in the radar literature are the order statistic (OS-CFAR) [3] and trimmed mean (TM-CFAR) [2] detectors. The OS-CFAR detector performs the rank-order operation over all *2n* reference cells unlike the generalized detectors that perform a rank-order operation over the lagging cells and other rank-order operation over the leading cells. The *Z* statistic is only the *k-th* sorted value (figure 2.10).

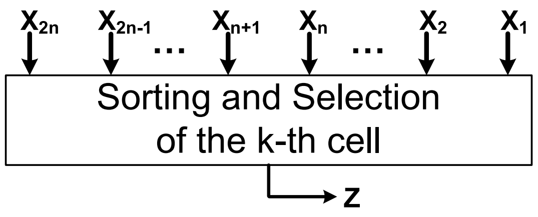


Figure 2.10: Order statistic CFAR detector.

The TM-CFAR detector performs more complex operations than the others detectors. First, it sorts the *2n* reference cells, then with the sorted data it discards the T_1 greatest and the T_2 smaller cells and the rest of the sorted data within this no discarded range is added. The statistic *Z* is formed by averaging the $2n$-T_1-T_2 remaining reference cells. Equation 2.15 shows how the *Z* statistic is calculated once the reference cells have been sorted:

$$Z = \frac{1}{N - T_1 - T_2} \sum_{n=T_1+1}^{N-T_2} X_{(n)} \tag{2.15}$$

where *N* = *2n* reference cells used in the sorting operation. Figure 2.11 shows the T_1 greatest and the T_2 smallest reference cells trimmed after the sorting operation.

2.3.3 CFAR Detectors Considerations

There are several problems that must be considered when a CFAR detector is developed. These problems that have been issue for investigation are: CFAR loss, clutter edges, multiple targets situations and radar range resolution.

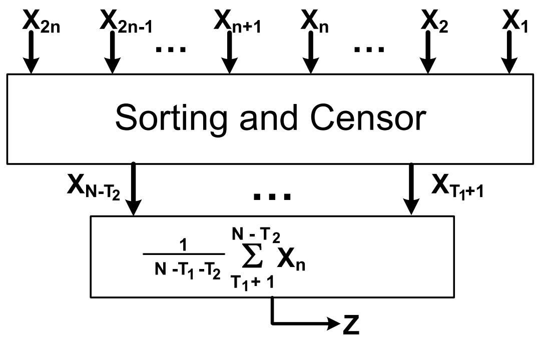


Figure 2.11: Trimmed mean CFAR detector.

CFAR Loss

The greater the number of reference cells used in CFAR better is the estimate of the background clutter or noise and the less is the loss in detectability (SNR), *i.e.* the greater of reference cells used the detection probability approaches to the optimum detector based on fixed threshold. However, there is a limit to the number or reference cells that can be used in practice since the clutter must be relatively homogeneous over the reference cells. Since there are only a finite number of reference cells, the estimate of the noise or clutter is not precise and there will be a loss in detectability. The CFAR loss is defined as the additional SNR needed to obtain the same detection performance as the associated with the fixed threshold detector where the interference level is known *i.e.* the SNR required when the CFAR is employed divided by the SNR required for the fixed threshold detection. This loss is decreased when the reference window size is increased and the loss is increased when the P_{fa} is decreased [1].

Clutter Edges

As the reference cells pass over the leading and lagging cells of a patch of clutter (clutter transition), not all the reference cells contain clutter; so the threshold will be lower than when all reference cells contain clutter. False alarms, therefore, can result at clutter edges. Threshold crossing from the clutter edges can be reduced by summing the lagging and leading cells separately and using the greater of the two to determine the threshold, *i.e.*, use a GO-CFAR detector [2].

Multiple Target Situations

When there are one or more targets within the reference cells along with the primary target in the CUT, the threshold is raised even in the absence of any clutter, and the detection of the primary target in the CUT might be suppressed. Order statistics detectors handle the multiple nearby target situations. Similar situations are presented when targets mask the presence of other targets (mutual target masking) or when extended targets occupy several resolution cells (self target masking), masking themselves by biasing the threshold level. These situations may be suppressed with the implementation of guard cells [2].

Range Resolution

Generally, two equal amplitude targets can be resolved if they are separated in range each 0.8 radar pulses. However, usual CFAR detectors considerably degrade the range resolution so that two equal targets can be resolved only if they are spaced greater than 2.5 pulse width. One reason for the poor resolution is that the range cells adjacent to the test cell are not used as part of the reference cells since the target energy in the test cell spills over to nearby cells and affects the threshold [1].

2.4 Summary

This chapter has covered the main theoretical aspects of radar and target detection. The radar signal processing chain demands a high computational performance in order to achieve its real-time constraints. The radar equation does not predict the range performance accurately due to the statistical nature of the radar echoes received and the uncertainties in many of the radar equation parameters. With the help of CFAR detectors the detection can be performed.

Although, a CFAR detector that can be considered optimal under any environmental circumstances has not been designed yet, each one of the presented detectors has advantages and disadvantages, and may be optimal under particular environmental conditions. Furthermore the detection performance is altered by varying the number of references cells, guard cells, the CFAR detector, the *k-th* rank-order sample, the trimming values T_1 and T_2 and the false alarm required (represented by the scaling factor α) [2]. In order to give robustness to the target detection process in radar applications, a scheme which supports some of these detectors, and allows changing the parameters that alter the detection performance is helpful for adapting the CFAR detector according to the environmental conditions.

In the next chapter, a review of the related work is presented, including the theoretical aspects, some implementations of the CFAR detector and, sorting schemes proposed for performing the rank-order operation in nonlinear detectors.

Chapter 3

Related Work

The radar target detection literature has been mainly focused on theoretical aspects. Several works have been focused on improvements of performance on target detection, development of different statistical noise models and CFAR detectors modifications for different environmental situations. In recent years digital hardware implementations for CFAR detectors have been suitable for being implemented, in spite of intensive computational and the high data rate requirements in radar signal processing. These digital implementations have been focused in systolic and specialized architectures since they meet the intensive radar requirements. This chapter presets the more relevant related works in both CFAR theoretical and CFAR implementations available in the literature. Also it covers related works about hardware sorters since sorting is needed in the CFAR nonlinear detectors implemented, in order to perform the rank-order operation.

3.1 CFAR Detectors

In the radar literature, the CFAR detectors can be found like first order detectors, refering to linear detectors; and order statistics detectors, refereing to the nonlinear detectors. These detectors have been developed in order to deal with different environmental conditions and target situations (multiple target situations and clutter transitions), trying to achieve a better detection performance while maintaining the false alarms in a low level.

3.1.1 First Order Statistic

The conventional CFAR algorithm is the CA-CFAR detector proposed by Finn and Johnson in [12] being the first detector proposed in the radar literature. CA-CFAR detector is the optimum CFAR detector (maximizes detection probability) in a homogeneous background when reference cells contain independent and identically distributed observations governed by an exponential distribution. However, the CA-CFAR detection performance degrades in multiple target situations and regions of power transitions. Both situations result in an excessive number of false alarms *i.e.* an inferior behavior in nonhomogeneous situations [2].

Hansen in [13] proposed the GO-CFAR detector in order to maintain a constant false alarm rate at clutter edge (regions of power transitions). Some studies have shown that during clutter power transitions a minor increase of the CFAR loss can be expected in the false alarm rate in the worst case when the lagging cells contain radar echoes from clear background (without target) the leading cells contain echoes from high clutter regions. This is simple because the detector includes only the clutter samples present in the reference cells to estimate the noise power in the worst case [2]. However its detection performance in multiple target situations is quite poor, suffering mutual target masking and being incapable of resolving closely spaced targets.

The SO-CFAR detector was introduced by Trunk in [14] trying to prevent the suppression of closely spaced targets (target masking). This detector resolves the primary target in multiple target situations when all the interferers are located in either the leading or lagging cells. However, SO-CFAR has undesired effects when interfering targets are located in both halves of the reference cells. Also, its detection performance degrades if interfering targets are located in both leading and lagging reference cells [2].

An analysis of some CFAR detectors in homogeneous and nonhomogeneous background is performed by Gandhi and Kassam in [2]. Their work has been taken as one of the reference works in the CFAR literature due it analyzes the detection performance in multiple target, clutter edges and nonhomogenous situations for five different detectors: CA-CFAR, GO-CFAR, SO-CFAR, OS-CFAR and TM-CFAR. Referring to the linear detectors, Gandhi and Kassam conclude that the linear detectors exhibit serious performance degradation in nonhomogeneous noise background. According to [2], the false alarm rate increases considerably at clutter transition regions, and target masking is experienced in multiple target situations when the CA-CFAR detector is used. On the other hand, the performance of GO-CFAR detector is better in regions of clutter transitions than the other linear detector analyzed, although its performance in multiple target situations is poor. Finally, the unique advantage of SO-CFAR detector is when a cluster of radar targets appears in the reference cells, *i.e.* a multiple target situation.

3.1.2 Order Statistic

Rohling in [3], proposed the use of order statistics and rank-order operators to improve the CFAR detection performance in situations where the linear detectors fail. In the same work, Rohling proposes the OS-CFAR detector which has better performance specially in cases where more than one target is present within the reference cells or where there are clutter edges. Also his result shows that window sizes of about *N = 24...32* and more are applicable, the *k-th* value should be greater than *N/2*, and the difference *N-k* should not be less than the double of the target length in the reference cells in order to avoid two targets from being mutually masked.

Elias-Fusté *et al.* [4] proposed two modified OS-CFAR detectors that require less processing time than the OS-CFAR detector and combine the GO-CFAR and SO-CFAR: OSGO-CFAR and OSSO-CFAR detectors. The OSGO-CFAR has all the advantages that OS-CFAR presents, requiring only a half of the OS-CFAR processing time. A generalization of these detectors are presented by You He in [16]: GOSCA-CFAR, GOSGO-CFAR and GOSSO-CFAR detectors. The generalized detectors are more robust than the detectors proposed in [3] and [4] because of the selection of the *k-th* and *i-th* rank-order sample and, because they require a half of the time for performing the sorting action compared with OS-CFAR. GOSCA-CFAR detector posses the best detection performance in both homogeneous background and multiple target situations. The GOSGO-CFAR has the same performance as the OS-CFAR and provides good transition clutter protection[5]. If the number of interfering targets in the reference cells is equal to the greatest interfering targets allowed, then the performance of these generalized processors are better than the OS-CFAR detector.

As mentioned in the previous section, Gandhi and Kassam [2] perform a detection performance analysis of two nonlinear CFAR detectors in multiple target, clutter edges and nonhomogeneous situations. According to their results, OS-CFAR detector performance is relatively unaffected if the clutter area is less than the length of all reference cells as long as the number of clutter samples present in this reference cells set is greater than *N-k*. The OS-CFAR detector can handle until five targets if the *k-th* sample is chosen correctly. The TM-CFAR detector has a slightly better performance in homogeneous background for isolated targets compared with the OS-CFAR detector and, in nonhomogeneous background only few higher ordered range cells need to be averaged in order to get a good detection performance. Also, they propose the use of adaptive versions of OS-CFAR and TM-CFAR called adaptive order statistic AOS-CFAR and variability trimmed mean VTM-CFAR. This idea is widely developed in [17], [18], and [19]. These works explore the use of a data-dependent rule for varying the number of samples that are required for the processing when the VTM-CFAR detector is used. The AOS-CFAR consists on choosing the *k-th* sorted value according to the outcome of a hypothesis test, which attempts to detect presence of a clutter region within each set of reference cells. Others works have proposed other CFAR detectors using different techniques:

- Excision (Ex-CFAR) detectors discard the reference cells whose value exceeds certain threshold.
- Censoring detectors (like TM-CFAR) once sorted the reference cells, they discard the greatest (or the smallest) sorted values and then perform an operation, like CMLD (Censored Mean Level Detector) and MX-CMLD (Max Censored Mean Level Detector).

- Hybrid approaches that combine one or more traditional CFAR detectors in one by fusioning their output, or by selecting the most appropriate output, or even with image processing techniques.

- Other techniques used include Bayesian, Biparametric Gaussian, Hofele and Weibull statistics, morphological operations, weighted averaging, logarithmic detectors, clutter maps, *LI* filters, among others.

3.2 CFAR Hardware Architectures

In the FPGA and radar literature, there are few FPGA architecture implementations, either based on systolic or specialized architectures, reported for CFAR detectors.

3.2.1 Systolic Architectures

One of the first systolic architectures for CFAR detectors was presented by Hwang and Ritcey in [20]. In this work, they proposed a systolic array capable of supporting two order statistics detectors: CMLD and MXCMLD CFAR. The first detector censors the largest reference samples sorted and then adds the remaining samples, meanwhile the second one gets the maximum of the sum of the remaining samples from the lagging and leading cells, after being censored. This systolic architecture is very versatile, and can accommodate a wide class of CFAR detector due to its nature. A modified OS-CFAR systolic architecture is presented in [21] by Han. This systolic architecture must be continuously fed in order to support the real-time processing demand. This architecture has less PEs, less interconnections, less gates and two times higher throughput than the architecture presented in [20]. Han also proposes a systolic architecture for the OSGO-CFAR and OSSO-CFAR detectors by slightly modifying the OS-CFAR systolic architecture. The throughput rate of this architecture is the same as the original proposed OS-CFAR, using less hardware resources.

Behar *et al.* in [22] presented a parallel systolic architecture for a CFAR detector with adaptive post-detection integration (API). This detector sorts and censors the reference cells in a similar way as the TM-CFAR detector does. The CFAR detector for this architecture was developed, analyzed and synthesized in a systolic architecture in order to detect targets in presence of pulse jamming. This proposed architecture has a linear structure, specially designed for real-time implementation of this API CFAR and it uses four sequential blocks of processing for: sorting, censoring, integration and comparison. These four blocks are constructed by five processing elements, which perform different logical operations. Other similar works are presented in [23] and [24] using as basis the work presented in [22] replacing one or two sequential blocks used in the signal processing. These architectures were targeted for different jamming levels and background models and they are using the odd-even transportation as a sorting scheme.

3.2.2 Specialized Architectures

Among the specialized architectures, Torres *et al.* [9] presented an architecture for CA-CFAR, GO-CFAR and SO-CFAR detectors. This architecture implements the average computations with two accumulating processing elements (APE) and a configurable threshold processing element (CTPE). These APEs compute the sum of their corresponding reference cells (lagging and leading) and the CTPE computes the threshold operation (cell-averaging, greatest of and smallest of). This architecture uses 12 bits for data, 32 reference cells, 8 guard cells and the internal temporal data of 18-bits precision in the accumulator for the worst case, and it has an operation frequency of 120 MHz achieving 840 MOPS (millions of operations per second) on a XC2V250 Virtex-II FPGA device.

In [25], Wei *et al.* presented an FPGA implementation of a matched filter with an OS-CFAR detector, focused on the adaptive pseudo noise (PN) code acquisition. In this work, the OS-CFAR detector uses the bubble sorting algorithm to find the *k-th* biggest sample in the reference cell. This OS-CFAR detector uses 16 bits for data, 16 reference cells and none guard cell. The resulting architecture with matched filter and OS-CFAR detector is implemented on a XCV400E Virtex-E FPGA device with a maximum clock frequency of 205 MHz. Other specialized architecture of a CA-CFAR detector is presented in [26]. This architecture is implemented on a XC9600 FPGA device and it was tested with 8-bits for each of the 16 reference cells. Its implementation consists of a storage circuit of 17 shift registers of 8-bits each one, two accumulator circuits, eight 8-bits adders, a multiplier circuit and an 8-bit comparator. Area results and performance were not reported.

3.2.3 Other Implementations

A patented artificial intelligence system called ES-CFAR (Expert System) was presented in [27] by the AirForce Research Laboratory in U.S. This system intelligently senses the clutter environment. By a set of rules along with a voting scheme, this system selects and combines, by an expert system, the most appropriate CFAR detector(s) to produce detection decisions that will outperform a single detector. This expert system is based on five CFAR detectors: CA-CFAR, GO-CFAR, SO-CFAR, OS-CFAR and TM-CFAR. ES-CFAR can be considered within the knowledge-based radar signal and data processing paradigm [28].

Other implementation developed by the the AirForce Research Laboratory was presented in [29]. This work presents the development of a 450-processors, 6-boards embedded signal processing system that provides 400 MFLOPS (millions of floating point operations per second) for the radar processing chain. Within this computer the target detection in a CFAR module achieves 9 MFLOPS.

In [30] it was presented a tool for target detection. This work shows a variation of CA-CFAR, GO-CFAR and SO-CFAR combining the use of morphological filters (erosion and dilatation) to improve the detection in PPI radar images. Also this implementation uses some clutter models (Gausssian, Weibull and K distributions) combined with a specific CFAR detector in order to obtain the minimum probability of false alarms combined with the maximum probability of detection. The statistic estimation is performed by determining the correlation coefficient, mean and standard deviation of the range cells.

Finally, in [31] was proposed an architecture that consists in multiple SIMD (single instruction multiple data) modules in order to achieve the high demands of the radar signal processing chain. This architecture makes all the computations needed for the radar beam forming, radar pulse compression, Doppler filters, envelope detection and CFAR detection. The CFAR module can work on one, two, three or more dimensions, requiring a throughput of 20 MOPS/PE with a bandwidth of 256 MB/s.

3.3 Hardware Sorters

Sorting is one of the most important operations performed by computers. Given their practical importance, algorithms for sorting data have been the focus of extensive research, resulting on several software algorithms proposed to address specific problems [32]. For certain applications, like median filters, Asynchronous Transfer Mode (ATM) switching, order statistics filtering and, in general, continuous data processing, sometimes software only implementations of sorting algorithms do not achieve the required processing speed [33].

In order to speed up the sorting operation, some custom hardware architectures have came up. These hardware sorters can be grouped into two kinds of architectures: sorting networks, including some systolic architectures, and linear arrays. The main idea behind sorting networks is to sort a block of data passing through a network of PE connected in such way that each datum takes its corresponding place. Linear sorters are based on the idea that data to be sorted come in a continuous stream one datum at a time; each datum is inserted into its corresponding place in a register group (sorting array) at the same time that one of the stored datum is deleted.

Figure 3.1.a represents the sorting network idea, where data are firstly stored and then sorted by a sorting network in a parallel fashion. The gray blocks represent the first and last stored data. The first stored datum is the first element leaving the file register, *i.e.* like in a First In First Out (FIFO) scheme. Figure 3.1.b represents the linear sorter idea, where the stored data are always sorted, thus the first and last datum are merged inside of the sorting array. On these sorters, a deleting mechanism must be used in order to free space for incoming data. Some examples of these mechanisms are deleting the oldest datum, selecting one datum or deleting the greatest or the smallest one.

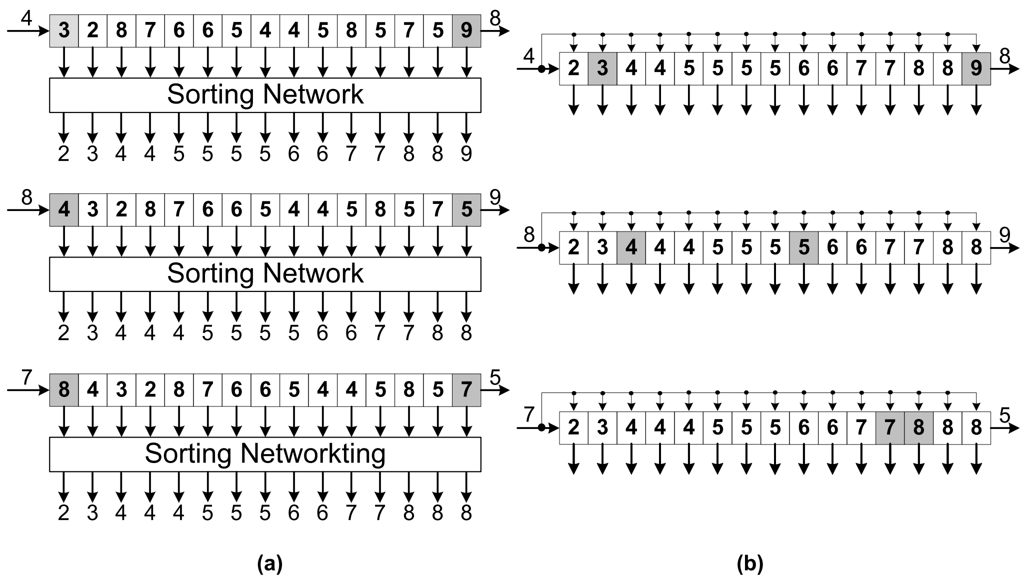


Figure 3.1: Sorting network (a) and Linear sorter (b) architectures.

3.3.1 Network Sorters

The sorting networks are based on a network constituted by several PEs, which consists on a comparator, located in the nodes of the network. The goal of each PE is to sort two input data in ascending (or descending) order by placing the larger (or smaller) datum in a specific output. This technique supposes that a block of data is available for being sorted in parallel fashion. Sorter networks can be pipelined in order to reduce their critical path and latency, thus resulting in a better throughput. The disadvantage of this approach is that the network can potentially require a large number of PEs and, depending on the algorithm, several clock cycles for sorting the whole block of data. On the other hand, if one input datum changes, the whole block of data must be re-sorted. The efficiency of these sorters can be measured by its total size (PEs amount) and by its depth (maximum number of PE from input to output). Both metrics are highly dependent on the number of data the architecture can sort. Figure 3.2 shows an eight elements input sorting network example, of 24 PEs size and depth of 6. Each PE is represented by two interconnected nodes.

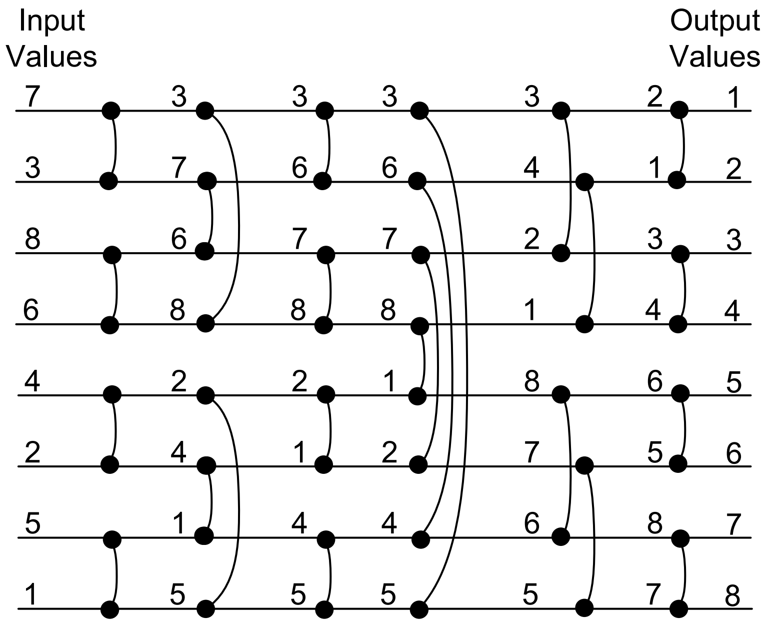


Figure 3.2: Sorting Network Example.

3.3.2 Network Sorters Related Work

In [34], Batcher was the first to introduce the concept of sorting networks. In his work he presented the odd-even merging and bitonic networks. The odd-even merging network consists of two networks that sort all the data contained in odd and even positions separately, applying an interactive rule. The bitonic network works, similarly to the odd-even, merging two monotonic sequences, one in ascending order and the other in descending order. These two monotonic sequences are built by sorting the input data in ascending and descending lists, and merging them. The nodes of both networks are built using PEs. The odd-even network can only sort an *N* fixed number of data. If *N* changes, the network must be rearranged. For this reason Kuo and Huang in [35] proposed a modification of the odd-even sorting network. They proposed a network that can sort any *M* input data smaller than *N*, which is the maximum number of data that the network can sort.

In [36] Tabrizi and Bagherzadeh use a different sorting scheme: basically, they use a tree as a network implemented in an Application-Specific Integrated Circuit (ASIC), where the leaves of the tree are the inputs and the root node is the output. This scheme works in a PISO (Parallel Input-Serial Output) fashion, thus requiring several clock cycles to flush the tree after the beginning of the process.

In [37] Hirschil and Yaroslavsky three different sorting architectures were proposed. One of these architectures does not work as a sorting network neither it sorts the elements; instead it ranks the input data. This Parallel Rank Computer (PRC) receives, in a parallel fashion, a vector of *N* numbers and produces their ranks in two clock cycles. The rank of each number is calculated by comparing every pair of numbers and summing the comparison values.

3.3.3 Linear Sorters

Linear sorters are useful when the sorting data streams and where sorting operation must be carried out after each input datum is received. Linear sorters are composed of a group of cells, each of them capable of deciding if an internal register should hold its current value or update it, either using the input datum or a datum stored in adjacent cells. The advantages of this approach are that it uses fewer area resources and data are always sorted. Figure 3.3 shows a linear sorter example, which inserts the input datum in its corresponding place and, as its deleting mechanism discards the greatest datum stored.

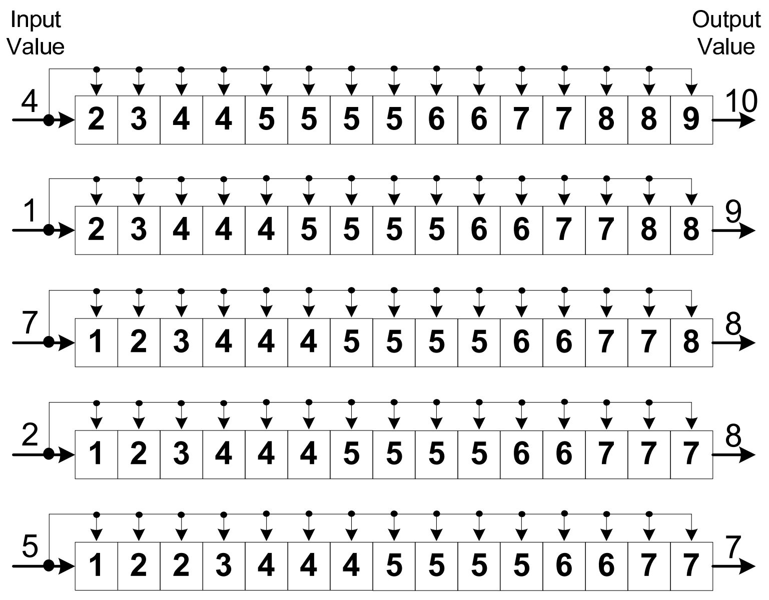


Figure 3.3: Linear Sorter Example.

3.3.4 Linear Sorters Related Work

The other two sorting architectures proposed in [37] are based on shift register architectures operating in a FIFO scheme. One of these architectures, called Serial Rank Computer (SRC), includes two attributes: value and rank. The incoming data are arranged according to their arrival sequence accompanying each number with its calculated rank. The other architecture, a Serial FIFO Sorter (SFS), stores an input vector of data in the order that it is received. This scheme is different from regular FIFO schemes as it keeps the data ordered by magnitude, still data leave the sorter in a FIFO fashion.

A Very-large-scale integration (VLSI) sorter implementation was presented in [33] by Colavita *et al.* They proposed a shift register architecture based on a Basic Sorting Unit (BSU) which contains two registers to store the data and an associated tag, a comparator, and a small logic circuit. This implementation is able to continuously process an input data stream while producing a sorted output in the same way. The data are sorted according to the tags preserving the order of words with identical tags.

Chin-Sheng and Bin-Da Liu in [38] proposed a sorter that uses a column of *N* PE to progressively sort *N* data. These PE are composed of two registers, and a Compare-Swap Cell (CS), which is built by a comparator and a swap unit and they are layout in cascade so their outputs are attached to the inputs of their successors. The idea of the PE is to allow the previous data being held by the PE or shifted to the successor PE at each clock cycle. In [39], Lluís Ribas *et al.* proposed a sorting array (linear shifter) built on data-slice cells. This scheme requires minimal control logic and it is easily expandable. The idea of this sorter is based on the insertion sorting algorithm, which for every unsorted datum, looks for the right position in the sorted list in order to perform the insertion of the unsorted datum into its corresponding place. This architecture only shifts data to one direction, discarding the smallest datum. The data-slice cell is composed of a multiplexer, a register and a comparator, resulting in a compact and simple architecture. A similar sorting scheme was proposed in [40], where data contained in the sorting array can be left o right shifted depending if the datum is going to be inserted or deleted. Both, the datum to be inserted and the one to be deleted are specificated by an input signal. To perform the inserting or deleting process, the cell performs four basic operations: shift right, shift left, load and initialize.

3.4 Summary

This chapter has covered the main theoretical related works about CFAR detectors. These detectors deal with different situations like clutter transitions, multiple target situations, homogeneous or nonhomogeneous background. Even though in the literature can be found more than 30 different detectors, only a few ones are used due to its simplicity when they are implemented in analogical systems. The most common detectors are the linear detectors because they only require computing an average, which does not require a lot of time. On the other hand, although nonlinear detectors, like the order statistics, require more processing time because they perform a sorting operation; they have a better performance than linear detectors.

In the literature reviewed, there are few CFAR detectors implementations reported, and almost all these works are implemented using systolic architectures. These systolic architectures perform the sorting operation on the basis of several PE working in parallel, taking some time to complete their work *i.e.*, they use a network sorting scheme. Specialized architectures that explore nonlinear CFAR detectors using other sorting schemes have not been developed, neither architectures that support modifications of CFAR parameters in run-time.

In the hardware sorting literature, there are two main families: network and linear sorters. The network sorters need to have all the data available in order to perform the sorting, while the linear sorters insert a new datum inside of a sorted linear structure. Network sorters are not recommended for continuous data stream due if one datum is altered, then it is needed to resort all the data block previously sorted. On the other hand, linear sorters do not perform efficiently the sorting over data blocks. In the reviewed works, there can be found several approaches of both sorting families, useful for different applications.

In next chapter it is proposed the CFAR hardware detector architecture, which uses a FIFO linear sorter in order to support nonlinear detectors and the incoming data in stream fashion coming from the radar processing chain. The CFAR hardware architecture supports six different CFAR detectors in one architecture, taking advantage of the similarities of these six variants. The supported detectors are: CA-CFAR, GO-CFAR, SO-CFAR, GOSCA-CFAR, GOSGO-CFAR and GOSSO-CFAR. Also, the next chapter analyze the trade-off for trying of implementing the TM-CFAR and OS-CFAR detectors as well as for modifying the parameters of these detectors.

Chapter 4

Proposed CFAR Hardware Architecture

As shown in previous chapters, a CFAR detector that can be considered optimal under any environmental circumstances has not been designed yet. The idea of the research work is to develop a hardware architecture for CFAR detectors that provides robustness to the target detection process in radar environments as well as to explore these detectors for being implemented in an specialized architecture. An architecture that supports several detectors, allowing to change both the CFAR detector used and the detector's parameters is a solution for dealing with several situations presented in radar applications. The CFAR hardware architecture developed is presented on this chapter. Some of its parameters can be changed in off-line work and other parameters can be changed during execution time. For performing the sorting operation needed for the rank-order selection, a linear insertion sorter based on a FIFO schema is presented too, as part of the CFAR architecture. Also the selection analysis for the implemented detectors is presented.

4.1 Architecture Specifications

The specifications that the architecture must meet are presented as well as the radar characteristics which was employed for the design of this work.

4.1.1 Transmitter-Receiver Used

This work was developed for a commercial non-coherent pulsed radar [11] which transmitter-receiver characteristics are:

• Band	X
• Frequency	9410 MHz
• PRF	3 KHz,750 Hz, 375 Hz
• Pulse Width	0.07 μs, 0.28 μs, 0.9 μs
• Range Resolution	10.5 m, 42 m, 135 m
• Range	1.5 mn, 24 mn, 96 mn
• Bandwidth	20 MHz
• Antenna Rotation	24 RPM
• PPI Resolution	4096x4096 pixels

4.1.2 CFAR Detector Architecture Requirements

Given the radar characteristics mentioned, the signal processing radar chain must process information of both target detection and other previous process, as explained in section 2.1.2, figure 2.4. According to the available radar characteristics, an antenna rotation time lasts 2.5 seconds (24 rotations each 60 seconds). During this time, the radar antenna completes a 360^o rotation sampling its environment. The radar takes a sample of the returned echoes approximately each 0.087^o, *i.e.* 4096 samples are taken during one rotation. Each one of these samples are digitized by the radar in 4096 different discrete values (ranges cells) *i.e.* a range silhouette is formed each 0.087^o and it has 4096 discrete values. Therefore a total of 16,777,216 (4096x4096) samples are digitized during a rotation. These samples must be processed in 2.5 seconds inside the whole radar signal processing chain in order to meet real-time processing. As these data are generated they are stored and shifted in the reference, guard and cell under tests in order to perform the target detection.

Independently of the PRF, the same amount of data is generated. For a PRF of 3 KHz, a range silhouette is formed by the echoes received each 1.8310 generated pulses; and for a 750 Hz and 375 Hz PRF, the range silhouette is formed by 0.4557 and 0.2288 echoes respectively.

In general, putting in together the processing time constraints and the need for a flexible architecture, the CFAR hardware architecture detector must meet with the next requirements:

Real-time processing The resulting architecture must be able to process the 16,777,216 samples in less than the 2.5 seconds, giving time to other processes to perform their functionality.

Modifications of CFAR parameters The architecture must be able to change the CFAR parameters in execution time or change its parameters in off-line work.

Selection of CFAR detector in execution time The resulting architecture must be able to change its functionality among some CFAR detectors previously explained in chapter two.

Low area requirements The architecture must take advantage of the similarities among the CFAR detectors implemented in order to use the same functional units.

In order to present clearly the proposed CFAR hardware detector, the sorter schema used is firstly introduced then it is defined the CFAR detectors implemented and the reasons of selecting these detectors and the parameters that might be altered are exposed. Later the general CFAR architecture is presented.

4.2 Linear Insertion Sorter

In the CFAR detector, each range cell (either reference, guard or CUT) stores an input sample. The first sample received is the first sample to leave the reference window, performing a FIFO functionality. When a new sample is going to be introduced into the detector, the stored samples are right shifted, discarding the oldest stored sample (the right most reference cell). This action makes an empty space for the new sample. When all cells in the reference window have their stored samples, the Z statistic is calculated and the detection process is performed according to the equation 2.7 presented in chapter 2. The same process (inserting, shifting and Z statistic estimation) is performed as long as new samples are introduced into the CFAR detector, *i.e.* the CFAR detector works in a stream fashion.

The CFAR architecture presented uses a sorter in order to perform the sorting operation needed in the order statistic detector. A sorting network schema is not used because it uses high hardware resources. Although further improvements can be achieved in these network schemas, the cost is increased by the amount of PE and their interconnection complexity, making these networks difficult to translate into hardware architectures [39]. Some of these improvements consist on pipelining the network in several stages or a kind of network folding, so data are recirculated through the net. The costs of these improvements is adding some extra circuitry and increasing time latency. Also, the main disadvantage of sorting networks is that if one input datum changes, the whole block of data must be resorted; which make them unpracticable for applications that require processing data in a stream way. On the other hand, although in hardware linear sorter literature there are several schemas for performing the sorting operation, there is only one architecture that works in a FIFO fashion [37]. The architecture requires *n+1* PEs in order to sort *n* values, requiring two memory levels for each PE and working on both falling and raising clock edges. These characteristics make the sorter need more circuitry and a long signal stabilization time. Due these characteristics some simple ideas of other linear sorters implemented in [37], [38], [39] and [40] were considered for developing a new linear sorter [41].

4.2.1 FIFO Insert Sort Algorithm

The linear sorter used is based on the insertion sort algorithm. The algorithm performs, for every unsorted datum, a procedure that looks for the appropriate position in the sorted list to insert the input data [39]. The algorithm is presented in the next pseudo-code:

```
Function 1 InsertSort
Require: Incoming Data each Clock Cycle
Ensure: $D =$Incoming Data
1: $i \Leftarrow 0$
2: while $(i < n)$ $AND$ $(D > R[i])$ do
3:     $R[i] \Leftarrow R[i+1]$
4:     $i \Leftarrow i+1$
5: end while
6: $R[i-1] \Leftarrow D$
```

The algorithm inserts incoming data in the vector *R* of infinite length. However, in practice this characteristic can not be met, thus a deleting condition must be used. In [39], the condition used for deleting is to erase the smallest stored datum, meanwhile in [40] the data to be erased is indicated by an external input signal.

The linear sorter used in this architecture uses a FIFO scheme as deleting condition, allowing the incoming datum to be inserted in its corresponding position. In order to achieve this FIFO-like functionality, it is necessary to keep a life period value for each sorted data. If the datum is shifted, then its corresponding life period value is shifted with it. The life period value is increased by one every time that a new datum is inserted. When the life period value has expired, that is, when it reaches a value equal to the number of elements in the array, the corresponding datum is discarded, making an empty space in the vector and thus allowing the insertion of a new datum. In order to meet this FIFO functionality, three different actions may be performed for keeping the array sorted:

1. Shift the datum stored in the array and its corresponding life period value to the left size.

2. Shift the datum stored in the array and its corresponding life period value to the right size.

3. Hold the data and its corresponding life period value.

To know the direction the data should be shifted to, every element in the array must know on which side, on relation to itself, the datum that is going to be discarded is located. Also, it must know on which side the incoming datum must be stored. This whole functionality can be achieved by creating an array of PEs, called Sorting Basic Cell (SBC). The behavior of an *i-th* SBC can be described by the functions 2, 3, 4 and 5. In order to understand these functions some variables must be described: *CNT[i]* represents the life period value of the *i-th* SBC, *R[i]* the data stored on the *i-th* SBC, *cnti* is a flag that indicates that life period value from a SBC to the right has expired. *D_right* and *D_left* are the output ports to the right and left sides of the SBC respectively. The first function is shown next:

Function 2 SBC_SendData

```
Require: Incoming Data each Clock Cycle
Ensure: $D =$Incoming Data
1: if $R[i] < D$ then
2:     $D\_right \Leftarrow R[i]$
3:     $D\_left \Leftarrow D$
4: else
5:     $D\_right \Leftarrow D$
6:     $D\_left \Leftarrow R[i]$
7: end if
```

The first function *SBC_SendData* is in charge of sending to its left and right neighbors the currently stored value and the incoming data, *D_left* and *D_right* SBC respectively. If the first condition is met, it indicates that this SBC must send to its right its current value (*R[i]*) and to the left the incoming datum, otherwise its current value must be send to its left and the incoming datum to the right.

Function 3 SBC_ResetPeriodLife

```
Require: Incoming Data each Clock Cycle
Ensure: $D =$Incoming Data
1: if $(CNT[i] = 0)\ OR\ ((R[i] < D)\ XOR\ (cnti = 1))$ then
2:     if $(R[i] < D)\ AND\ R[i+1] \geq D$ then
3:         $CNT[i] \Leftarrow 0$
4:     end if
5:     if $(R[i] \geq D)\ AND\ R[i-1] < D$ then
6:         $CNT[i] \Leftarrow 0$
7:     end if
8: end if
```

The first condition of *SBC_ResetPeriodLife* checks if the *CNT[i]* value must be updated while the inner conditions check for those cases where the SBC's counter must be set to zero. This action takes place when the incoming datum will be stored on the *i-th* SBC therefore setting the life period value to zero is needed.

Function 4 SBC_UpdateValues

```
Require: Incoming Data each Clock Cycle
Ensure: $D =$Incoming Data
1: if $(CNT[i] = 0)\ OR\ ((R[i] < D)\ XOR\ (cnti = 1))$ then
2:    if $R[i] < D$ then
3:       $R[i] \Leftarrow R[i + 1]$
4:       $CNT[i] \Leftarrow CNT[i + 1]$
5:    else
6:       $R[i] \Leftarrow R[i - 1]$
7:       $CNT[i] \Leftarrow CNT[i - 1]$
8:    end if
9: end if
```

In the *SBC_UpdateValues* function the first condition checks if the *R[i]* value must be updated by the value coming from its left o right neighbor as indicated by the second condition. Even though the first condition is the same as the one shown in *SBC_ResetPeriodLife* function, they are separated because there is a priority order, if both conditions are met then only the *SBC_ResetPeriodLife* function should be performed.

Function 5 SBC_PropagateFlag

```
Require: Incoming Data each Clock Cycle
Ensure: $D =$Incoming Data
1: if $CNT[i] = 0$ then
2:    $cnti \Leftarrow 1$
3: else
4:    $cnti \Leftarrow 0$
5: end if
```

The final function, *SBC_PropagateFlag*, checks if the life period value of the SBC has expired. The flag, *cnti*, is used by the functions as one of the conditions checked to update the SBC.

In order to fulfill the FIFO sorting functionality, the SBCs must be interconnected (figure 4.1) in a simple linear structure, called sorting array. This linear structure can be easily expandable as long as needed depending on the number of range cells needed.

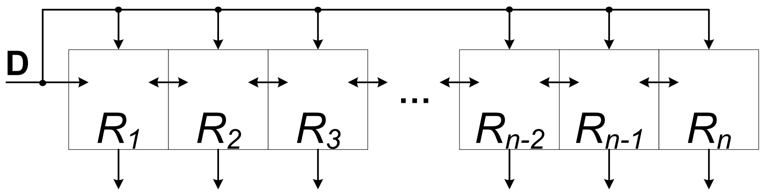


Figure 4.1: Sorting array structure.

For each incoming data *D*, one of the SBCs must discard its value. At the same time, all the SBCs hold their previous value, or store the value coming from the cell to the left or to the right. Only one clock cycle is needed to perform these actions (discarding the oldest data, holding data, right or left shifting). Under this FIFO sorting functionality, there are three insertion cases that are considered and solved by the SBC (shown in figure 4.2, where the gray cell indicates the data to be discarded) are:

1. The datum value is inserted to the left of the cell that discards its stored value (R_{n-2}). In this case data from R_i to R_{n-2} must be shifted to the right side in order to make an empty space for the incoming datum. Then this incoming datum is inserted in R_i.

2. The incoming datum value is inserted to the right of the cell that discards its stored value (R_3). In this case data from R_i to R_3 must be shifted to the left side in order to make an empty space for the incoming datum. Then this incoming datum is inserted in R_i.

3. The incoming datum is inserted at the same position of the discarded value *i.e.* R_i. The rest of the cells hold their values.

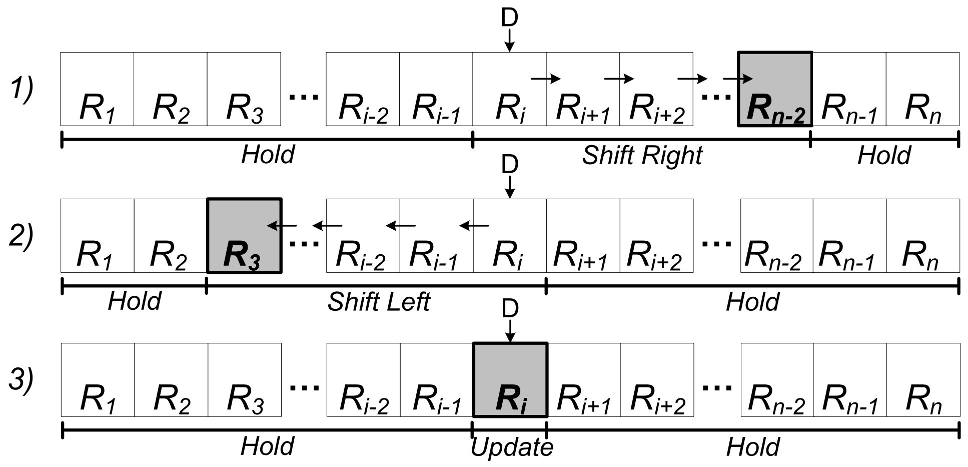


Figure 4.2: Insertion cases inside the sorting array.

4.2.2 Sorting Base Cell Architecture

Each SBC has a register with synchronous load to store the data, a counter with synchronous reset and load to store the period life value of the datum, a comparator, four 2-1 multiplexers and control logic. Figure 4.3 shows how each one of these elements are connected inside of the SBC.

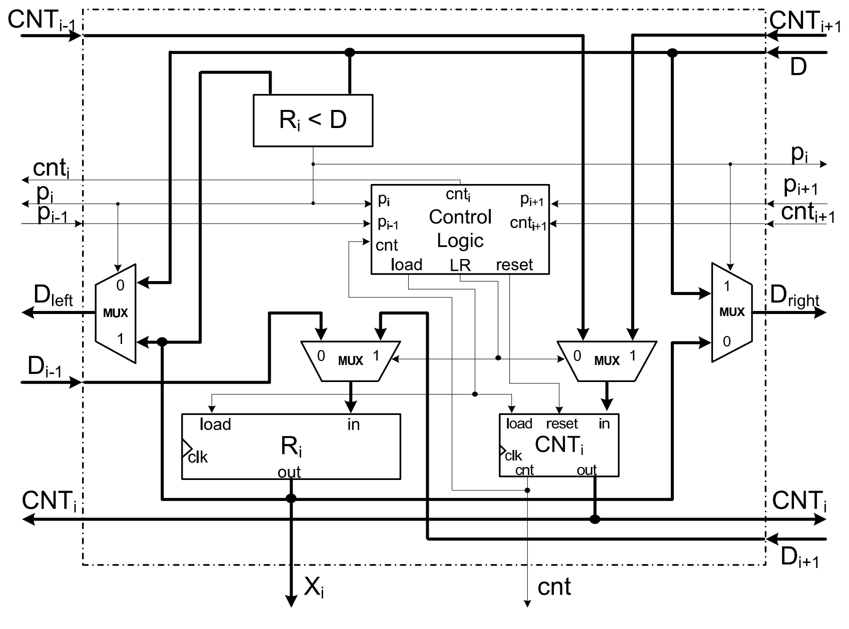


Figure 4.3: SBC internal architecture.

The SBC internal control logic consists of four boolean equations, three of them are in charge of governing the register, the counter, and two of the four multiplexers. The other two multpliexers are governed by the comparator output (p_i). The *load* signal allows both the register and the counter to update their values by an external source, while the *reset* signal sets the counter to certain initial value. The *LR* signal allows to receive the period life value and the value stored from the left or right SBC. As it was mentioned, the p_i signal is the other one that governs the other two multiplexer. This signal is asserted when the comparator condition is met and it is in charge of sending the correct datum to be inserted to the left and right SBC.

Figure 4.4 shows how two SBC must be interconnected in order to assemble them in the sorting array structure shown in figure 4.1.

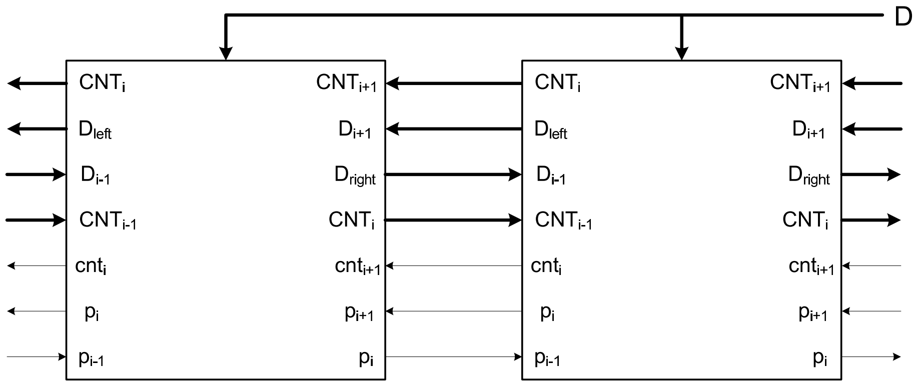


Figure 4.4: Connection example of two SBCs.

The four internal equations can be viewed as representations of the conditions functions explained in the previous section. The first of these equations is equation 4.1. It detects and propagates to the left, information about if the life period value of one of the SBCs to the right has expired as indicated by the *SBC_PropagateFlag* function:

$$cnt_i = cnt_{i+1} + cnt \tag{4.1}$$

where signal *cnt* indicates when the life period value has expired inside of one SBC and *cnt*$_{i+1}$ is the flag coming from the SBC immediately to the right, as shown in figure 4.5.

The *cnt*$_{i+1}$ signal has the information about whether or not the life period value has expired in one of the *n-i* SBCs placed on the right side of the *i-th* SBC. This functionality is simply the boolean operator OR.

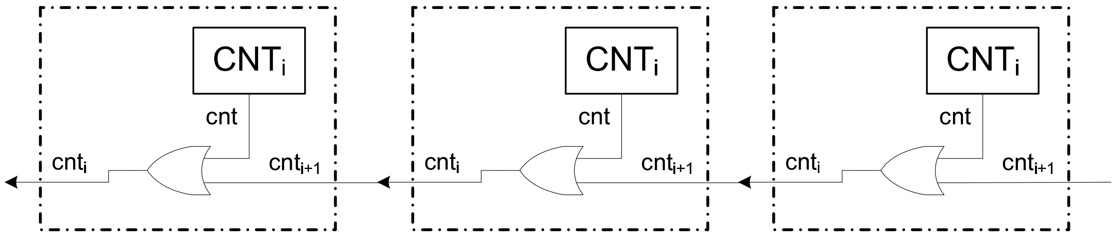


Figure 4.5: Propagation of signal *cnt*$_i$ inside the sorting array.

The second equation of the internal control logic is used as a condition in the function *SBC_ResetPeriodLife* and in *SBC_UpdateValues* function:

$$load = (p_i \oplus cnt_{i+1}) + cnt \qquad (4.2)$$

where p_i is the comparator output, *cnt* indicates when the life period value has expired inside of one SBC and *cnt*$_{i+1}$ is the flag coming from the SBC immediately to the right. This equation controls when the register (*R[i]*) and the counter (*CNT[i]*) inside the SBC must be updated either by the left or right neighbour, *i.e.* load a new value. This equation states that *load* signal is asserted when either one of the next three cases is met:

1. When the value stored on the SBC is the oldest in the sorting array, *i.e.* *cnt* = 1.

2. When a incoming datum value is greater than the value stored inside the SBC and when none of the life period values of the right SBCs has expired, *i.e.* p_i = 1 and *cnt*$_{i+1}$ = 0.

3. When the incoming datum value is smaller or equal than the value stored inside the SBC and when one of the life period values of the right SBCs has expired, *i.e.* p_i = 0 and *cnt*$_{i+1}$ = 1.

When the first case is met, this means that an update operation must be performed in order to discard the oldest value in the array. The second and third cases are represented by the boolean operator XOR and they indicate from when it must start and until where it must stop the updating operation of the SBC values. This functionality is shown in the look-up table (LUT) 4.1 as a truth table.

p$_i$	***cnt***$_{i+1}$	***cnt***	***load***
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Table 4.1: Truth table for equation 4.2

The origin of the data (left or right side) is selected by the internal equation 4.3, which is used in function *SBC_UpdateValues*.

$$LR = p_i \cdot load \tag{4.3}$$

where *p*$_i$ is the comparator output as described by condition in the *SBC_SendData* function and the *load* signal is the result of the equation 4.2. This equation controls where the values that must be inserted come from (left or right side). When the load signal is asserted, it allows to the *p*$_i$ signal decide from which side the values must be inserted. If the incoming value is greater than the one stored in the SBC, it means that the right value must be inserted inside the SBC making a left shifting action in order to preserve the values sorted; otherwise, if the incoming value is not greater than the one stored in the SBC, it means right shifting action must be performed. This equation functionality is simply the boolean operator AND.

The fourth equation that represents the function *SBC_ResetPeriodLife* is:

$$reset = load \cdot [(p_{i-1} \cdot \overline{p_i}) + (p_i \cdot \overline{p_{i+1}})] \tag{4.4}$$

where p_i is the comparator output as described by condition in the *SBC_SendData* function. The signals p_{i+1} and p_{i-1} correspond to the right and left SBC neighbors respectively. The *load* signal is the result of the equation 4.2. This signal is only asserted in the SBC that will take the incoming datum *D*. Therefore it controls when the counter must be set to zero, providing a proper life period value for the new datum. According to the equation, there are two cases when this must occur:

1. When it is allowed to write in the register and counter, *i.e. load* is asserted, the left SBC value is minor than the value of the incoming datum and the value of the SBC is major or equal than the value of the incoming datum.

2. When it is allowed to write in the register and counter, *i.e. load* is asserted, the SBC value is minor than the value of the incoming datum and the value of the right SBC is major or equal than the value of the incoming datum.

If either of these two conditions is met the internal reset signal for the counter will be asserted. The functionality is shown in the look-up table 4.2.

There are some considerations that must be taken into account to ensure proper behavior and to perform correctly the insert sort algorithm. Before performing the insert sort algorithm, all registers R_i must be initialized to zero, while life counter values CNT must be initialized according to *CNT[i] = i*. This is achieved by adding an external reset signal to the register and the counter. Other consideration is that the leftmost p_{i+1} signal's value is always 1 and the rightmost p_{i-1} signal's value is 0. This can be viewed as the leftmost datum having the largest value while the rightmost has the smallest one. Finally, the CNT counter word size depends on the sorting array's length, being a function of $\lceil \log_2 n \rceil$ where *n* is the sorting array length.

load	***p***$_{i-1}$	***p***$_i$	***p***$_{i+1}$	***reset***
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Table 4.2: Truth table for equation 4.4

4.2.3 Sorting Array Example Functionality

Figure 4.6 and figure 4.7 exemplify how the SBC's control signals work in two different situations. In both figures, the first row contains the sorted data currently stored in the sorting array, the second row contains the corresponding life period values and the following rows contain the control signals values needed to perform the insertion operation. The gray column indicates the oldest data to be discarded, whose life period value is 12. Different clock cycles are represented by different tables in the same figure. Only one SBC can have the reset signal (equation 4.4) asserted at each clock cycle. When one SBC asserts the reset signal, it means that this SBC is where the incoming datum *D* will take place in next clock cycle. The *cnt* signal is only asserted when the SBC's life period value has reached the same value of the sorting array length, meaning that this is the oldest datum stored (it is showed by the gray columns). Similarly to the reset signal, only one SBC can have the *cnt* signal asserted at each clock cycle. Note how *cnt*$_i$ signal is propagated through the sorting array to the left side once it is activated according to equation 4.4. Although *LR* signal is always calculated according to equation 4.3, it is only considered when in the same SBC the load signal is asserted.

Figure 4.6 exemplifies how the SBC's control signals work at each clock cycle allowing the sorting array to perform the sorting algorithm. Different clock cycles are represented by different tables in the same figure. This figure illustrates the three previously mentioned insert cases (figure 4.2) in a 3 steps sequence. In this example there is a given sorted sequence (figure 4.6.a). At the first clock cycle the incoming datum value is D = 2. The control signals take their corresponding values allowing the inserting, shifting and deleting operations. Note that at this moment, the incoming datum has not been inserted yet and the oldest datum is still in the sorting array. At the next clock cycle the sorting array is updated (figure 4.6.b) and the second incoming datum D = 18 is also inserted in its corresponding position performing similar actions as the first incoming datum. In this case, there is another datum inside of the sorting array which has the same value. When this case occurs the new datum is inserted to the left side of the datum with the same value, having the oldest datum always at the right most position. This behavior is because the SBC has the comparator unit which performs the comparison in a strictly minor than fashion between its stored datum and the incoming datum. Finally, in figure 4.6.c, the incoming datum value is D = 11 which is placed in the SBC that just discarded its datum.

Figure 4.7 exemplifies how the SBCs must be initialized. After the reset signal is asserted, all the stored data in the SBCs take a zero value, while the period life values are set according to *CNT[i]* = *i* (figure 4.7), *i.e.* the SBC position inside the sorting array. This special initialization for the *CNT[i]* is because it is always needed to discard only one datum from the sorting array in order to make possible the insertion operation. At this moment, the incoming data value is D = 6, thus the control signals take their value in order to perform the insertion of this datum. Like in the previous example, in the next clock cycle, D = 6 is inserted and the datum that has the oldest life period value is deleted, following the sorter normal functionality. Figures 4.7.b - 4.7.f show the insertion process after the initialization, being D = {3, 5, 0, 1, 4} the respective incoming datum value sequence for these figures. Note that in figure 4.7.d the incoming datum value D = 0 is inserted in the left most side of the sorting array and the rest of

Hold *D* *Right Shift* *Hold*

Sorted Value	**1**	**1**	**3**	**4**	**8**	**11**	**15**	**16**	**17**	**17**	**18**	**20**	**22**
CNT_i	4	8	6	7	11	**12**	10	3	0	5	1	2	9
p_i	1	1	0	0	0	**0**	0	0	0	0	0	0	0
cnt	0	0	0	0	0	**1**	0	0	0	0	0	0	0
cnt_i	1	1	1	1	1	**1**	0	0	0	0	0	0	0
load	0	0	1	1	1	**1**	0	0	0	0	0	0	0
LR	0	0	0	0	0	**0**	0	0	0	0	0	0	0
reset	0	0	1	0	0	**0**	0	0	0	0	0	0	0

(a)

Hold *Left Shift* *D* *Hold*

Sorted Value	**1**	**1**	**2**	**3**	**4**	**8**	**15**	**16**	**17**	**17**	**18**	**20**	**22**
CNT_i	5	9	0	7	8	**12**	11	4	1	6	2	3	10
p_i	1	1	1	1	1	**1**	1	1	1	1	0	0	0
cnt	0	0	0	0	0	**1**	0	0	0	0	0	0	0
cnt_i	1	1	1	1	1	**1**	0	0	0	0	0	0	0
load	0	0	0	0	0	**1**	1	1	1	1	0	0	0
LR	0	0	0	0	0	**1**	1	1	1	1	0	0	0
reset	0	0	0	0	0	**0**	0	0	0	1	0	0	0

(b)

Hold *D* *Hold*

Sorted Value	**1**	**1**	**2**	**3**	**4**	**15**	**16**	**17**	**17**	**18**	**18**	**20**	**22**
CNT_i	6	10	1	8	9	**12**	5	2	7	0	3	4	11
p_i	1	1	1	1	1	**0**	0	0	0	0	0	0	0
cnt	0	0	0	0	0	**1**	0	0	0	0	0	0	0
cnt_i	1	1	1	1	1	**1**	0	0	0	0	0	0	0
load	0	0	0	0	0	**1**	0	0	0	0	0	0	0
LR	0	0	0	0	0	**0**	0	0	0	0	0	0	0
reset	0	0	0	0	0	**1**	0	0	0	0	0	0	0

(c)

Figure 4.6: Sorting array example functionality.

Hold ***D*** ↓

Sorted Value	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
CNT$_i$	0	1	2	3	4	5	6	7	8	9	10	11	**12**
p_i	1	1	1	1	1	1	1	1	1	1	1	1	**1**
cnt	0	0	0	0	0	0	0	0	0	0	0	0	**1**
cnt$_i$	1	1	1	1	1	1	1	1	1	1	1	1	**1**
load	0	0	0	0	0	0	0	0	0	0	0	0	**1**
LR	0	0	0	0	0	0	0	0	0	0	0	0	**1**
reset	0	0	0	0	0	0	0	0	0	0	0	0	**1**

(a)

Hold ***D*** ↓ ***Hold***

Sorted Value	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**6**
CNT$_i$	1	2	3	4	5	6	7	8	9	10	11	**12**	0
p_i	1	1	1	1	1	1	1	1	1	1	1	**1**	0
cnt	0	0	0	0	0	0	0	0	0	0	0	**1**	0
cnt$_i$	1	1	1	1	1	1	1	1	1	1	1	**1**	0
load	0	0	0	0	0	0	0	0	0	0	0	**1**	0
LR	0	0	0	0	0	0	0	0	0	0	0	**1**	0
reset	0	0	0	0	0	0	0	0	0	0	0	**1**	0

(b)

Hold ***Left Shift*** ← ***D*** ↓ ***Hold***

Sorted Value	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**3**	**6**
CNT$_i$	2	3	4	5	6	7	8	9	10	11	**12**	0	1
p_i	1	1	1	1	1	1	1	1	1	1	**1**	1	0
cnt	0	0	0	0	0	0	0	0	0	0	**1**	0	0
cnt$_i$	1	1	1	1	1	1	1	1	1	1	**1**	0	0
load	0	0	0	0	0	0	0	0	0	0	**1**	1	0
LR	0	0	0	0	0	0	0	0	0	0	**1**	1	0
reset	0	0	0	0	0	0	0	0	0	0	**0**	1	0

(c)

	D					Right Shift						Hold	
Sorted Value	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**3**	**5**	**6**
CNT_i	3	4	5	6	7	8	9	10	11	**12**	1	0	2
p_i	0	0	0	0	0	0	0	0	0	**0**	0	0	0
cnt	0	0	0	0	0	0	0	0	0	**1**	0	0	0
cnt_i	1	1	1	1	1	1	1	1	1	**1**	0	0	0
load	1	1	1	1	1	1	1	1	1	**1**	0	0	0
LR	0	0	0	0	0	0	0	0	0	**0**	0	0	0
reset	1	0	0	0	0	0	0	0	0	**0**	0	0	0

(d)

					Hold					D		Hold	
Sorted Value	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**3**	**5**	**6**
CNT_i	0	4	5	6	7	8	9	10	11	**12**	2	1	3
p_i	1	1	1	1	1	1	1	1	1	**1**	0	0	0
cnt	0	0	0	0	0	0	0	0	0	**1**	0	0	0
cnt_i	1	1	1	1	1	1	1	1	1	**1**	0	0	0
load	0	0	0	0	0	0	0	0	0	**1**	0	0	0
LR	0	0	0	0	0	0	0	0	0	**1**	0	0	0
reset	0	0	0	0	0	0	0	0	0	**1**	0	0	0

(e)

					Hold				Left Shift		D	Hold	
Sorted Value	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**3**	**5**	**6**
CNT_i	1	5	6	7	8	9	10	11	**12**	0	3	2	4
p_i	1	1	1	1	1	1	1	1	**1**	1	1	0	0
cnt	0	0	0	0	0	0	0	0	**1**	0	0	0	0
cnt_i	1	1	1	1	1	1	1	1	**1**	0	0	0	0
load	0	0	0	0	0	0	0	0	**1**	1	1	0	0
LR	0	0	0	0	0	0	0	0	**1**	1	1	0	0
reset	0	0	0	0	0	0	0	0	**0**	0	1	0	0

(f)

Figure 4.7: Sorting array example initialization.

the values are shifted. Note that, although the value of this incoming datum is similar to the rest of the initialization values, its period life value is reseted and increased in figures 4.7.e and 4.7.f. This is similar to the behavior showed in figure 4.6.b where the incoming datum inserted has the same value that one stored in the right SBC.

4.3 CFAR Detectors and Parameters Selected

Once the sorter schema was designed, it continued the analysis for the hardware architecture for several detectors, taking as a reference the linear sorter implemented. Since keeping the values sorted does not affect the averaging process needed in the linear detectors, the CA-CFAR, GO-CFAR, SO-CFAR detectors can be considered for the architecture. The other detectors considered were: OS-CFAR, TM-CFAR, GOSCA-CFAR, GOSGO-CFAR and GOSSO-CFAR. Also the parameters that vary their detection performance were taken into account: number of reference and guard cells, the *k-th* and *i-th* rank-order samples, the trimming values T_1 and T_2 and the scaling factor α.

4.3.1 CFAR Detectors Selection

Firstly, two different schemas for using the sorting structure needed in the order statistic detectors were considered:

1. Using a sorting array for each lagging and leading reference cells and shift register for the guard cells and CUT. Figure 4.8 shows an example of this schema with two guards cells and the CUT, which values are stored in the shift registers meanwhile the reference cells values are stored in two sorting arrays. In this schema the lagging and leading reference windows can be clearly distinguished.

2. Using only one sorting array for both reference cells, guard cells and CUT. Figure 4.9 shows an example of this schema. Note that in this schema the CUT and the lagging and leading windows are mixed in the sorting structure.

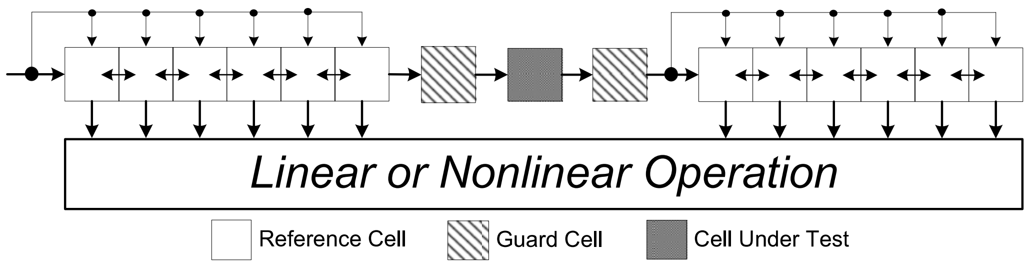


Figure 4.8: Example schema analyzed using two sorting arrays and shift register.

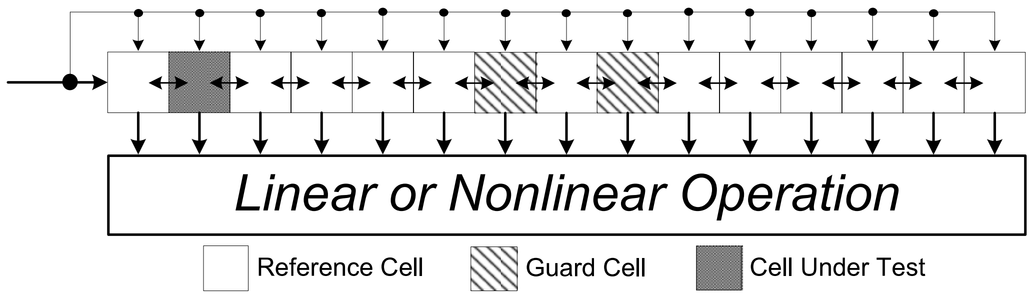


Figure 4.9: Example schema analyzed using one sorting array.

The schema with two sorting arrays has the advantage that both reference windows are clearly separated. For this approach, computing the average needed in CA-CFAR, GO-CFAR and SO-CFAR detectors is easier than for the one sorting schema. Also, the *k-th* and *i-th* rank-order selection for the GOSCA-CFAR, GOSGO-CFAR and GOSSO-CFAR detectors is favoured. This is due to these six detectors perform their calculation in separately reference windows. The one sorting array schema favours the *k-th* rank-order selection for the OS-CFAR detector and the trimming operation of TM-CFAR detector. This is because these detectors perform their operations over the whole reference window, taking these values as one set. Besides, if the schema were implemented, it would require a mechanism for ignoring the CUT and the guard cells, because they do not take part in the computation in TM-CFAR and OS-CFAR detectors.

If the first schema were used for the OS-CFAR and TM-CFAR detectors, it would be needed one extra operation: perform a merge sort action of both sorting arrays in order to have only one set of sorted data, given that these two detectors perform their respective operation over the whole reference window. On the other hand, if the second schema were used for the other detectors, it would be necessary to implement other mechanism (as well as the mechanism for discarding the CUT and guard cells) for distinguishing the values that belong to the lagging window and the other ones that belong to the leading window. The cost implementation of these three mechanisms (the merge sort operation, the discrimination of lagging and leading cells and the discrimination of CUT and guard cells) demands high area resources. There are network sorters that perform the merge sort action but, as it was mentioned in section 4.2, they use more hardware resources. Beside, for distinguishing each cell (CUT, guards cells and the cells that belong to the lagging and leading cells), the mechanism for discrimination could be an *n*-1 multiplexer for each cell that wants to be discriminated, where *n* is length of the sorting array, which is high hardware resources consuming too.

4.3.2 CFAR Parameters Selection

Referencing to the parameters of the CFAR detectors, the *k-th* and *i-th* rank-order samples can be easily implemented in both schemas as well as the scaling factor α. These three parameters can be implemented achieving a modification of their values in run-time. The trimming values T_1 and T_2 of TM-CFAR are complicated to be implemented even if they were not modified in run-time. Figure 4.10 exemplifies this situation with four clock cycles sequence represented by figure 4.10.a - figure 4.10.d. In this figure, a one sorting array structure is used, supposing that it has been used only one sorting array. The values inside the cells are the values stored and the values below are the life period values corresponding to the above cells. The example supposes that T_1 = 3 and T_2 = 3 for the four cases and the incoming data values are D = {3, 1, 10, 6} for each subfigure on this sequence. The reference cells and the CUT are represented by grey cells. Note how these cells move inside the array at different clock cycles in different SBCs.

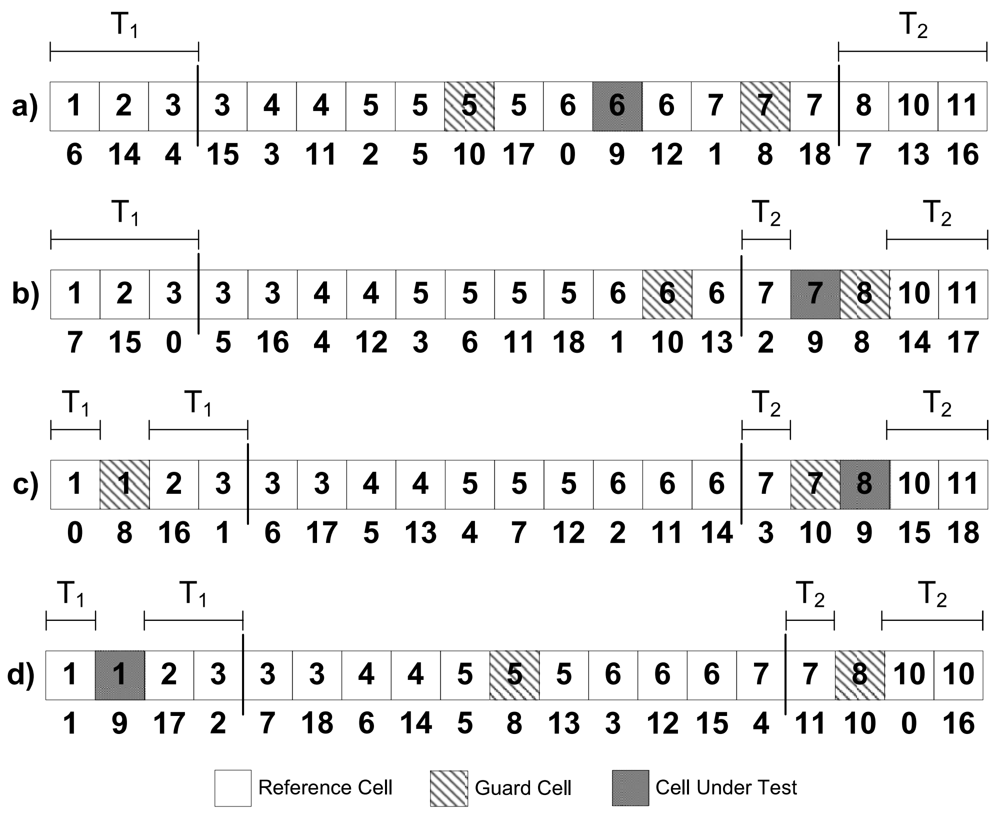


Figure 4.10: Example of some cases for the TM-CFAR detector implementation in a one sorting array structure.

In figure 4.10.a the trimmed values (the three major and three minor values) are clearly distinguished. The CUT and the guard cells are mixed among the values that must be added according to equation 2.15, therefore the discarding mechanism must discard these mixed values. In figure 4.10.b, the three major values are not the three values that must be trimmed because one reference cell is included. In fact, in this case the fifth value from the right size shown in figure 4.10.b must be taken as trimmed value. Similar cases occur in figures 4.10.c and 4.10.d. In order to perform the computing of the *Z* statistic for the TM-CFAR detector, the discarding mechanism must be able to detect if the cell is a CUT, a guard cell, trimmed reference cell or a reference cell used in the *Z* statistic computing. Using *n*-1 multiplexers of each one of the cells is a high cost solution for supporting the TM-CFAR detector in the hardware architecture, making it not suitable for practical implementation. Thus, both the TM-CFAR detector and the one sorting array structure were discarded for the final architecture.

Six of the seven detectors left perform their operation over the leading and lagging window separately. The only detector that need both reference windows in order to perform its operation is OS-CFAR. For incorporating this detector it is needed a merge sorting mechanism that requires high area resources too. Besides it is impractical to implement the merge sort network for only one detector at the cost of area resources. Therefore, the OS-CFAR was also discarded.

At this point of the analysis, the CFAR detectors to be implemented have been selected as well as the sorting array structure in basis of the designed linear sorter. The CA-CFAR, GO-CFAR, SO-CFAR, GOSCA-CFAR, GOSGO-CFAR and GOSSO-CFAR detectors are the ones to be supported by the architecture. The *k-th* and *i-th* rank-order and the scaling factor α can be altered during run-time and be supported by the architecture too. The amount of reference and guard cells is the only parameter that has not been analyzed yet, in order to know if they can be altered during run-time. Figure 4.11 shows a simple example that illustrates the functionality needed if the amount of reference and guard cells were modified in run-time during a sequence of four clock cycles. The incoming data values for each subfigure on the sequence are *D* = {3, 4, 7, 1}. In the first cycle (figure 4.11.a), there are two sorting array structures (reference cells) of four elements and four shift registers (guard cells). In order to move from the first cycle to the second cycle (figure 4.11.b) it is needed to add a new SBC for maintaining the four values stored, allowing the incoming datum to be stored on each sorting array. Also in this cycle, it must be discarded a shift register in both reference windows. Note that in the leading sorting array there are incorporated the two values stored in the shift register, and in the lagging sorting array, no one of the stored values are discarded. This functionality is not performed by the designed linear sorter, because it always inserts and discards only one value at the same time. Therefore for implementing the variation in run-time of the amount of reference and guard cells, the FIFO functionality of the linear sorter should be altered. For moving from the second cycle to the third cycle (figure 4.11.c) it is needed to discard two SBCs and to incorporate two shift registers. At the same time, the lagging sorting array must discard two values and incorporate the incoming one, and the leading sorting array must discard only one value and incorporate two new ones. A similar case occurs in the final transition between the third and fourth dock cycles.

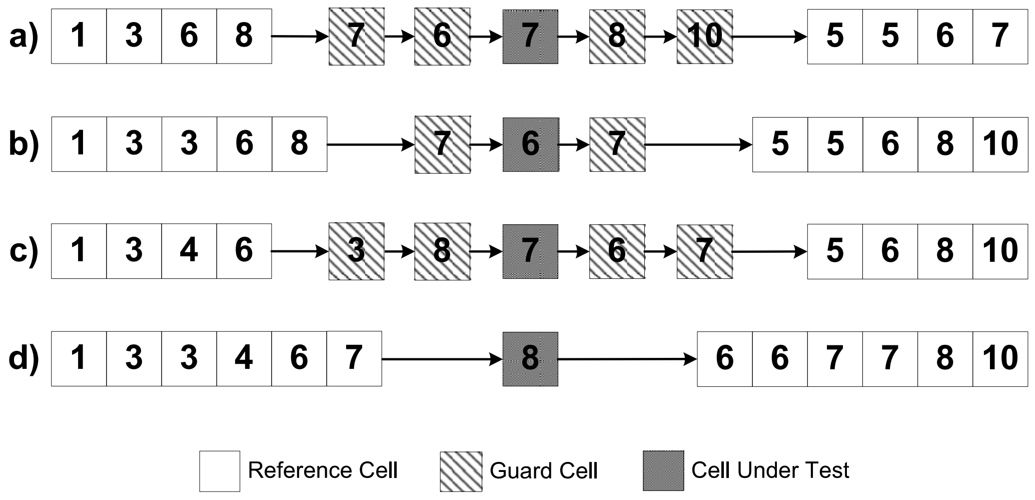

a) 1 3 6 8 → 7 → 6 → 7 → 8 → 10 → 5 5 6 7
b) 1 3 3 6 8 → 7 → 6 → 7 → 5 5 6 8 10
c) 1 3 4 6 → 3 → 8 → 7 → 6 → 7 → 5 6 8 10
d) 1 3 3 4 6 7 → 8 → 6 6 7 7 8 10
Reference Cell
Guard Cell
Cell Under Test

Figure 4.11: Exemple of modification in run-time of the amount of reference and guard cells.

Although the amount of reference and guard cells used in the CFAR also alter the target detection performance, it is uncommon that both parameters are altered during run-time. Besides the changes previously explained in run-time are more factible if partial or total dynamic reconfiguration inside the FPGA is used. These run-time reconfigurations (RTR) are a method of computing using FPGAs that exploit the ability of the FPGA to embed new hardware configurations [42]. In RTR techniques, an FPGA changes configurations from phase to phase of a computation as in figure 4.11. Therefore, for the uncommon change of both parameters in run-time and the insertion and deletion of hardware elements, the modification of both parameters was not used; instead these parameters can be modified in offline work, *i.e.* they are parameterizable.

4.4 CFAR Detector Architecture

With the CFAR detectors selected for the architecture and the parameters that can be altered in run-time, the CFAR detector architecture is presented. The CFAR detector architecture, figure 4.12, consists of two SBC's sorting arrays for *2n* reference cells, *2m+1* shift registers for the guard cells and CUT, which is at the middle of these registers. Also, the architecture has two *n-1* multiplexers that perform the rank operation for the lagging and leading sorting arrays, needed in GOSCA-CFAR, GOSGO-CFAR and GOSSO-CFAR detectors. Given that the reference cells values are ordered, the *k-th* and *i-th* values can be selected by the control signals *Sel-k* and *Sel-i* respectively. The result of this selection is the $Y_{(1)}$ and $Y_{(2)}$ values needed in the nonlinear operations shown in equations 2.12, 2.13 and 2.14.

For the linear operations presented in equations 2.8, 2.9 and 2.10, it is needed to add all the values stored in the leading and lagging sorting arrays for computing the average. In order to perform this operation, it is not necessary to add all values each time that one value from the sorting array is inserted and deleted. Once a value is inserted and other one deleted, the preceding result can be used to compute the next result without adding all values. Only by adding and subtracting the newest and oldest values respectively, the next result is obtained. This whole operation can be performed by the PE Accumulator, which computes the average of Y_1 and Y_2 values on each sorting array. The PE Accumulator, figure 4.13, consists of an adder, which receives the incoming value, a subtracter, connected to one of the multiplexers which selects the oldest value stored in sorting array, a register to store the accumulated and a left shifter that performs the division needed to compute the average Y_n value. Because of the left shifter, only amount of reference cells that are power of two multiple can be used.

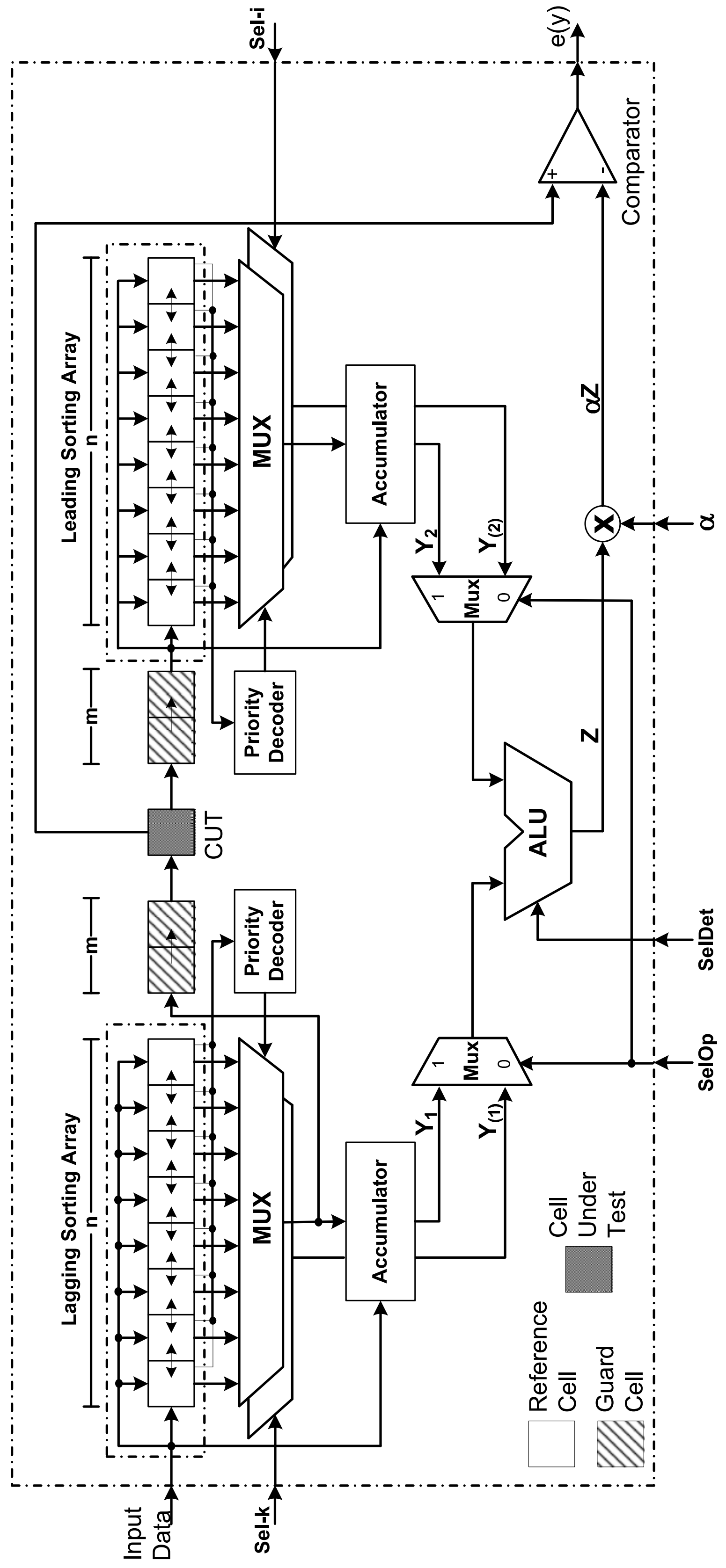


Figure 4.12: CFAR detector architecture.

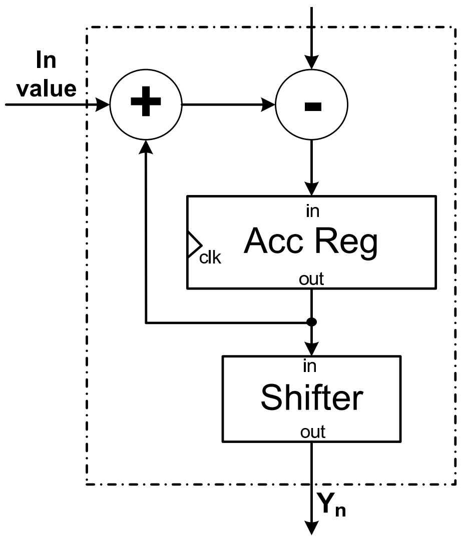


Figure 4.13: PE accumulator.

The oldest value stored in the sorting array is gotten by the n-1 multiplexer whose control line value is generated by a priority decoder. The input of this decoder is a data bus formed by the *cnt*$_n$ signals coming from the SBC in the sorting array and the output bus is *SelOldest* (Select Oldest), as shown in figure 4.14. The *cnt* signal indicates when the life period value has expired in only one SBC's, *i.e.*, the oldest value that must be subtracted and passed to the shift registers.

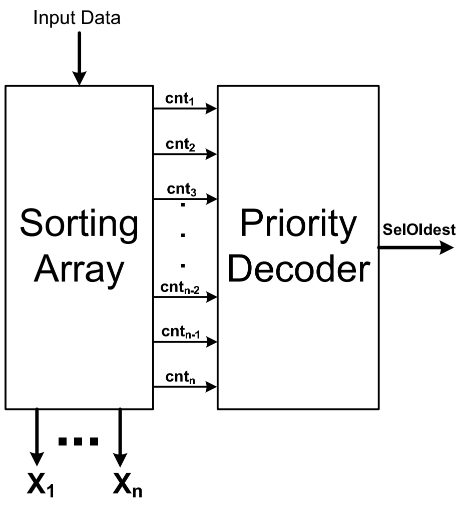


Figure 4.14: Priority Decoder and its connection with the sorting array.

Two 2-1 multiplexers perform the selection between Y_1 and $Y_{(1)}$ from the lagging window and Y_2 and $Y_{(2)}$ from the leading window. The selection is performed by the one bit signal *SelOp* (Select Operation). An ALU-like module provides the three modalities for computing the *Z* statistic: the average, the maximum and the minimum of the rank-order or the accumulated value. The desired modality is chosen using the control signal *SelDet* (Select Detector) established either manually by the user or by an automatic control expert system. A multiplier scales up the *Z* statistic with the scaling factor α and a comparator decides whether a target is present or absent as indicated by equation 2.7.

The control signals, SelDet and SelOp can be grouped into a data bus in order to perform the selection among the six detectors with their corresponding operations with only a data bus. The arithmetic precision of scaling factor α can be modified in off-line work.

In this architecture, on each clock cycle, values flow from the lagging sorting array, to the shift registers and to the leading sorting array. In order to begin the target detection processing, a reset signal must be applied to the SBC in order to initialize the counters inside of the SBCs and PE Accumulators. Once data begin to flow as mentioned above, *N+M+1* clock cycles of latency are required for having all the values stored in the sorting arrays and in the shift registers. After this latency time, the architecture produces a valid output for each clock cycle allowing the continuous operation of the target detection process.

4.5 Summary

This chapter presented a CFAR hardware architecture, which is parameterizable in terms of its reference cells, guard cells and arithmetic precision and capable of changing in run-time the values of *k-th*, *i-th* rank-order factors and the scaling factor α and select six different CFAR detectors: CA-CFAR, GO-CFAR, SO-CFAR, GOSCA-CFAR, GOSGO-CFAR and GOSSO-CFAR.

The architecture uses a linear insertion sorter based on a FIFO schema. The linear insertion sorter is composed of an array of identical PEs called SBCs, wich implements the insert sort algorithm in a compact and efficient way by performing a number of tasks in a single clock cycle. The architecture is based on four functions whose characteristics are translated into four boolean equations, working as an internal control logic for each of these processing elements. The linear sorter can be easily adapted to any length and data width according to specific application needs.

Also, this chapter has analyzed the trade-off for implementing the TM-CFAR and OS-CFAR detectors taking as reference the designed linear sorter. Both detectors need a schema for merging the two sorted arrays, like a merge sort sorting network. Merging both arrays need a lot of area resources and the issues concerned with the trimmed values make the TM-CFAR detector too expensive to be implemented in terms of area resources. It is recommended to explore dynamic reconfiguration in order to implement this detector as well as for modifying the amount of guard and reference cells.

Chapter 5

Implementation and Results

This chapter shows the results of the linear sorter compared against other sorter works as well as scalability results for different combinations of word sizes and sorting array lengths. It also shows the synthesis result of the CFAR hardware architecture implemented and results with distinct amount of guard and reference cells.

5.1 Implementation

The CFAR hardware architecture was developed for a commercial X-band non-coherent radar, whose characteristics were described in section 4.1.1. The linear sorter and the CFAR hardware architecture were modeled in VHDL (VLSI Hardware Description Language) using Active-HDL 7.1 for simulation. The simulation was performed on a notebook whose characteristics are:

- Intel Core 2 Duo Processor at 2.20 GHz
- 2 GB in RAM
- Windows XP Professional.

Once both architectures were validated, they were synthesized with Xilinx ISE 9.1 and targeted for different FPGA devices for comparison purpose. The CFAR hardware architecture was targeted for a Nallatech XtremeDSP Development Kit PCI board, with a Xilinx's Virtex-4 XC4VSX35 FPGA device. This board was installed on a personal computer whose characteristics are:

- Intel Pentium 4 Duo at 2.20 GHz
- 1 GB in RAM
- Windows XP Professional.

5.2 Implementation Results

This section shows the results, comparison and discussion for both the linear sorter proposed and the CFAR detector. Also, the section shows hardware complexity equations for comparison purpose for both architectures. These equations shows the amount of hardware elements (multiplexors, comparator, registers, adders and multiplicators) in order to make fair comparisons with other related works.

5.2.1 Linear Sorter

For the purpose of validation and comparison against other works, the linear sorter architecture was targeted for different FPGA devices. The design was also synthesized for a more up to date FPGA device in order to show results for scalability. Table 5.1 summarizes the FPGA hardware resource utilization and timing performance for the proposed linear sorter and other related sorters, using a Virtex-II. Table 5.2 shows a comparison of the linear insertion sorter against other works in terms of the number of hardware elements they require. In the table 5.2, *n* refers to the amount of data being sorted.

Data for the Bitonic, Odd-Even, Column, and Shifter sorters were taken from [39], while data for the SFS, PRC, and SRC sorters were taken from [37]. Note that a direct comparison between the FIFO linear insertion sorter and other sorters is not possible, except by the SFS sorter which performs the same FIFO functionality.

Sorter	FPGA Used	Speed (MHz)	Latency Clock Cycles	Gate Count	Flip Flops	LUTs Count	Data Sorted	Word Size (Bits)
Bitonic	Virtex II	127	14	153k	-	-	32	16
Odd-Even	Virtex II	147	14	137k	-	-	32	16
Column	Virtex II	66	32	23k	-	-	32	16
Shifter	Virtex II	216	32	12k	-	-	32	16
SFS	VirtexE	115	49	35k	1,372	3,430	7x7	8
PRC	VirtexE	159	2	384k	1,634	11,809	7x7	8
SRC	VirtexE	96	49	19k	784	2,548	7x7	8
FIFO Scheme	**VirtexE**	**72**	**49**	**15k**	**325**	**1,895**	**49**	**8**
FIFO Scheme	**Virtex II**	**126**	**32**	**25k**	**672**	**2,726**	**32**	**16**
FIFO Scheme	**Virtex 4**	**170**	**32**	**26k**	**672**	**3,156**	**32**	**16**

Table 5.1: Performance results with others sorting architectures.

	Sorter					
Number of	**Bitonic**	**Odd-Even**	**Column**	**Linear Shifter**	**SFS**	**FIFO Scheme**
Multiplexors	$n(\log^2 n + \log n)/2$	$n(\log^2 n - \log n + 4)/2 - 2$	*2n*	*n*	*5n+5*	***4n-2***
Comparators	$n(\log^2 n + \log n)/4$	$n(\log^2 n - \log n + 4)/4 - 1$	*n*	*n*	*2n+2*	***n***
Register	$n(\log^2 n + \log n)/2$	$n(\log^2 n - \log n + 4)/2 - 2$	*2n*	*n*	*4n+4*	***2n***
Counters	*0*	*0*	*0*	*0*	*n+1*	***n***
Clock Cycles	$(\log^2 n + \log n)/2$	$(\log^2 n + \log n)/2$	*4n*	*n*	*n*	***n***

Table 5.2: Comparison with others sorting architectures.

It is important to emphasize that the proposed linear sorter differs from other sorters as it implements a FIFO-like scheme where the oldest datum in the sorting array is discarded to make room for every incoming data. Although this sorter performs the same FIFO functionality of the SFS presented in [37], they differ in their internal functionality. The SFS stores an input vector of data in the order that it is received, discarding the oldest datum. At each clock cycle, one datum enters taking its corresponding place inside the sorter according to its value and other datum leaves the sorter. These two characteristics are met by the linear sorter too. Also, both sorters need to store the value age or life period value. The difference between the linear sorter and the Serial FIFO Sorter is that, the SFS sorter consists of two levels of memory elements: main and auxiliar; meanwhile the linear sorter only requires one memory level. Moreover, the SFS needs *n+1* cells to sort *n* elements, requiring an overflow cell. On the other side the linear sorter needs only *n* cells to sort *n* elements. Although the SFS and the proposed sorter operate in one clock cycle, the SFS works during both clock edges: rising and falling edges. During the rising edge the incoming datum is inserted by shifting the needed data to the overflow cell side, having *n+1* sorted elements. In the next falling edge the oldest stored datum in the *n+1* cells is discarded by shifting since the position of the overflow cell until the oldest datum. The proposed linear sorter is able to decide if it is needed to shift to the left or to the right side in one clock cycle instead of performing two shifting operations, allowing a major signal stability in order to access the data stored in the sorting array.

Even though the proposed FIFO linear sorter is not the fastest among all the sorters shown in table 5.1, it uses less hardware resources (gates, FFs and LUTs count) than the SFS sorter that performs similar functions. The SFS sorter is faster than the proposed linear sorter, but this SFS sorter needs longest clock period to ensure the signal stability. Both sorters are able to discard a datum and to insert a new datum in a single clock cycle while maintaining the rest of the data sorted. Network sorters on the other hand would require a large number of clock cycles to sort the data even if only a single datum is replaced.

Although the Bitonic and Odd-Even sorters have a greater maximum operation frequency and a smaller latency than the proposed FIFO scheme, they need to re-sort the data once a datum has changed. This make them impractical for continuous data processing. Also both sorters require less time to sort the *n* data than the linear sorters, however they require that all data to be sorted are available at the same time, which is not always possible specially in applications that produce data in a stream fashion, like CFAR detectors. Moreover they require a larger number of hardware elements.

According to table 5.2, the linear shifter requires the least quantity of hardware elements, followed by the column shifter. Although the linear insertion sorter requires more hardware elements than the column shifter, it is capable of sorting *n* data in the same amount of clock cycles, similar to the linear shifter and the SFS sorter. Also, the proposed linear sorter requires less hardware elements than the SFS, even though they differ in their internal functionality.

Table 5.3 shows the scalability results, in terms of amount of slices, FFs, LUTs and maximum operation frequency, of the linear sorting architecture for the Virtex-4 device. For this comparison, different word sizes and sorting array length combinations were used. The scalability data results are grouped by number of sorted elements (amount of SBCs) and their word size in bits. For clarity, figures 5.1 and 5.2 show the maximum operation frequency and LUTs scalability results in graphs. In figure 5.1, by increasing the sorting array size, the number of LUTs used grows more than twice as the SBC amount is increased at the same proportion. One exception for this is when a 16 bits word size is used and the sorting array is increased from 128 to 256 elements. The rest of the data grows at the same rate. Figure 5.2 shows the maximum frequency operation results when the number of SBCs is increased for the same word size. Generally, the 8 bits sorter has the best performance, except by the 8 bits 64 elements sorter. In this case a 12 bits sorter for the same element number is faster than the others, followed by the 24 bits sorter. The 24 bits sorter is slightly slower than the other ones, but for a register file of size 256, its performance declines more that the other sizes.

SBC	8 Bits				12 Bits				16 Bits			
Amount	**Slices**	**FF**	**LUT**	**MHz**	**Slices**	**FF**	**LUT**	**MHz**	**Slices**	**FF**	**LUT**	**MHz**
8	220	88	423	223	248	120	480	211	317	152	606	204
16	532	192	1,028	212	586	256	1,130	192	758	320	1,458	190
32	1,216	416	2,350	182	1,518	544	2,941	176	1,632	672	3,156	170
64	2,930	896	5,680	156	3,072	1,152	5,965	165	3,389	1,408	6,553	159
128	6,129	1,920	11,772	150	6,432	2,432	12,348	144	7,734	2,954	14,840	137
256	12,635	4,096	24,272	124	14,100	5,120	27,257	121	13,388	6,144	26,080	120

SBC	20 Bits				24 Bits			
Amount	**Slices**	**FF**	**LUT**	**MHz**	**Slices**	**FF**	**LUT**	**MHz**
8	376	184	720	204	425	216	870	198
16	932	384	1,796	189	1,145	449	2,140	194
32	1,978	800	3,806	178	1,983	928	3,824	162
64	3,984	1,664	7,709	156	4,713	1,920	9,081	162
128	8,854	3,456	16,968	136	9,657	3,968	18,496	138
256	17,574	7,169	34,073	120	18,147	8,192	35,256	89

Table 5.3: Scalability results of the sorting architecture.

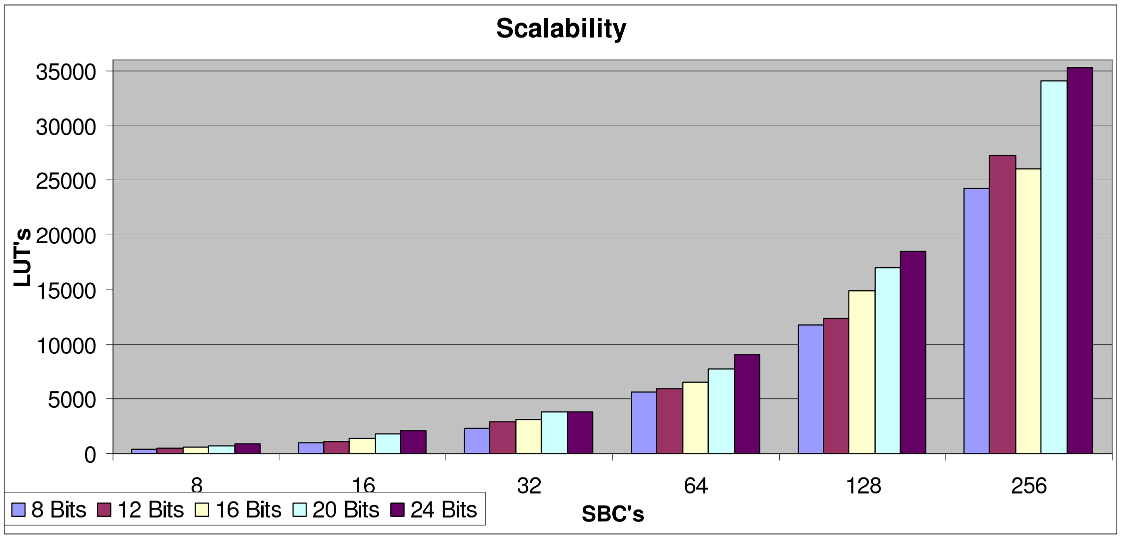


Figure 5.1: LUTs Comparison Results.

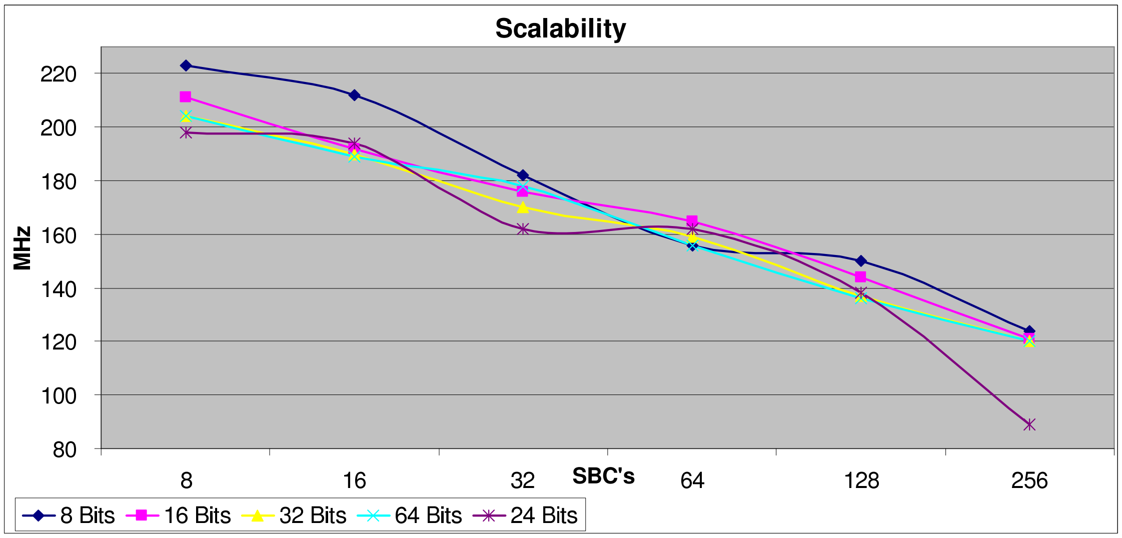


Figure 5.2: Maximum Operation Frequency Comparison Results.

5.2.2 CFAR Detector

The CFAR hardware architecture was targeted for a Xilinx's Virtex-4 XC4VSX35 FPGA device. As it was explained in chapter 4, the CFAR detector architecture is parameterizable in terms of its reference cells and guard cells. The architecture was synthesized for several configurations of reference and guard cells, but the result for the same amount of reference cells varies slightly when different guard cells are used. Table 5.4

summarizes the results of the FPGA hardware resources utilization for the maximum amount of guard cells (four cells at each side of the CUT) with different amount of reference cells. Also this table shows the throughput measured in millions of operations per second (MOPS) and the required processing time for the radar data set of 4096x4096 samples. All these four configurations use 12-bit for data. In fact, a common configuration used for most radar-based applications with a good performance-accuracy trade-off is that one that uses 12-bits for data, 32 reference cells and 8 guard cells [1], which is shown in the third row.

Ref. Cells Amount	**Slices Count**	**LUTs Count**	**FF Count**	**Speed (MHz)**	**Throughput (MOPS)**	**Processing Time (msec)**
8	315	266	581	256	5,888	64
16	602	423	1,147	218	8,502	77
32	1,364	690	2,637	198	14,058	84
64	2,790	1,260	5,430	165	22,275	99

Table 5.4: Results for eight guard cells and some combinations of reference cells.

The proposed architecture requires 84 milliseconds to process a radar data set of 4096x4096 samples, which is 30x times faster than the required theoretical processing time of 2.5 seconds needed for this application parameters; thus this module can be potentially used in radars with much higher resolution or CFAR processor with larger *N* and *M* values, *i.e.*, the amount of reference and guard cells. In fact, with a greater configuration of the CFAR detector of 14-bits of data, 64 reference cells and 8 guard cells, the architecture has a maximum operation frequency of 165 MHz, requiring 97 milliseconds to process the same data amount. This configurations is 25x times faster than the theoretical required processing time.

In [9] the proposed CFAR detector uses only three linear processors, yet it does not implement the sorting functionality. In [25] a nonlinear CFAR detector was implemented with only one nonlinear detector. Since [9] and [26] and [25] do not perform the same functionality, a direct comparison between our proposed CFAR architecture and other architectures is not possible. Nevertheless, a parallelism grade can be applied between [9] and the proposed architecture. This grade can be the architecture's throughput measured in MOPS. Concurrently our architecture performs *N+7* arithmetic operations per clock cycle, while the other architecture performs only 7 arithmetic operations. This means that for the CFAR detector configuration and the maximum operation frequency of this implementation, the architecture achieves a throughput of 14,058 MOPS, as shown in table 5.4. The throughput achieved in [9] is 840 MOPS on a XC2V250 Virtex-II. For throughput comparison purpose, the CFAR hardware detector architecture was also synthesized for this last FPGA device, getting a throughput of 7,526 MOPS which is nine times better. This is the result of the sorting schema used to support the rank-order operation, since each SBC used for storing the values in a sorted way requires two operations and the number of SBC used is proportional to the number of reference cells used.

In table 5.5 a comparison of the CFAR hardware detector against other works in terms of the number of hardware elements they require is shown. In this table *p* refers to the amount of cells in a reference window (section 2.3), *i.e.*, the sum of reference cells (*N*), guard cells (*M*) and CUT, *P* = *N+M+1*. The clock cycles refer to the latency required to process the data and the delay elements refers to hardware elements that are needed to store data.

	CFAR Detector Hardware Complexity				
Number of	**Hwang [20]**	**Han [21]**	**Behar [22]**	**Torres [9]**	**Proposed**
Comparators	$2P+1$	$2P+1$	$3(P^2+P+4)/2$	2	$N+2$
Multiplexors	$2P+1$	$2P+1$	P^2+P+2	1	$4N+7$
Delay Elements	$8P+5$	$5P+7$	$2P^2+2P+1$	P	$2P+M+2$
3-input Adders	0	$P-1$	0	0	0
2-input Adders	$2P$	2	$P+2$	3	3
Multiplicator	1	1	1	1	1
Clock Cycles	$2P+3$	$P+3$	$2P+2$	P	P

Table 5.5: Comparison with others CFAR detector architectures.

Data for Hwang and Han architectures were taken from [21], data for Behar architecture were taken from [22] and data for Torres from [9]. A direct comparison among these CFAR detector architectures is not possible, because they implement different CFAR detectors and Hwang, Han and Behar architectures are systolic designs. Beside the Behar systolic architecture implements a two dimension CFAR detector (*p* rows and *q* columns), thus it needs more hardware elements. However, data shown in table 5.5 were obtained considering that *q = 1* in formulas given in [22].

The architectures in [20] and [21] implement the OS-CFAR, OSGO-CFAR and OSSO-CFAR detectors. In [22] it is used a censored technique over the sorted samples and then the uncensored samples are added. The architecture presented in [9] implements the CA-CFAR, GO-CFAR and SO-CFAR detectors in an specialized architecture. Even though that the proposed architecture implements a sorter functionality, it has the same latency than the architecture in [9] which only implements three of the six detectors that the proposed architecture does. Also, both architectures use the same amount of adders and multiplicators. Due the proposed CFAR architecture performs a sorting functionality, it uses more comparators than in [9], which does not perform this functionality. Among the other three systolic architectures, the proposed CFAR architecture uses lower number of comparators, delay elements and adders, but it uses higher amount of multiplexors. However, compared with the systolic architectures presented in [20] and [21], the proposed solution implements the OSGO-CFAR and OSSO-CFAR detectors with a minor amount of hardware elements.

5.3 Architecture Validation

For validate the results of the CFAR hardware architecture, the CFAR detector was implemented in software using C language for modeling the six detectors that the proposed architecture uses. The input data, *i.e.* range silhouettes, for both cases were obtained from radar scans. Each radar scan consists on several range silhouettes. Figure 5.3 shows a range silhouette example of two thousand samples. Both implementations were fed by the radar data and then their corresponding outputs were compared, *i.e.* when the CFAR hardware architecture performs an erroneous detection or miss detection. Since the proposed architecture has various parameters that modify the detection performance, a complete validation for several combination of different values of *k-th*, *i-th* rank-order factors and the scaling factor α is difficult to estimate, since, this is a task that corresponds to the CFAR designer. However, some tests are included on this work.

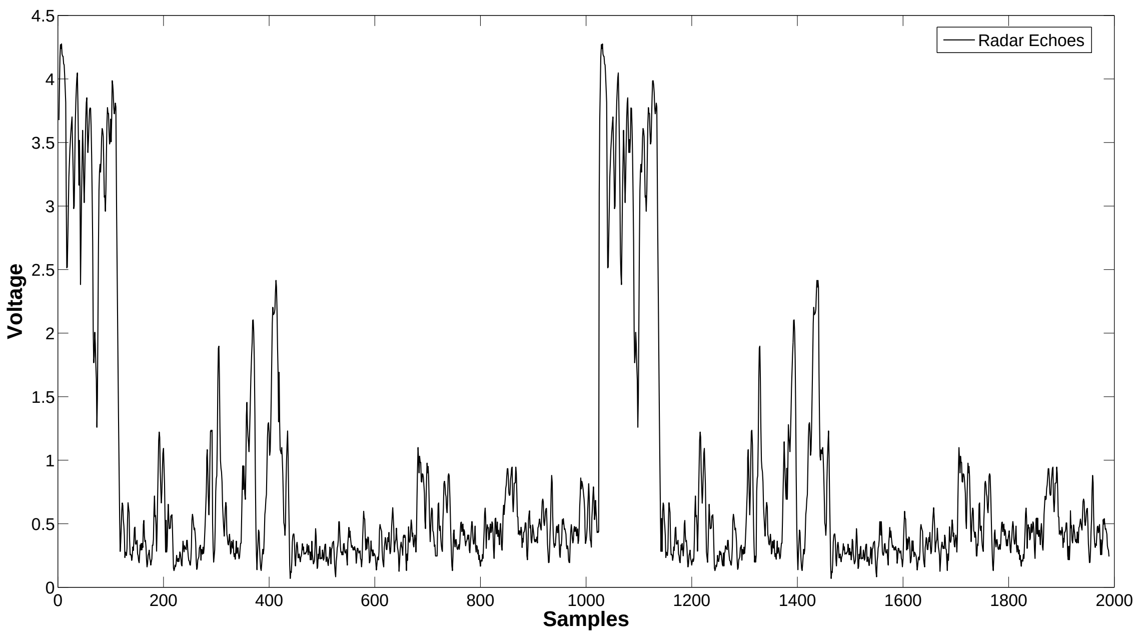


Figure 5.3: Radar receiver range silhouette example.

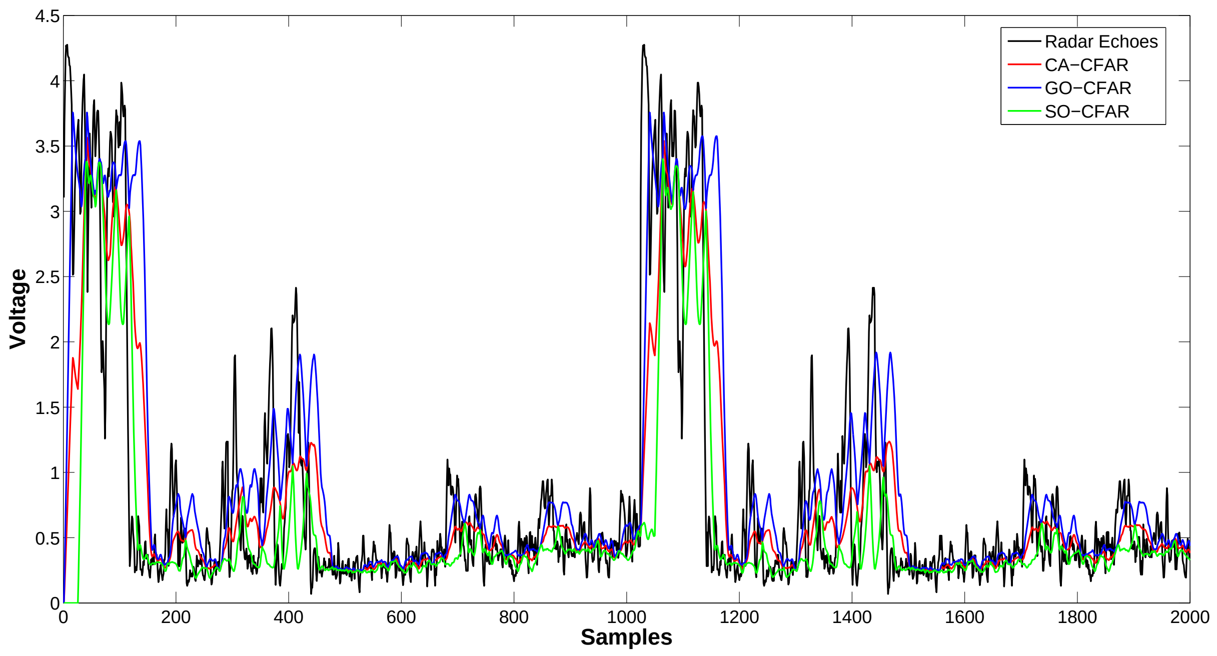
For these tests it was used a typical CFAR configuration *i.e.* 12-bits for data, 32 reference cells and 8 guard cells and the *k-th* and *i-th* rank-order samples = 12. According to [2] a value of 0.75*n* has the best detection performance, where *n* is the size of the lagging and leading windows. The value used for the scaling factor was α = 0.95. However, an exact fixed point representation for the scaling factor is not suitable for the CFAR hardware detector. Therefore, a scaling factor of α = 0.9501953125 was used in the proposed architecture. This approximation is very close to the previous value of 0.95 which was used for the software implementation.

Figure 5.4 shows the resulting threshold calculated by the six CFAR detectors implemented in C language, using as input the radar receiver range silhouette shown in figure 5.3. For clarity purpose, the six CFAR detectors are shown in different graphs containing the linear and nonlinear detectors. Figure 5.5 shows the resulting threshold calculated by the proposed CFAR hardware architecture. At first glance, there are not significant differences among the threshold calculated by the different implementations. However, there are small differences between the thresholds calculated in software and hardware. Figure 5.6 shows these differences at each sample between the C language implementation and the proposed CFAR hardware architecture. Note that the x-axis scale, indicating the difference between the software implementation and the CFAR hardware architecture, is in *mV* meanwhile the thresholds calculated for both implementations are in *V*. This difference is caused by an error in the scaling factor fixed point representation and the average computing need on the linear detectors.

In order to compare the CFAR hardware architecture against the software implementations using an exact and an approximation of the scaling factor value, 1,063,936 radar samples were tested and compared. Table 5.6 shows the differences between the CFAR hardware architecture using α = 0.9501953125 and the software implementation using α = 0.95 and α = 0.9501953125. The radar range silhouettes were tested for each CFAR detector, getting the total of errors or differences compared with the hardware implementation and the percentage of this differences.

	α = 0.95		**α = 0.9501953125**	
Detector	Errors	Error Percentage	Errors	Error Percentage
CA-CFAR	166	0.02096%	31	0.00291%
GO-CFAR	149	0.02049%	18	0.00169%
SO-CFAR	166	0.02132%	15	0.00141%
OSCA-CFAR	118	0.01513%	0	0.0%
OSGO-CFAR	132	0.01711%	0	0.0%
OSSO-CFAR	168	0.02002%	0	0.0%
Total	1,224	-	64	-
Average	204	0.01917%	11	0.00100%

Table 5.6: Comparison results with exact fixed point number and non exact fixed point number of 1,063,936 radar samples.



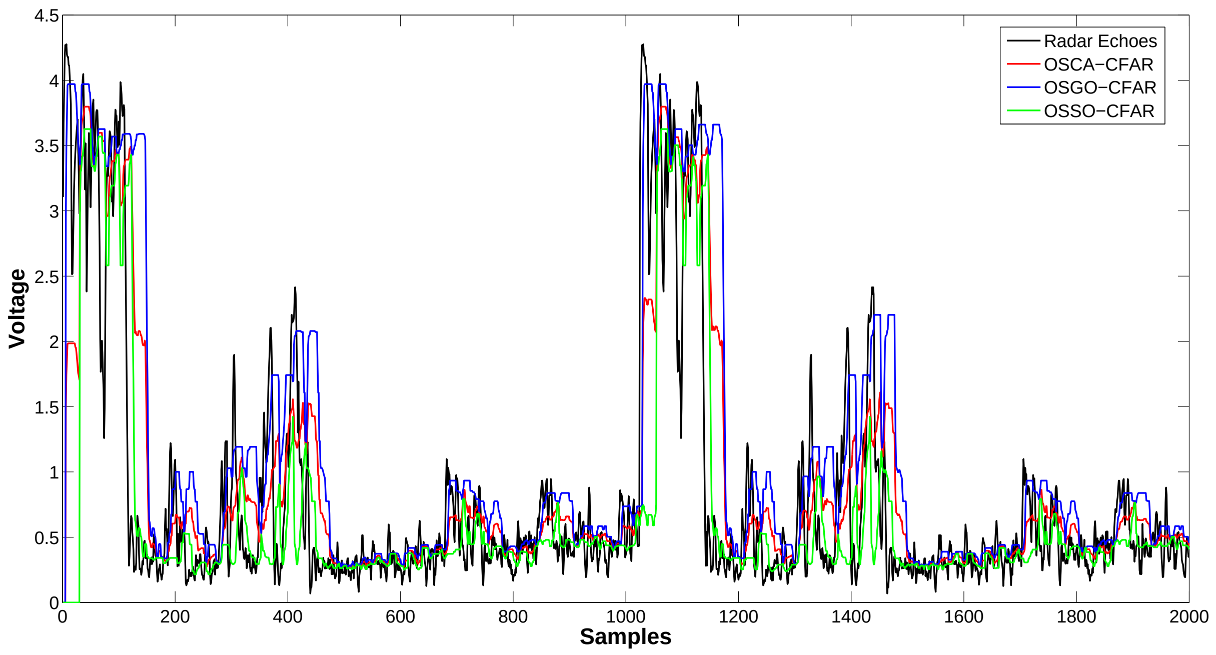
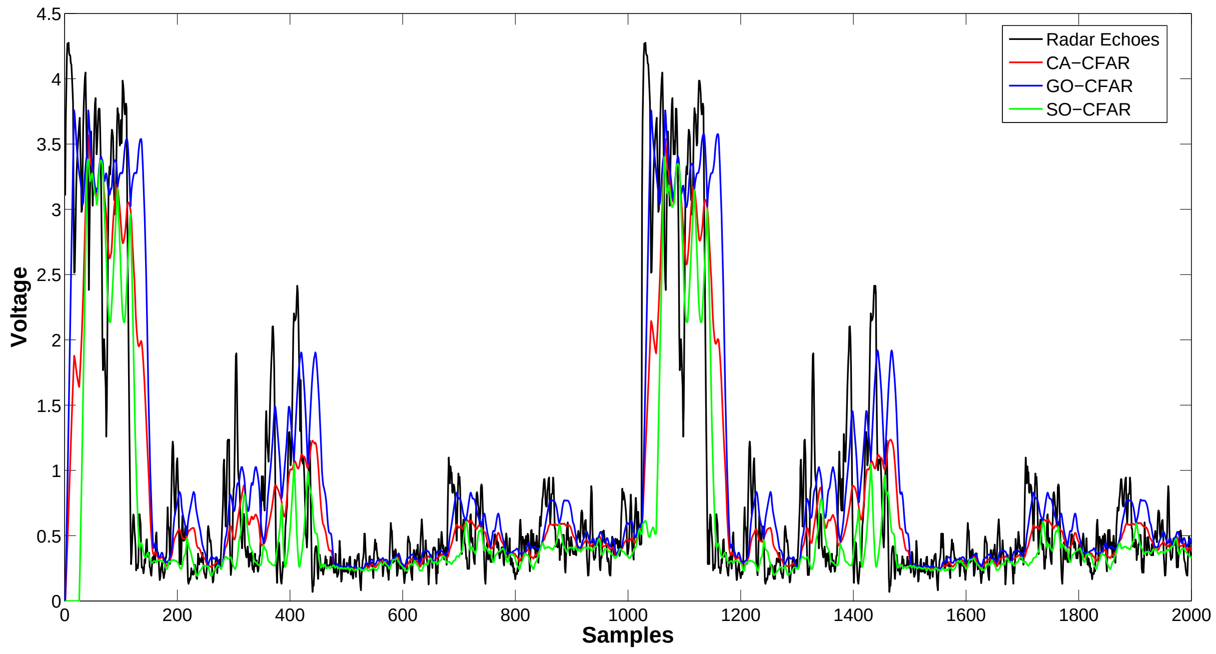


Figure 5.4: Radar receiver range silhouette and thresholds calculated by the software implementation.



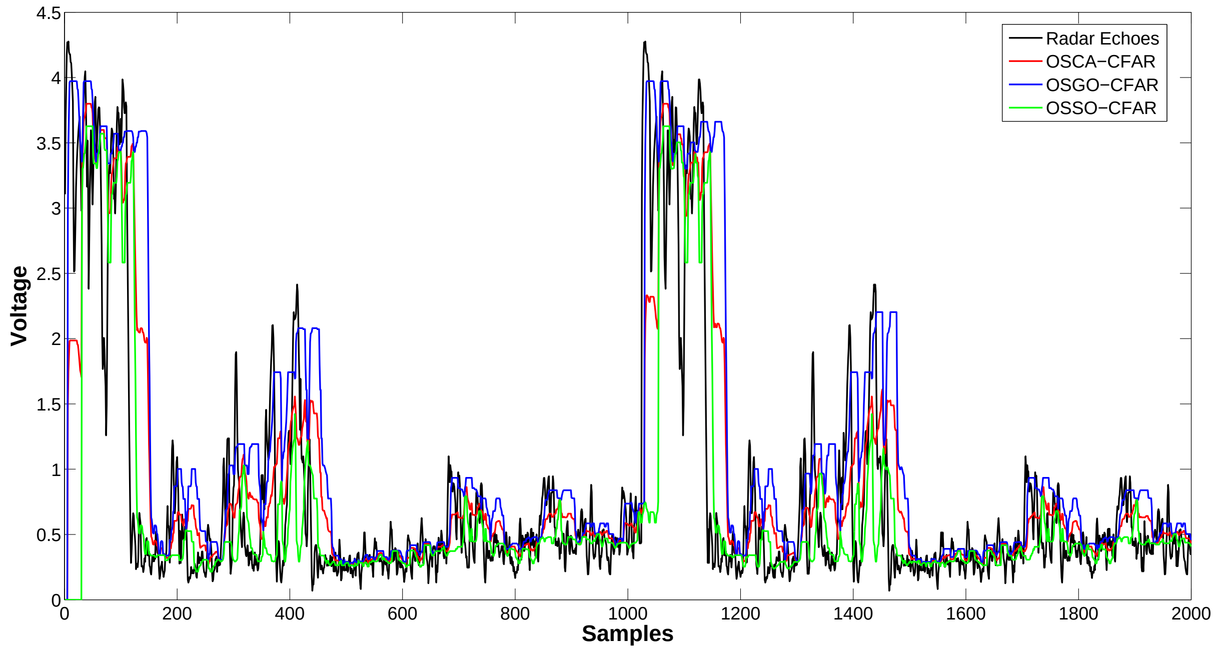
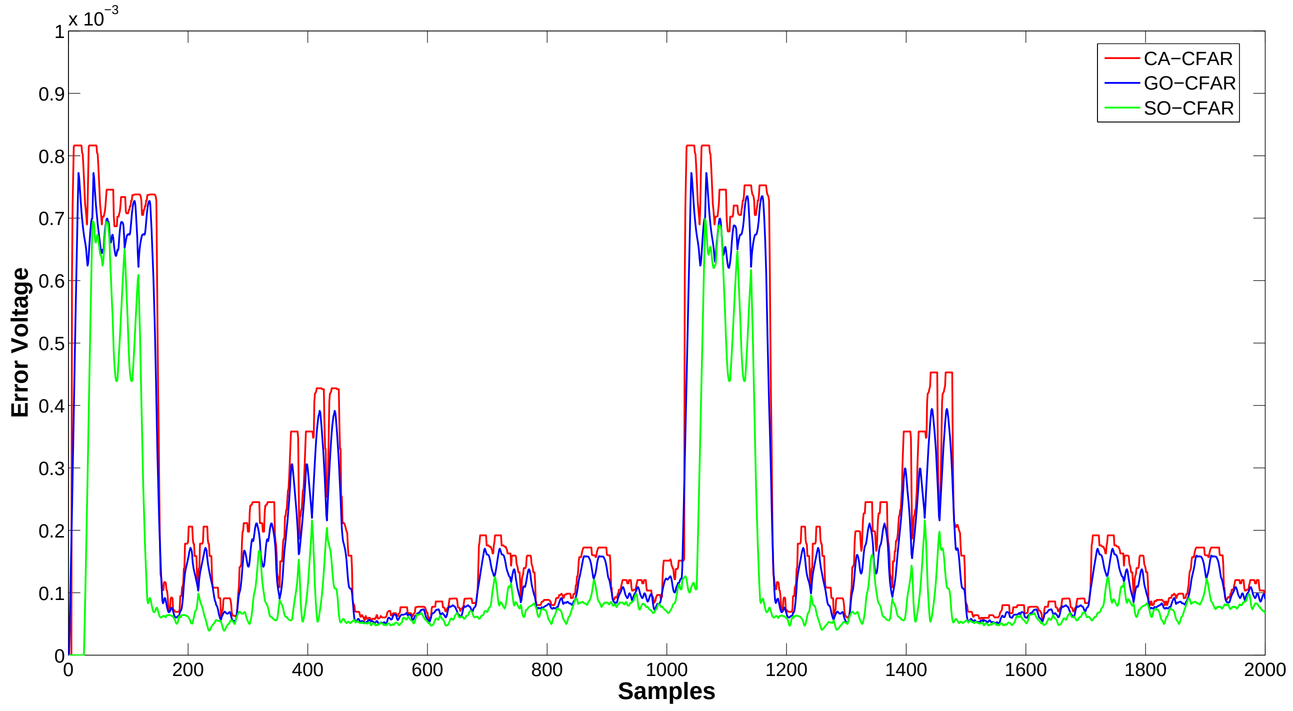


Figure 5.5: Radar receiver range silhouette and thresholds calculated by the proposed architecture.



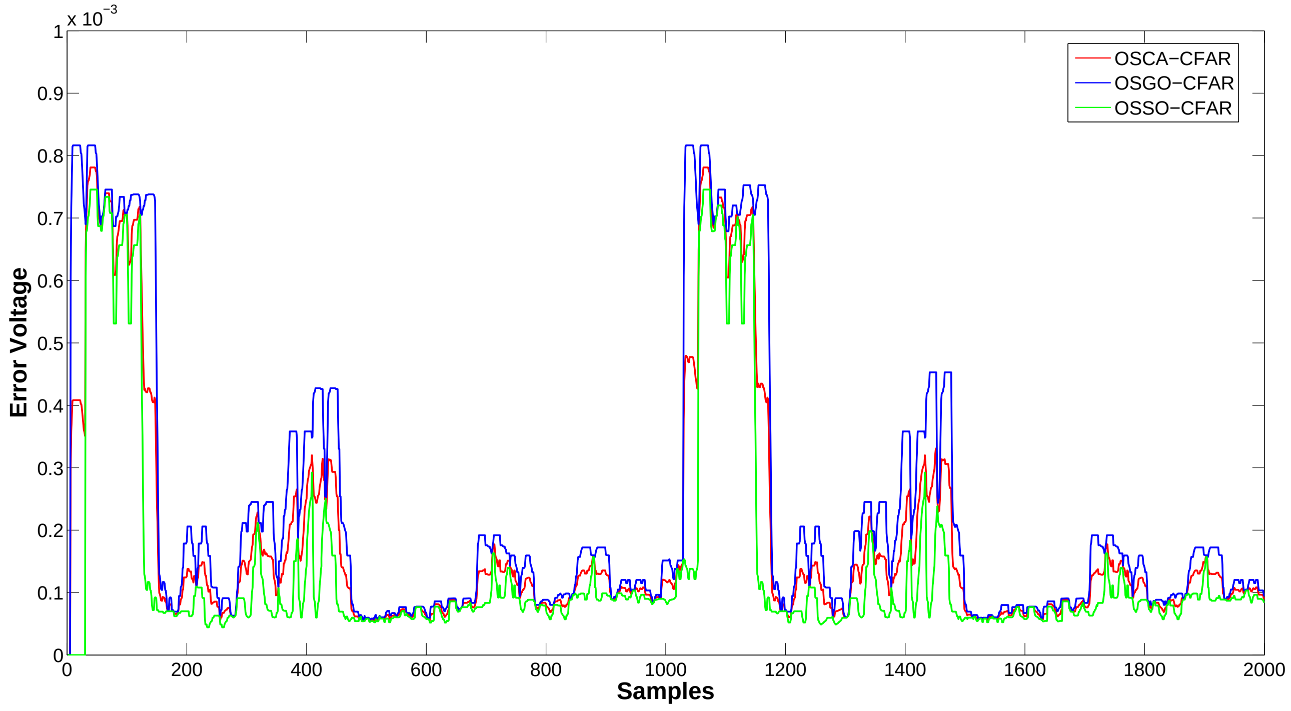


Figure 5.6: Difference among each one of the CFAR detector implemented in software and hardware.

5.4 Discussion

This chapter has presented the implementation results of the linear sorter and the CFAR detector proposed in this thesis. The linear sorter has been compared against other hardware sorters (linear and networks), but only a direct comparison is possible with the SFS sorter. For a same FPGA device, amount of data sorted and word size, the proposed linear sorter uses less hardware elements than the SFS sorter. Although the SFS is faster than the proposed linear sorter, the former uses both clock edges to perform its functionality, which is not a good design practice for digital systems.

The scalability results of the proposed sorter shows that by increasing the sorting array size, the number of hardware elements used grows more than twice when the amount of SBC needed is increased in power of two. The maximum operation frequency is above of 120 MHz in all configurations, excepting the greatest one.

The proposed CFAR hardware architecture uses few hardware resources and it has a maximum operation frequency of above 160 MHz for complex CFAR detector configurations, achieving a performance 25x times faster than the theoretical processing time of 2.5 seconds required for the radar. Also, for more complex CFAR configurations, it is achieved a higher throughput measured in MOPS. Compared with other works, the proposed CFAR architecture implements more CFAR detectors in one schema. It also uses less hardware resources than systolic architectures and it has a minor latency time.

The implementations results of the CFAR hardware architecture shows that an error is introduced because of the truncation of the scaling factor and the average computation needed in CA-CFAR, GO-CFAR and SO-CFAR detectors. The CA-CFAR detector is the more affected by the error introduced by the average computation, because of it needs to compute the average of the *2n* reference cells in spite of *n* cells that are used in GO-CFAR and SO-CFAR detectors. However, when the scaling factor used has an exact fixed point representation these error are reduced.

The threshold computed by the CFAR hardware architecture and the software implementation have a small differences that represent less than 0.02%. However, in spite of the error introduced, the target detection difference against a software implementation are not significant, and they are highly reduced when the scaling factor desired has an exact fixed point representation.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

CFAR detectors are used in signal processing applications to extract target signals from noisy backgrounds. For radar applications, the theoretical aspects of CFAR detectors are advanced, with a number of CFAR schemas proposed for several environment conditions. As no optimal CFAR detector has been proposed, for practical implementations, a versatile processing architecture that is able to switch among different CFAR detectors and performs in real time is required. General purpose processors and DSP are not suitable for implementing these CFAR detectors in SDR systems, because they do not meet the high computational requirements of performance and flexibility required in these systems. This thesis proposed a specialized architecture as an alternative in order to support the workload of radar processing chain. The proposed architecture allows to select among six CFAR detectors (CA-CFAR, GO-CFAR, SO-CFAR, OSCA-CFAR, OSGO-CFAR and OSSO-CFAR), scaling factor α and the *k-th* and *i-th* rank-order sample in run-time, and it is parameterizable in the amount of guard and reference cells, giving robustness to the target detection process.

In order to support the nonlinear processing, the architecture performs a linear insertion sort based on a FIFO schema. The linear sorting operation is performed with an array of PE called SBC, which offers a compact and practical solution for the rank-order operation. The linear sorter is based on four functions whose characteristics are translated into four boolean equations, working as an internal control logic for each of these processing elements. The architecture can be easily adapted to any length and data width according to specific application needs. The nature of this sorter exploits the parallel properties of the insert sort algorithm and achieves excellent performance due to the use of identical processing elements that perform a number of tasks in parallel without the need of a complex control unit.

The CFAR hardware architecture exploits the parallel nature of the CFAR signal processing and it can be easily extended to accommodate larger CFAR detectors as required by more demanding applications. Thus, this high performance, yet compact, architecture can be used as a specialized processing module or as a co-processor in the radar processing chain for conventional systems or even in SDR systems.

The main contributions of this research work are:

- A compact digital implementation of six variants of CFAR detectors, that is able to modify some of its parameters improving the detection performance needed in navigation radars.

- A hardware architecture for target detection, suitable for being implemented in SDR applications and due its high data rate, is able to perform the radar detection task, avoiding to be a bottle neck for future digital implementations of the whole radar signal processing chain.

- A small sorter for applications that are fed in a continuous fashion and need to access to all the sorted values, commonly used in rank-order filtering techniques.

- An analysis of the implementation of detectors that performs its operations over all samples in the reference window taking them as one set.

6.2 Objectives Review

The main objective of this work was: the design and the implementation of a hardware architecture for different CFAR detectors, able to modify some of its parameters. This main objective was accomplished because the resulting architecture implements six distinct variants of CFAR detectors in a single hardware architecture, and it is capable of modifing some parameters in run-time or in off-line work. The modification of these parameters improves the detection performance needed in navigation radars. Also, the proposed architecture supports the high data rate involved in the radar signal processing chain.

As one of the secondary objectives established, the resulting architecture uses few hardware resources, even with greatest configurations of the CFAR detector. This is result of the compact sorting processing element and the similarities among the CFAR detectors implemented, *i.e.*, the operation of averaging, selection of the greatest or smallest values.

The resulting architecture implements a linear sorter based on a FIFO schema. This sorter was implemented in order to deal with the nonlinear detectors implemented, which was another secondary objective.

The last of these objectives was to analyze the trade-off of implementing the CFAR detectors and the modification of its parameters. This was accomplished by analyzing and then selecting the parameters that can be altered in run-time or in off-line work. This analysis suggests the use of partial or total dynamic reconfiguration in order to implement the modifications of all CFAR's parameters.

6.3 Future Work

This thesis also gives the basic ideas for the future implementation of other CFAR detectors and the modification of the amount of reference and guard cells in run-time. This work proposes the exploration of partial or total dynamic reconfiguration in order to implement the OS-CFAR and TM-CFAR detectors as well as for modifying its parameters. However, a discarding or selecting mechanism can be implemented in order to modify the amount of reference and guard cells. Once this mechanism is implemented, other CFAR detectors that work in only one reference window can be supported by the same linear structure.

This work can be tested in real radar environments like in systems that use SDR for performing the radar signal processing chain. Also the proposed architecture can be tested substituting the traditional CFAR analogical implementations.

The proposed architecture works in a one dimension, *i.e.*, over the samples rows of the 4096x4096 matrix of a PPI. There are some architectures like in [22] that process the data in reference windows of two dimensions (columns and rows of the matrix). The proposed architecture can be modified to process data in two dimensions, specially the linear sorter architecture can be redesigned in order to work with two dimension windows, allowing to insert and discard more than one value in the same clock cycle.

Bibliography

[1] M. L. Skolink, *Introduction to RADAR Systems*, 3rd ed. Mc Graw Hill, New York 2001.

[2] P. P. Gandhi and S. A. Kassam, "Analysis of CFAR processors in nonhomogeneous background," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 24, no. 4, pp. 427–445, Jul 1988.

[3] H. Rohling, "Radar CFAR thresholding in clutter and multiple target situations," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 19, no. 4, pp. 608–621, Jul 1983.

[4] A. R. Elias-Fuste, M. G. de Mercado and E. de los Reyes Davo, "Analysis of some modified order statistic CFAR: OSGO and OSSO CFAR," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 26, no. 1, pp. 197–202, Jan 1990.

[5] Y. He, "Performance of some generalised modifed order statistics CFAR detectors with automatic censoring technique in multiple target situations," *IEE Proceedings in Radar, Sonar and Navigation*, vol. 141, no. 4, pp. 205–212, Aug 1994.

[6] P. Tsakalides, F. Trinic, and C. L. Nikias, "Performance assessment of CFAR processors in pearson-distributed clutter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 36, no. 4, pp. 1337–1386, Oct 2000.

[7] D. Wu, Y.-H. Li, J. Eilert, and D. Liu, "Real-time space-time adaptive processing on the STI CELL multiprocessor," in *EuRAD European Radar Conference*, Munich, Germany, Oct 2007, pp. 71–74.

[8] Y. Zhao, J.-F. Gagne, and K. Wu, “Adaptive baseband architecture for software-defined RADAR application,” in *IEEE Canadian Conference on Electrical and Computer Engineering, CCECE*, vol. 2, Montreal, Canada, May 2003, pp. 1087–1090.

[9] C. Torres, R. Cumplido, and S. Lopez, “Desing and implementation of a CFAR processor for target detection,” in *14th International Conference on Field Programmable Logic, FPL’04*, Antwerp, Belgium, 2004, pp. 934– 947.

[10] USA Navy, “Electronics technician, vol 4 - radar systems.”

[11] Santos Martin Lopez Estrada, “Deteccion de blancos en señales de radar mediante seleccion adaptativa de umbrales,” Master’s thesis, INAOE, Tonatzintla, Mexico, December 2004.

[12] H. M. Finn and R. S. Jonhson, “Adaptive detection mode with threshold control as a function of spatially sampled clutter-level estimates,” *RCA Review*, no. 29, pp. 414–464, Sep 1968.

[13] V. G. Hansen and J. H. Sawyers, “Detectability loss due the greatest of selection un a Cell-Averaging CFAR,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 16, no. 1, pp. 115–118, Jan 1980.

[14] G. V. Trunk, “Range resolution of targets using automatic detectors,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 14, no. 5, pp. 750–755, Sep 1978.

[15] A. Farina and A. Russo, “Radar detection of correlated targets in clutter,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 22, no. 5, pp. 513–532, Sep 1986.

[16] Y. He, “Performance analysis of the censored mean-level detector,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 22, no. 4, pp. 443–454, Jul 1986.

[17] P. P. Gandhi and S. A. Kassam, “Adaptive order statistic and trimmed mean CFAR radar detectors,” in *Fourth IEEE Region 10 International Conference*, Bombay, India, Nov 1989, pp. 832–835.

[18] I. Ozgunes, P. P. Gandhi, and S. A. Kassam, “A variably trimmed mean CFAR radar detector,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 28, no. 4, pp. 1002–1114, Oct 1992.

[19] P. P. Gandhi and S. A. Kassam, “An adaptive order statistic constant false alarm rate detector,” in *IEEE International Conference on Systems Engineering*, Fairborn, USA, Aug 1989, pp. 85–88.

[20] J.-N. Hwang and J. A. Ritcey, “Systolic architectures for radar CFAR detectors,” *IEEE Transactions on Signal Processing*, vol. 39, no. 10, pp. 2286–2295, Apr 1991.

[21] D.-S. Han, “VLSI architectures for CFAR based on order statistics,” *Elsevier Signal Processing*, vol. 62, no. 1, pp. 73–86, Oct 1997.

[22] V. P. Behar, C. A. Kabakchiev, and L. A. Doukovska, “Adaptive CFAR PI processor for radar target detection in pulse jamming,” *Journal of VLSI Signal Processing Systems*, vol. 26, no. 3, pp. 383–396, Nov 2000.

[23] I. Garvanov and C. Kabakchiev, “Systolic architecture of adaptive post detection integration CFAR processor in binomial distribution pulse jamming,” *Large-Scale Scientific Computing*, vol. 2907, pp. 448–455, Feb 2004.

[24] C. Kabakchiev and I. Garvanov, “Systolic architecture for adaptive CFAR PI detector,” *Large-Scale Scientific Computing*, vol. 3734, pp. 655–662, Feb 2006.

[25] B. Wei, M. Y. Sharif, T. D. Binnie, and A. E. A. Almaini, “Adaptive PN code acquisition in multi-path spread spectrum communications using FPGA,” in *IEEE International Symposium on Signals, Circuits and Systems ISSCS*, vol. 2, Iasi, Romania, Jul 2007, pp. 1–4.

[26] Saed Thamir R. and Ali Jawad K. and Yassen Ziad T, “An FPGA based implementation of CA-CFAR processor,” *Asian Journal of Information Technology*, no. 4, pp. 511–514, 2007.

[27] M. C. Wicks, W. J. Baldygo, and R. D. Brown, “Expert system constant false alarm rate (CFAR) processor,” United States Patent 1996.

[28] G. T. Capraro, A. Farina, H. Griffiths, and M. C. Wicks, “Knowledge-based radar signal and data processing: a tutorial review,” *IEEE Signal Processing Magazine*, vol. 23, no. 1, pp. 18–29, Jan 2006.

[29] C. Puschak and V. W. Ross, “A power efficient embedded high performance computer for spaced based radar signal processing,” ARFL Space-Based Radar Workshop.

[30] S. Lopez-Estrada and R. Cumplido, “DETECTA: Una herramienta software para evalauar esquemas de deteccion de blancos en presencia de ruido marítimo,” in *Congreso Internacional de Computacion*, D.F., Mexico, Sep 2004.

[31] M. Taveniku, A. Anlander, M. Jonsson, and B. Svensson, “A multiple SIMD mesh architecture for multi-channel radar processing,” in *International Conferecne on Signal Processing Applications and Technology, ICSPAT’96*, Boston, USA, Oct 1996, pp. 1421–1427.

[32] D. E. Knuth, *Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed. AddisonWesley Professional, 1998.

[33] A. A. Colavita, A. Cicuttin, and F. F. G. Capello, “SORTCHIP: A VLSI implementation of a hardware algorithm for continuous data sorting,” *IEEE Journal of Solid-State Circuits*, vol. 38, no. 6, pp. 1076–1079, Jun 2003.

[34] K. E. Batcher, “Sorting networks and their applications,” *Proceedings of the AFIPS Spring Joint Computer Conference*, vol. 32, pp. 307–314, 1968.

[35] C. Kuo and Z. Huang, “Modified odd-even merge-sort network for arbitrary number of inputs,” in *IEEE International Conference on Multimedia and Expo, ICME*, Tokyo, Japan, Aug 2001, pp. 929–932.

[36] N. Tabrizi and N. Bagherzadeh, “Design of a Novel Pipelined and Parallel Sorting Accelerator for a Multiprocessor-on-a-Chip,” in *International Conference On ASIC, ASICON*, vol. 1, Shanghai, China, Octuber 2005, pp. 46–49.

[37] B. Hirschil and L. Yaroslavsky, “FPGA implementations of sorters for non-linear filters,” in *Proceedings of the XII European Signal Processing Conference*, vol. 1, Viena, Austria, Sep 2004, pp. 541–544.

[38] Chi-Sheng Lin and Bin-Da Liu, “Design of a pipelined and expandable sorting architecture with simple control scheme,” in *IEEE International Symposium on Circuits and Systems, ISCAS*, vol. 4, Phoenix, USA, May 2002, pp. 217–220.

[39] L. Ribas and D. J. Carrabina, “A linear sorter core based on a programmable register file„” in *XIX Conference on Design of Circuits and Integrated Systems, DCIS*, Bordeaux, France., Nov 2004, pp. 635–640.

[40] Chen-Yi Lee and Jer-Min Tsai, “A shift register architecture for high-speed data sorting,” *The Journal of VLSI Signal Processing Systems*, vol. 11, no. 3, pp. 273–280, Dec 1995.

[41] R. Perez-Andrade, R. Cumplido, F. Martin Del Campo, and C. Feregrino-Uribe, “A versatile linear sorter based on a FIFO scheme,” in *IEEE Computer Society Annual Symposium on VLSI, ISVLSI’08*, Montpellier, France, Apr 2008, pp. 357–362.

[42] R. Vaidyanathan and J. L. Trahan, *Dynamic Reconfiguration: Arhictectures and Algorithms*. Kluwer Academic Publishers, 2003.