Des Autom Embed Syst (2013) 17:109–127
DOI 10.1007/s10617-013-9121-6

# An adaptive CFAR embedded system architecture for target detection

**Ridha Djemal · Kais Belwafi · Saleh Alshebeili**

Received: 23 September 2011 / Accepted: 17 August 2013 / Published online: 21 September 2013


**Abstract** This paper presents field-programmable gate array (FPGA)-based novel forward and backward automatic censored cell algorithms using a Nios II core processor embedded on a Stratix II FPGA programmable device. These algorithms were recently presented for target detection in a nonhomogeneous environment, and they operate in a complementary manner to allow for high-resolution target detection with a time constraint fixed below 0.5 μs. The ACOSD-based constant false alarm rate detector does not require any prior information regarding the background environment and employs statistical analysis to dynamically calculate the threshold at which the ordered cells under investigation are accepted or rejected. The advantages of the proposed system lie in its simplicity and short processing time while maintaining a low development cost. For a reference window of 16 range cells, the experimental results obtained using the Stratix II development kit demonstrate that the proposed architecture works properly with a processing speed of 100 MHz and an overall detector execution time of 0.11 μs for each range cell. The designed hardware, which is an example of system-on-chip architecture, was physically realized in a Stratix II FPGA device, and the results are presented and discussed.

**Keywords** Constant false alarm rate · System on chip · Real-time system · Radar system

## 1 Introduction

A radar system detects and locates objects or targets by transmitting an electromagnetic signal and receiving echoes from target objects within a volume of coverage. From the received

R. Djemal (✉) · S. Alshebeili
Electrical Engineering Department, King Saud University, Riyadh 11421, Saudi Arabia
e-mail: rdjemal@ksu.edu.sa

K. Belwafi
Electronic and Microelectronic Laboratory, Monastir University, Monastir 5000, Tunisia

S. Alshebeili
KACST Technology Innovation Centre (RFTONICS), King Saud University, Riyadh 11421, Saudi Arabia

signal, the system extracts necessary information on the targets, related to the object type (target or clutter), and the locations of the identified objects. If the echo is associated with a clear or empty background, it can be simply compared with a fixed threshold, and the target is detected whenever the signal exceeds this threshold. However, in actual cases, the echo is accompanied by clutter that varies in time and position; therefore, in target extraction, the threshold should be calculated dynamically from the local background noise/clutter power and should not be constant. In this respect, the use of adaptive signal processing with a variable detection threshold is required to decide whether a target is present. The main idea of the technique is to define a window of cells around the cell under analysis and to determine the clutter information in that window to calculate the actual threshold in a dynamic manner [1].

In practice, improving target detection involves suppression of the undesired signal (clutter) over the echo signal present. The situation may differ over the observation space. We must define the clutter type and describe its properties to readily discriminate targets from the clutter. Several constant false alarm rate (CFAR) techniques for use in radar systems have been proposed in the literature, such as the application of cell averaging (CA) and ordered statistics (OS) [2, 3]. For example, an appropriate reference cell is used in the OS-CFAR detector to estimate the background noise power level. The OS-CFAR detector incurs a small additional detection loss over the CA-CFAR detector for homogeneous backgrounds but can resolve closely spaced interferences. However, it requires a longer processing time than the CA-CFAR detector; thus, the CA-CFAR technique is the optimal CFAR approach for homogenous environments.

Other well-developed OS algorithms, such as the Greatest-of-CFAR (GO-CFAR) algorithm, the Smallest-of-CFAR (SO-CFAR) algorithm [4], the Censored Mean-Level Detector (CMLD) [5], and other OS algorithms [6], have been studied for different scenarios. However, the assumption of a homogenous environment is no longer valid when the number of targets changes abruptly (i.e., the case of multiple interfering targets). In such situations, the performance of the CA-CFAR processor is significantly degraded. Various classes of CFAR techniques have been proposed to enhance their robustness against a non-homogeneous environment for different applications [7, 8] in accordance with the background distribution; however, these implementations have been experimental in the software environment and have not been validated for a real-time system.

In contrast, a major concern in radar signal processing is the maintenance of the false alarm rate at a desired constant value (i.e., the CFAR). To achieve a CFAR, the processed target signal is compared with the cell under consideration using an adaptive threshold detector in a process that requires sufficient knowledge of the statistics (Rayleigh, Weibull, K-, or lognormal distribution) of the clutter. Another concern is the performance of target detection within a limited delay to satisfy real-time constraints, which is particularly important for high-resolution target detection. To accomplish this task, a design exploration of the solution space should be carried out for the CFAR detector.

Although the theory of CFAR radar detection has been well established, the implementation of hardware for a real-time environment is still beyond the currently available high-computational signal processing operations. When designing or implementing a CFAR system, one should anticipate challenges associated with complexity, restricted timing, and large-scale computation. The recent technological developments in complex hardware software systems have allowed for the exploration of digital implementations of the CFAR system and new algorithms in a real-time environment. Due to the real-time constraints of target detection using a high-resolution radar system, system-on-chip (SoC) architecture is an attractive solution for the real-time CFAR processor. In SoC architecture, all components

of a computer, such as the processor and memory, are integrated onto a single chip and operate in an organized manner. In addition to their flexibility, SoC designs consume less power and are less expensive and more reliable than multi-chip systems. Recent advances in field-programmable gate array (FPGA) technology have made SoC fabrication faster and easier.

In this paper, a Nios II processor FPGA-based platform is used to implement automatic censored ordered statistics detector (ACOSD) CFAR algorithms. These recently proposed CFAR-based detectors should be able to operate robustly to detect a target automatically and to determine the number of interferences close to the target to obtain a lognormal clutter distribution. The SoC architecture of the CFAR detector is implemented on an Altera Stratix II EP2S60 FPGA chip. The design goal is to implement a forward ACOSD (F-ACOSD) and backward ACOSD (B-ACOSD) to achieve a processing delay of less than 500 ns, which is suitable for high-resolution radar applications in a desert environment [9].

The remainder of this paper is organized as follows. In Sect. 2, the fundamentals of CFAR theory and the related research on hardware realization for several types of CFAR algorithms are described. Section 3 presents the relevant mathematical formulas and algorithms for both the F-ACOSD and B-ACOSD. The SoC FPGA-based design architecture for the detectors is explained in Sect. 3.4. Section 4 presents the simulation and realization of the system. The conclusions and future research plans are discussed in Sect. 5.

## 2 Related work

A detection method is needed in radar systems to determine the power threshold above which any return can be considered as originating from a target. If the threshold is too low, then more targets are detected, but there are a relatively high number of false alarms. Conversely, if the threshold is too high, then fewer targets are detected but with fewer false alarms. For most radar detectors, the threshold is set to realize the required probability of a false alarm $P_{fa}$. If the background of noise, clutter, and interference are constant in time and space, then a fixed threshold level that provides a specified probability of a false alarm can be chosen. However, under natural conditions, unwanted clutter and interference sources affect the noise level spatially and temporally. In this case, an adaptive threshold can be used in which the threshold level is raised and lowered to maintain the constant probability of a false alarm. This technique is called CFAR detection.

A typical CFAR processor is shown in Fig. 1. The input signals are set serially in a shift register. The contents of the cells surrounding the cell under test, $X(0)$, are processed using a CFAR processor to obtain the adaptive threshold, $T$. The value of $X(0)$ is then compared with the threshold to make a decision. The cell under consideration is declared a target if its value exceeds the threshold value.

The conventional cell-averaging (CA)-CFAR technique proposed by [10] sets the threshold adaptively by estimating the mean level in a window of $N$ range cells. Many CFAR algorithms have been developed in recent years. We can categorize a CFAR algorithm into one of four models according to the clutter power distribution and interfering targets.

- When a transition occurs in the clutter power distribution, we can use, for example, greatest-of-selection logic for the CA-CFAR detector (GO-CFAR) [11] to control the increase in the probability of a false alarm. If one or more interfering targets are present, the GO-CFAR detector performs target detection poorly. Thus, previous studies have suggested that an SO-CFAR algorithm employing smallest-of-selection logic should be used instead of the GO-CFAR algorithm for the CA-CFAR detector [12].

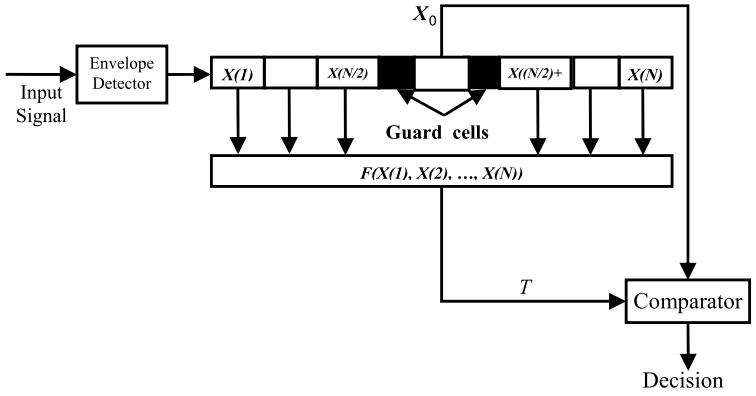


**Fig. 1** Block diagram of a typical CFAR algorithm

- When the clutter background is composed of homogeneous white Gaussian noise in addition to interfering targets, the CMLD can be used as a target detector. The CMLD censors target samples and estimate the noise level from the remaining noise sample. Additionally, the trimmed mean-level CFAR (TM-CFAR) detector [3] implements trimmed averaging after ordering the samples in the window. When the number of interfering targets is not known a priori, the generalized CMLD (GCMLD), for which the number of interfering targets is determined and their corresponding samples are then sampled, can be used as well as the OS-CFAR detector, which chooses one ordered sample to represent the estimated noise level in the cell under consideration. If transition occurs not only in the clutter power distribution but also in the interfering targets, a commonly used technique is the generalized two-level CMLD (GTL-CMLD) [13], which uses an automatic censoring algorithm of the unwanted samples when both interfering targets and extended clutter are present in the reference window of the cell under consideration.
- The last category considers non-Gaussian clutter distribution. The lognormal, Weibull, gamma, and K-distributions have been used to represent the envelope-detected non-Gaussian clutter distribution. Publications on CFAR detection for Weibull clutter have been reported. For example, the maximum-likelihood CFAR (ML-CFAR) algorithm has been presented [14], and its performance was analyzed for the case in which both the scale and shape parameters are unknown. Furthermore, the optimal Weibull CFAR (OW-CFAR) algorithm, for which the test statistics are expressed according to the estimate of the mean power of the Weibull clutter, has been proposed [15].

The theoretical developments of CFAR detection have clearly not been followed by hardware implementation. Few attempts to implement the hardware of CFAR processors have been reported. In particular interest is the parallel pipelined hardware implementation of a CA-CFAR-based target detection system in a noisy environment using TMS320C6203 digital signal processor (DSP) and FPGA devices [16]. The processing time achieved for this implementation was approximately 420 ms using 32 reference cells with eight guard cells. Another example of OS-CFAR using the Virtex-II-V2MB100 development kit has an execution time for the detection algorithms of 0.48 ms (0.6 μs for each range cell) for a dataset of 800 samples using only 16 reference cells [17]. These proposed delays are not suitable for high-resolution detection, which requires a delay of less than 0.5 μs per range cell. Versatile hardware architectures for CFAR processors based on a linear insertion sorter implementing

CA, OS, SO, and GO variants of the CFAR algorithm have been presented [18]. Even if the proposed architecture is configurable to switch between all variants within one clock cycle, no information related to real-time applications has been presented. Other implementations of CA-CFAR and OS-CFAR using parallel structures have been presented using a DSP and FPGA [19], but the execution delays were not optimized. As mentioned above, many parts of the target detector have been implemented using a DSP and hardware technologies; the performance of the new CFAR techniques in the presence of noise with a lognormal distribution operating in real time remains unknown. Alsuwailem et al. [20] implemented an automatic censoring CFAR detector, known as the TM-CFAR and Automatic Censored Cell Averaging (ACCA) ordered data variability CFAR detector. However, the implementation did not consider the real-time aspects, and offline validation was carried out without allowing interactive interaction with the architecture environment. Furthermore, no standard interface was given to facilitate communication with the radar system environment. Winkler et al. [21] used a SoC design containing a reconfigurable processor for an automotive radar sensor in which the processor controlled all custom logic and input/output tasks.

Almarshad et al. [22] recently introduced automatic censoring detectors called ACOSD CFAR detectors. The algorithm is able to censor an unknown number of interfering targets in lognormal clutter automatically. Because of the enhanced radar resolution, the lognormal distribution is more reliable than the Rayleigh distribution in representing the amplitude of clutter. Meanwhile, the automatic censoring algorithms developed for Rayleigh clutter, as presented in [23], cannot be directly extended to cases in which the clutter samples are drawn from a lognormal distribution. It has recently been reported that a hardware implementation of the F-ACOSD target detector on a Stratix II development board can operate at up to 115 MHz with a delay of 0.29 μs for each range cell under a lognormal distribution [24]. Therefore, it is necessary to investigate the performance of the system after including appropriate interface delays related to interfaces and wrappers. In the remainder of this paper, we consider a new implementation, namely, the ACOSD-CFAR algorithms proposed in [22] for SoC implementation using Nios II core processors embedded within an FPGA. Using the MicroC/OSII real-time operating system integrated with the Nios II processor and an adequate optimization technique, we attempt to divide the total delay presented in [24, 25] by two without any restriction on the performance of the ACOSD algorithms.

## 3 ACOSD detection algorithms

In the case of ACOSD CFAR algorithms, detection consists of two steps: removing the interfering reference cells (censoring step) and the actual detection (detection step). Both steps are performed dynamically using a suitable set of ranked cells to estimate the unknown background level and to set the adaptive thresholds accordingly. This detector does not require any prior information regarding the clutter parameters, nor does it require the number of interfering targets. In a CFAR processor, the radar outputs $\{X(i) : i = 0, 1, \ldots, N\}$ are stored in a tapped delay line. The cell with subscript $i = 0$ is the cell under consideration and contains the signal that will be judged as either belonging to a target or not. The last $N$ surrounding cells are auxiliary cells used in the CFAR procedure. In the ACOSD CFAR algorithm, the $N$ surrounding cells are ranked in ascending order according to their magnitudes:

$$X(1) \leq X(2) \leq \cdots X(p) \leq \cdots \leq X(N) \tag{1}$$

After sorting, the sorted cells are sent to the detection stage.

### 3.1 B-ACOSD detection algorithms

In the B-ACOSD algorithm, the sample $X(N)$ is compared with the adaptive threshold, $T_{ci}$, where $i$ takes a value from zero to $X$. To begin the comparison, the threshold is defined by

$$T_{c0} = X(1)^{1-\alpha_0} X(p)^{\alpha_0}, \tag{2}$$

where $X(p)$ is the $p$th largest sample and $\alpha_0$ is a constant chosen to achieve the desired probability of false censoring ($P_{fc}$). Using Monte Carlo simulation, values of $p > N/2$ are observed to yield sufficient detection performance [22]. Given the first threshold:

- If $X(N) < T_{c0}$, then $X(N)$ belongs to a clutter sample without interference and the algorithm terminates;
- If $X(N) > T_{c0}$, then $X(N)$ is a return echo from an interfering target. In this case, $X(N)$ is censored and the algorithm proceeds to compare the sample $X(N-1)$ with the following threshold to determine whether it corresponds to an interfering target or a clutter sample without interference:

$$T_{c1} = X(1)^{1-\alpha_1} X(p)^{\alpha_1}. \tag{3}$$

At the $(k+1)$th step, the sample $X(N-k)$ is compared with the threshold $T_{ck}$, and a decision is made according to the test

$$X(N-k) \underset{H_0}{\overset{H_1}{\gtrless}} T_{ck}; \quad 0 \le k < N-p; \text{ where } T_{ck} = X(1)^{1-\alpha_k} X(p)^{\alpha_k}. \tag{4}$$

$H_1$ represents the case where $X(N-k)$, and thus, the subsequent samples $X(N-k+1), X(N-k+2), \ldots, X(N)$ correspond to clutter samples with interference. $H_0$ denotes the case in which $X(N-k)$ is a clutter sample without interference. Successive tests are repeated as long as the hypothesis $H_1$ is deemed true. The algorithm stops when the cell under investigation is deemed homogeneous (i.e., a clutter sample only) or, in the extreme case, when all $N-p$ highest cells have been tested, that is, $k = N-p$. Figure 2 presents a block diagram of the B-ACOSD algorithm.

In the detection step, the cell under consideration, $X_0$, is compared with the threshold $T_{ak}$ to decide whether a target is present according to

$$X_0 \underset{H_0}{\overset{H_1}{\gtrless}} T_{ak}; \quad 0 \le k < N-p+1. \tag{5}$$

Hypothesis $H_1$ denotes the presence of a target in the test cell, whereas hypothesis $H_0$ denotes that there is no target.

In the B-ACOSD CFAR algorithm, the threshold $T_{ak}$ is defined as

$$T_{ak} = X(1)^{1-\beta_k} X(N-k)^{\beta_k}, \tag{6}$$

where the value of $\beta$ is selected for the designed probability of a false alarm ($P_{fa}$) and $k$ is the number of interfering targets found in the censoring step.

### 3.2 The F-ACOSD detection algorithm

The F-ACOSD algorithm starts by comparing sample $X(p+1)$ with the threshold $\hat{T}_{c0}$ given by

$$\hat{T}_{c0} = X(1)^{\hat{\alpha}_0} X(p)^{1-\hat{\alpha}_0}, \tag{7}$$

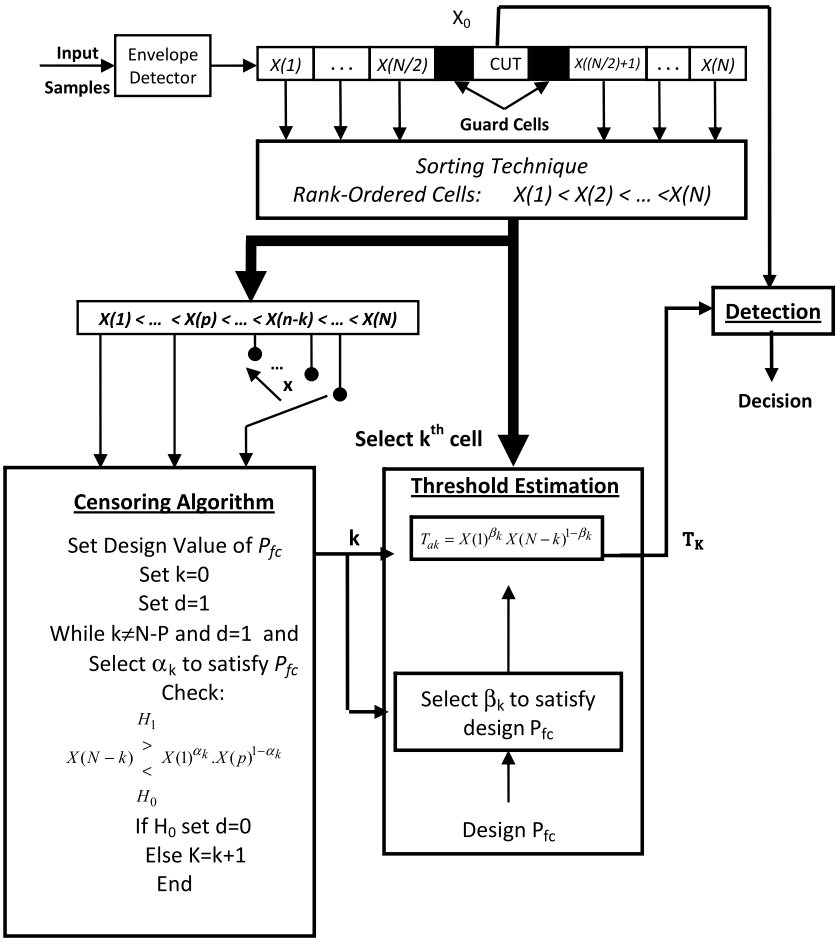


**Fig. 2** Block diagram of the B-ACOSD algorithm

where $\hat{\alpha}_0$ is a constant chosen to achieve the desired $(P_{fc})$ for the F-ACOSD algorithm. In contrast to the B-ACOSD algorithm, if $X(p+1) > \hat{T}_{c0}$, the algorithm decides that $X(p+1)$ is a return echo from an interfering target and terminates. If, by contrast, $X(p+1) < \hat{T}_{c0}$, the algorithm decides that the sample $X(p+1)$ corresponds to a clutter sample without interference, and then the detector compares the sample $X(p+2)$ with the threshold given by

$$\hat{T}_{c1} = X(1)^{\hat{\alpha}_1} X(p)^{1-\hat{\alpha}_1} \tag{8}$$

This comparison is performed to determine whether the sample corresponds to an interfering target or a clutter sample without interference. At the $(k+1)$th step, the sample $X(p+k+1)$ is compared with the threshold $\hat{T}_{ck}$, and a decision is made according to the test

$$X(p+k+1) \underset{H_0}{\overset{H_1}{\gtrless}} \hat{T}_{ck}; \quad 0 \le k < N-p \tag{9}$$

where

$$\hat{T}_{ck} = X(1)^{\hat{\alpha}_k} X(p+k)^{1-\hat{\alpha}_k} \tag{10}$$

$H_1$ represents the case in which $X(p+k+1)$; thus, the subsequent samples $X(p+k+2), \ldots, X(N)$ correspond to clutter samples with interference, whereas $H_0$ denotes the case in which $X(p+k+1)$ is a clutter sample without interference. The successive tests are repeated as long as hypothesis $H_0$ is declared true. The algorithm terminates when the cell under investigation is deemed non-homogeneous (i.e., clutter plus interference sample) or, in the extreme case, when the entire $N-p$ highest cells are tested, that is, $k = N-p$.

In the detection step, the CUT $X_0$ is compared with the threshold $\hat{T}_{ak}$ to determine whether a target is present according to

$$X_0 \underset{H_0}{\overset{H_1}{\gtrless}} \hat{T}_{ak}; \quad 0 \le k < N-p+1 \tag{11}$$

Hypothesis $H_1$ denotes the presence of a target in the test cell, whereas hypothesis $H_0$ denotes the absence of a target. In the F-ACOSD algorithm, the threshold $\hat{T}_{ak}$ is defined as

$$\hat{T}_{ak} = X(1)^{\hat{\beta}_k} X(p+k)^{1-\hat{\beta}_k} \tag{12}$$

where the value of $\beta_k$ is selected according to the design probability of false alarm $P_{f\alpha}$ for $k$ interfering targets found in the censoring step.

To eliminate noise, which corrupts the potential target, we considered the lognormal distribution that is close to the distribution for the desert environment [26] and calculated the associated statistical parameters, such as $\alpha_k$ and $\beta_k$, which were obtained using the B-ACOSD depending on the probability of a false alarm $P_{fa}$ and the window dimensions $p$ and $N$. These parameters are useful in calculating the adaptive threshold, which is a key element of the proposed algorithms.

Threshold selection is an important aspect of the algorithms [22]. The thresholds should be selected such that there is a low probability of hypothesis test error in a homogeneous environment. A Monte Carlo simulation comprising 500,000 independent runs was performed using computer simulation to obtain the threshold values, maintaining low values of $P_{fa}$ and $P_{fc}$ based on statistical analysis. Table 1 provides the threshold parameters $\alpha_k$ and $\beta_k$ obtained using the B-ACOSD with $P_{fa} = 0.001$ and $P_{fc} = 0.01$. Table 2 provides values of $\hat{\alpha}_k$ and $\hat{\beta}_k$ for the F-ACOSD with the same probabilities. These parameters are used in the proposed system implementation to simplify the adaptive threshold computation and improve target estimation. Furthermore, the proposed ACOSD algorithms are implemented and simulated to demonstrate functionality and check the correctness of the algorithm, including all components regardless of the timing analysis.

**Table 1** Threshold parameters for B-ACOSD ($P_{fc} = 0.001$ and $P_{fc} = 0.01$)

| $(N, p)$ | | $K$ | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| (16, 12) | $\alpha_k$ | 2.596 | 2.038 | 1.709 | 1.443 | – | – | – | – | – | – | – | – | – |
| | $\beta_k$ | 1.635 | 1.889 | 2.12 | 2.37 | 2.64 | – | – | – | – | – | – | – | – |
| (36, 24) | $\alpha_k$ | 2.538 | 2.154 | 1.953 | 1.812 | 1.7 | 1.601 | 1.523 | 1.443 | 1.369 | 1.3 | 1.225 | 1.153 | – |
| | $\beta_k$ | 1.35 | 1.465 | 1.566 | 1.65 | 1.73 | 1.8 | 1.87 | 1.94 | 2.02 | 2.1 | 2.18 | 2.265 | 2.345 |

**Table 2** Threshold parameters for F-ACOSD ($P_{fa} = 0.001$ and $P_{fc} = 0.01$)

| $(N, p)$ | | $K$ | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| (16, 12) | $\hat{\alpha}_k$ | 1.442 | 1.465 | 1.535 | 1.745 | – | – | – | – | – | – | – | – | – |
| | $\hat{\beta}_k$ | 2.64 | 2.37 | 2.12 | 1.889 | 1.635 | – | – | – | – | – | – | – | – |
| (36, 24) | $\hat{\alpha}_k$ | 1.15 | 1.152 | 1.154 | 1.158 | 1.16 | 1.167 | 1.174 | 1.191 | 1.21 | 1.264 | 1.311 | 1.467 | – |
| | $\hat{\beta}_k$ | 2.345 | 2.265 | 2.18 | 2.1 | 2.02 | 1.94 | 1.87 | 1.8 | 1.73 | 1.65 | 1.566 | 1.465 | 1.35 |

### 3.3 Nios II based ACOSD system development flow

Our high-resolution and real-time target detection requires critical timing, less than 500 ns, and a constant false alarm rate obtained by maintaining the desired values of the probability of false censoring $P_{f\alpha} = 10^{-3}$ and the probability of false censoring $P_{fc} = 10^{-1}$. To achieve the desired probabilities, the threshold parameters ($\alpha$ and $\beta$) are obtained from Monte-Carlo simulations and from a performance evaluation of the proposed target detector using Matlab without considering the timing constraints, which will be studied in the embedded system design process. All of these parameters should be considered as requirements in the system design. The main challenges associated with embedded systems are maintaining a minimal cycle time and reducing the system cost with increasing system requirements and complexities. Three alternatives can be explored, according to the requirements of the application, as follows:

- *Develop embedded design software:* In this design flow, the application is defined as software code embedded within FPGA-based hardware architecture, running on the integrated core processors. This solution requires a complete software development environment, such as the Nios II IDS environment provided by Altera. If the adequacy between the target architecture and embedded software is well designed to satisfy the system requirements with respect to the application, this solution becomes very interesting because it allows the virtual prototype of our application to be obtained within a reasonable time and at a reasonable cost.
- *Develop an Intellectual Property* (*IP*)*—based architecture* using VHDL or Verilog language at the RTL level and integrate this IP in the embedded system as a coprocessor or accelerator. This approach can be used when the previous solution cannot provide the required performance. The time to obtain the prototype is highly important, and the cost of the design is increased in comparison with that of the previous approach. A typical design example is presented in [27] in which the application is a real-time watermarking technique for video application.
- *Combine both the hardware and software implementation* for the same application and perform a co-simulation around the FPGA-based processor architecture. In this case, many solutions can be applied using glue logic components, a custom logic instruction approach, and accelerator components for critical functions. In [28], we presented a hardware/software implementation based on the integration of a custom instruction approach to implement the BACOSD technique for radar target detection. However, this implementation provided a delay close to 500 ns, which is the critical time for high-resolution target detection.

In the remainder of the paper, we consider the first approach for both FACOSD and BACOSD CFAR-based high-resolution target detection techniques as depicted in Fig. 3.

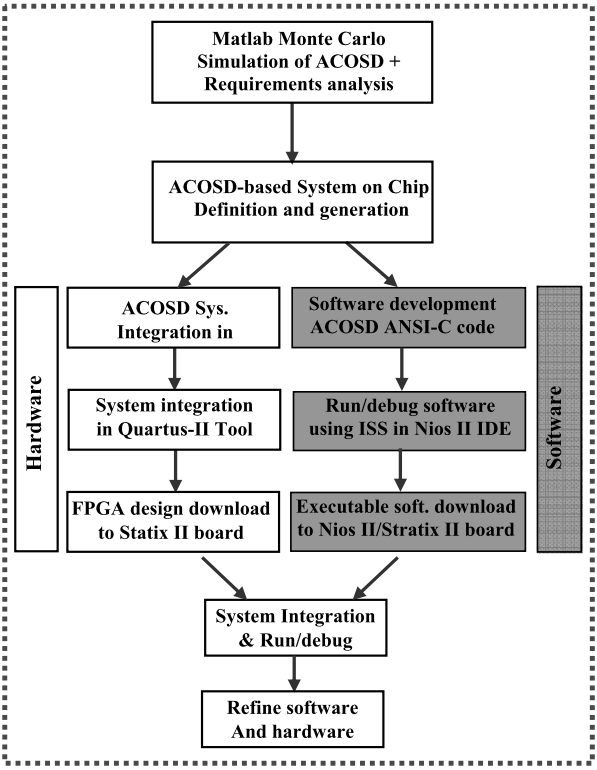


**Fig. 3** Proposed design flow for the ACOSD system

Our main contribution consists of providing an adequate combination of target architecture and the embedded design software to satisfy the system requirements.

The design and implementation of a real-time target detection system in the context of SoC architecture requires the consideration of several problems with respect to the following design flow:

- *Hardware design steps*: In this step, the embedded system-based hardware architecture is defined. It incorporates a fast version of the Nios II core processor with on-chip memories and a JTAG-UART interface interconnected using the Avalon fabric. A MicroC/OS real-time operating system is also selected and integrated within the Nios II core processor to execute the ANSI-C software code related to the proposed CFAR application.
- *Software design steps*: This step consists of the design of a pure software architecture using a high-level language (HLL). The target detection code is developed with the ANSI-C language and is run at first on the Instruction Set Simulator (ISS) of the Nios II core processor in the Nios II-IDE environment of Aletra. Once simulated and checked, the code is integrated on the FPGA code runs on the Nios II processor within the FPGA using a micro-C operating system, and real-time validation is performed.
- *System design step*: This step consists of the integration of both FPGA-based hardware architecture and software code within the same platform. The adequacy between the system architecture and target techniques is explored by operating many optimizations on the algorithm (sorting, look up, threshold computation) as well as the system architecture (cache optimization, memory organization) to meet the high-resolution and real-time requirements.

### 3.4 Implementation of the ACOSD-CFAR architecture on a Nios II development board

The software implementation of the CFAR algorithms can be achieved using microprocessors or DSPs, and the performance of such implementation depends on the complexity of the implemented CFAR techniques and the required performance according to the environment in which the detector will be used. However, a high-resolution detector in a non-homogeneous environment has a severe timing constraint where the false alarm rate is maintained under a fixed $P_{fa}$. Hardware/software integrated architecture with embedded core processors is required. In our case study, the Altera Nios II development board was considered for the implementation of the COSD-CFAR target detector. The core of the board is the Altera Stratix II EP2S60F672C3 FPGA. To implement the SoC CFAR-based architecture, we must first define the SoC architecture, which is the basis for our design implementation. This architecture is generated automatically using a system on a programmable chip (SoPC) development tool to integrate all IPs, the Altera mega-function, memory and interfaces, and HDL code with the appropriate test benches. For our application, the following components are required to build the CFAR-based SoC.

- A software Nios II core processor. We selected a powerful version of the Nios II/f core processor with a 32-bit data path, 2-kB data cache, and 4-kB instruction cache.
- JTAG UART interfaces with a simplified configuration, allowing target connection-downloaded software and possessing an interface for on-chip trace data.
- A timer for time-based system routines. A 32-bit timer is selected.
- A flash and SDRAM memory controller. This component is comprised of common flash interface-compliant (CFI) flash memory devices. According to the hardware platform, we selected the AMD29LV128M123R_Byte interface. Furthermore, an external memory controller was integrated into our SoC configuration.
- An Avalon tri-state bridge. The Avalon interface represents the system bus that interconnects all memories with the Nios II core processor. In our configuration, the Avalon MM tri-state bridge shares external system connections with the SRAM and flash memories.
- A phase-locked loop (PLL) clock to synchronize the system design.

Once all of the components are generated, each mapping phase can be obtained by auto-assigning addresses to generate the entire SoC architecture and carrying out pin assignments. Based on this configuration, the total architecture is synthesized using the predefined VHDL components to provide a feature called real-time ISP, which allows a device's configuration flash memory to be loaded with the proposed design.

### 3.5 A layered approach to software design development

Software development is a significant aspect of the proposed ACOSD design architecture. It consists of the HLL code intended for execution on a processor Nios II ISS. The Nios II ISS is a software model of an Instruction Set Architecture that is used to debug code before downloading it on a target board. It provides limited models of a few hardware peripherals (timer, UART, flash, and SDRAM on-chip memories).

Our conceptual software layered architectural approach suggests that programs can be structured as a series of three layers with a sequence of well-defined interfaces between the layers. This design effectively isolates each layer from the layers above and below it so that one can change the internal aspects of any layer without having to change any of the other layers in the program. The proposed layered architecture isolates the hardware/software components, enabling the organization shown in Fig. 4.

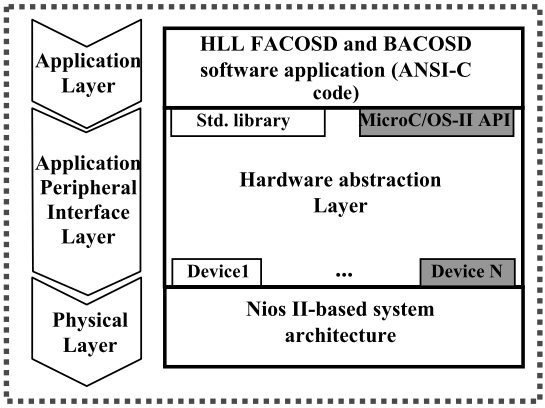


**Fig. 4** Layered representation of the ACOSD model for an embedded system architecture

Because this code is intended to run in real time, the ISS is equipped with a real-time operating system (MicroC/OS). Figure 4 presents a typical ACOSD-CFAR-based application, which is intended to run on a Nios II-based architecture integrating a real-time operating system.

- *Software application layer:* In this layer, we defined the ANSI-C codes for both FACOSD and BACOSD using the appropriate application programming interface, MicroC/OS II API, for software development. Many optimizations were considered to meet the real-time requirements.
- *Application Programming Interface Layer:* This layer, which abstracts details of the hardware complexity for software development, integrates three main parts, the specific API (MicroC/OS II), the communication interconnect (Avalon interconnect), the hardware communication abstraction, and the abstract interface (device) [29].
- *Hardware component layer*: In our design, this layer is based on the Nios II processor core, represented by its ISS, which is tailored to support the software application and its real-time operating system. This real-time operating system architecture has several objectives, one of which is to centralize the control of the limited hardware resources.

### 3.6 Block diagram of the architecture

The overall SoC design consists of five main modules, as shown in Fig. 5, including the Nios II processor, which is dedicated to the execution of the F-ACOSD and B-ACOSD CFAR algorithms, on-chip ROM input/ROM interface, output/RAM interface, and JTAG UART interface. All blocks are connected by an Avalon interface. This interface allows our system to interact easily with external devices, such as external memories or any other component capable of integration with the Avalon interface. The processor masters all communications between the hardware and executes the CFAR program using the MicroC/OS II operating system.

The SoC utilizes a Nios II soft-core processor. Nios II is a 32-bit embedded-processor architecture designed specifically for the Altera family of FPGAs. Nios II has key features, such as custom instructions and easy custom peripheral management. Nios II is offered in three different configurations, including Nios II/f (fast), Nios II/s (standard), and Nios II/e (economy). We chose to use the Nios II/f core because it is faster and able to meet the real-time constraints. The input signal from an envelope detector can be sent directly to the Avalon bus through an input interface or can first be stored in the 16-MB flash ROM

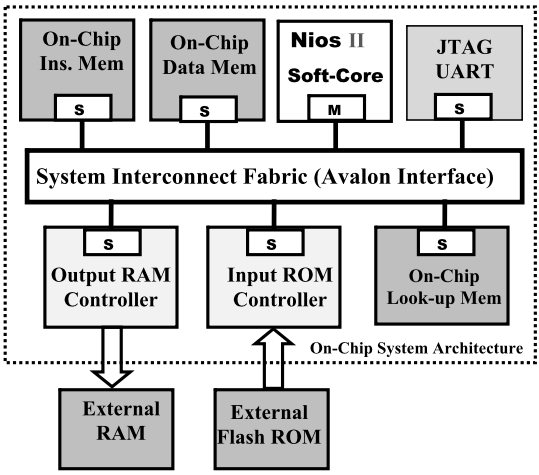


**Fig. 5** Block diagram of the implemented ACOSD target detector

provided by a development board and connected to the system through a flash ROM driver. The censoring results are stored in a 2-MB external SSRAM that is controlled by a SSRAM driver.

## 4 Experimental implementation and validation

The main objective in this section is to demonstrate the functionality of the hardware for the proposed F-ACOSD CFAR architecture. To this end, the complete architecture, including the associated input target data ROM and output target detection RAM, was built around a single FPGA chip, with a minimum set of function units and minimal features and cost. Specifically, the proposed CFAR processor was targeted for the Stratix FPGA chip (EP2S60ES device). The Stratix II development kit EP2S60 DSP was selected to prototype the proposed target detector. This board serves as a platform for use in high-performance digital signal processing applications. It is normally employed to design, verify, and evaluate systems prior to final stand-alone single-chip implementation. We used SoPC Builder, which is an integrated tool of the Quartus II tool that allows designers to build SoC architectures easily without the need to integrate the system manually. In our implementation, we considered the following configuration, including

- A Nios II/f core with a 2-kB data cache and a 2-kB instruction cache
- An internal ROM with controller
- An external flash memory of 2 MB with a controller
- JTAG UART adapter
- An Avalon interface
- PLL device operating at 100 MHz for clock generation

### 4.1 Design issues

For the software, we considered MicroC/OS II for the development language of the ACOSD CFAR algorithms using the Nios II integrated development environment. In the first trial implementation of F-ACOSD and B-ACOSD, we found that the computations of $T_{ck}$ and $T_{ak}$

are critical points, accounting for approximately 92 % of the total time (200 µs), which does not meet the delay requirement for the real-time execution of the ACOSD target detector. Only 8 % of the total time is devoted to the shifting and sorting operations. To reduce the required time of approximately 200 µs, we considered many design explorations related to critical components, such as the $T_{ck}$ and $T_{ak}$ threshold values, sorting module, and adaptive threshold computation component.

- *Optimization of the threshold computation:* In terms of hardware, computation of the exponential Eqs. (4), (6), and (10)–(12) is difficult and expensive, especially for a floating-point calculation. To reduce the hardware complexity and to decrease the computational time, Eqs. (4) and (6) are converted to logarithmic form as Eqs. (13) and (14), respectively.

$$\log T_{ck} = (1 - \alpha_k) \log X(1) + \alpha_k \log X(p), \tag{13}$$

$$\log T_{ak} = (1 - \beta_k) \log X(1) + \beta_k \log X(N - k). \tag{14}$$

In the proposed form, exponential computations become simple multiplication operations, and multiplication becomes an addition operation. After implementation, the solution appears inappropriate because at each point in time, we must calculate the logarithm of the cell under test and the logarithm of all cells $X(p + k + 1)$; therefore, we propose to use a look-up table (LUT), as defined below.

- *Implementation of a LUT:* For Eqs. (13) and (14), it is difficult and time-consuming to calculate the logarithms for all cell values directly. To circumvent this problem, a LUT is built. The LUT contains the range of a lognormal distribution with $\mu = 1$ and $\sigma = 1.1$ and is based on the real radar input measurements given in [22]. The proposed LUT supports up to 2,000 values, which allows resolution following the change in the number representation to logarithmic form. For example, with 64-kB ROM, we achieve a resolution of approximately 0.0312, and using 32-kB on-chip ROM, which is available for our Stratix II prototyping board, this resolution is reduced to 0.0624.
- *Approximation of the logarithmic function:* To optimize the available size of the memory used for the LUT and to achieve acceptable resolution in calculating the logarithmic function, we approximated the function by computing its values with steps of 0.03 for the range of $X$ from one to 10, 0.1 for the range of $X$ from 10 to 100, and one for the range of $X$ from 100 to 2,000.

Figure 6 presents the interpolated logarithmic function and original logarithmic function, which differ by approximately $10^{-3}$. This technique allows us to obtain the fixed resolution with a small on-chip memory limited to 32 kB. Additionally, we considered a fixed-point COSD CFAR simulation to minimize the complexity of the architecture because the floating point does not provide any significant information for the given resolution. To implement the logarithmic function required for the threshold calculation, we considered two on-chip memories, each with a M4K block [30]. In the first configuration, we put all logarithmic values for all inputs up to 400, where the second configuration provides the logarithmic value for the remaining values of $X(i)$ up to 2,000. As depicted in Fig. 7, the pseudo-code presents the memory access to these look up tables with different base addresses (06440000h and 06448000h) using the Nios II Hardware Abstraction Layer macro, IORD_16DIRECT, to read access at the location with base + offset by bypassing the data cache.

- *Optimization of the sorting module:* As reported in [24], the even-odd bubble-sorting module yields a delay that is proportional to the number of cells in the tape delay memory

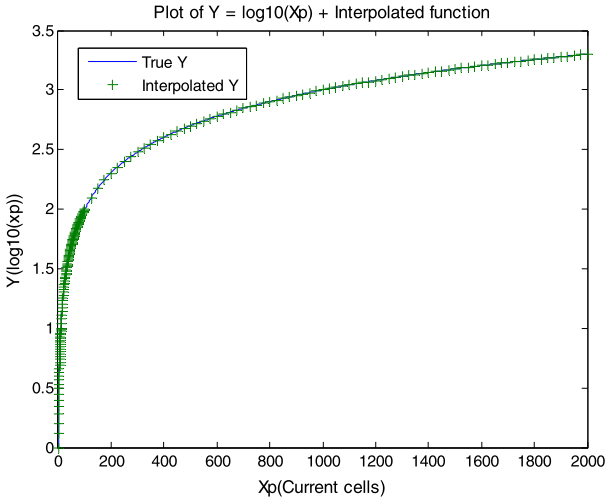


**Fig. 6** Approximation of the logarithmic function for the threshold calculation

```
alt_u16 log_val(alt_u16 x)
{
 if(x<=400)
      return (IORD_16DIRECT (0x06440000, x*80));
         else
      return (IORD_16DIRECT (0x06448000, (x-400)*2));
}
```

**Fig. 7** ANSI-C pseudo code of the logarithmic calculation running on a Nios II core processor

**Table 3** Timing results of the ACOSD CFAR architecture

| ACOSD implementation | B-ACOSD | F-ACOSD |
|---|---|---|
| Direct Implementation | 200 µs | 200 µs |
| Using the cordic method | $5.10^6$ µs | $5.10^6$ µs |
| With the logarithmic function | 20 µs | 20 µs |
| Using the logarithmic function interpolation | 4.3 µs | 4.3 µs |
| Using LUT optimization | 0.7 µs | 0.7 µs |
| After sorting optimization | 0.49 µs | 0.49 µs |
| After cache memory optimization | 0.11 µs | 0.11 µs |

(16 clock cycles for the proposed architecture). We carried out an optimization procedure to reduce this delay to two clock cycles using a sorted list and inserting the new cell in the appropriate location using pointers.

- *Cache optimization:* To accelerate execution, we increased the cache memory size from 8 to 64 kB and found that the total delay was reduced to 0.11 µs, which is a good result for high-resolution target detection. Table 3 summarizes the timing results of the implementation of the ACOSD CFAR algorithms.

**Table 4** The hardware implementation results

| Allocated resources | Percentage of occupation |
|---|---|
| Logic gates | 20 % |
| Total register | 5,810 |
| Total pins | 44 % |
| Total blocks memory | 76 % |
| DSP blocks | 3 % |
| Total PLL | 17 % |

### 4.2 Design results and resource usage

To evaluate our embedded COSD-CFAR technique, we considered a software code for the overall algorithm after carrying out optimization over the Nios II processor; the software is downloaded to the FPGA. The core coprocessor is responsible for delivery to all cells in the architecture and for sending the results back to an interactive window via the JTAG UART interface connected to the Avalon interconnect. The synthesis results for the overall CFAR architecture are presented in Table 4. The memory requirement for our architecture is approximately 76 %, where only 20 % of the logic gates have been used. Furthermore, we have used 3 % of the DSP blocks and 44 % of the input/output blocks with 5,810 registers. We have only checked the design using 2,000 cells because of the memory limitation. If we increase this number, we cannot complete the routing of the design within the Stratix II FPGA device; thus, we are working within the limited memory resources.

The FPGA implementation result for the SoC architecture with $N = 16$ and $p = 12$ illustrates that the Nios processor can achieve a maximum operating frequency of 100 MHz. After many optimizations, the processing time to perform a single run is 0.11 s. This processing time is below the real-time requirement of 0.5 s [24]. For demonstration purposes, the default configuration of the processor is 16-bit data samples, 16 reference cells, and two guard cells. The processor was tested and verified by generating 256 data samples drawn from a lognormal distribution. The dataset was downloaded to $16 \times 256$ ROM. The output array of the target presence indication result was saved in $1 \times 256$ RAM.

During the validation phase, we checked both the timing features and performance of the proposed system in terms of false alarm and false censoring. The clutter follows a lognormal distribution, where the pulses are distributed according to the Rayleigh shape. These assumptions are performed to emulate the desert environment. The noise of the signal-to-clutter ratio (SCR) varies from 10 to 25 dB. No degradation is registered in this respect for all signals for which the SCR exceeds 10 dB. However, if the SNR is decreased by less than 10 dB, the $P_{fa}$ exceeds $10^{-3}$. This degradation does not affect our target detector because the typical SCR exceeds 15 dB in practice, and in this case, our algorithm performs well, as depicted in Fig. 8.

In order to verify that the approximations carried out on the logarithmic function do not affect the required performance of the proposed algorithms in terms of false alarm, we measured the false alarm rate directly from the embedded system. The following figure presents the results of these measurements displayed by the Nios-II core processor for different values of SNR (20 and 30 dB). We confirm that the $P_{fa}$ is quite below $10^{-3}$ required for high resolution target detection and the approximation used before is well justified.

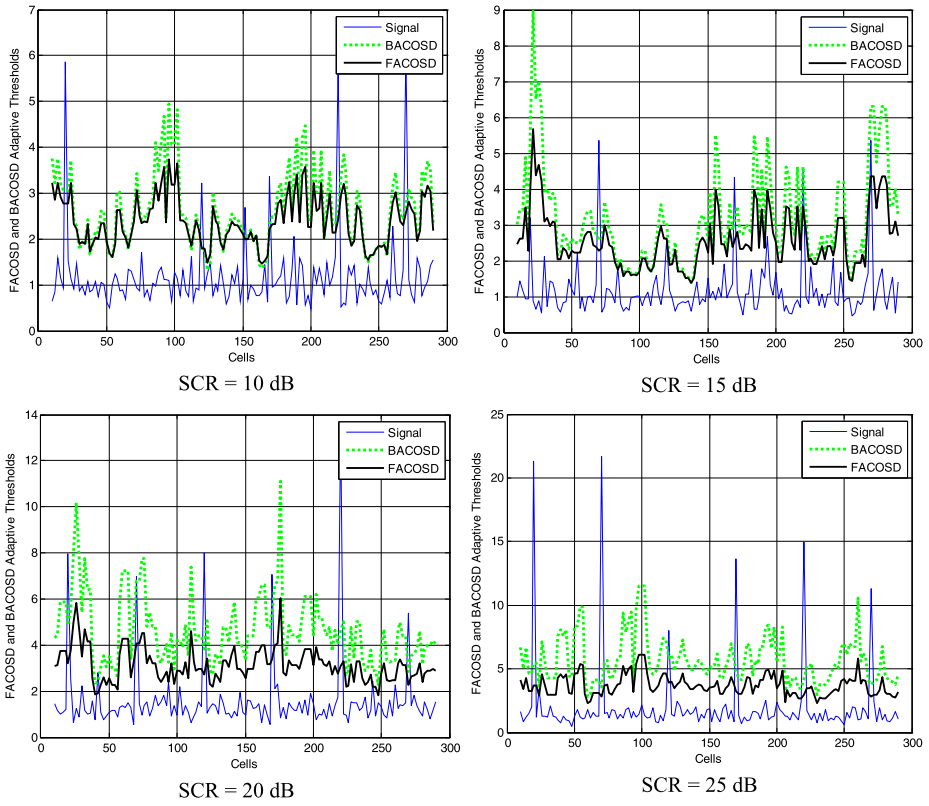


**Fig. 8** Threshold estimation using the FPGA-based ACOSD embedded system architecture

## 5 Conclusion

This paper reported the hardware implementation of an ACOSD CFAR target detector for lognormal clutter based on the ACOSD algorithm. This proposed SoC architecture has the advantages of being simple and fast and requiring low development cost. The performance of the prototype hardware setup demonstrated the concept of the proposed CFAR processor. This programmable hardware, with its significant improvement in processing speed, will make it possible to increase the resolution of radar range cells and will open new avenues for further useful real-time applications.

The proposed FPGA implementation integrates a Nios II core processor within the FPGA with internal memories and controllers using the Avalon switch fabric. The proposed architecture allows for the detection of each cell under consideration within a delay of 0.11 μs, which meets the real-time requirement of 0.5 s. The total processing delay for the 256 cells given by the Stratix II development Kit (EP2S60ES device) is approximately 0.028 ms for a configuration using 16 reference cells and two guard cells. Furthermore, the required resources are less than the available FPGA resources with significant use of the internal memories. If we consider 32 cells, for example, we must optimize the sorting module and computation of the logarithmic function to fit the overall architecture within the same platform. We checked the false alarm rate for different values of the SCR (varying from 10 to 25 dB) on the proposed embedded system as shown in the Fig. 9, and we found results that were con-

```
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

**************************************************************
* ********************************************************** *
* *    Expiremental test on the board: Stratix II of Altera  * *
* *                    Frequency= 100 Mhz                    * *
* ********************************************************** *
* *                     Number of Cells=1000000                *
* *                     Number of target=10000                 *
* ********************************************************** *
* *                         F/B-ACOSD-CFAR                   * *
  * ======================================================== * *
* *                 Signal to clutter ratio   SCR=20 db      * *
* * -------------------------------------------------------- * *
* *   Processing time for each cell=0.11 microsecond         * *
* *            Number of detected targets=9998               * *
* *         Measured False Alarm rate Pfa=0.0002             * *
  * ======================================================== * *
* *                Signal to clutter ratio   SCR=30 db       * *
* *  ------------------------------------------------------- * *
* *   Processing time for each cell=0.11 microsecond         * *
* *            Number of detected targets =10000             * *
* *         Measured False Alarm rate Pfa=0.00               * *
* * -------------------------------------------------------- * *
* ********************************************************** *
* *                              END                         * *
* ********************************************************** *
**************************************************************
```

**Fig. 9** Performance evaluation of the Real-time Target ACOSD detectors

sistent with those obtained by Monte Carlo simulations. A small degradation was registered when the SCR was less than 10 dB. However, this result did not affect the performance of our system because the SCR is greater than 10 dB in practice. For future work, other types of CFAR algorithms, such as the Automatic Censored Cell Averaging Detector (ACCAD) CFAR detector, can be implemented within the same SoC to realize a generic CFAR detector architecture for different algorithms.

**Acknowledgements** The work reported in this paper was supported by the National Plan for Science and Technology (NPST) at King Saud University (project number: ADV-170-2-08).

## References

1. Barkat M (2005) Signal detection and estimation. Artech House, Norwood
2. Blake S (1982) OS-CFAR theory for multiple target and nonuniform clutter. IEEE Trans Aerosp Electron Syst 24(6):785–790
3. Rohling H (1983) Radar CFAR thresholding in clutter and multiple target situations. IEEE Trans Aerosp Electron Syst 19(4):608–621
4. Meziani HA, Soltani F (2006) Performance analysis of some CFAR detectors in homogenous and non-homogenous Pearson-distributed clutter. Signal Process 86:2115–2122
5. Larouissi T, Barkat M (2006) Performance analysis of order-statistic CFAR detectors in time diversity systems for partially correlated chi-square targets and multiple target situations. Signal Process 86(7):1617–1631

6. Mezache A, Soltani F (2007) A novel threshold optimization of ML-CFAR detector in Weibull clutter using fuzzy-neural networks. Signal Process 87:2100–2110
7. Khalighi MA, Bastani MH (2000) Adaptive CFAR processor for nonhomogenous environment. IEEE Trans Aerosp Electron Syst 36(3):889–897
8. Saarnisaari H, Henttu P, Juntti M (2005) Iterative multidimensional impulse detectors for communications based on the classical diagnostic methods. IEEE Trans Commun 53(3):395–398
9. Alshebeili S, Alhumaidi SM, Obied AM, Seddiq YM (2009) FPGA-based implementation of a CFAR processor using Batcher's sort and LUT arithmetic. In: 4th international design and test workshop (IDT), Riyadh-KSA, pp 1–6
10. Finn HM, Johnson RS (1968) Adaptive detection mode with threshold control as a function of spatially sampled clutter level estimates. RCA Rev 29:414–463
11. Sawyers JH, Hansen VG (1980) Detectability loss due to greatest of selection in a cell-averaging CFAR. IEEE Trans Aerosp Electron Syst 16:115–118
12. Weiss M (1982) Analysis of some modified cell-averaging CFAR processors in multiple target situations. IEEE Trans Aerosp Electron Syst 15(1):102–114
13. Barkat M, Himonas SD, Varshney PK (1989) CFAR detection for multiple target situations. IEE Proc, F, Radar Signal Process 136(5):193–210
14. Ravid R, Levanon N (1992) Maximum-likelihood CFAR for Weibull background. IEE Proc, F, Radar Signal Process 139(3):256–264
15. Anastassopoulos V, Lampropoulos G (1995) Optimal CFAR detection in Weibull clutter. IEEE Trans Aerosp Electron Syst 31(1):52–64
16. Torres C, Lopez S, Cumplido R (2004) A configurable FPGA-based hardware architecture for adaptive processing of noisy signals for target detection based on constant false alarm rate (CFAR) algorithms. In: Global signal processing conference, Santa Clara, CA, pp 214–218
17. Bencheikh ML, Magaz B (2008) An efficient FPGA implementation of the OS-CFAR processor. In: International Radar symposium, Wroclaw, pp 1–4
18. Cumplido R, Uribe C, Del Campo F, Perez R (2010) A versatile hardware architecture for a constant false alarm rat processor based on alinear insertion sorter. Digit Signal Process 20:1733–1747
19. Ali JK, Yassen ZT, Saed TR (2007) An FPGA-based implementation of CA-CFAR processor. Asian J Inf Technol 6(4):511–514
20. Alsuwailem AM, Alshebeili SA, Alamar M (2008) Design and implementation of a configurable real-time FPGA-based TM-CFAR processor for radar target detection. J Act Passiv Electron Devices 3(3–4):241–256
21. Detlefsen J, Siart U, Buchlert J, Wagner M, Winkler V (2004) FPGA-based signal processing of an automotive radar sensor. In: First European Radar conference, Amsterdam, pp 245–248
22. Almarshad MN, Barkat M, Alshebeili SA (2007) A Monte Carlo simulation for two novel automatic censoring techniques of radar interfering targets in log-normal clutter. Signal Process 88(3):719–732
23. Alsuwailem AM, Alhowaish MH, Alshebeili SA, Qasim SM (2009) Field programmable gate array-based design and realization of automatic censored cell averaging constant false alarm rate detector based on ordered data variability. IET Circuits Devices Syst 3(1):12–21
24. Djemal R (2010) A real-time FPGA-based implementation of target detection technique in non-homogenous environment. In: Design and technology of integrated system in nanoscale era (DTIS), Hammamet, Tunisia, pp 1–6
25. Rosyadi I, Djemal R, Alshebeili S (2009) Design and implementation of real-time automatic censoring system on chip for radar detection. In: World academic of science, engineering and technology (WASET), Penang, Malaysia, pp 318–324
26. Winter EM, Schlangen MJ, Hendrickson CR Comparisons of target detection in clutter using data from the 1993 FOPEN experiments (Technical report). Naval Command, Control and Ocean Surveillance Center (NCCOSC), RDT&E Division, San Diego, CA 92152-5001
27. Karmani S, Djemal R, Tourki R (2007) A blind watermarking algorithm implementation for digital images and video. Int J Soft Comput 2(2):292–301
28. Djemal R, Belwafi K, Kaaniche W, Alshbeili SA (2011) An FPGA-based implementation of HW/SW architecture for CFAR radar target detector. In: International conference on microelectronics (ICM), Tunisia, pp 1–6
29. Cesario W, Gauthier L, Lyonnard D, Nicolescu G, Jerraya AA (2004) Object-based hardware/software component interconnection model for interface design in system-on-a-chip circuits. J Syst Softw 70:229–244
30. Atera Corporation (2007) Stratix II architecture, Stratix II device family— Data sheet SII1002-4.3, May 2007