International Journal of Engineering and Technology Innovation, vol. 9, no. 4, 2019, pp. 314-326

Simple Communication Interface for a Radar Detector in the Moments Space

Camilo Guillén*, Greysi Casas, David Frómeta, Nelson Chávez

Department of Telecommunication and Telematics, Technological University of Havana
José Antonio Echeverría, CUJAE, La Habana, Cuba

Received 16 February 2019; received in revised form March 16 2019; accepted 29 April 2019

Abstract

The method of Radar Target Detection by Analysis and Statistical Classification of the Cellular Emission (DRACEC) can be functionally divided into three fundamental stages: Acquisition, Adaptation, and Detection. During Acquisition it is required to continuously exchange information between the hardware in FPGA and the software, without affecting the performance of the latter. This paper proposes the simplest communication interface that satisfies these requirements when DRACEC is applied to a small searching window. The solution is implemented on the serial port and ensures that samples stored in FPGA are available at the computer for the remaining processing stages. One fundamental characteristic of the proposal is a protocol designed to control the communication flow, which is implemented through a dedicated program thread. This allows software performance to not deteriorate during communication and lays the foundations for using multithreading techniques to develop the stages of DRACEC.

Keywords: DRACEC method, radar, communication interface, UART

1. Introduction

The novel technique known as DRACEC [1-2] improve the detection probability for low signal-to-noise ratio environments [1]. Furthermore, it is applicable to scenarios where the statistical distributions of the random variables used for decision making are unknown or difficult to estimate. DRACEC is based on the use of large-size samples of parameters such as the amplitude, frequency, polarization, etc., of the signals scattered by the resolution cells that compose the surveillance region. As a fundamental characteristic, a classification vector (pattern) is obtained, whose components (features) are a determined number of moments of the selected signal parameters, enabling the classification of the resolution cells in one of two classes: background (target absent) or anomaly (target present). Since the decision is carried out through a finite set of moments, DRACEC is usually referred to as a detection method in the moment's space, which differentiates it from traditional techniques. In addition, the amplitude of the video signal at the radar receiver is the most commonly selected parameter, due to the simplicity in obtaining measurements.

Taking into account that the random variables used for detection are a set of normal-distributed moments [1-2], DRACEC matches the environmental conditions better than methods based on processing the signal parameters, which assume models for their distributions. Therefore, by means of DRACEC, the problem of the a priori indetermination is significantly reduced. Another distinctive characteristic of DRACEC is the multidimensional approach to detection and the concept of anomaly-to-background ratio instead of the classical signal-to-noise ratio. This gives a higher detection and classification

* Corresponding author. E-mail address: camilo.gs@tele.cujae.edu.cu
Tel.: +53-7-266-33-34

capacity, which makes it possible to achieve a distinction among targets attending to the statistical behavior of the eco-signals. The method is suitable as a complementary technique in systems that use others up-to-date detection methods and could be employed only in certain areas of the surveillance region, where it is required to detect with a high probability low observable target.

DRACEC is functionally divided into three stages: Acquisition, Adaptation, and Detection, which could be implemented in hardware or software depending on their characteristics and the available technology [3-4]. The temporary requirements for samples acquisition involve dedicated hardware, usually FPGA (Field Programmable Gate Array) [5]. On the other hand, the mathematical complexity of the Adaptation and Detection algorithms and the necessity of displaying results to the user, demand the development of software with Graphical User Interface (GUI) in a computer.

The Acquisition stage requires uninterrupted information exchange between hardware and software. For example, once the FPGA is configured with the operating modes chosen by the user, the samples are continuously transmitted to the computer. In this case, it would be unfeasible to dedicate a software function to communication, since the application would be "attending" it continuously, at the cost of not performing well the remaining tasks and affecting the overall system performance. All these statements manifest the need for an interface for achieving an uninterrupted communication between the FPGA and the computer, which allows acquiring the samples for further processing. The communication through this interface should not deteriorate the performance of the remaining stages of DRACEC and should be as simple as possible.

This paper presents a solution to the above-mentioned problem, by implementing the hardware and software elements that allow computer-FPGA communication through the simplest interface: the serial port. From the software point of view, a GUI developed in Qt [6-7] is proposed, whose main objectives are to configure the hardware and receive the samples previously stored in FPGA. On the other hand, the hardware components are implemented in a development kit of Altera [8-9], dedicated to control the samples reading and its transmission to the computer. As a fundamental characteristic, a protocol is designed to control the communication flow, which is implemented by means of a dedicated program thread. The latter allows software performance to not deteriorate during communication and lays the foundation for developing the remaining stages of DRACEC using multi-threading techniques. In the next section, the use of the serial port as a communication interface is justified. Subsequently, its software and hardware elements are presented and finally, the interface is verified through the obtained results.

2. Selection of the Communication Interface

Currently, the use of DRACEC is limited to a certain number of resolution cells, particularly those where small targets are to be detected, without affecting the radar performance during surveillance [1]. Therefore, this method should be applied only over a small searching window when compared to the surveillance region. Taking the Navi Radar 4000 (NR-4000) [10] as a reference radar, a typical configuration of the searching window would be 8 angular sectors and 64 range rings, for a maximum of 512 resolution cells [3]. Table 1 shows some features of the NR-4000 that determine the formation of the searching window and the amount of data to be sent to the computer.

The four operating modes are selected according to the range scale in which the radar work [10], establishing different pulses durations (with the consequent change in range resolution) and repetition frequencies. Also of interest are the rotation speed of the antenna, the half power beam width [11] and the number of received pulses. The first feature imposes an upper limit on the time required to send the acquired samples to the computer, and therefore, directly affects the choice of the communication interface. The half power beam width establishes the angular resolution of the radar, while the number of received pulses represents the number of echoes per resolution cell for each angular sector and is determined by $\rho = \Delta\alpha \cdot f_r / 6 \cdot \omega$ [12], where the meaning for each symbol is in the last column of table 1.

Table 1 NR-4000 Scanner Technical Specifications

Feature	Value				Symbol
Half Power Beam Width (azimuth)	1.8^o				$\Delta\alpha$
Antenna Rotation Speed [RPM]	22				ω
Operating Modes	1	2	3	4	-
Pulse Duration [ns]	80	200	400	800	τ
Range Resolution [m]	12	30	60	120	ΔR
Pulse Repetition Frequency [Hz]	2000	1000	750	500	f_r
Received Pulses	27	13	10	6	ρ
Samples per Antenna Revolution	13824	6656	5120	3072	-
Bytes per Antenna Revolution	27648	13312	10240	6144	-

The maximum amount of data to be sent to the computer after each antenna revolution is indicated in the last two rows of the table. The operation mode 1 is the most critical in this sense, since for any angular sector 27 pulses are received per resolution cell, and after a full scan of the searching window (with a maximum of 512 cells) a maximum of 13824 samples will be available. Although the acquisition system is external to the communication interface [3, 13], it should be mentioned that it has an Analog-To-Digital Converter (ADC) of 14 bits [3, 14], hence each sample is represented by two bytes and the maximum amount of data available per revolution will be 27648 bytes.

Another element to keep in mind is the way of sampling required by DRACEC since the accumulated samples must be statistically independent. This is essential if we want a set of normal-distributed moments for decision making, which is a fundamental advantage of DRACEC as a parametric method [1]. Each sample is taken from the random process $\xi_k(t)$, which describes the amplitude of the video signal for the resolution cell *k* according to

$$\xi_k(t)\Big|_{t=i\cdot T_{360}+j\cdot T_R+t_k} \qquad i = 0,\ldots,M-1;\ j = 0,\ldots,\rho \tag{1}$$

where T_{360} is the duration of one antenna revolution, $T_R = 1/f_r$ and $t_k = 2\cdot R_k/c$, where R_k are the cell's range and *c* is the light speed. In Eq. (1) *M* stands for the number of antenna revolutions necessary to obtain a suitable set of samples for moments computation, which in statistical argot is the sample-size [1]. For each revolution, ρ samples per cell are acquired, which are correlated and therefore are not useful for computing the moments directly. In marine clutter environments, decorrelation is achieved at intervals greater than 10 ms [15-16], but the maximum value of T_R reached with the NR-4000 is only 2 ms, for operating mode 4. So the most feasible way to accumulate decorrelated samples without applying mechanical or other modifications to the NR-4000 is through consecutive 360^o sweeps. This offers the possibility to compute ρ moments for each resolution cell after *M* antenna revolutions [3], associating the samples in a sweep-to-sweep way, instead of from pulse-to-pulse.

The most common computer-FPGA communication interfaces in the radar field are UART, USB, and Ethernet. Among the representative examples, the authors of [17] implement a radar controller in FPGA, which sends commands to the transmitter and receiver modules via serial port in order to achieve beam steering of the radar antenna. In [18] are detailed the primary systems of ocean observatories, which have sensors that communicate through a backup RS-232 serial interface. On the other hand, [19] describes a protocol used to reliably transmit a time-critical synchronization pulse, as well as multi-channel bidirectional RS-232 data, over a single low-quality twisted pair cable. The work of [20] describes the interactive software and firmware for a ground-penetrating radar that uses the Universal Serial Bus (USB) for communication between the computer and radar peripherals. This USB interface was configured as a virtual serial port with typical connection settings. The paper [21] address the modernization of an old analog radar through a dedicated library over the USB port, which

facilitates the exchange of the antenna direction and other parameters. In [22] a system based on a 1 Gbit-Ethernet can achieve high-speed data transfers to a computer functioning as a multi-beam forming processing unit, while in [23] it is proposed a GUI that establishes the control specifications for a ground-penetrating radar. This interface uses TCP/IP link for communication with hardware.

UART is generally used to transmit small amounts of data [17-19], as well as to exchange commands for configuration and control. On the other hand, both USB [20-21] and Ethernet [22-23] allow exchanging large volumes of data in quasi-real time, and are commonly used in the so-called radar-PC [20, 23], modern digital radar systems [22] or to digitize analog radars to extend their useful life [21]. As it is known, Ethernet and USB transmit at a considerably higher speed than UART, although they increase the complexity of the necessary controllers. However, the required time to accumulate a new set of independent samples will be the same for all interfaces, since it only depends on the antenna rotation speed. Each revolution takes 2.73 seconds, which establishes the maximum allowable time for transmission of the acquired samples before a new acquisition begins. So it does not matter how quickly you can transmit during a revolution if you must wait 2.73 seconds to send a new set to the computer [3].

Taking advantage of these reasons, any of the above-mentioned interfaces allow transmitting the maximum of 27648 bytes (corresponding to one set of samples for the critical case in mode 1) in less than 2.73 seconds. So the chosen communication interface is UART since it is the simplest to implement and satisfies the needs of DRACEC when using the NR-4000 and a small searching window. The software elements of the designed interface will be treated below.

3. Software Elements

The proposed software takes advantage of the parallel processing capabilities of current computers and consists of two program threads: the main thread to handle the GUI and a thread dedicated to communication with FPGA. As mentioned before, the software was developed using Qt due to the facilities of this platform for implementing high-performance GUI [7, 24-25]. In addition, the classes offered by Qt are used for serial port communication (QSerialPort) and the work with multiple program threads (QThread). Next, the GUI is described in order to contribute to the understanding of the protocol and the communication thread.

3.1. Graphical user interface

The mains objectives of the GUI are to select the searching window, to establish the radar operating mode and the display of user-useful information. Fig. 1 shows the GUI, which has two tabs: one to configure the acquisition specifications and another to represent the acquired samples. The second tab is only for system verification, so the emphasis will be on the elements in the first one ("Configuration").

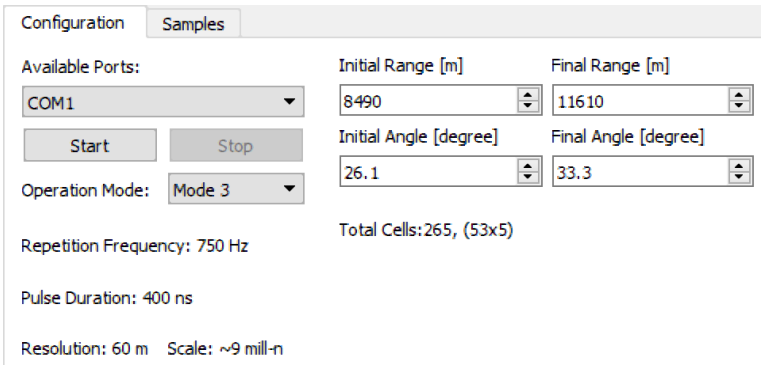

Fig. 1 Graphical user interface

The four spin-boxes identified under the labels "Initial Range", "Final Range", "Initial Angle", and "Final Angle", configure the searching window. The default variation of the initial and final ranges depends on the selected mode, while the

angular sectors remain unchanged unless the user modifies them directly. The searching window by default contains the maximum of 512 cells, distributed in 64 range rings and 8 angular sectors. However, the figure shows an arbitrary configuration of 53 rings and 5 sectors, hence the 53*5 notation used in the informative label about the total cells. The user can vary the value of each spin box to locate the searching window between the scale limits and the 360º. The increase allowed for the rings is equal to the range resolution, which depends on the operating mode, while the sectors increase is always 1.8°.

The spin-boxes values that establish the limits of the searching window are not directly useful for the acquisition system in FPGA since these values are used as indexes for angle and range counters in hardware and therefore must be positive integers. The indexes for the rings need 2 bytes since they can take any value between 0 and 499 [3]. Each index is determined through Eq. (2), which uses the number from the spin-box, as well as the minimum range ($R_{\min} = 2.5 \cdot \Delta R$) and resolution (ΔR)

$$R_m = \frac{\text{spinBox.value()} - R_{\min}}{\Delta R} \qquad m = i, f \tag{2}$$

where $m = i$ for the initial ring and $m = f$ for the final ring.

On the other hand, the indexes for the initial and final angular sectors will take a value between 0 and 199 [3], therefore a single byte is enough. This value will be given by Eq. (3), which takes into account the number entered by the user through the spin-box, the minimum angle of 0.9º and the resolution by the azimuth of 1.8º.

$$A_m = \frac{\text{spinBox.value()} - 0.9}{1.8} \qquad m = i, f \tag{3}$$

Returning to the example of Fig. 1, a total of 265 cells were selected, distributed in 5 angular sectors and 53 range rings. After evaluating Eqs. (2) and (3), the indices $R_i = 139$ and $R_f = 191$ are obtained for the initial and final rings, while those corresponding to the initial and final sectors are $A_i = 14$ and $A_f = 18$. By means of these indexes, the hardware system will define precisely the sampling intervals of the video signal [3].

Finally, the "Start" button gathers the specifications and prepare the initial configuration frame. This frame starts the communication with the hardware through the selected serial port. The confirmation of the initial frame and the protocol will be treated in the next section. Both software and hardware will remain running continuously until the user presses the "Stop" button, which indicates that the system must be restored to its initial state. The normal operation could also be stopped due to communication errors, which will be referred to next.

3.2. Communication protocol

In order to achieve communication between computer and FPGA, a set of rules must be established to organize the information exchange, hereinafter referred to as a protocol [17, 19, 26]. Transmission by serial port consumes considerable time, although it does not represent the major system delay, which is associated with the accumulation of statistically independent samples [3]. For this reason, the protocol must satisfy the temporary requirements of DRACEC, and be simple, so as not to cause unnecessary delays in the configuration and information transmission. It is emphasized that none of the stop-and-wait schemes [27] commonly used by UART interfaces are required since the computer and the FPGA will not transmit data at the same time (except for a case that will be analyzed below). Furthermore, none of the components involved in communication (computer and FPGA) will exceed the other in speed.

The proposed protocol is based on the 5 bytes with the specific meaning shown in Table 2, represented by hexadecimal format. The first three bytes serve as delimiters between the different data types of the initial configuration frame, while the last two contribute to control the information flow. All communication is made taking into account the format of 8 data bits, 1 stop bit, 1 start bit and speed of 115200 bit per seconds, following the RS-232 standard [28].

Table 2. Commands used in the protocol

Command	Meaning
0xAA	Searching Window Beginning
0xBB	Mode Beginning
0xCC	Total Samples Beginning
0xDD	Send Samples Request
0xEE	End of Communication

The communication is established by an initial frame of 11 bytes arranged as shown in Fig. 2. The bytes marked in red color contain delimiters characters. Between 0xAA and 0xBB it is established the searching window through the index of Eqs. (2) and (3). The eighth byte, denoted as mode, may take the values 0x00, 0x01, 0x02 or 0x03, to establish the radar operating mode.

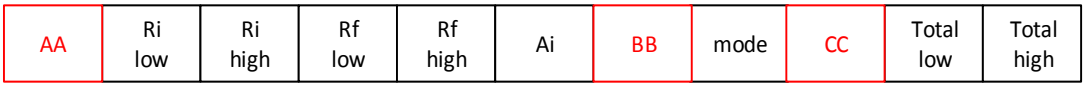

Fig. 2 Bytes of the initial configuration frame

On the other hand, the last two bytes define the number of addresses to be read from memory, which stores the samples. This number is determined by Eq. (4), and it is transmitted to the FPGA in order to free the hardware of this computation and simplify it. Note that only the index of the initial angular sector is included in the frame since only this and the total number of samples are required to acquire the desired set.

$$Total = \left[\left(R_f - R_i + 1\right)\cdot\left(A_f - A_i + 1\right)\right]\cdot\rho \tag{4}$$

The typical flow of communication is shown in Fig. 3. Once the FPGA receives the initial configuration frame, it sends back the same 11 bytes, as acknowledgment (ACK). In the computer, each byte of the ACK is compared with the corresponding one of the initial frames. If at least one is different, the user is notified and the byte 0xEE is sent to the FPGA, returning the system to its initial condition. The above is also done if no ACK is received by the computer at a previously established time. In case the ACK is correct, the computer remains "listening" to the port, waiting to receive the samples.

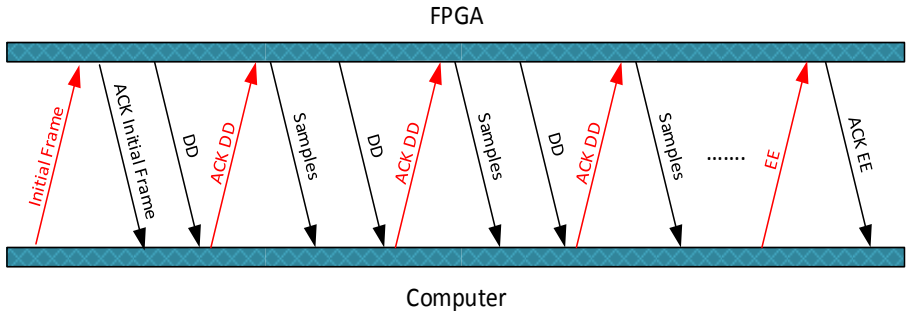

Fig. 3 The communication flow between computer and FPGA

Once the samples are available in the FPGA, this notifies the computer by transmitting the byte 0xDD. The computer will respond with another byte 0xDD, denoted as ACK DD, to indicate its disposal to receive the samples and then, the FPGA begins the transmission. This flow of byte 0xDD, ACK DD, and samples will be repeated until the user ends the communication sending 0xEE, which is the case shown to the right of Fig. 3. The 0xEE byte could be sent to the FPGA at any time required by the user (using the "Stop" button), so this will be the only possibility of simultaneous transmission between hardware and software. As it will be treated in the design of the hardware, the reception of 0xEE has the highest priority, because its objective is to restart the system and immediately interrupt any running process.

In addition to serving as delimiters, the characters 0xAA, 0xBB and 0xCC constitute a mechanism to verify that no bytes have been lost. The protocol states that if the configuration frame received by the FPGA is not correct, it does not send any

confirmation. From the computer side, if there is a lapse of three seconds without receiving an ACK, the communication is interrupted by sending 0xEE and the system is returned to its initial state. This procedure of waiting and sending 0xEE is common to handle any error, such as the absence of 0xDD or the unexpected interruption of samples transmission.

3.3. Program thread dedicated to communication

After knowing the protocol characteristics, its implementation through a dedicated program thread will be addressed. The communication is carried out by an object of class CxThread, developed by the authors. This class inherits from QThread [6] and begins to run as an independent thread when the inherited method *start()* is called. The only functionality of *start()* is to invoke the pure virtual method of QThread named *run()*, which will execute the tasks associated with the new thread. In this way, to develop any task parallel to the main thread (the GUI thread), it will be enough to create an object whose class inherits from QThread and re-implement the *run()* method with the desired task. Fig. 4 shows the way in which the general tasks of the communication thread have been organized.

Pressing the "Start" button (see Fig. 1) will gather the specifications that define the initial configuration frame. Then several variables are established to control the communication flow, whose meanings will be clear later. Next, it is verified whether the thread is executed for the first time, or is inactive after a previous execution. In the first case, its execution begins with the *start()* method for invoking *run()*, while in the second, the thread is "awakened" using the *wake()* method of the class QWaitCondition [6]. The thread may be "sleeping" due to completed acquisition sections, and by calling *wake()* it resumes execution from the same line of code where it was blocked.

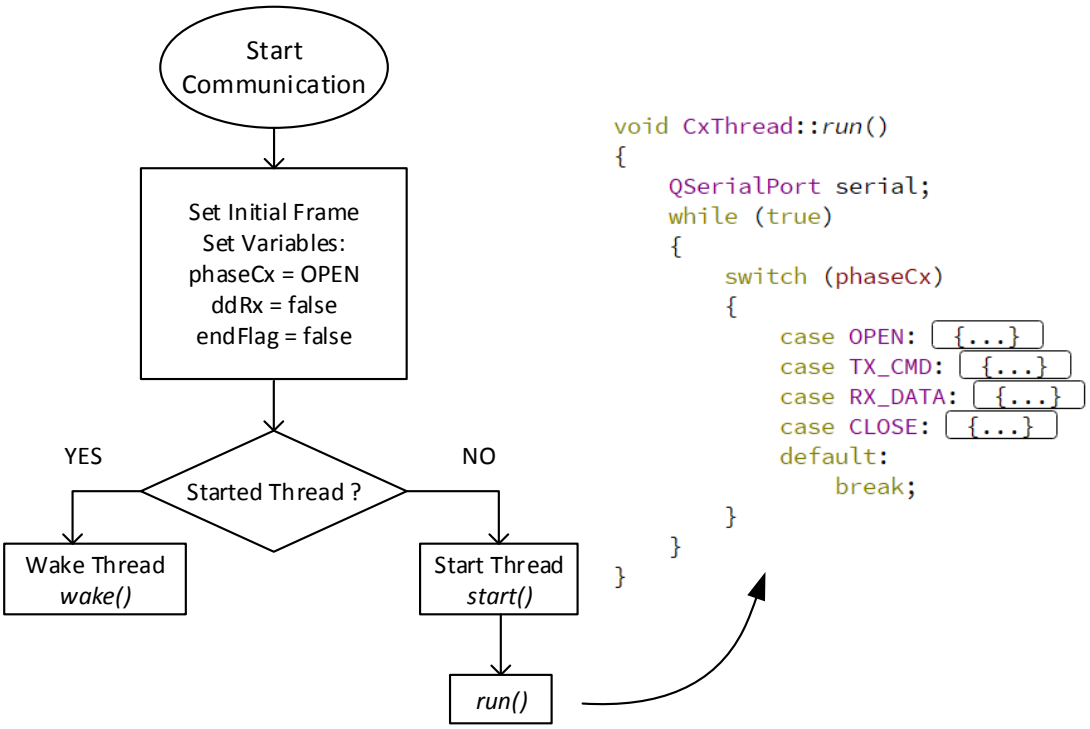

Fig. 4 General operation of the program thread dedicated to communication

In the right part of Fig. 4, the implementation of the *run()* method is summarized. The thread will be running "forever" by the while loop unless it is blocked, either by waiting for some data or by the user's decision. The variable named phaseCx indicates the communication status and is the key to control its flow. The procedures for the four possible cases of this variable are shown in Fig. 5 and 6. Note also, the object of class QSerialPort [6] that serves to perform communication through the serial port.

As shown in Fig. 5, when starting the thread execution, the variable phaseCx takes the value OPEN. During this phase, the port is configured for the speed and format established by the protocol. If an error occurs when trying to open the port (busy

port, for example) the communication is terminated, so the variable phaseCx takes the value CLOSE and the next iteration to execute the corresponding code (see Fig. 6). If the port was opened correctly, phaseCx takes the value TX_CMD and proceeds to send the initial configuration frame in the next iteration.

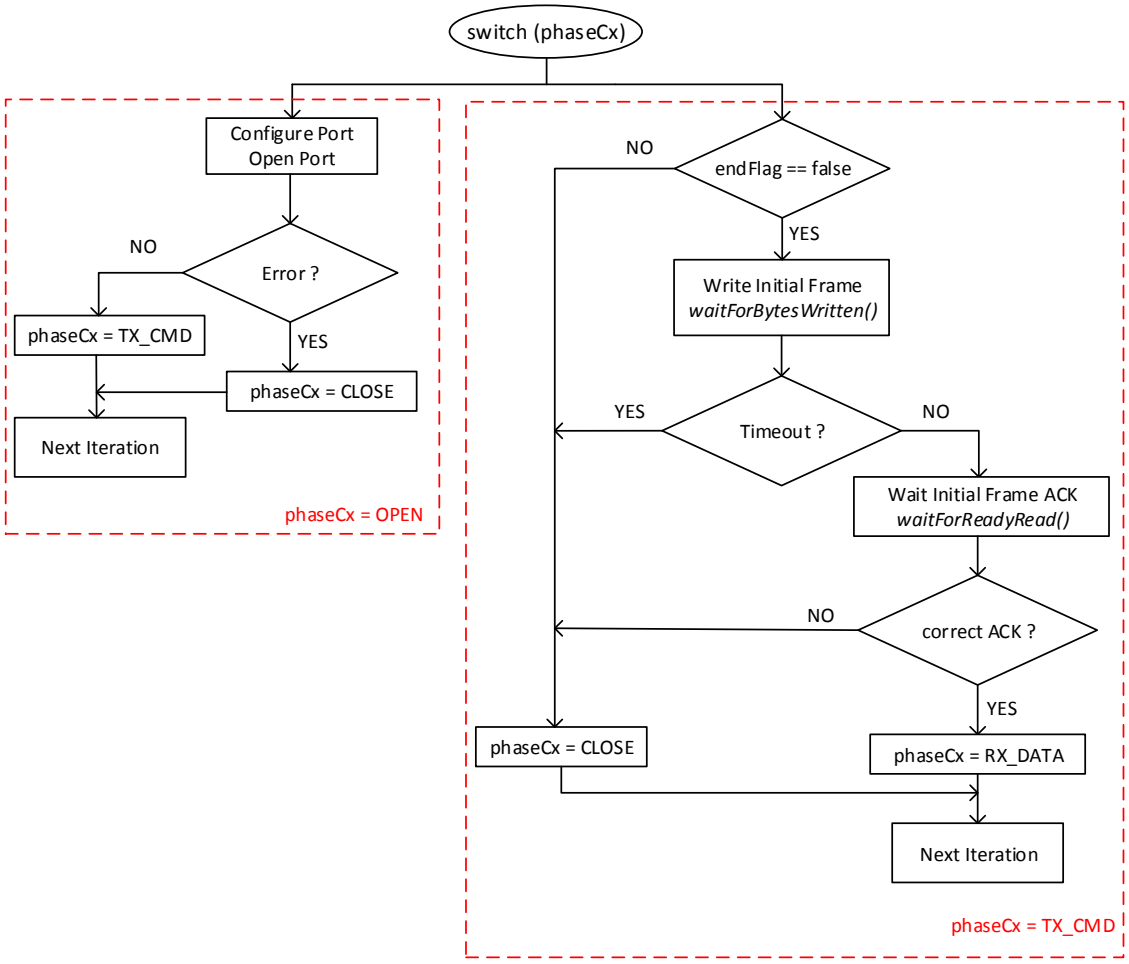

Fig. 5 General Cases OPEN and TX_CMD of the variable phaseCx to control the communication flow

The first step of the TX_CMD phase is to verify that the end of the process indicator (Boolean variable endFlag) is false, so that transmission of the configuration frame begins. This is done by writing the data to the serial port using the *waitForBytesWritten()* function, which locks the thread until at least 1 byte has been written. In case that the elapsed time is one second and the data has not been written, the communication is terminated. If the bytes were written without trouble, a wait for ACK of the initial frame is done and the thread is blocked by *waitForReadyRead()* until receiving the 11 bytes. If the received frame differs from the one sent, either in the number of bytes or in its content, the communication is terminated. The communication will also be terminated if the elapsed time without receiving a response from hardware is five seconds. If the ACK is correct, the next iteration will execute the RX_DATA phase.

The execution of the thread continues as shown in Fig. 6. As in TX_CMD, the RX_DATA phase starts verifying the end of process indicator. Depending on the value of the ddRx variable, the thread is blocked while it is waiting for byte 0xDD from FPGA to indicate the availability of the samples. If this byte is received by the computer, 0xDD is written to the port so that the hardware begins the transmission of the samples in the next iteration (see Fig. 3). The bytes received by the computer are stored in an array of type QByteArray [6] in order to be emitted by the signal *acquiredBytes()*, which will be attended by a slot in the main thread. In future work this signal could be connected to another thread that implements other functionalities of DRACEC, for example, the moment's computation [3] or anomalies detection [1, 29-30]. This procedure continues running until an error occurs or by pressing the “Stop” button, which indicates that acquisition is finished. In any of these cases, the phase executed in the next iteration will be CLOSE.

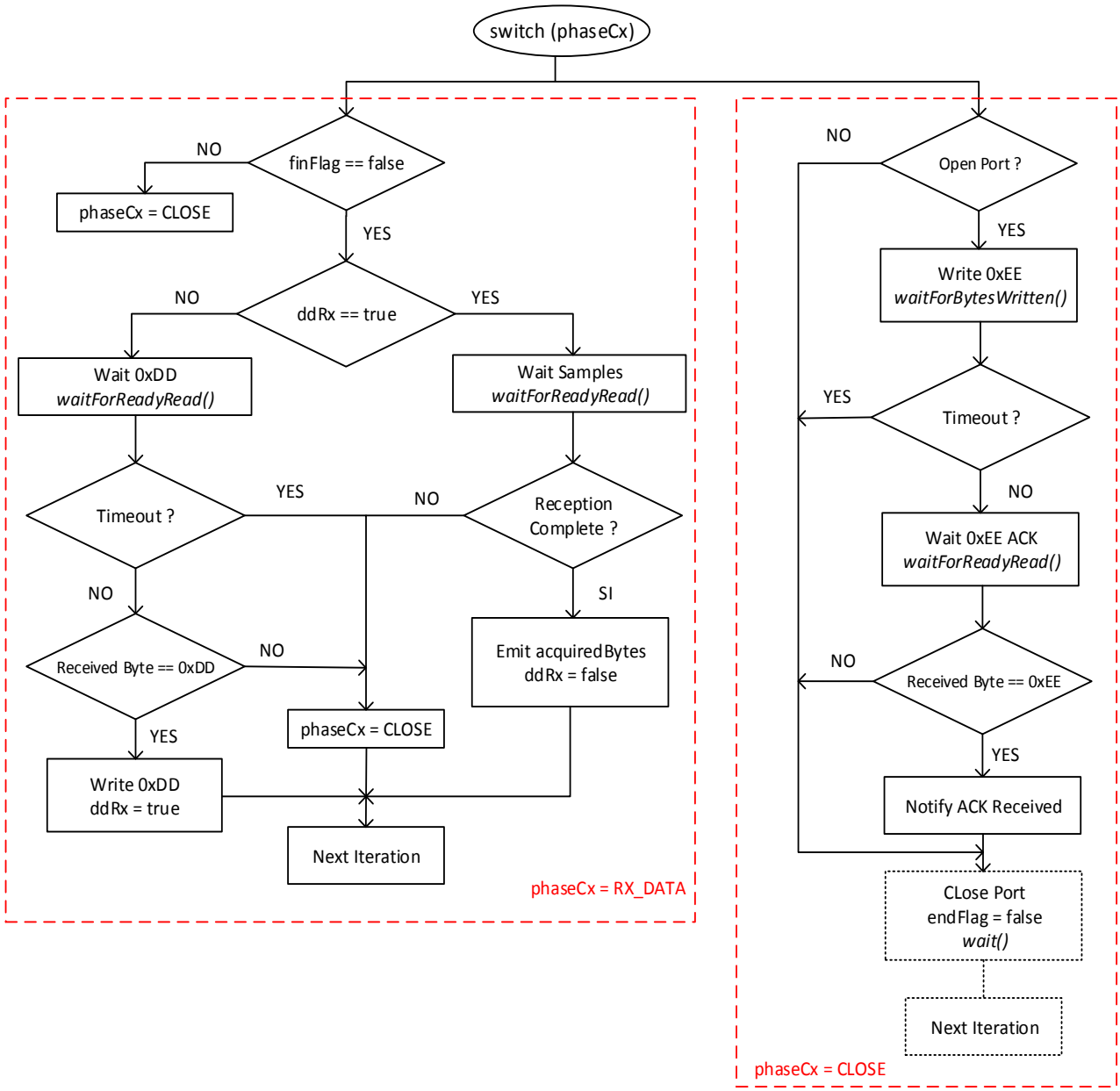

Fig. 6 General Cases RX_DATA and CLOSE of the variable phaseCx to control the communication flow

According to Fig. 6, the fundamental operations of the CLOSE phase are to send the 0xEE command to the FPGA and block the thread until the user presses the "Start" button again. The actions of this phase are executed for two reasons: after the occurrence of some error during the communication (busy port, timeout, etc.) or by pushing the "Stop" button, whose effect is to assign "true" to the endFlag variable. The thread is blocked by the *wait()* method of the QWaitCondition class [6] and "wakes up" with the *wake()* method, once the "Start" button is pressed. The dashed lines of the last two blocks in Fig. 6 indicate that the thread will pass to the next iteration only after a call to *wake()*, since before it will remain blocked.

4. Hardware Elements

After addressing the fundamental software components, it is possible to have a global idea of the hardware characteristics, whose flow chart is shown in Fig. 7 and follows the communication protocol. Initially, the hardware waits for the configuration frame and sets the specifications defining the operating mode and searching window. This comprises the storage of 8 bytes (five for the searching window, one for operating mode and two for the samples total) in a group of registers that will be used by the acquisition system to establish the sampling intervals of the video signal [3]. Then the ACK of the initial frame is sent to the computer so that the software validates the configuration and it is ready to receive the byte 0xDD.

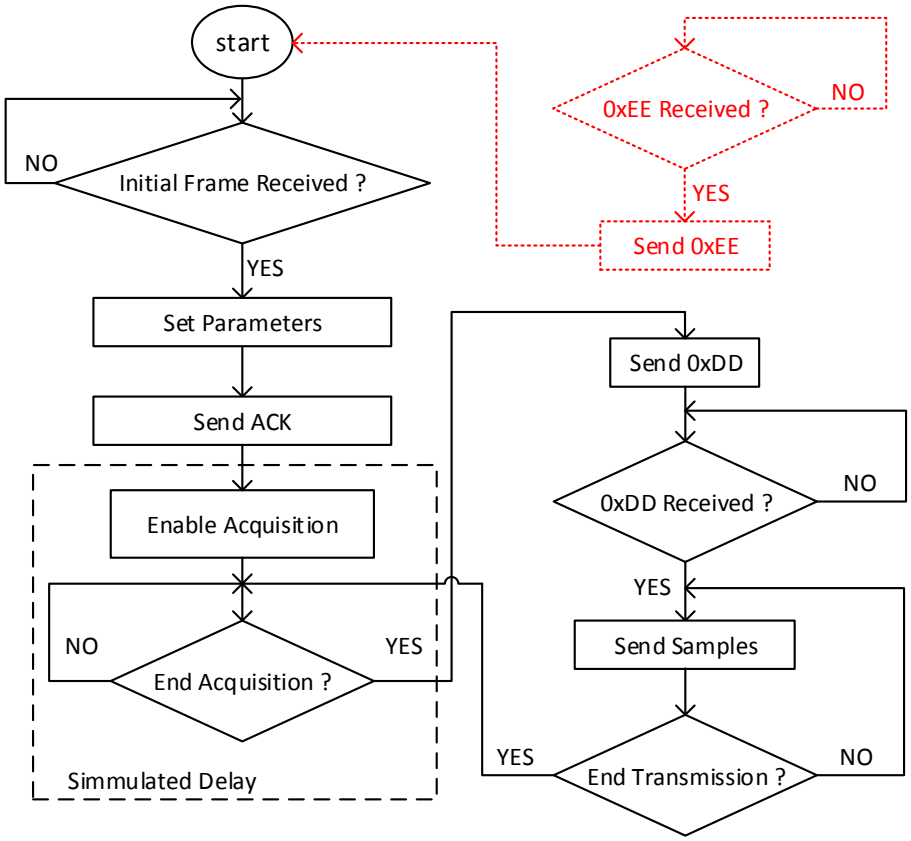

Fig. 7 Flow chart for the hardware system

Next, it is enabled the system responsible for sampling the video signal, which is external to the communication interface and does not constitute the center of this work. In a brief way, its objective is to store the samples obtained through A/D conversion, taking into account that each one corresponds to a specific triad of range, angle and received pulse [3]. Due to the characteristics of the NR-4000, every 2.73 seconds new samples will be available, as it was addressed in section 2. Therefore, this delay must be simulated by the hardware in order to get a better approximation to the real operation of the communication interface. For this reason, after 2.73 seconds a timer indicates that the "simulated acquisition" has been completed and the hardware sends the byte 0xDD to the computer. Then, the communication thread in the software will respond with 0xDD to begin the transmission (see Fig. 3). Once the transmission is over, the next simulated delay begins and the process repeats indefinitely. The system returns to its initial state only when the "Stop" button of Fig. 1 is pressed and the byte 0xEE is sent to the FPGA. This is highlighted in Fig. 7 by the red dashed lines, since the logic for the reception of 0xEE is executed continuously and independently of the rest of the system.

Fig. 8 shows the hardware components, where the central element to guarantee the flow of Fig. 7 is the Finite State Machine (FSM) called Control. It stores the bytes corresponding to the searching window, the operating mode and the total samples in the set of registers identified as PARAMETERS. These bytes are received through the UART receiver proposed in [31], which is represented as RX_UART in the figure. Another functionality of Control is to use the multiplexer identified as MUX to share the transmitter TX_UART [31] between three subsystems with specific functions, represented as ACK, Samples_TX and END.

The ACK logic is responsible for sending the ACK of the initial configuration frame to a computer for validating the received data. Once the ACK transmission is completed, the transmitter control goes to the logic Samples_TX, whose first task is to enable the delay that simulates the acquisition process and upon completion, sends to the computer the 0xDD byte, indicating that the samples stored in RAM memory are available. According to Table 1, the RAM memory must store a maximum of 13824 samples, so it has a capacity of 16384 locations of 14 bits. The RAM reading is carried out by the

aforementioned Samples_TX logic, while the writing is handled by the external acquisition system [3]. Since this system is not considered in this work, the memory contains known samples (describing a saw-tooth function) with the purpose of verifying their correct transmission to the computer.

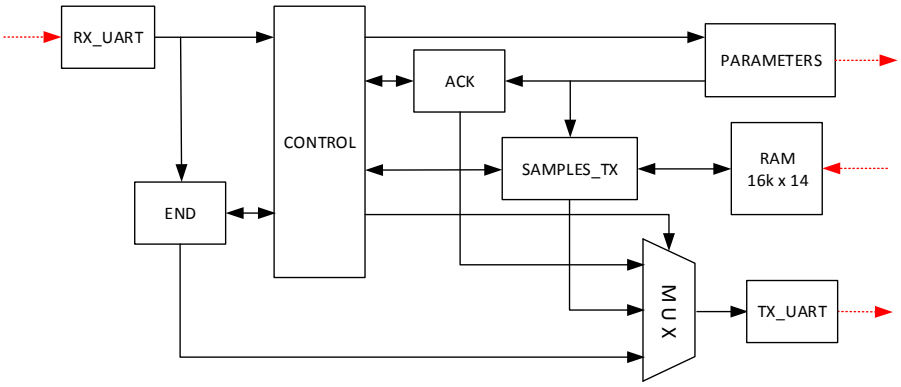

Fig. 8 Fundamental components of the communication interface hardware

Independently of the described components, the END FSM has the sole purpose of attending the software request for ending all processes. This device continually compares any received byte with 0xEE, and if it is equal, restarts the whole system. If byte 0xAA is received, which suggests a transmission with the configuration frame, the Control state machine disables the end-of-process logic until the frame reception is completed. This is due to the possibility of receiving a byte with value 0xEE as part of the operating specifications.

5. Verification of the Communication Interface

With the aim of verifying the proposed interface, Fig. 9 shows an image of the Qt Application Output after pressing the "Start" button. By means of the function *qDebug()* [6], the communication phases can be observed in close relation with Fig. 3, 5, and 6, thus illustrating the flow of the communication thread. The red rectangle highlights the repetitive processes of samples reception, which run uninterruptedly until the user presses the "Stop" button.

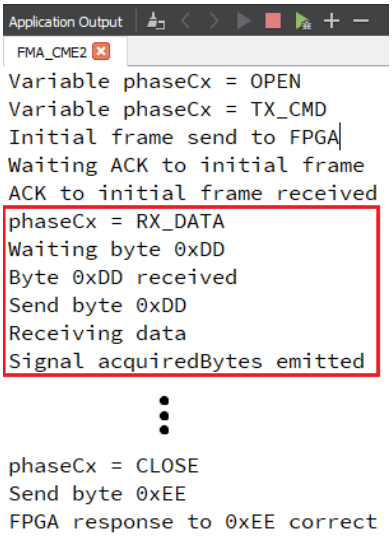

Fig. 9 Application output in Qt to indicate the different communication phases

As mentioned in the previous section, the RAM in FPGA contains known samples of a saw-tooth function, loaded by means of the memory initialization file. In order to consider the critical quantity of samples, the RAM locations are filled consecutively with values from 1 to 1728 (27 received pulses for each one of the 64 range rings) and this arrangement repeats eight times (for the eight angular sectors). Fig. 10 displays the received samples for this critical case, which correspond to a searching window of 512 cells and operating mode 1.

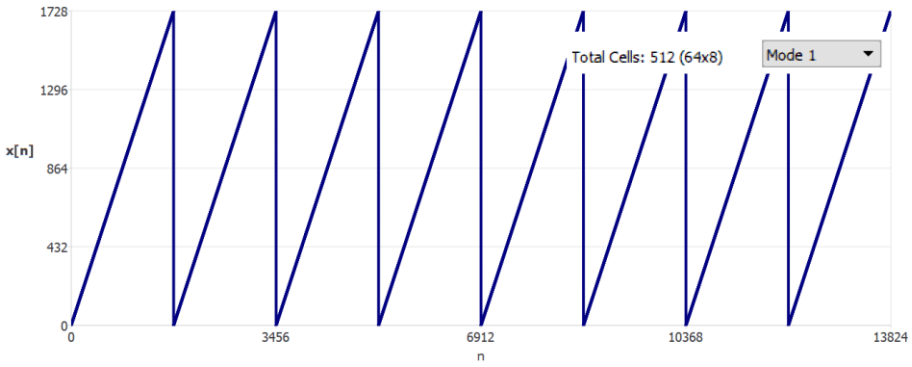

Fig. 10 Samples received on the computer for the critical searching window

On the other hand, Fig. 11 plots the received samples for a searching window of 265 cells and operating mode 3, which is the same case shown in Fig. 1. Both figures demonstrate that the communication between software and hardware is done correctly since the samples match with those stored in RAM. Operations like tab switching, zoom-in and zoom-out were performed on the graphics and it was confirmed at run time that the GUI does not "freeze", despite being continuously receiving the samples. These facts confirm that overall software performance is not affected.

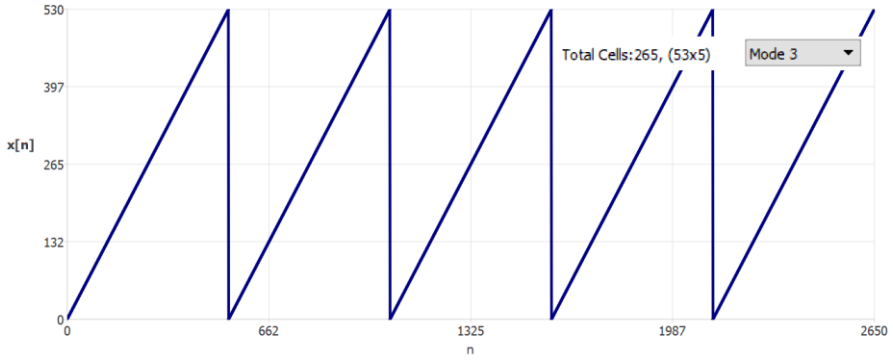

Fig. 11 Samples received on the computer for the example of Fig. 1

6. Conclusions

The proposed communication interface allows the samples stored in the FPGA to be available on the computer without affecting the software performance. Although the serial port is a slow communication interface, it is the simplest among those that satisfy the DRACEC requirements when it is analyzed a small window in comparison with the surveillance region. If it is necessary to increase the number of resolution cells in the window or the antenna rotation speed, another communication interface should be used with higher transfer speeds, such as USB or Ethernet.

The capacity of current computers for using multiple program threads improves the performance of tasks that can run independently. This constitutes a considerable advantage for the implementation of DRACEC method, whose stages execute a set of algorithms that perform computations independently and exchange their results. The designed software uses a thread dedicated to communication and establishes the principles to develop the algorithms of Adaptation and Detection, following the multithreading philosophy.

Conflicts of Interest

The authors declare no conflict of interest.

References

[1] N. Chávez and C. Guillén, "Radar detection in the moment's space of the scattered signal parameters," Digital Signal Processing, vol. 83, pp. 359-366, 2018.

[2] N. Chávez, "Detección y alcance de radar: la alerta temprana de blancos en fondos enmascarantes y una solución al problema," Ph. D. dissertation, Instituto Técnico Militar "José Martí", La Habana, 2002.
[3] C. Guillén, "Formación de la muestra aleatoria y cálculo de los momentos estadísticos para un detector DRACEC," M.Sc. dissertation, Universidad Tecnológica de La Habana "José Antonio Echeverría", CUJAE, La Habana, 2018.
[4] C. Guillén, C. L. Marcos, and N. Chávez, "Variantes de sistema de almacenamiento para un detector de radar en el espacio de los momentos," Revista de Ingeniería Electrónica, Automática y Comunicaciones, RIELAC, vol. 38, pp. 65-71, 2017.
[5] L. A. Miller, "The role of FPGAs in the push to modern and ubiquitous arrays," Proceedings of the IEEE, vol. 104, no. 3, pp. 576-585, 2016.
[6] Qt Creator Manual: Vers. 4.3.1, The Qt Company Ltd., 2017.
[7] L. Zhi, Qt 5 C++ GUI programming cookbook, Birmingham, UK., Packt Publishing Ltd., 2016.
[8] Cyclone II FPGA Starter Board Schematic Diagram: Rev. 1.1A, Altera Corporation, 2006.
[9] Cyclone II FPGA Starter Development Board Reference Manual: Vers. 1.0, Altera Corporation, 2006.
[10] NAVI-RADAR 4000 Technical Reference: Vers. 1.11.002, Transas Ltd., 2007.
[11] Scanner Unit 25 kW with Serial Control SU70-25H: Rev. 0, GEM Elettronica, 2007.
[12] M. I. Skolnik, Radar handbook, 3ra ed. New York: MacGraw-Hill, 2008.
[13] G. Casas and D. Frómeta, "Implementación de una interfaz de comunicación por puerto serie para un detector DRACEC," B.Sc. dissertation, Universidad Tecnológica de La Habana "José Antonio Echeverría", CUJAE, La Habana, 2018.
[14] ADS5500 14-Bit, 125 MSPS, Analog-To-Digital Converter, Texas Instruments, Austin, Texas, 2008.
[15] M. I. Skolnik, Introduction to radar systems, 3rd ed. New York: McGraw-Hill, 2001.
[16] E. Conte, A. De Maio, and C. Galdi, "Statistical analysis of real clutter at different range resolutions," IEEE Transactions on Aerospace and Electronic Systems, vol. 40, no. 3, pp. 903-918, 2004.
[17] N. Pallavi, P. Anjaneyulu, P. B. Natarajan, V. Mahendra, and R. Karthik, "Design and interfacing of T/R modules with radar system through TR module controller using FPGA," Proc. IEEE Symp. International Conference on Electronics, Communication and Aerospace Technology (ICECA 17), IEEE Press, 2017, pp. 231-233.
[18] M. Palanza, S. Petillo, and P. Matthias, "Ocean observatories initiative coastal surface mooring technology," Proc. IEEE Symp. OCEANS 2017, IEEE Press, 2017, pp. 1-4.
[19] J. P. Taylor and J. G. Hoole, "Robust protocol for sending synchronization pulse and RS-232 communication over single low quality twisted pair cable," Proc. IEEE Symp. International Conference on Industrial Technology (ICIT), IEEE Press, 2016, pp 684-689.
[20] M. Chizh, A. Pietrelli, V. Ferrara, and A. Zhuravlev, "Development of embedded and user-side software for interactive setup of a frequency-modulated continuous wave ground penetrating radar dedicated to educational purposes," Proc. IEEE Symp. International Conference on Microwaves, Antennas, Communications and Electronic Systems (COMCAS 17), IEEE Press, 2017, pp. 1-5.
[21] D. Tyagi, K. V. Krishnan, and R. Tyagi, "Digital redesign of analog search radar," Proc. IEEE Symp. International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT), 2017, pp. 27-31.
[22] W. Shang, Z. Dou, W. Xue, and Y. Li, "Digital beamforming based on FPGA for phased array radar," Progress in Electromagnetics Research Symposium (PIERS), 2017, pp. 437-440.
[23] P. Prabhakar, R. Rao, S. Panbude, A. Kulkarni, and A. Khandare, "SFCW ground penetrating radar for soil profile measurement simulation mode user interface," Proc. IEEE Symp. International Conference on Trends in Electronics and Informatics (ICEI 17), IEEE Press, 2017, pp. 1106-1108.
[24] M. Summerfield, Advanced Qt programming, Upper Saddle River, NJ: Addison-Wesley, 2011.
[25] J. Thelin, Foundations of Qt development, New York: Springer-Verlag, 2007.
[26] A. Kumar, S. Kumar, and K. Harlalka, "Optimization of the performance of high-speed serial communication," Proc. IEEE Symp. International Conference on Power, Control, Signals, and Instrumentation Engineering (ICPCSI 17), IEEE Press, 2017, pp. 1693-1697.
[27] L. L. Peterson and B. S. Davie, Computer Networks: a systems approach, 4th ed. San Francisco, CA: Elsevier, Inc., 2007.
[28] Interface Circuits for TIA/EIA-232-F, Texas Instruments, Austin, Texas, 2002.
[29] C. Guillén, C. Hernández, and N. Chávez, "Graphical user interface to show the results of the adaptation algorithms in the DRACEC method," VIII Simposio Internacional de Telecomunicaciones, Informática 2018, La Habana, Cuba, 2018.
[30] C. Guillén and N. Chávez, "Two-dimensional determination of the decision boundary for a radar detection method in the moment space," Journal of Aerospace Technology and Management, in press.
[31] P. P. Chu, RTL Hardware Design using VHDL, Hoboken, NJ: John Wiley & Sons, 2006.