Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Programmable-System-on-Chip-based data acquisition system for educational purposes

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Fernando Manzano Rubio

DIRECTOR: Ernesto Serrano Finetti

DATE: April 23 th 2012

Título: Programmable-System-on-Chip-based data acquisition system for educational purposes

Autor: Fernando Manzano Rubio

Director: Ernesto Serrano Finetti

Fecha: 23 de abril de 2012

Resumen

El objetivo del proyecto es desarrollar un sistema DAS con fines docentes basándose en un PSoC DVK. El sistema está orientado principalmente a ser un soporte para desarrollar conocimientos básicos sobre la adquisición de señales, así como una herramienta para llevar a cabo medidas sencillas.

Para ello se ha implementado una parte hardware y una parte software. El hardware consiste en el PSoC DVK, el cual tiene las funciones de adquirir y generar señales AC y DC, mostrar información sobre la función seleccionada y sobre la señal adquirida mediante un display y transmitir las muestras adquiridas a través de una comunicación RS-232. El software permite la interacción con el PSoC así como la visualización de las muestras adquiridas tanto en función del tiempo como en un histograma.

El sistema ofrece la capacidad de seleccionar el tipo de ADC entre integrador, SAR y doble integrador; así como varias resoluciones y tasas de muestreo. Esto permite disponer de varias configuraciones, con el fin de estudiar algunas características del ámbito de la adquisición de señales como por ejemplo la capacidad del integrador para el rechazo de interferencias en comparación con el SAR y el testeo de circuitos analógicos mediante el doble integrador. Además, el sistema dispone de un DAC para la generación de señales AC y DC con amplitudes y frecuencias seleccionables, como fuente de señal para el testeo de los circuitos analógicos o el estudio de los ADCs.

Finalmente, el sistema permite seleccionar el número de muestras a adquirir entre varios tamaños predefinidos para poder observar con más o menos detalle la representación temporal de la señal, así como una opción de escalar la entrada para aprovechar la resolución del ADC en señales de baja amplitud.

Title: Programmable-System-on-Chip-based data acquisition system for educational purposes

Author: Fernando Manzano Rubio

Director: Ernesto Serrano Finetti

Date: April, 23 th 2012

Overview

The project aims to implement a PSoC DVK based DAS system learning resource. The idea is to provide a way to develop signal acquiring basic knowledge, as well as a tool to implement simple measures.

In order to do it, hardware and software have been implemented. Hardware consists of PSoC DVK, which has the functions of AC and DC signal acquiring and generating, showing information about the selected function and the acquired signal through a display, and transmitting the acquired samples through the RS-232 communication. Software allows PSoC interacting and showing the acquired samples over the time and in a histogram.

The system provides the capability for selecting the ADC type between an integrating, a SAR and a dual integrating. Also, different resolutions and sample rates can be selected. This allows having several configurations in order to study some signal acquiring features, such as the interference rejection capability of the integrating ADC compared to the SAR and the analog circuit response, or to study each ADC type signal acquiring.

Finally, the system allows selecting the number of samples to be acquired between some preset quantities. It allows showing a detailed temporal representation of the samples acquired. Also, the input can be scaled in order to profit the ADC resolution for small amplitude signals.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my tutor Ernesto Serrano, whose expertise, understanding and time invested had helped me in all the time research for and writing of this project.

I must also acknowledge my family for the support they have provided me during the project development and the encouragement to going on in the hard moments.

TABLE OF CONTENTS

ACRONYMS LIST

ADC: Analog to Digital Converter.
AC: Alternating Current.
API: Application Programming Interface.
CPU: Central Processing Unit.
CRC: Cyclic Redundancy Check.
DAC: Digital to Analog Converter.
DAQ: Data acQuisition.
DAS: Data Acquisition System.
DC: Direct Current.
DNL: Differential Non-Linearity.
DTMF: Dual-Tone Multi-Frequency.
DVK: DeVelopment Kit.
EEPROM: Electrically Erasable Programmable Read-Only Memory.
ENOB: Effective Number Of Bits.
FSR: Full-Scale Range.
GPIO: General Purpose Input/Output.
I2C: Inter-Integrated circuit Communication.
INL: Integral Non-Linearity.
IO: Input/Output.
IRDA: Infrared Data Association.
ISSP: In-System Serial Programming.
LCD: Liquid Cristal Display.
LED: Light-Emitting Diode.
LSB: Least Significant Bit.
MCU: Micro-Controlling Unit.
MIPS: Million Instructions Per Second.
PGA: Programmable Gain Amplifier.
PSRR: Power Supply Rejection Ratio.
PRS: Pseudo-Random Sequence.
PWM: Pulse-Width Modulation.
RAM: Random-Access Memory.
RMS: Root Mean Square.
ROM: Read-Only Memory.
RS232: Recommended Standard 232.
SAR: Successive Approximation Register.
SINAD: Signal to Noise And Distortion ratio.
SPI: Serial Peripheral Interface.
SQNR: Signal to Quantization Noise Ratio.
SRAM: Static Random-Access Memory.
UART: Universal Asynchronous Receiver Transmitter.

1. INTRODUCTION

1.1 Objectives

The main objective of this project is to implement a DAS learning resource based on a PSoC DVK for a wide variety of engineering students, from those closely related to electronic systems like telecommunications to those who need to understand the basic features of signal acquisition but only as a necessary tool to perform measurements. This learning resource will cover many basic features of a DAS such as ADC resolution and sampling rates, aliasing, particular characteristics of different ADC types, DAC AC-DC signal generation, data logging capability, noise and input/output characterization. Also, in this study the main PSoC limitations referring to dynamic reconfiguration or system resources will be assessed.

Finally, a user interface will be developed in order to interact with the PSoC. It will allow the configuration and running of the PSoC to work and test the previously mentioned DAS features.

1.2 DAS

In some areas, data of interest is present in the real world in a difficult manner to treat it. It is present as physical signals that, because of its nature, can be in a great quantity, with fast variations or small dimensions in order to appreciate the information they can give us.

A DAS is a system designed to acquire these signals from the real world to a device. The main objective of this system is to measure, process or store these signals. Typically, the DAS needs some other elements to do it. These elements can be seen as stages, which everyone has a concrete task in the acquisition of the signal. **Fig. 1.1** shows the common stages in a DAS.

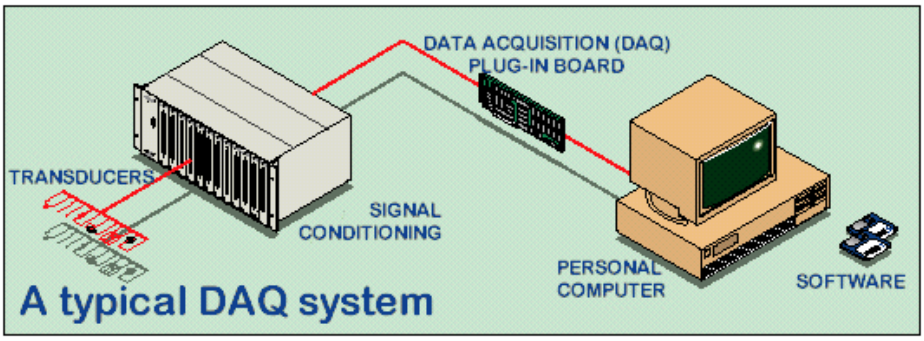


Fig. 1.1 A typical DAS system.[1]

Usually, the goal of a DAS is to acquire signals coming from one or more sensors which are used to get the real signal and to convert it to an electrical one, normally expressed as a voltage signal. Sensors can be classified in a great number of groups, depending on the way they sense the signal and the kind of output they provide. Alternatively, a DAS might be used together with well-known signals to study input/output characteristics of a certain electronic device or circuit.

Whatever its source, the signals will be converted into digital codes by an ADC which is the key component of the DAS. The signal is sampled and quantified and a binary value is assigned to each signal level, which represents the analog value in the digital world. This sampling is triggered by the DAS main clock at a periodic rate. So, the main characteristics of the DAS depend on the basic ADC performance parameters: the resolution, the full scale voltage and the sampling rate.

The resolution, determined by the number of bits (N) of the ADC, gives the number of levels that the analog signal can be quantified into. Therefore, the number of bits corresponds to the size of the digital code.

The voltage reference is a voltage in the system that determines the FSR and the dynamic range of the ADC, which is defined as the difference between the maximum and the minimum voltages that can be detected at the input. This one, altogether to the resolution, gives the minimum voltage variation that the system can detect in the analog signal at the input, named LSB and corresponds to:

$$1LSB = \frac{FSR}{2^N} \tag{1.1}$$

[1] Source: http://www.kostic.niu.edu/DAQ-LabVIEW.html

From this parameter, another aspect of the DAS can be obtained, the quantization error (e_q), expressed as:

$$e_q = {}^{1}/_{2}\, LSB \tag{1.2}$$

This error is the uncertainty between the levels that can be quantified by the ADC. It is usually regarded as noise (quantization noise) and therefore, its relation with the signal at the input can be calculated as:

$$SQNR = 6{,}02 \cdot N + 1{,}76 \tag{1.3}$$

Also, if important levels of noise or distortion are present at the ADC input, some bits can be unavailable to represent the signal because they are masked. So, another parameter used to characterize the system, which also considers the quantization noise, is the ENOB. It specifies the number of bits in the digital signal above the noise floor and it is calculated as:

$$ENOB = \frac{SINAD - 1{,}76}{6{,}02} \tag{1.4}$$

Where the SINAD is a parameter that shows the relation between the signal and the noise and distortion present.

The sampling rate is the rate at which the quantization of the analog signal is done. In order to represent correctly the real signal in the digital world, it is mandatory to be greater than the double of the analog signal bandwidth.

Apart from this, the benefits of an ADC depend on the type of ADC. The most common are the Flash ADC, the SAR, the integrating ADC and the Sigma-Delta. The main difference between them is the way to implement the conversion, achieving different performance.

Another important step in the process of signal acquisition is signal conditioning, which adapts the electrical signal coming from the sensor to the DAS input to match dynamic range, offset, common mode and bandwidth to mention some of the key performance parameters. This stage is named the Analog Front End and it is previous to the DAS input. This adaptation is necessary to obtain the maximum quality in the conversion.

Filtering is done to attenuate noise and other unwanted signals in order to keep the noise floor as close as possible to keep the expected DAS resolution while also avoiding aliasing effects.

Amplifying or attenuating allows the level of the signal match the DAS dynamic range. This is usually done to take full profit of the number of bits offered by the DAS. For example, if only half the full scale voltage is used then the effective resolution will be one bit less than expected.

Offset shifts the average level of signal to match the maximum and minimum values accepted by the DAS input. For example, a signal might swing between -2 to 3 V while the DAS input might only accept values from 0 to 5 V. It is clear that a shift of 2 V is necessary for the signal to be accurately sampled.

Multiplexing the signal is a secondary aspect of adaptation that depends on the quantity of signals to acquire. It allows the DAS to acquire several signals in parallel.

Once the signal is adapted to the DAS input, it is converted to digital and is finally transferred to the final device. This device normally is referred to some kind of computer with powerful software that shows this data and allows processing it if needed in order to measure and study some processes. Also, more concretely systems can be used to store data for future manipulations.

1.3 PSoC

This section will explain some features about the PSoC, such as its description, the resources needed to implement the project, the Dynamic Reconfiguration and the DVK.

1.3.1 Why use a PSoC?

PSoC stands for Programmable System on Chip, and it is a variety of the Cypress programmable microcontrollers' family. It has a core, a configurable system integrated which includes mixed-signal arrays of configurable analog and digital blocks, and a programmable routing and interconnect. **Fig. 1.2** shows these parts.

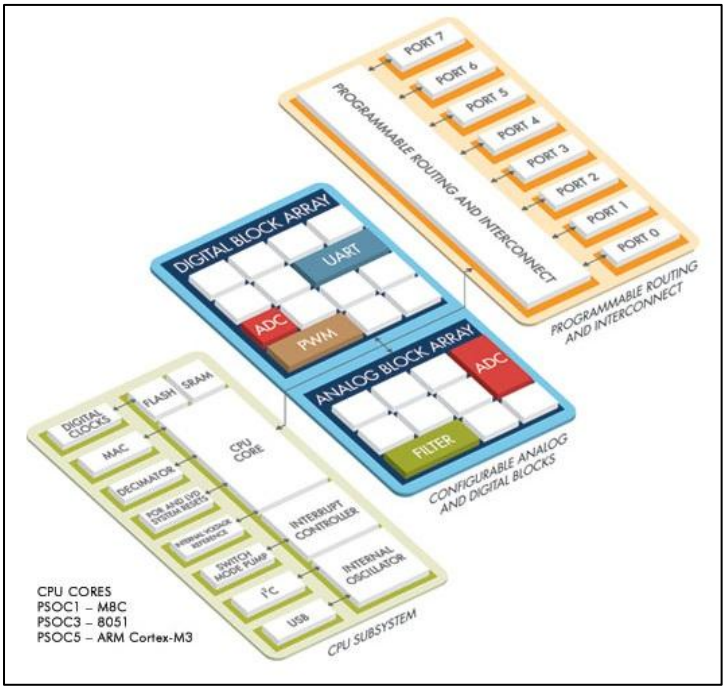


Fig. 1.2 The PSoC main parts.[2]

These configurable blocks can be programmed as peripherals to perform a wide variety of functions, and then they can be interconnected to perform a specific task. Between these functions there are ADCs, PGAs, filters, DACs, counters or UART communication.

Furthermore, PSoC allows to create several configurations for different tasks, and to switch between them in real time. This characteristic, called Dynamic Reconfiguration, gives to the PSoC the main difference from other microcontrollers.

PSoC is programmed by a powerful software provided by the manufacturer, PSoC Designer. It has a graphic interface to configure the analog and digital blocks with user modules, which represents the peripherals such as ADC, PGA, DAC, etc. and their parameters. Also, parameters of the entire PSoC such as clocks, reference voltages, power, etc. and the device pins can be set through the software. Apart from this, the software allows to program the PSoC through C++ or assembler code. All user modules parameters and functions can be accessed through code, in order to achieve a more complexity configuration. To allow it, the manufacturer has implemented a series of libraries for each user module, which provides an API to configure them.

[2] Source: http://www.cypress.com

Therefore, versatility and capability to switch between different tasks are the strengths of these microcontrollers. Aspects that make them a good choice to develop the functionality presented in this project.

1.3.2 PSoC architecture

There are three families of PSoC: PSoC 1, PSoC 3 and PSoC 5. The principal difference is the MCU that feeds the core, which increases its performance in terms of clock rate and therefore the calculation rate. Also, the instruction length is incremented from 8 to 16 and 32 bits. **Fig. 1.3** shows a comparative between them. In this project we will work with a PSoC 1 because of its availability and technical support.

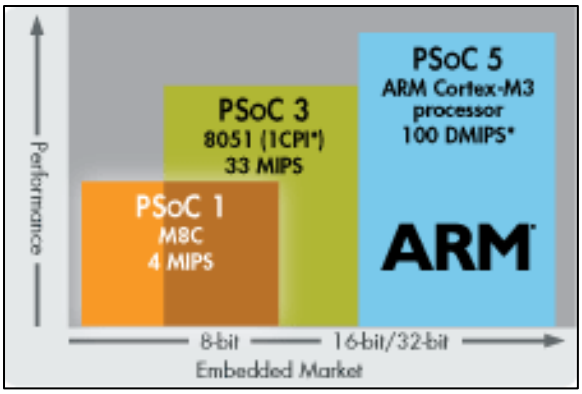


Fig. 1.3 PSoC family cores.[3]

Then, depending on the PSoC model, the digital and analog systems may have 16, 8, or 4 digital blocks and 12, 6, 4 analog blocks. Also, the number of pins and the memories size may differ from each other categories. To implement the project, a CY8C29x66 model of PSoC 1 has been chosen, which is powered by a M8C MCU of up to 24 MHz and 4 MIPS and has the capability of Dynamic Reconfiguration.

As mentioned earlier in this section for general PSoCs, PSoC 1 devices are formed by a core, a configurable system based on analog and digital blocks, and a programmable routing and interconnect.

The most important elements included in the core are CPU, memory, clocks, and configurable GPIOs. Memory refers to RAM memory and Flash memory to locate

[3] Source: http://www.cypress.com

the firmware. PSoC uses 16 KB of flash for program storage, 256 bytes of SRAM for data storage, and up to 2 KB of EEPROM emulated using the flash.
The clocks are used to synchronize the internal clock of the analog and the digital blocks. They are part of the PSoC global resources. PSoC has a wide variety of clock sources in order to increase the flexibility. They provide the following clocking signals:

- SYSCLK: Is the system main reference clocking signal, it can be created by the Internal Main Oscillator (IMO) which runs at 24 MHz or and external clock (EXTCLK).
- SYSCLKx2: Twice the frequency of SYSCLK.
- CPUCLK: Determines the speed of the CPU. It is created by SYSCLK divided down to one of eight possible frequencies.
- VC1, VC2 and VC3: Are Variable Clocks (VC), related between them, the SYSCLK and the SYSCLKx2 through a divisor.
- CLK32K: Can be created by the Internal Low Speed Oscillator or the External Crystal Oscillator (ECO).
- CLK24M: Is the internally generated 24 MHz clock by the IMO.

Fig. 1.4 shows the relationship between the different PSoC clocks.

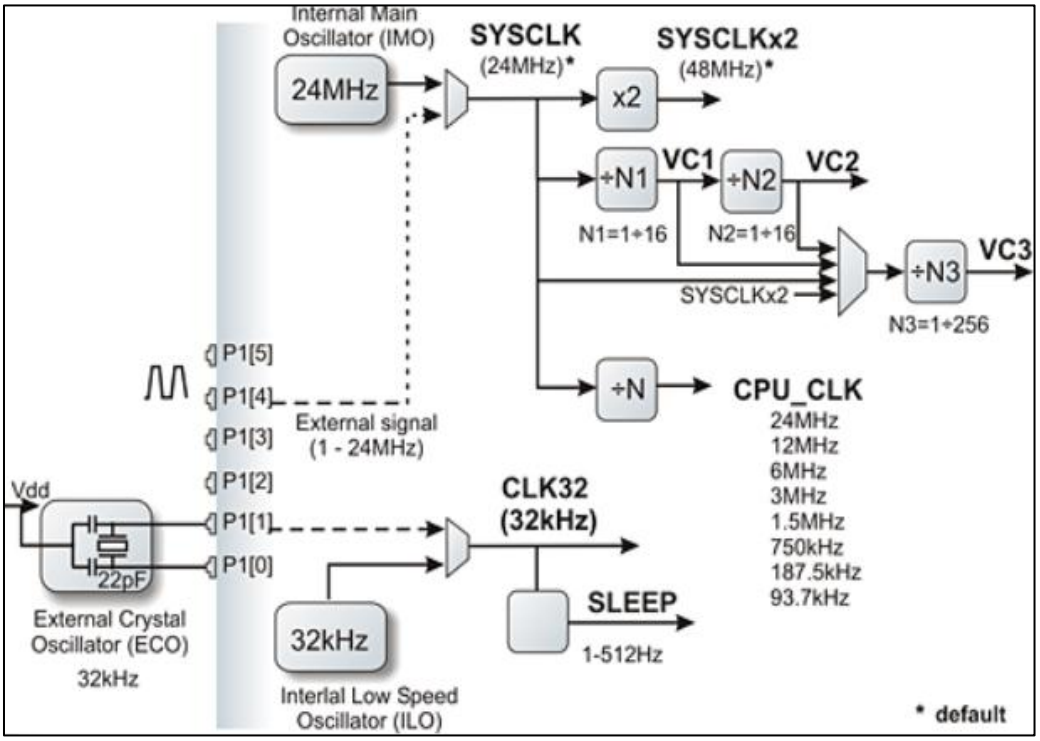


Fig. 1.4 Source clock PSoC internal distribution.[4]

[4] Source: http://www.psoc-chile.es.tl/Clock-y-Frecuenias.htm

The GPIO blocks provide the interface between the CPU and the outside world. Each GPIO block is formed by different I/O pins, which have several drive modes, as well as and interrupt capabilities. They can be used for the following types of I/O:

- Digital I/O (used by the CPU to send out and get into information)
- Global I/O (digital PSoC block input and output)
- Analog I/O (analog PSoC block input and output)

The programmable routing and interconnect allows any connection between the configurable system and the GPIO. It is done through a series of global buses that can route any signal to any pin. The buses also allow signal multiplexing and performing logic operations.

The digital part of the CY8C29x66 category configurable system is composed by 16 digital blocks provided in rows of four, where each row has two blocks for communication purposes (DCB) and two blocks for basic purposes (DBB). **Fig. 1.5** shows the digital system distribution.

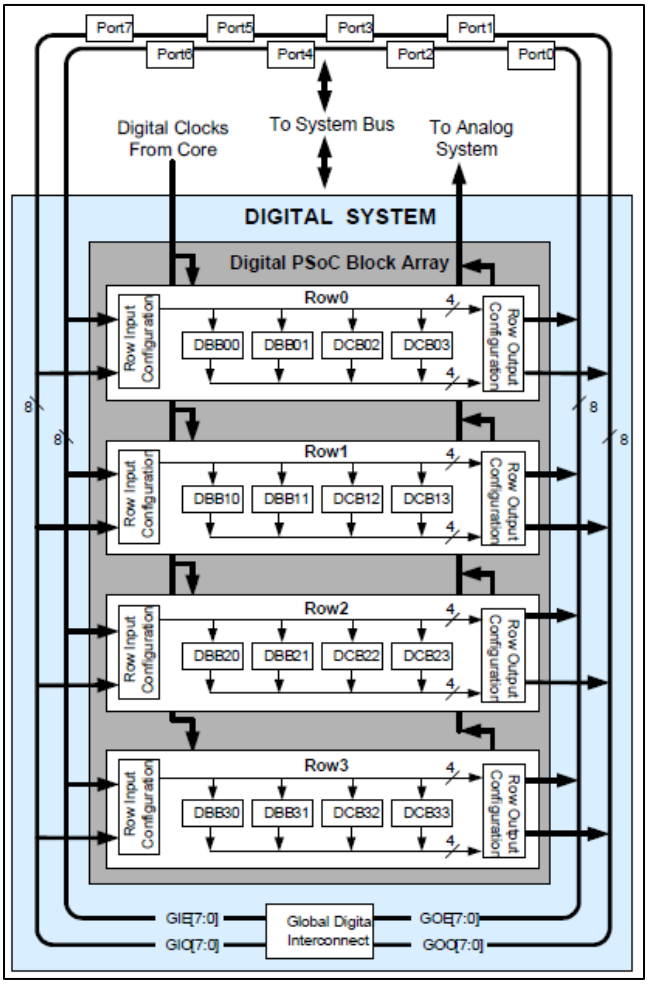


Fig. 1.5 PSoC digital system distribution.[5]

These blocks can be used alone or combined with other blocks to form 8-, 16-, 24-, and 32-bit user modules. The user modules included are:

- PWMs (8- to 32-bit)
- PWMs with dead band (8- to 32-bit)
- Counters (8- to 32-bit)
- Timers (8- to 32-bit)
- UART 8-bit with selectable parity (up to 2)
- SPI slave and master (up to 2)
- I2C slave and multi-master (one available as a system resource)
- CRC generator (8- to 32-bit)
- IrDA (up to 2)
- PRS generators (8- to 32-bit)

[5] Source: Cypress CY8C29x66 datasheet.

The analog part of the CY8C29x66 category configurable system is composed by 12 analog blocks provided in columns of three, where each column includes one continuous time (CT) and two switched capacitor (SC) blocks. **Fig. 1.6** shows the analog system distribution.

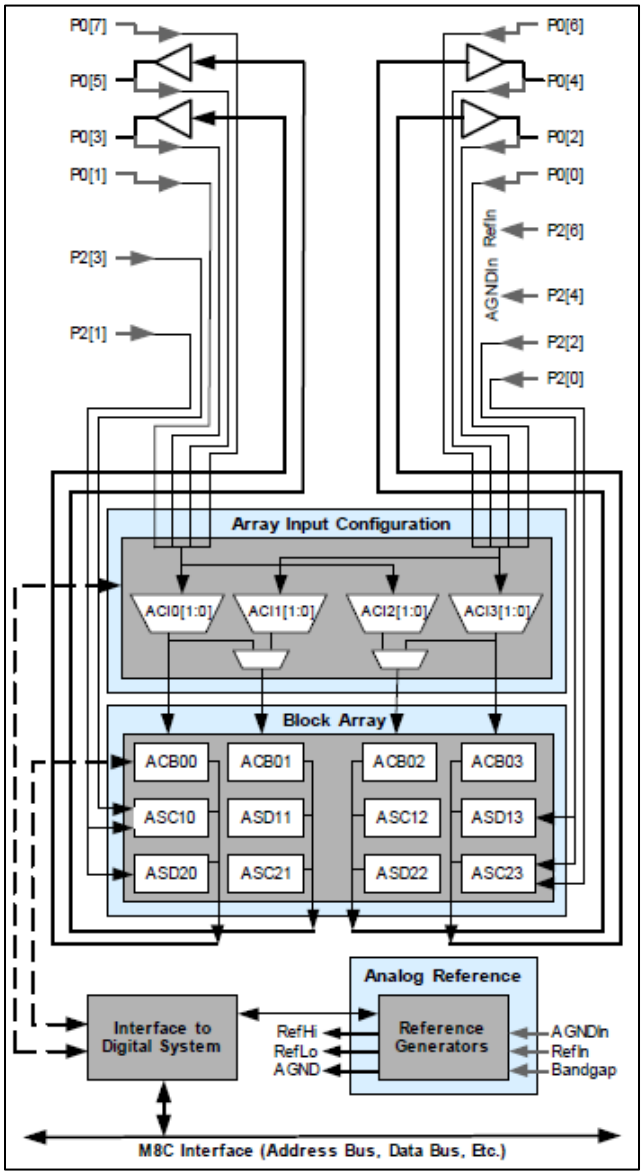


Fig. 1.6 PSoC analog system distribution.[6]

[6] Source: Cypress CY8C29x66 datasheet.

These blocks, each containing an opamp circuit that allows the creation of complex analog signal flows, can be used as a wide variety of peripherals. The user modules that can be configured as these peripherals are:

- ADCs (up to 4, with 6- to 14- bit resolution; selectable as incremental, delta sigma, and SAR)
- Filters (2-, 4-, 6-, and 8-pole band pass, low pass, and notch)
- Amplifiers (up to 4, with selectable gain to 48x)
- Instrumentation amplifiers (up to 2, with selectable gain to 93x)
- Comparators (up to 4, with 16 selectable thresholds)
- DACs (up to 4, with 6-bit to 9-bit resolution)
- Multiplying DACs (up to 4, with 6-bit to 9-bit resolution)
- DTMF Dialer
- Modulators
- Correlators
- Peak detectors
- Etc.

Also, a reference voltage and a nominal operation voltage are needed for setting the analog ground and the peak-to-peak voltage limit of the analog system. They are provided by the V_{cc} and the Ref Mux parameters of the PSoC and also are part of the PSoC global resources together the clocks. The V_{cc} corresponds to the power supply voltage of the system.

1.3.3 Technical specifications of resources needed

As explained at the beginning of this chapter, the PSoC peripherals are represented by user modules. These user modules have a series of technical specifications that characterize them. To implement the different system functions some user modules are needed, as well as other resources considered global resources. The user modules needed, as well as their technical specifications, are shown below. They corresponds to a temperature of 25 ºC, a system power supply of V_{cc} = 5 V and a high power supply for analog blocks.

UART

- Asynchronous receiving and transmitting.
- RS-232 serial-data format.
- Burst rates up to 6 Mbps.
- Data frame composed by start, optional parity and stop bits.
- Optional interrupt on receive register full and/or transmit buffer empty.
- Parity, overrun and framing error detection.

LCD

- Requires seven I/O pins.
- Allows for printing RAM or ROM strings.

- Allows for printing numbers.
- Allows for displaying horizontal and vertical bar graphs.

Counter
- Up to 32-bit resolution.
- Source clock rates up to 48 MHz.
- Automatic reload of period on terminal count.
- Programmable pulse width.
- Selectable continuous counter operation.
- Optional interrupt on compare output or terminal count.

DAC
- 6-bit resolution.
- 2's complement, offset binary and sign/magnitude input data formats.
- Up to 250 ksps.
- Output voltage range depending on the system reference voltage.
- DNL of 0,09 LSB, INL of 0,07 LSB.
- Including reference gain error of 3,4 %FSR, excluding reference gain error of 0,45 %FSR.[7]
- Offset voltage of +- 7,5 mV.
- Output noise of 4,6 mV RMS.

PGA
- Thirty-three user-programmable gain settings with a maximum gain of 48,0.
- High impedance input.
- Input voltage range depending on the system reference voltage.
- PSRR of 73 dB.
- Slew rate (20% to 80%) of 9,5 V/µs.
- Settling time of 1 µs.
- Noise of 99 nV/√Hz.

ADCINCVR
- 7- to 13-bit resolution.
- 2's complement and unsigned output data format.
- Sample rates from 4 to 10 000 sps.
- Input voltage range depending on the system reference voltage.
- SNR of 77 dB.
- DNL of 0,4 LSB, INL of 1,0 LSB.
- Offset error of 9 mV.
- Including reference gain error of 2,0 %FSR, excluding reference gain error of 0,1 %FSR.
-

DUALADC
- 7- to 13-bit resolution.

[7] Cypress condition: reference gain error measured by comparing the external reference to $V_{RefHigh}$ and V_{RefLow} routed through the test mux and back out to a pin.

- 2's complement and unsigned output data format.
- Sample rates from 4 to 10 000 sps.
- Input voltage range depending on the system reference voltage.
- SNR of 77 dB.
- DNL of 2,0 LSB, INL of 1,0 LSB.
- Offset error of 9 mV.
- Including reference gain error of 3,0 %FSR, excluding reference gain error of 0,1 %FSR.

SAR6

- 6-bit resolution.
- 2's complement output data format.
- Internal update rate of 32 to 333 kHz.
- Input voltage range depending on the system reference voltage.
- DNL of 0,25 LSB, INL of 0,75 LSB.
- Offset error of 8 mV.
- Including reference gain error of 1,5 %FSR, excluding reference gain error of 0,4 %FSR.

The global resources needed, as well as their technical specifications are shown below.

Clocks

As introduced above in the section Basic explanation of the architecture, PSoC has a wide variety of clock sources in order to increase the flexibility. The clock sources needed for implementing the project are SysClk, VC1, VC2 and VC3.

- SysClk: It can be sourced by the IMO or an external clock (ExtClk pin). The frequencies to be selected from the IMO are 24 MHz and 6 MHz.
- VC1: Is sourced by SysClk. An integer from 1 to 16 divides down the frequency.
- VC2: Is sourced by VC1. An integer from 1 to 16 divides down the frequency.
- VC3: It can be sourced by the VC1, VC2, SysClk or SysClk*2. An integer from 1 to 256 divides down the frequency.

Reference voltages

The reference voltages needed by the user modules are given by the Ref Mux and V_{cc} global resources parameters. They set the analog ground and the peak-to-peak voltage limits of the analog system.

- V_{cc}: Is the system supply voltage. It is selectable between 3,3 V and 5 V.
- Ref Mux: Sets the analog ground and the peak-to-peak voltage limits. The options offered are:
    - $V_{dd}/2$ +/- BandGap
    - $V_{dd}/2$ +/- $V_{dd}/2$
    - BandGap +/- BandGap

- 1,6 BandGap +/- 1,6 BandGap
- 2 BandGap +/- BandGap
- 2 BandGap +/- P2[6]
- P2[4] +/- BandGap
- P2[4] +/- P2[6]

1.3.4 Dynamic Reconfiguration

The Dynamic Reconfiguration is a technique developed by Cypress that allows PSoC 1 devices to reuse analog and digital resources to maximize the functionality. It consists in creating different layers, each of them with the user module configuration that have to work as a system.

The PSoC Designer allows easily the implementation of the Dynamic Reconfiguration by creating these layers in the user interface in a graphical way. These layers are located in the firmware and can be switched between and interacted with them at a firmware API levels. To do it there are some instructions in the code level that allow loading or unloading a concrete layer. Load a layer is referred to make it active and unload a layer is referred to make it inactive.

When the PSoC is started the boot code is executed. It includes loading the base configuration layer. This base layer must be always loaded. The rest of the layers are considered overlays. Apart from the base layer, more than one overlay can be loaded at the same time. So the user modules placed in these layers are always active. The digital and analog blocks used in these layers can't be used in other layers; otherwise they will be overlapped and as a consequence there will appear conflicts.

Global resources as reference voltages or clocks are set by the last loaded layer, so they may be reconfigured to accomplish with the user modules of the current layer. In this case, those resources used by other user modules already active have to be respected so that these user modules can work properly.

The result is a system composed by parallel systems that can be switched on and off at any time. And the blocks used at the same time by different layers can't be overlapped.

1.3.5 PSoC DVK

The PSoC DVK provides a common development platform, as well as optional peripherals such as RS-232, LCD, potentiometers and Input Peripherals, to help to prototype and to evaluate applications using a PSoC device. It is used to examine

and explore the peripherals and hardware features that are integrated into the PSoC device. Also, it allows programming the PSoC from the PSoC Designer.

There are many PSoC DVKs, oriented to any PSoC family, which have more or less capabilities and other optional components. To implement this project, the CY3210 PSoCEval1[8] has been selected, which works with the PSoC 1 family. This evaluation kit demonstrates the function of PSoC 1 devices. It connects the device to onboard peripherals such as potentiometers, LEDs, LCD, and RS-232. It also has additional features such as a general prototype area of bread board and an ISSP programming header to program the PSoC through the MiniProg programming unit also included. **Fig. 1.7** shows the PSoCEval1 DVK and its main components.

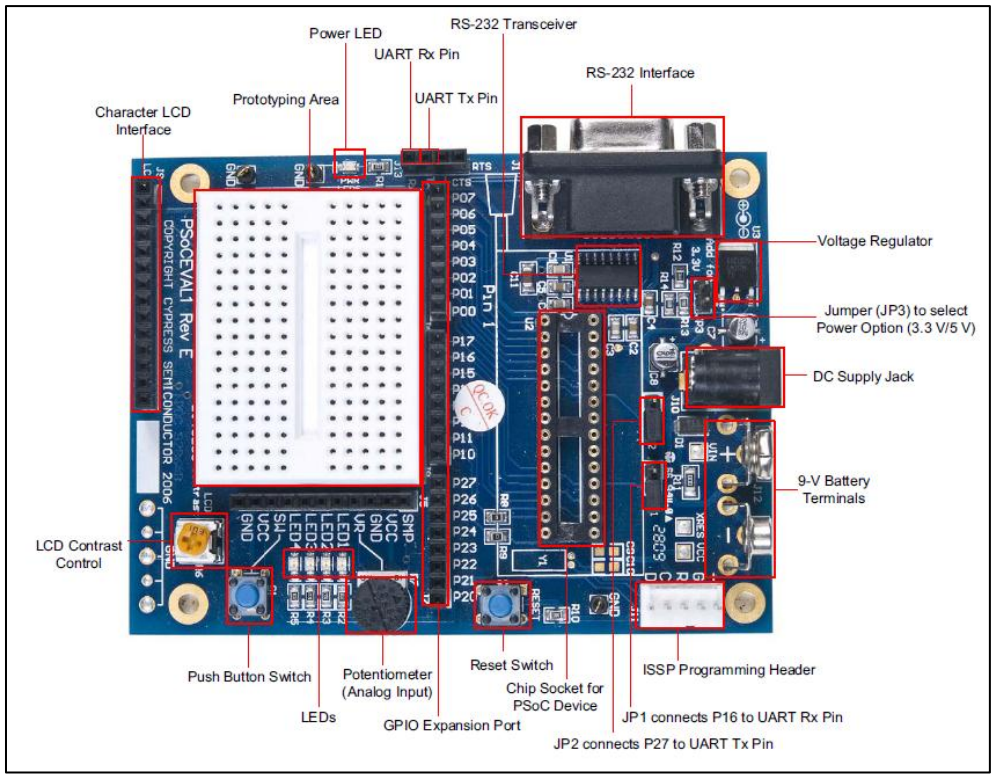


Fig. 1.7 CY3210 PSoCEval1 DVK.[9]

[8] Web reference: http://www.cypress.com/?rID=2541
[9] Source: CY3210-PSoCEVAL1 PSoC 1 Evaluation Kit Guide.

2. DESIGN AND IMPLEMENTATION OF A PSOC-BASED DAS SYSTEM

This chapter will explain the system functions as well as each of its modules. Then, a detailed explanation about the design and the implementation of each module will be given.

2.1 Functional description

As it has been said in the introduction chapter, this project consists in the implementation of a PSoC-based DAS system for teaching purposes. For this reason the system has the functions of a DAS and also the capability to generate some test signals. These functions are: getting a signal from outside through different types of ADCs, generating test signals through a DAC, showing some basic information about the process by an LCD display and finally providing a way to interact and transmit the data through the RS232 interface of the board.

In order to choose one of them, the PSoC have a main code to process all the external user interactions[10]. This code activates the proper layer with the global resources defined and allows setting the desired parameters through the user modules libraries[11]. **Fig. 2.1** shows a diagram of the main code with the different system functions.

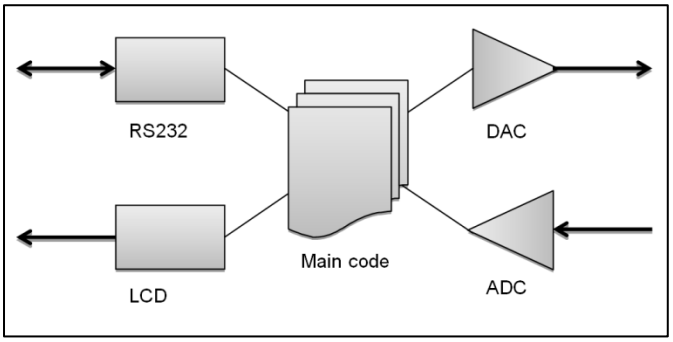


Fig. 2.1 System general functions diagram.

The functional description of these functions is explained below.
First, the communication between the user and the system is made through the RS232 interface in the board. In order to achieve that, the UART user module is

[10] See the source code in the annex.
[11] See the user modules library functions used in the annex.

used in the PSoC. It is the responsible for transmitting the signal information gathered by the PSoC to the RS232 and then outside and of allowing the user to send the proper command for switching between the system functions.

The LCD shows information about the active function and the set parameters. Depending on the function it also shows information about the signal that is being acquired.

The ADCs are the main objective of the project. This part is composed by several ADCs types and configurations in order to compare their results and to study the main aspects of signal acquiring. Resolution and sample rates can be selected, as well as the ADC type between integrator and SAR. These configurations can be switched from outside, and their results are sent out to make the appropriate comparisons. Also this part includes a dual ADC in order to test the response of analog circuits external to the PSoC. In this case, two signals are acquired at the same time by the two ADC present in the dual ADC. In this case the resolution and the sample rate are fixed.

The DAC allows obtaining test signals to do the ADC comparisons by the generation of AC and DC signals. Amplitude and frequency can be set. It provides some autonomy to the system because is not necessary the use of external signal generators.

The main code makes possible to switch between these functions. It depends on the RS232 communication. When a command is received from the user, the code goes to the function called, sets the parameters desired, or loads the proper layer. Also, it is responsible for transmitting the signal information through the RS232.

All functions of the system are provided by the PSoC, through the different user modules loaded. So, the PSoC is the principal process unit in the DVK board, the rest of the board only provides connectivity between PSoC ports as well as inputs and outputs such as the RS232 transceiver, the potentiometer or the LCD display.

The design and implementation of these functions are explained below.

2.2 Global resources

As it has been explained before, the most important global resources for the project are the clocks and the reference voltages.
They define the internal clock of the analog and the digital blocks and the reference voltage of the analog system. Consequently, parameters of the user modules such as ADCs sample rates and input ranges, DAC frequency or UART bitrate are also defined.

These global resources are set by the V_{cc}, Ref Mux and clocks parameters. V_{cc} is set to 5 V. It corresponds to the PSoC power supply, so it is always the same value. Ref Mux is always set to $V_{dd}/2$ $\pm V_{dd}/2$. They give a ground of 2,5 V and an input range of ±2,5 V to the ADCs. Also, it determines the DAC output range to 0 to 5 V. The clocks parameters used are SysClk, VC1, VC2 and VC3. They are taken as follows:

- SysClk is set to 24 MHz. It corresponds to the PSoC CPU clock, so is always the same value.
- VC2 is divided down from VC1, and this is divided down from the SysClk. They are combined in order to obtain the sample rates for the converters. They change for every overlay loaded in order to provide the sample rate desired.
- VC3 can be used only for digital blocks, so it is used as a fixed resource for digital blocks-based user modules that are always active, for example UART and the counter. It is derived directly from the SysClk to be independent from the variable VC1 or VC2 parameters.

2.3 RS232 transmission

The RS232 transmission is implemented by the UART user module, which needs two communication type digital blocks, one of them for receiving and the other for transmitting. Finally, these blocks are connected to the external pins through the digital interconnect rows.

Fig. 2.2 shows the UART blocks and the way they are connected to the outside.

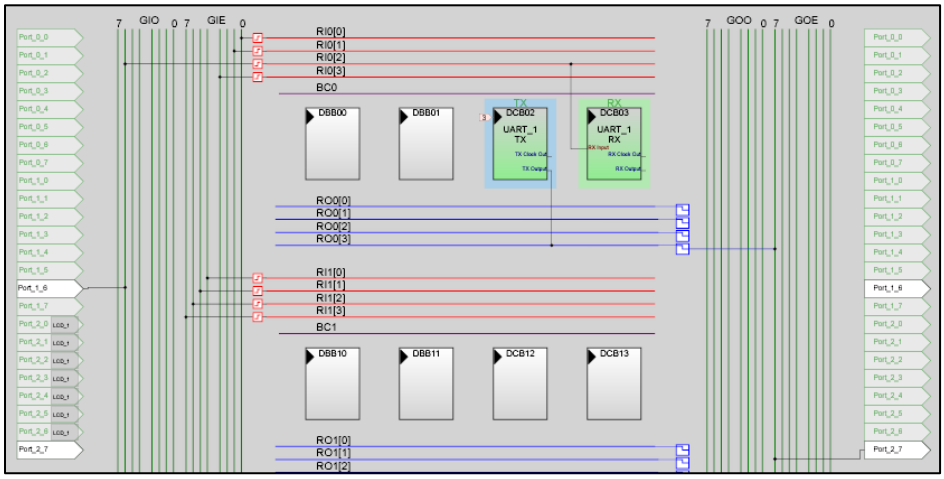


Fig. 2.2 UART digital blocks.

The UART user module is located in the base layer in order to be active at any moment because it transmits data and listens for a command continuously. Its parameters set are a bit rate of 115 384 bps and none parity; a start bit, a stop bit and 8 data bits are used as default set. The bit rate has been chosen in order to provide enough bandwidth for transmitting the fastest ADC, which is 7 500 samples per second at 7 bits per sample that it corresponds to a 52 500 bps of output. A more accurate bit rate isn't possible as it has discrete values because it is divided down from other clock, it can't be finely adjusted and in addition, it has to match with a preset bit rates list[12]. The clock used to synchronize the UART module and to provide the bit rate is the VC3 set directly from the SysClk. No interrupts are used with the UART user module.

2.4 LCD

The LCD is represented by the LCD user module, which no needs any digital or analog block to be implemented. The user module only includes the LCD parameters to be set. It allows setting the pin port where the display is connected. **Fig. 2.3** shows the pins used for the LCD display.

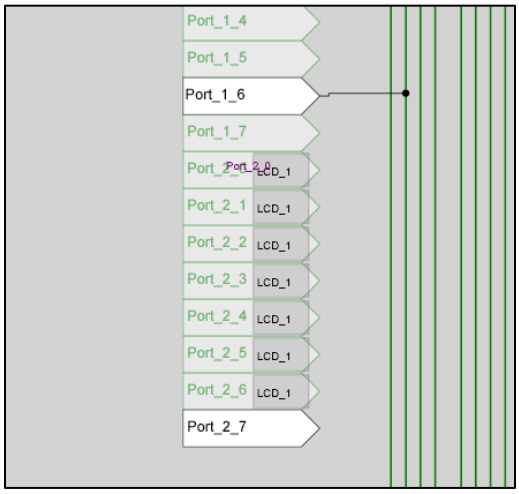


Fig. 2.3 Pins used by the LCD user module.

It is located in the base layer in order to be accessible at any moment because is the way to show the PSoC state. The LCD user module doesn't need any interruption and clock signal for running.

[12] Windows recognizes the serial port as a COM device with a preset bit rates list. The PSoC bit rate has to match with it.

2.5 ADC

The ADC function offers acquiring a signal through two ADC types: SAR and integrating, and testing analog circuits through the dual ADC, composed by two integrating ADC.

The SAR ADC compares the input signal to a voltage level produced by an internal DAC. Depending on if it is higher or lower, the DAC voltage is incremented or decremented and the corresponding bit is added to the digital output. This type of ADC usually has a medium-high resolution and sample rates under other types of ADC. Its strength is its low power consumption and its size.

The integrating ADC uses an integrator to convert an input voltage into its digital representation. The one provided by the PSoC is algorithmically equivalent to the dual-slope ADC architecture. The conversion process of this architecture consists in two phases, the run-up and the run-down. In the run-up phase, the input signal voltage is integrated during a predetermined period of time. In the run-down phase, the input switches to a known reference voltage and a counter measures the period of time that the integrator takes to return a zero value. Finally, the input voltage is calculated from the run-up phase period of time, the reference voltage and the measured run-down phase period. This ADC type can achieve high resolution trading off its speed, and as consequence decreasing its bandwidth. Other advantages are accuracy, a good noise performance and the frequency rejection capability.

The ADC function is implemented by three different user modules: SAR6, ADCINCVR and DUALADC. Any ADC user module has to acquire their signal through a PGA user module of unit gain.

2.5.1 PGA

The PGA user module provides the signal to the ADC from outside. It also allows amplifying the input signal by a predetermined selected value of 1, 2 or 8 in order to optimize the ADC resolution. To do it, the Reference parameter is set to V_{ss} in order to amplify ranges from 0 V to $V_{dd.}$

This user module needs a continuous time type analog block to be implemented. It is connected to the analog column multiplexers, which drive the signal from the input pin to the proper block, and to the analog column clock to be synchronized. **Fig. 2.4** shows this user module. As the PGA is common to the all ADCs and it can

be always active, is located in the base layer. No interrupts are needed for running this user module.

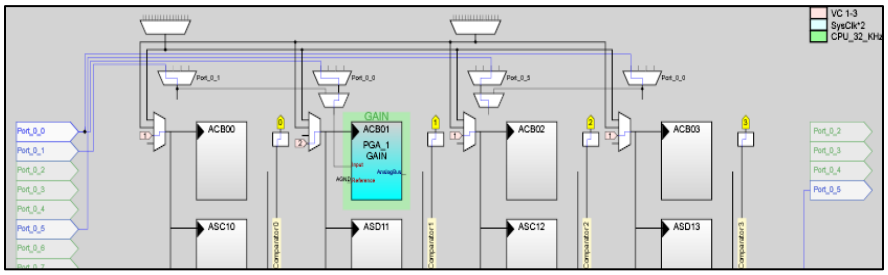


Fig. 2.4 PGA analog block.

2.5.2 SAR6

The SAR6 user module only needs a switched capacitor type analog block to be implemented. Its source signal is provided by the PGA and the clock signal is provided by the analog column clock. **Fig. 2.5** shows the SAR6 user module.

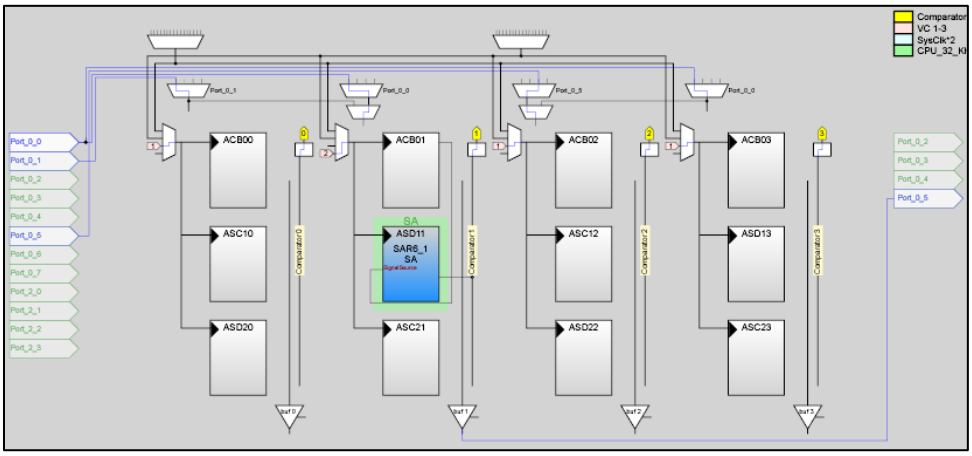


Fig. 2.5 SAR6 analog block.

This user module is located in an overlay in order to be activated or inactivated when desired. SAR6 has a resolution of 6 bits and a 2's complement value data format predetermined[13]. Sample rate is set to 3 900 samples per second. It is given by the VC2. No interrupts are used to run this user module.

[13] See annex: Data format used by the user modules.

2.5.3 ADCINCVR

The ADCINCVR needs more than one block to be implemented. On one hand, a switched capacitor analog block is needed for getting the signal from the outside. On the other hand, three digital blocks are needed to obtain the digital value: one for a counter and two to form a 16-bit PWM. The source signal for the analog block is provided by the PGA, and its clock signal is provided by the analog column clock. For the digital blocks, which also need to be synchronized, is done through the counter source clock input. **Fig. 2.6** and **Fig. 2.7** show the ADCINCVR user module.

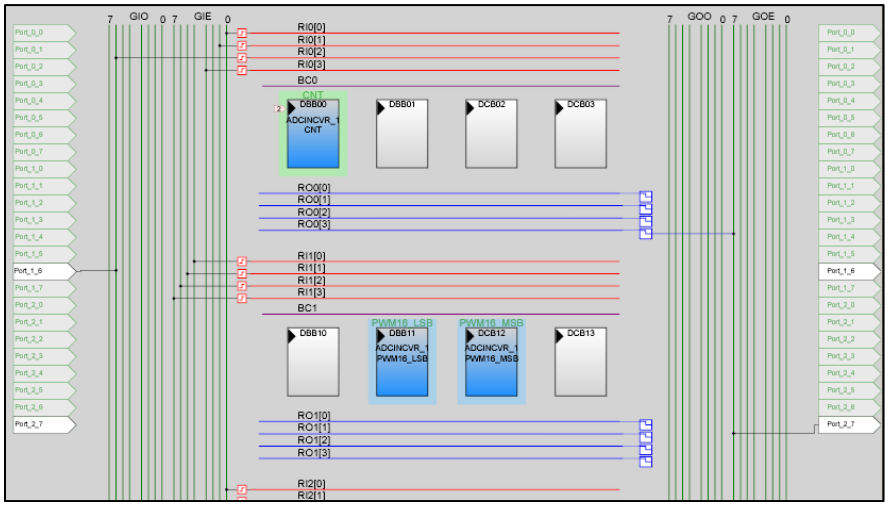


Fig. 2.6 ADCINVR digital blocks.

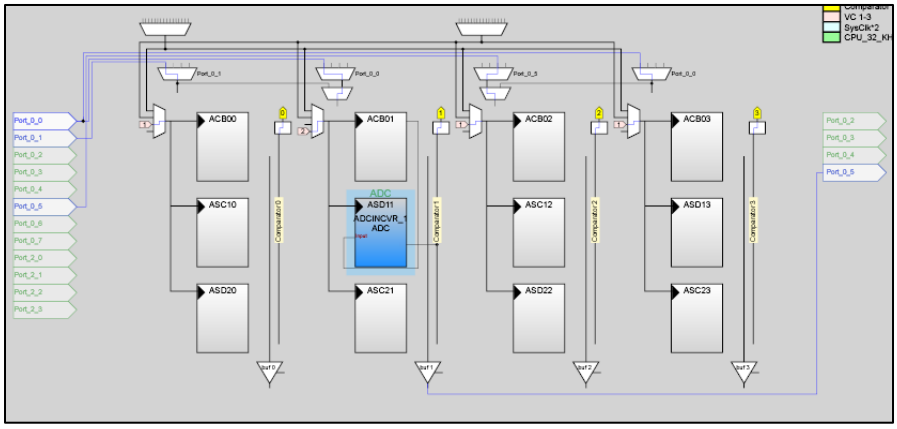


Fig. 2.7 ADCINCVR analog block.

The ADCINCVR is located in four overlays in order to be activated or inactivated when desired. These overlays have different configuration. They are explained below. The first overlay provides a selectable resolution of 7, 10, and 13 bits with a sample rate of 6, 50 and 400 samples per second respectively. These resolutions can be selected through the properly function from the user module library. The sample rates change because they depend on the resolution and the analog column clock, which is given by the VC2 clock, fixed to the layer loaded.

The second, third and fourth overlays have the same fixed resolution, set to 7 bits, and only the sample rates change from each other, which are 2 000, 5 000 and 7 400 samples per second respectively. These sample rates are also given by the VC2 clock[14]. The ADCINCVR user module needs that the interruptions are enabled in order to generate the digital value for the current sample. Also, it is needed for detecting the end of the sample conversion[15]. All the four overlays return the sample value in an unsigned data format[16].

2.5.4 DUALADC

The DUALADC user module consists of two ADCINCVR because it performs two conversions at the same time. It needs two switched capacitor analog blocks and four digital blocks to be implemented. The analog ones in order to acquire the two different signals and the digital ones in order to obtain the two digital values: two for two counters and two more to form the 16-bit PWM.

Each of the two analog blocks must be in a different column and can't share a column with another switch cap block that connects to the comparator bus. They are connected to two different PGA and their clock is provided by the analog column clock. The two counters may be placed in any digital block, but the pulse width modulator has to be placed in any two consecutive digital blocks. They are synchronized through the first counter source clock input. **Fig. 2.8** and **Fig. 2.9** show the DUALADC user module.

This user module only has a configuration, so is placed in one overlay. It has a fixed resolution of 10 bits and a sample rate of 1 000 samples per second, given by the VC2 clock. The DUALADC user module needs that the interruptions are enabled in order to generate the digital value for the current samples. Also, it is needed for detecting the end of the samples conversion. It returns the sample value in an unsigned data format[17].

[14] The way to change them is explained in this section: How the ADCs sample rates are obtained.
[15] See annex: How ADCINCVR makes the conversion.
[16] See annex: Data format used by the user modules.
[17] See annex: Data format used by the user modules.

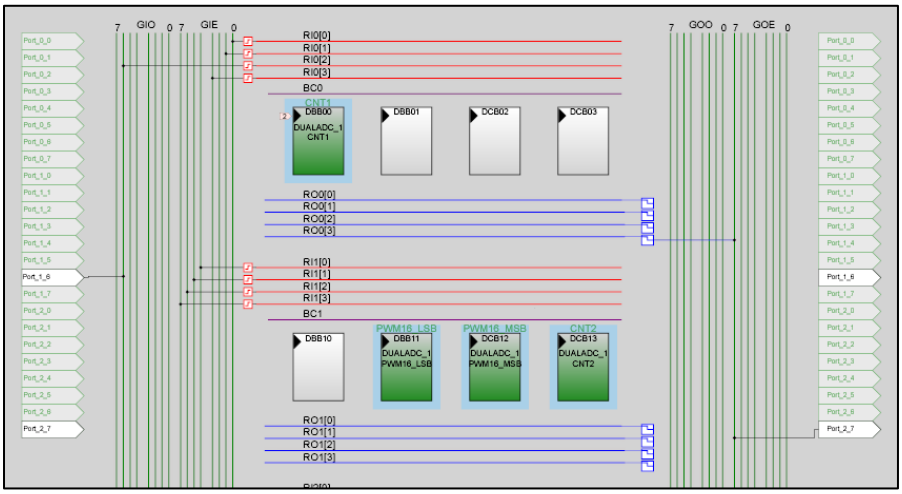


Fig. 2.8 DUALADC digital blocks.

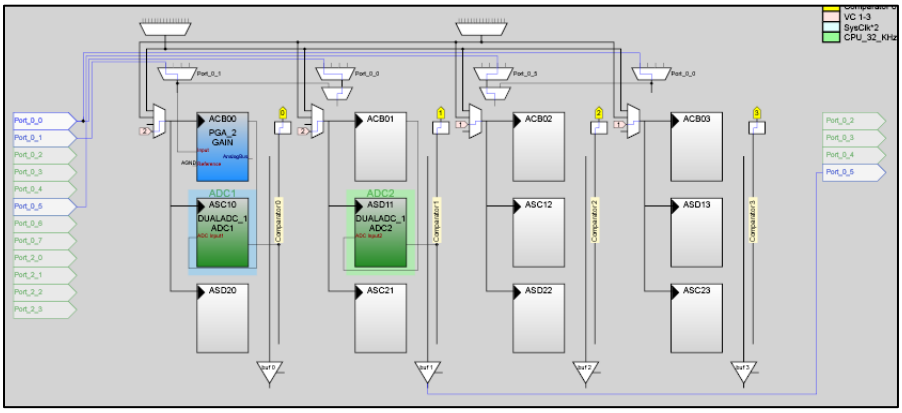


Fig. 2.9 DUALADC analog blocks.

2.5.5 How the ADCs sample rates are obtained

The ADCs user modules analog block has an internal clock in order to acquire the samples. It needs an external clock signal in order to synchronize the internal clock, as well as any user module implemented by analog blocks. This external clock is provided by the analog column clock, which is obtained from the system main clock SysClk through the different variable clock used for this purpose: VC1 and VC2. So, the sample rate for these peripherals will be given by these clocks.

Apart from this, depending on the ADC type, there are other parameters that influence in the sample rate calculation, they are explained below.

ADCINCVR

The ADCINCVR sample rate (SR) is given by the DataClock (DC), the resolution (N) and the CalcTime (CT) parameters, as the equation **(2.1)** shows.

$$SR = \frac{DC}{2^{N+2} + CT} \tag{2.1}$$

DataClock is the name for the analog block internal clock, which is actually composed by two clocks, ϕ_1 and ϕ_2[18], with a frequency of one fourth the analog column clock frequency. The resolution gives the integrate time and the CalcTime defines the time to do the result calculations, which can be selected from a minimum required in order to optimize the sample rate. This minimum value can be calculated as follows in the equation **(2.2)** and its units are DataClock cycles. **Table 2.1** shows the values used for these parameters in each ADCINCVR.

$$CT \geq \frac{180 \cdot DC}{CPU\_Clock} \tag{2.2}$$

Table 2.1 Parameters values used for each ADCINCVR.

User module	SysClk [MHz]	VC1 [Hz]	DC (VC2) [Hz]	N [bits]	CT_{min}	CT	SR [sps]
ADCINCVR_1	24	1 846 153	205 128	7	1,53	2	399
ADCINCVR_1	24	1 846 153	205 128	10	1,53	2	50
ADCINCVR_1	24	1 846 153	205 128	13	1,53	2	6,25
ADCINCVR_2	24	2 000 000	2 000 000	7	15	488	2 000
ADCINCVR_3	24	3 000 000	3 000 000	7	22,5	88	5 000
ADCINCVR_4	24	4 000 000	4 000 000	7	30	31	7 366

DUALADC

The DUALADC user module is composed by two ADCINCVR analog blocks. By this reason, the sample rate calculation is done as the same way that for the ADCINCVR. The only difference is that the minimum CalcTime needed corresponds to 260 CPU cycles instead of 180 needed by the ADCINCVR. So, the equation is:

$$CT \geq \frac{260 \cdot DC}{CPU\_Clock} \tag{2.3}$$

[18] ϕ_1 and ϕ_2 are the phase clocks used to acquire and transfer the signal.

Table 2.2 shows the values used for the implementation of this user module.

Table 2.2 Parameters values used for the DUALADC.

User module	SysClk [MHz]	VC1 [Hz]	DC (VC2) [Hz]	N [bits]	CT_{min}	CT	SR [sps]
DUALADC_1	24	4 000 000	4 000 000	10	44	45	966

SAR6

The SAR6 sample rate is given by the SampleClock. It is the analog block internal clock name; which is actually composed by two clocks, ϕ_1 and ϕ_2[19], with a frequency of one fourth the analog column clock. On the other hand, the SAR6 needs six times the period of the sample clock to perform the sample total conversion. The equation **(2.4)** shows the SAR6 Conversion Time calculation.

$$Conversion\ Time = 6 \cdot \frac{1}{f_{sample\ clock}} = 6 \cdot 4 \cdot \frac{1}{f_{analog\ column\ clock}} \quad \textbf{(2.4)}$$

Finally, the sample rate corresponds to:

$$Sample\ Rate = \frac{f_{sample\ clock}}{6} = \frac{f_{analog\ column\ clock}}{24} \quad \textbf{(2.5)}$$

So, the values used for the implementation of the SAR6 are 93 750 Hz for the analog column clock and a sample rate of 3 906 samples per second.

2.5.6 ADCINCVR frequency rejection

The ADCINCVR user module is able to reject interferences added to the signal of interest due to its integrating architecture. To reject a frequency, the integration time must be equal to the interference integral cycle that corresponds to its period. It implies that the harmonics will be also rejected. To reject more than one interference and their harmonics the integration time must be an integral cycle of both interferences.

[19] ϕ_1 and ϕ_2 are the phase clocks that control each successive approximation step.

For this project, the first ADCINVR configuration layer has been designed in order to reject the 50 Hz power supply interferences, so the integration time must be equal to 20 ms. This parameter is given by the ADCINCVR DataClock and the resolution (N) as the equation **(2.6)** shows.

$$Integration\ time = \frac{2^{N+2}}{DataClock} \tag{2.6}$$

The selected resolution chosen is 10 bits, so the DataClock has to be 204,800 Hz. As this clock signal is given by the discrete parameter VC2, the nearest value achieved is 205 128,205 Hz, so the final integrate time will be 19,968 ms. Then, selecting the proper CalcTime, the ADCINCVR sample rate will be obtained as explained in the previous section.

2.6 Test of linear circuits

The system also has a function to generate some signals in order to do the ADC comparisons or to test analog circuits. This function needs two user modules to be implemented: a DAC and a counter. They are placed in a switched capacitor analog block and in a digital block, respectively. The DAC is connected to the analog out bus to output the corresponding signal voltage level, and its clock signal input is connected to the analog column clock. The counter is synchronized through its source clock input. **Fig. 2.10** and **Fig. 2.11** show these two user modules.

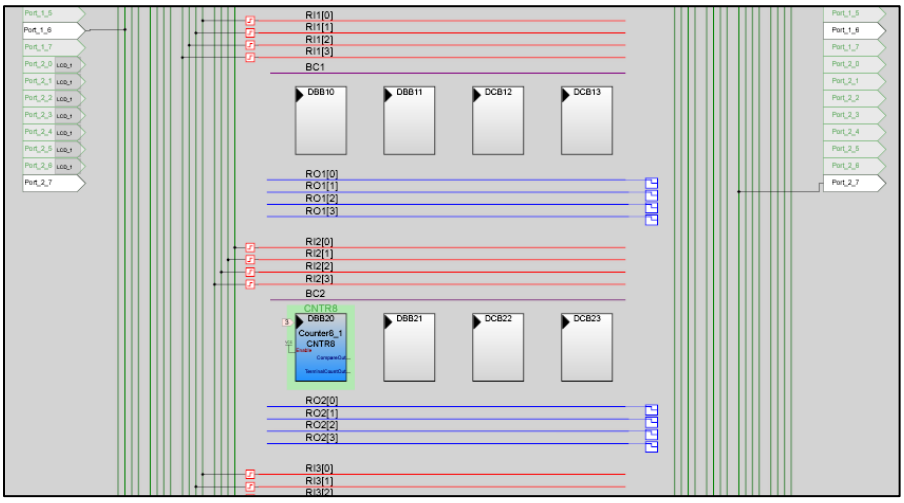


Fig. 2.10 Counter digital block.

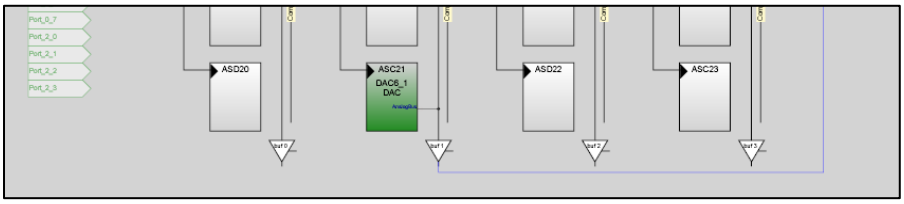


Fig. 2.11 DAC analog block.

These modules are located in the base layer in order to be available at any moment. The signals implemented are a sinusoidal with three selectable frequencies and a continuous one. Both of them have a $V_{dd}/4$ to V_{dd} amplitude, selectable in steps of ¼. A 64-samples lookup table is needed for generating the sinusoidal signal. Every time that the counter arrives to the value of period set it sends to the DAC the next sample from the table and this sets the voltage level corresponding to the sample. The frequency of the sinusoidal signal is 57, 100 and 200 Hz. It is defined by the internal clock of the counter, the period of the lookup table and the period of the counter, which can be selected to obtain the sinusoidal frequency desired[20].

The ground and the amplitude of the generated signal, either continuous or sinusoidal are given by the reference voltage set for the analog system in the global resources. The internal clock of the counter is sourced by the VC3, and the DAC is sourced by VC2. The DAC user module doesn't need any interruption for running. The counter needs an interruption for generating the sinusoidal signal. The data format for the DAC input is set to offset binary[21].

2.6.1 How the DAC generates the signals

As explained through the report, the DAC6 user module is used to generate continuous and sinusoidal signals. This user module is implemented by an analog block.

It has an internal clock to update the output analog voltage level with the current digital code value. It needs an external clock signal in order to synchronize the internal clock. This external clock is provided by the analog column clock, which is obtained from the system main clock SysClk through the different variable clocks used for this purpose: VC1 and VC2. These variable clocks haven't a concrete value for the DAC because they are used by the last loaded ADC overlay and the DAC is located in the base layer. The reason is that the DAC function must be available at any moment to provide the signal and so, it can't have an own overlay

[20] See annex: How the DAC generates the signals.
[21] See annex: Data format used by the user modules.

for setting these parameters. Also, the Counter8 user module is needed for generating the sinusoidal signal. This user module provides a counter that sets the proper sample from a look up table in the DAC6. This look up table looks like as follows.

```
int SINtable[SIN_PERIOD] = {31, 33, 36, 39, 41, 44, 46, 49, 51, 53, 55,
56, 58, 59, 59, 60, 60, 60, 59, 59, 58, 56, 55, 53, 51, 49, 47, 44, 42,
39, 36, 33, 31, 28, 25, 22, 19, 16, 13, 11, 9, 7, 5, 3, 2, 1, 0, 0, 0, 0,
1, 2, 3, 4, 6, 7, 10, 12, 14, 17, 20, 23, 26, 29};
```

The Counter8 has an internal clock to increment the counter and a Period parameter that defines the counter maximum value. When the counter reaches the Period parameter value set, an interruption is triggered and the next value in the look up table is set in the DAC6. Its internal clock and the counter period ($T_{counter}$) give the Counter8 sample rate (SR) as the equation **(2.7)** shows.

$$SR = f_{internal\ clock} \cdot \frac{1}{T_{counter}} \tag{2.7}$$

So, the sinusoidal signal frequency f_{out} is given by the Counter8 sample rate and the look up table number of samples (N). The equation **(2.8)** shows the relation between these parameters.

$$f_{out} = Sample\ Rate \cdot \frac{1}{N} \tag{2.8}$$

The internal clock is synchronized through the variable clock VC3. This clock is also needed for the UART user module, so the value is set by this one because is more restrictive in order to set the bitrate of the RS232 transmission. **Table 2.3** shows the values used to obtain the three sinusoidal frequencies used in the system.

Table 2.3 Values used for generating the three sinusoidal signals.

$f_{internal\ clock}$ [Hz]	$T_{counter}$ [cycles]	SR [sps]	N [samples]	f_{out} [Hz]
923 076,923	255	3 619,909	64	56,56
923 076,923	144	6 410,256	64	100,16
923 076,923	72	12 820,512	64	200,32

For the sinusoidal signal generation, the value of the variable clocks VC1 and VC2 may limit the sinusoidal frequency if the DAC6 internal clock was slower than the Counter8 sample rate. This isn't the case because the slowest DAC6 internal clock is 205 128 Hz and the fastest Counter8 sample rate is 3 619,909 Hz. This means that the DAC6 analog block always has a new sinusoidal sample to output, and that the frequency is respected.

2.7 PSoC dynamic reconfiguration

PSoC dynamic reconfiguration has been used to reuse digital and analog resources in order to provide all the proposed functions. It has been done as follows.

Blocks from functions that are always activated, such as UART for communicating, LCD for displaying information and PGA for giving the input signal to the ADCs, and blocks from functions that may be used parallel to other functions, such as the DAC and the counter for generating signals, are located in the base layer, which is always loaded. Other blocks from functions that are not always active are located in different overlays. The main code loads the proper overlay corresponding to the function selected. Blocks that are always loaded must use different parameters than the blocks that are activated and inactivated in order not to interference each other. The global resources parameters used by the blocks that are always loaded are Ref Mux used by the PGA and VC3 used by the counter and the UART. They must have always the same value, so these parameters must remain unaltered in each overlay. The global resources parameters used by the blocks that are activated and inactivated are VC1 and VC2, used by the different ADCs configurations. They change its value depending on the configuration selected, so these parameters must have its proper value in each overlay.

Global resources are defined in each layer, so the last loaded overlay will reset the current parameters. By this reason, these parameters must be set carefully to avoid that the functions which are always loaded remain without configuration.

2.8 Detected PSoC limitations

The first detected limitation is referring to the clock resources. All the digital or analog blocks need a clock signal to be synchronized. They are provided by a limited number of sources, which are basically the VC1, VC2 and VC3 parameters from the global resources. In this project, the sample rate of the different ADC configurations is given by a combination of these clock sources in order to obtain a range of different sample rates.

The need to have some user modules always active requires not having some resources available for the rest of the user modules. The UART for communicating, the LCD for displaying information or the counter for generating the AC test signals require that VC3 global resource clock parameter has always the same value in order to this functions work properly at any time. This implies that ADC obtains its clock source only from the VC1 and the VC2, and consequently to have a more limited range for their sample rates. Also, another consequence is that the DAC generated AC signals frequency is limited by the UART bit rate, which predetermines the VC3.

Another limitation is given by the CPU processing time. The user modules implementation has been done through the libraries provided by Cypress. They consist of a group of C++ functions in order to control and configure the user module. These functions, formed by different assembly instructions, need for some CPU cycles to be executed. Depending on the C++ function complexity, the number of CPU cycles may vary. When the ADC sample rate is increased, the time between two adjacent samples is decreased. This means that the sample processing is limited in time and therefore, the number of functions to be executed after getting the sample must be the smallest as possible in order to, for highest sample rates, leave the CPU free to acquire the next sample. Acquiring a sample, displaying it on the LCD and transmitting it through the RS232, hold the CPU for a time that limits the sample rate. By this reason, for all the ADC configurations except the ADCINCVR_1, the LCD displaying has been needed to avoid. Also, for ADCINCVR_3, ADCINCVR_4 and SAR6, the sample transmission is done in blocks of 100 samples. For it, a predetermined number of samples are acquired continuously before transmitting them through the RS232. So, the sampled signal is transmitted in blocks. This avoids the displaying and transmitting CPU tasks between samplings.

Also, a detected limitation is that SAR6 can't acquire properly 100 and 200 Hz AC signals generated by the DAC. The reason is that due to its architecture, it requires 100 % of the PSoC CPU during the sample conversion[22] and the DAC needs a CPU interruption to generate the AC signal.

[22] As explained in the Cypress Application Note 2239: Analog – ADC selection, (http://www.cypress.com/?rID=2641)

3. USER INTERFACE

The last stage of the system is a personal computer with a software that provides a user interface for the system.[23] It is used to show the signal information and to interact with the system. The next section explains its different functions in detail.

3.1 Functional description

The user interface can be divided in two different parts: the configuration area, which is composed by a tab for the RS232 communication and a tab for the system control panel, and the visualization area.

The RS232 communication tab allows setting the RS232 communication between the system and the personal computer. When all the communication parameters [24]are set the Open communication option is enabled. When the communication is opened, the Close communication option becomes visible in order to close it. Also, an option to save the set parameters is provided. **Fig. 3.1** shows a screenshot of the RS232 communication tab.

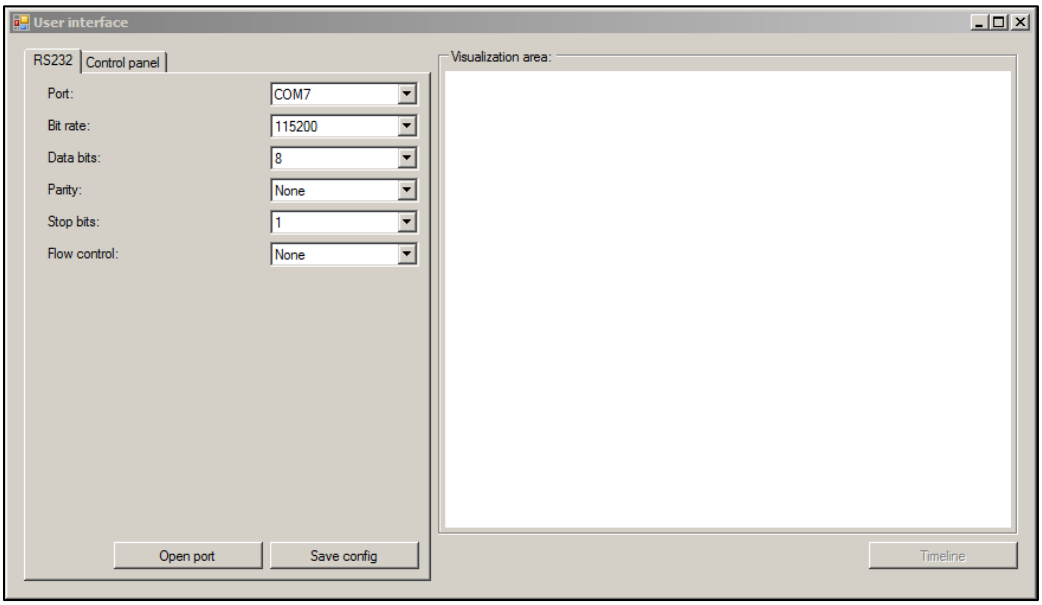


Fig. 3.1 RS232 communication tab screenshot.

[23] See the source code in the Annex.
[24] Parameters: 115 200 bps, 8 data bits, no parity, 1 stop bits and no flow control.

The system control panel tab is enabled only if the communication is opened. This tab allows configuring the system and powering on and off. It has two separated parts: the signal generator and the ADC selection. The first one activates the signal generation, allows selecting the signal type and setting its frequency and amplitude. The second one allows setting the type of converter, the resolution, the sample rate and the number of samples to acquire. Also, there is an option to select the scale for the signal acquired. With this function the resolution of the ADCs is optimized for small signals.

When the parameters corresponding to the ADC selection are set, the button Start is enabled and the process of acquiring the signal can be started. In this moment, the Stop button is enabled and all the parameters including the RS232 communication tab are blocked. The process can be stopped at any moment or it will be stopped automatically when all the samples indicated in the buffer size are get. **Fig. 3.2** shows a screenshot of the system control panel tab.

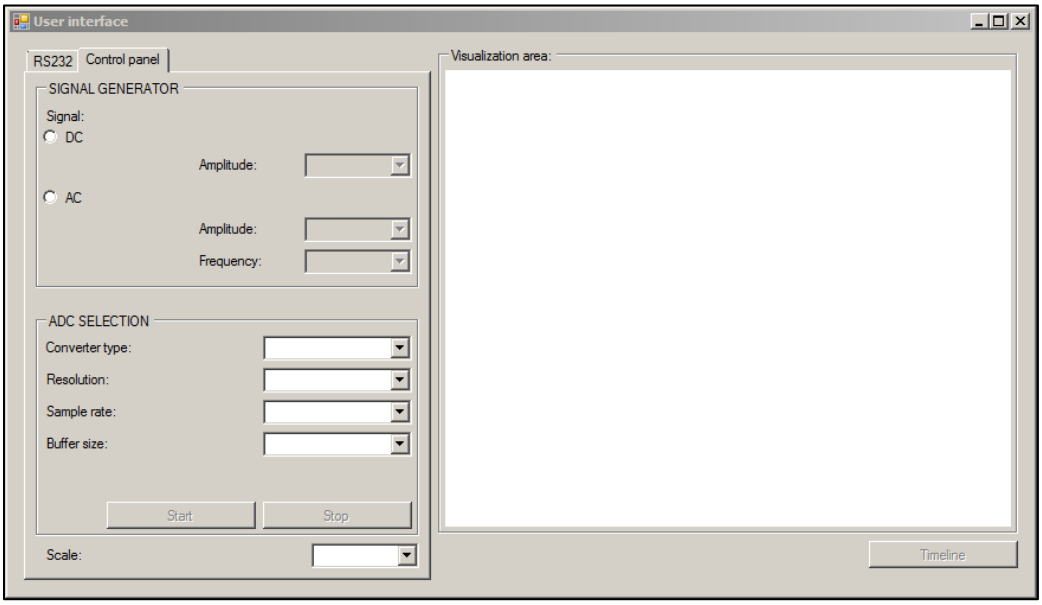


Fig. 3.2 System control panel tab screenshot.

When the process of acquiring the signal is finished, the signal is visualized along the time in the visualization area. The visualization area is the space from the interface where the signal is shown. Two types of visualizations can be selected: the signal along the time and a histogram of the acquired samples. The button under the visualization area allows switching between them. **Fig. 3.3** and **Fig. 3.4** show a screenshot with an example of these visualizations: a continuous signal represented along the time and its histogram, respectively.

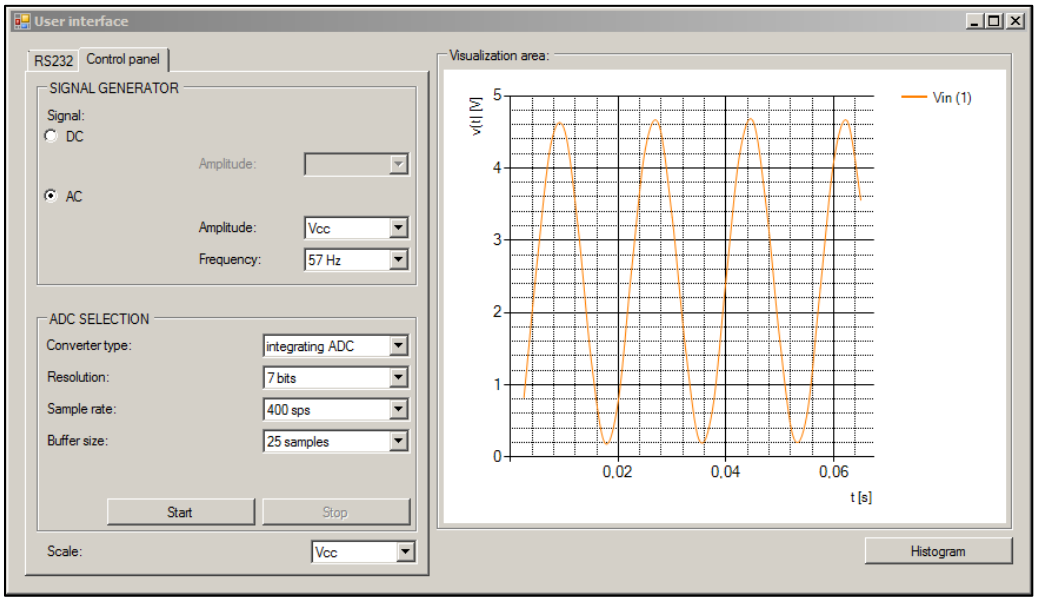


Fig. 3.3 Visualization area in time respresentation mode.

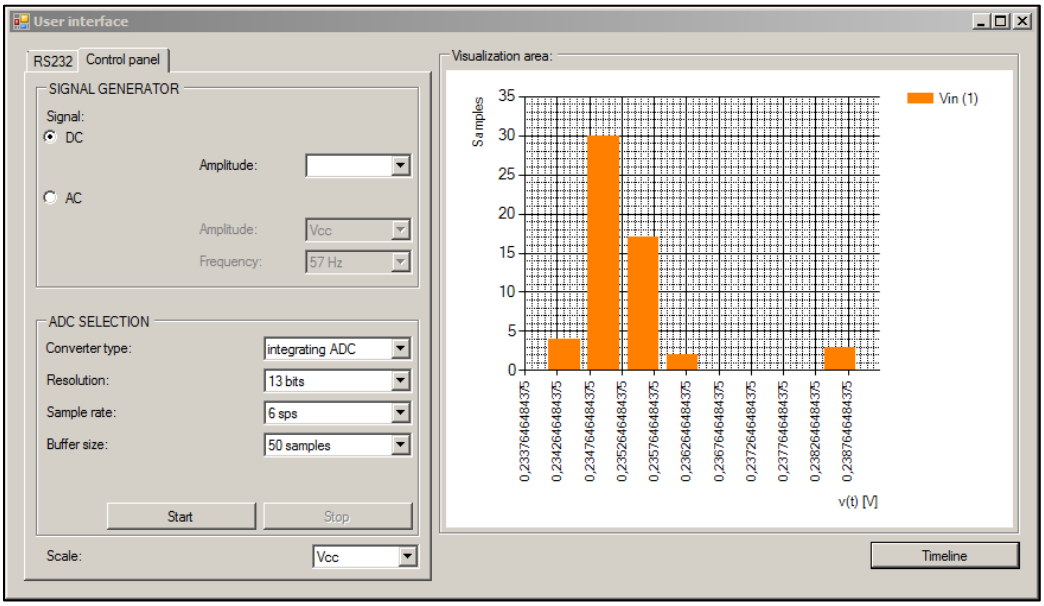
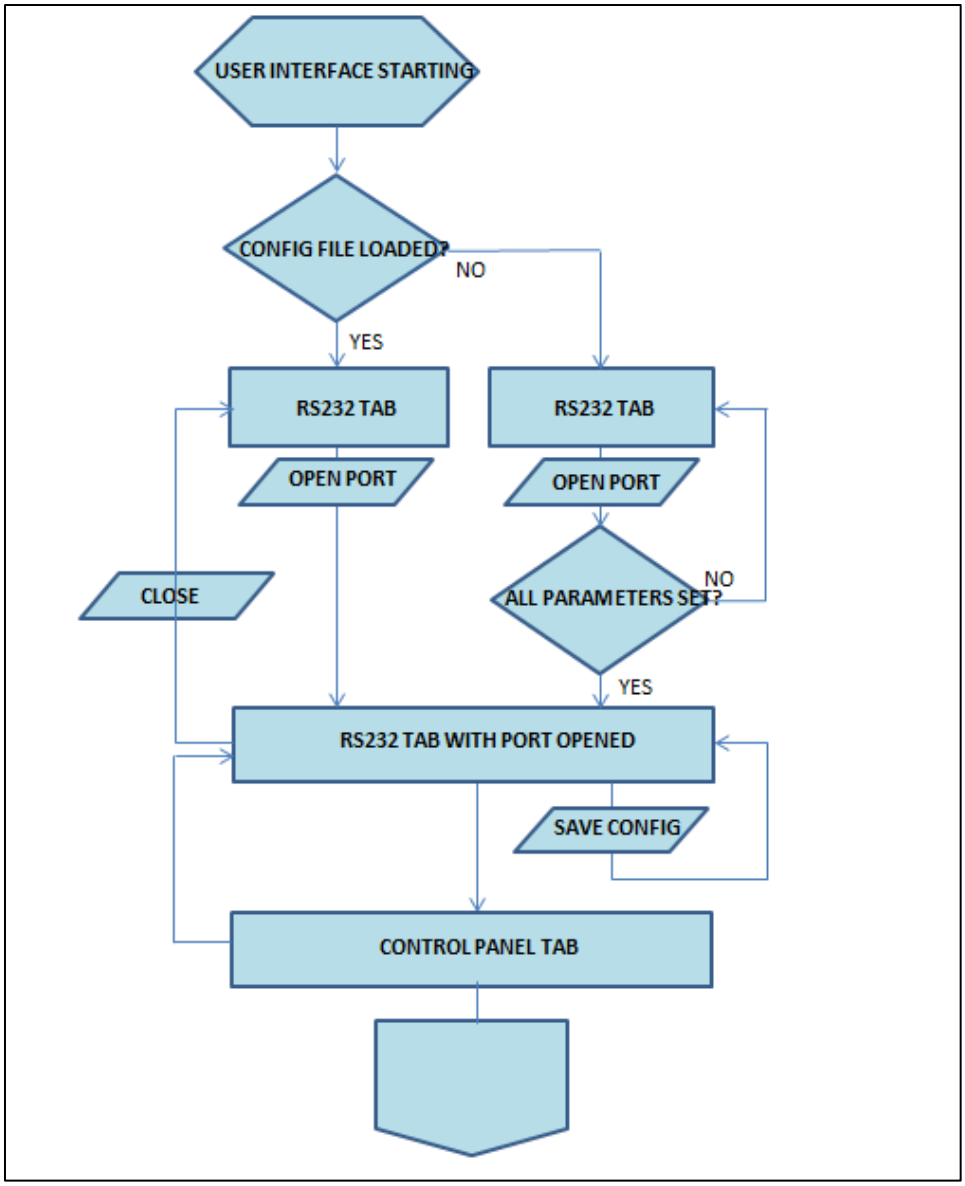


Fig. 3.4 Visualization area in histogram mode.

3.2 Implementation

The user interface is implemented in C# language for Windows. For the RS232 communication the COM serial ports from Windows have been used. C# provides a library to manage these ports in several ways.

For visualizing the signal along the time and the histogram, the MSChart library from Microsoft has been used. It allows several types of graphs and a lot of ways to configure them. **Fig. 3.5** shows the software flowchart.



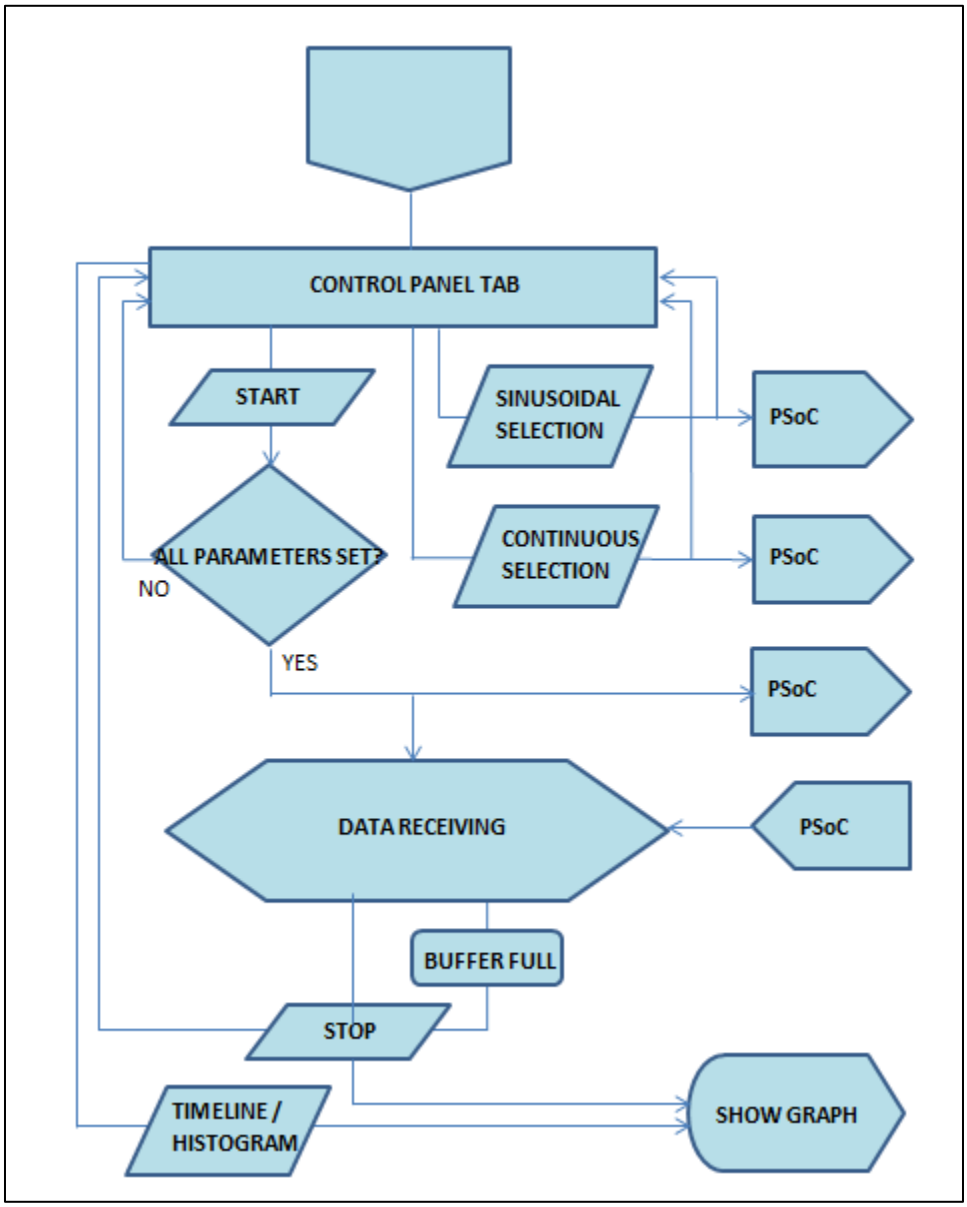


Fig. 3.5 User interface software flowchart.

4. EXPERIMENTAL RESULTS

4.1 Test of different ADC features

To observe the ADC linearity the 13-bits integrating ADC is used because its higher resolution is better to represent this characteristic. To see this characteristic, the ADC transfer curve is needed. It is obtained applying different DC voltage levels to the ADC input. The onboard potentiometer connected to the V_{cc} pin is used to obtain these DC voltage levels, previously measured with a tester of up to 200 mV scale.

The output code is compared with the ideal output (n) given by the equation **(4.1),** where V_{in} is the voltage level applied, FSR is equal to 5 V and N are the 13 bits.

$$n = \frac{V_{in}}{FSR} \cdot 2^N \tag{4.1}$$

The voltage level applied as well as its ideal output code (y-axe values) and its real output value (labels values) is given in the **Fig. 4.1**.

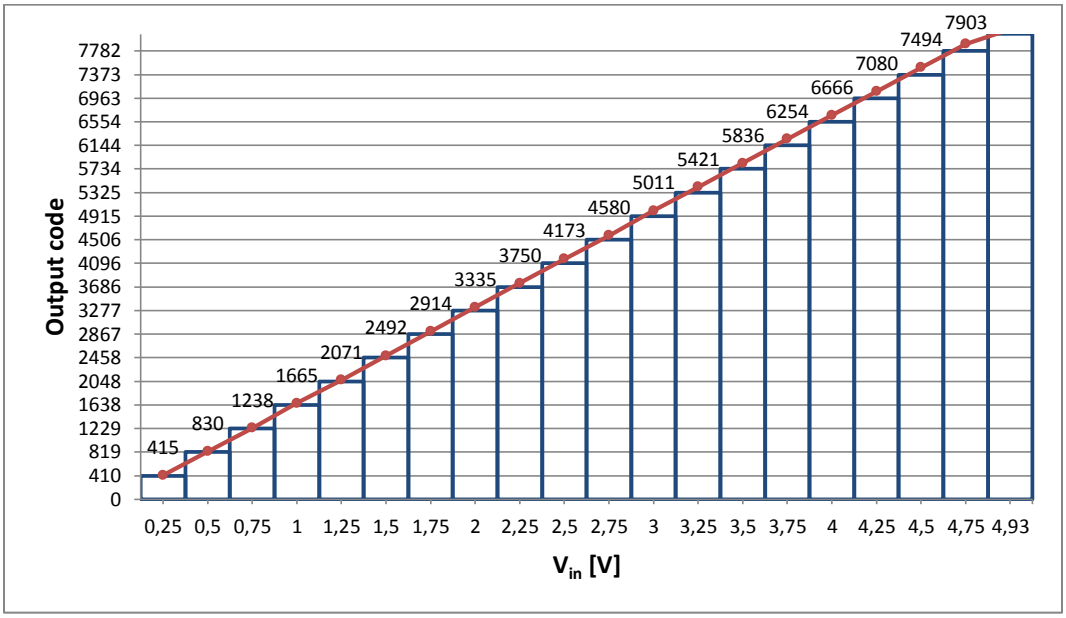


Fig. 4.1 ADC transfer curve.

To test the experimental dynamic range the input is connected to the board ground in order to measure the noise floor present in the device. Then, the noise RMS (x_{RMS}) is calculated from the histogram obtained. Finally, the experimental dynamic range can be obtained from this parameter.

To show the noise floor, the 13-bits integrating ADC is used with a sample rate of 6 sps and an integration time of 19,968 ms. The test is done with a buffer size of 1 000 samples. The noise floor histogram obtained is shown in the **Fig. 4.2**.

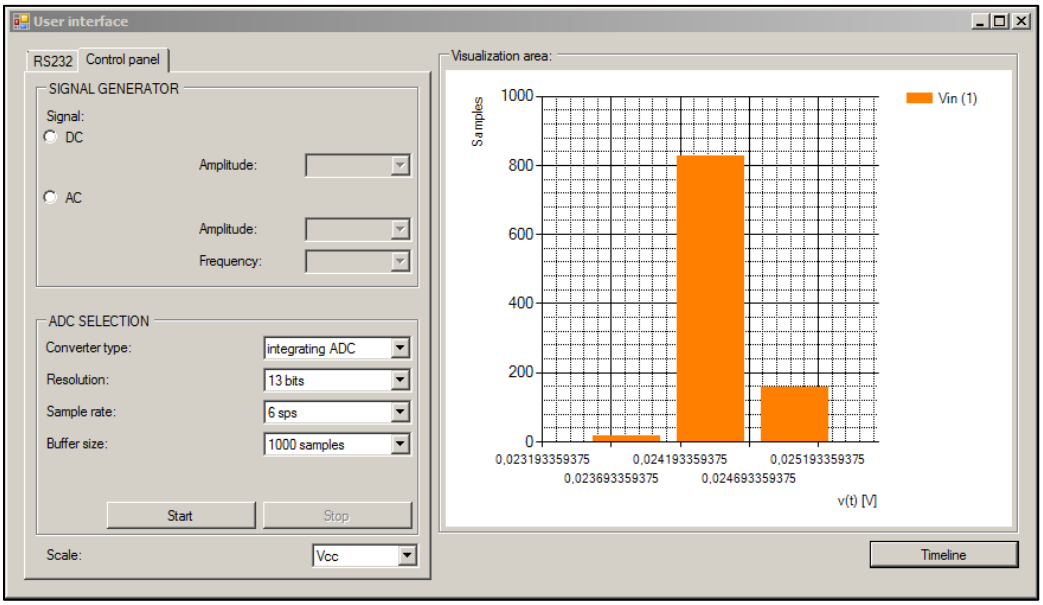


Fig. 4.2 Noise floor histogram for a 13-bits integrating ADC.

The histogram shows a Gauss distribution of media μ equal to 24,4 mV (value corresponding to the decimal code 40, obtained by the LCD display). It means that there is an offset between the ADC internal ground and the on board ground pin where the ADC input is connected. Also, a noise variation of ±1 LSB can be seen, equal to ±0,61 mV, value given by the resolution and the FSR which corresponds to 5 V.

An approximation of the $V_{n,RMS}$ noise value can be obtained from this distribution, applying the equation **(4.2)**. Where N are the total of samples acquired, corresponding to the buffer size and x_i is the voltage value corresponding to the different obtained codes. The $V_{n,RMS}$ value obtained is 1,337 mV.

$$V_{n,RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}$$

(4.2)

And the corresponding ADC dynamic range can be estimated through the equation **(4.3)**, where the $V_{in,RMS}$ is equal to 3,125 V, a value that corresponds to a FSR amplitude sinusoidal signal. The expected DR is equal to 67 dB.

.

$$DR = SNR = 20\log\frac{V_{in,RMS}}{V_{n,RMS}}$$

(4.3)

4.2 Interference rejection

This section aims to show the main power line 50 Hz interference rejection capability of the integrating ADC, the design to this purpose of which is explained in the section 2.5.6 ADCINCVR frequency rejection.

To do it, the AC signals generating system function is used in order to generate a 100 Hz AC signal, which is a proper signal for the test because it is a 50 Hz signal harmonic. Also, the integrating type ADC and the 10 bits resolution (with restrings the sample rate to 50 sps) are selected in the user interface, parameters that provide the proper integration time, equal to 19,968 ms. The samples acquired are 1 000 samples that gives the 20 seconds of acquisition time. The **Fig. 4.3** shows the acquired signal.

Only a signal approximated to a 2,5 V DC voltage level is shown. The reason is that the sinusoidal integration operation gives its average value, which corresponds to its DC voltage component or offset value. For this case, the generated AC signal reference voltage has a ground of $V_{dd}/2$, which is given by the system in order to generate both positive and negative semi-cycles. For a 0 V based-ground, the interference would be attenuated to the 0 V value.

Also, another test to check the rejection is to try to get some main power line interference leaving opened the ADC input and acquiring it with the SAR type ADC. If a main power line is near to the input the acquired signal looks like the **Fig. 4.4**.

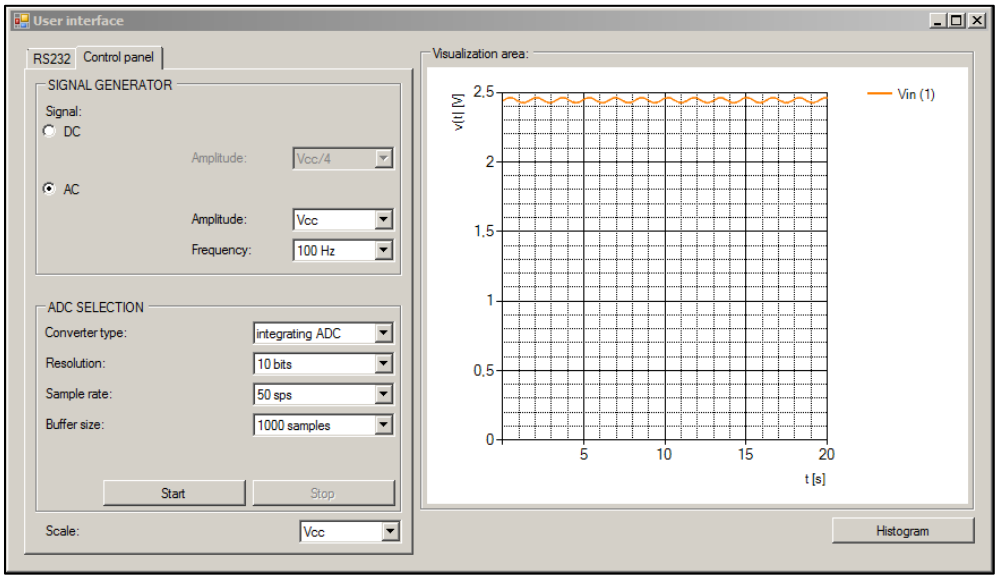


Fig. 4.3 Integrating ADC acquired signal.

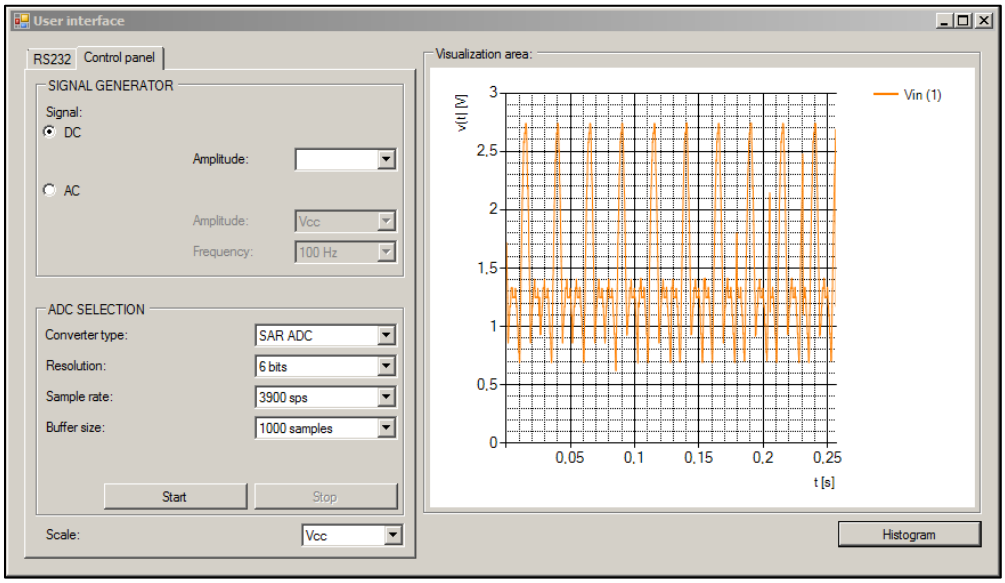


Fig. 4.4 User interface with the SAR ADC selected.

An interference with an approximated frequency of 50 Hz can be seen. The **Fig. 4.5** shows the input with the same conditions that the last test, but for an integrating ADC with 20 ms of integration time instead of the SAR type. The figure shows that the interference is not present.

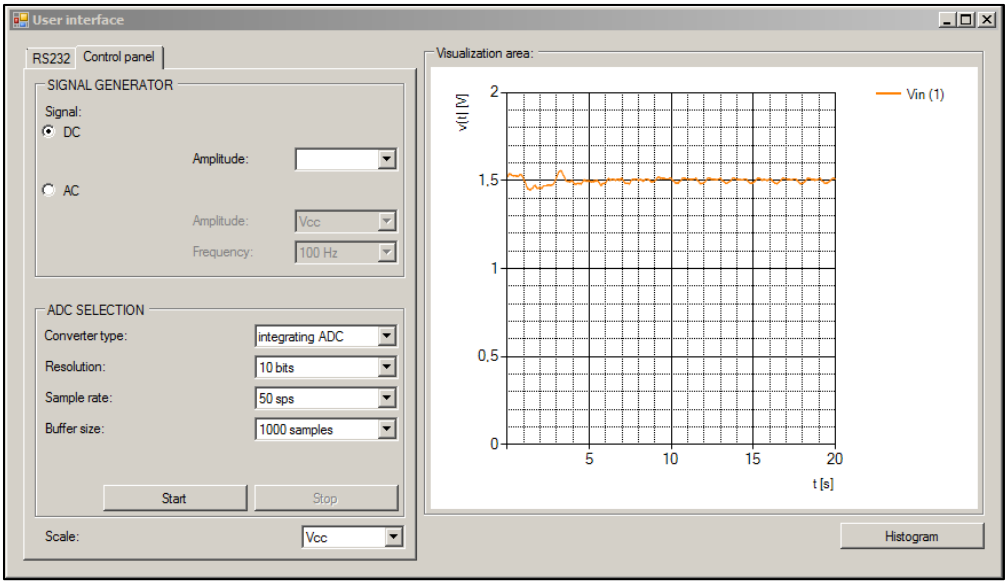


Fig. 4.5 User interface with the integrating ADC selected.

4.3 Test of DAC signal generation

This section shows the different signals that can be obtained from the DAC. The user can select between AC signals with an amplitude of 1,25, 2,5, 3,75 and 5 V and DC signals with a voltage level of 1,25, 2,5, 3,75 and 5 V. For the AC signal, the frequencies are 57, 100 and 200 Hz. To show it, the user interface has been used, although previously they have been already tested through an oscilloscope. Only the 100 Hz AC signal with an amplitude of 2,5 V will be analyzed in the section in order to not extend the report, the rest of the screenshots can be seen in the annex. The **Fig. 4.6** shows it.

An integrating ADC converter has been used to acquire the signal. The sample rate is set to 2 000 sps and the buffer size to 100 samples, which provide the shown capture of 50 ms, calculation given by the equation **(4.4)**.

$$Capture\ period = \frac{Buffer\ size}{Sample\ rate} \tag{4.4}$$

To calculate approximately the signal frequency the equation **(4.5)** can be used. The 5 cycles that can be shown in an approximate period of 44 ms are divided by this period, it corresponds to 113 Hz.

$$Frequency = \frac{Cycles\ captured}{Capture\ period} \tag{4.5}$$

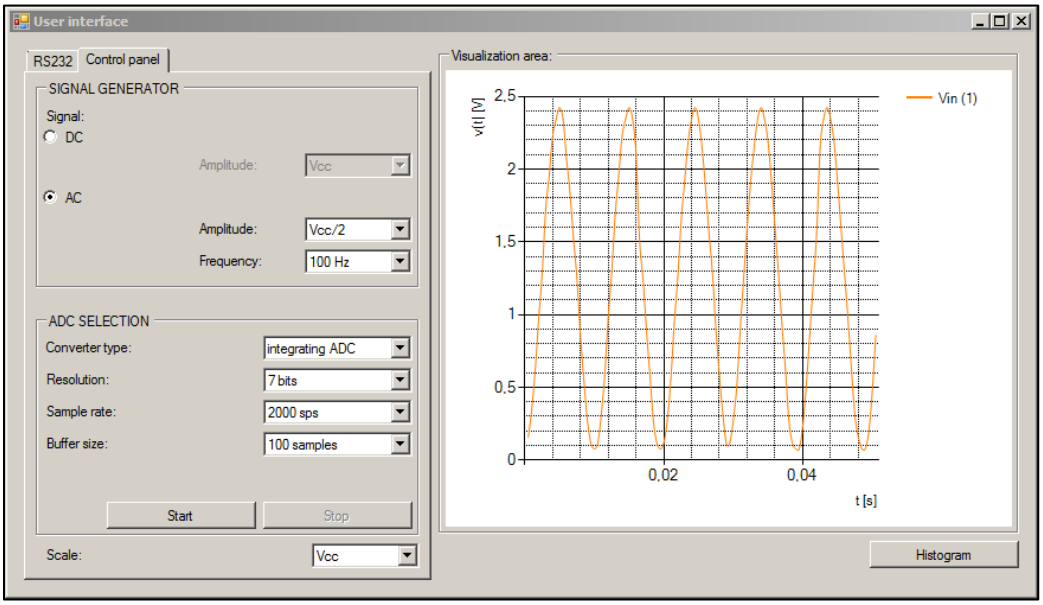


Fig. 4.6 100 Hz AC signal generation.

4.4 ADC and DAC working together

The ADC and DAC working together are tested through a RC circuit designed as a first order low-pass filter, as **Fig. 4.7** shows. In order to do this test, the dual ADC is needed for comparing the amplitude and phase of the two signals present in the filter. It is configured with the only parameters available that are 10 bits and a sample rate of 1000 sps.

The test consists in generating a 100 Hz AC signal with amplitude of 5 V. This signal is output from the DAC through the P05 PSoC pin to the circuit and to the P00 dual ADC input 1 pin. The circuit is formed by a resistor of resistance R and a

capacitor of capacity C, with the input V_{in} and the output V_{out}. The circuit output is connected to the P01 dual ADC input 2 pin.

This circuit configuration behaves as a first order low-pass filter, which attenuates the amplitude of the input signal f_c frequency to the 70,7 % of the maximum (the signal power is attenuated 3 dB) and changes its phase -45º. This frequency characterizes the circuit and is given by the resistance and capacitor values as the equation **(4.6)** shows.

$$f_c = \frac{1}{2 \cdot \pi \cdot R \cdot C} \qquad \textbf{(4.6)}$$

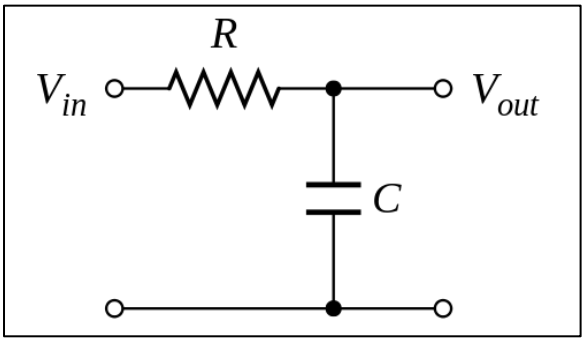


Fig. 4.7 RC filter schematic.[25]

In order to attenuate the 100 Hz AC signal, R is calculated through the equation **(4.6)** when C is fixed to 1 µF. Its value is to 1 591 Ω, but the nearest commercial resistor value is 1 500 Ω, with an experimental resistance of R = 1 493 Ω, so the final f_c is 106,6 Hz.

The **Fig. 4.8** shows the relation between the filter input signal $V_{in}(1)$ and the filter output signal $V_{in}(2)$.

[25] Source: http://es.wikipedia.org/wiki/Filtro_paso_bajo

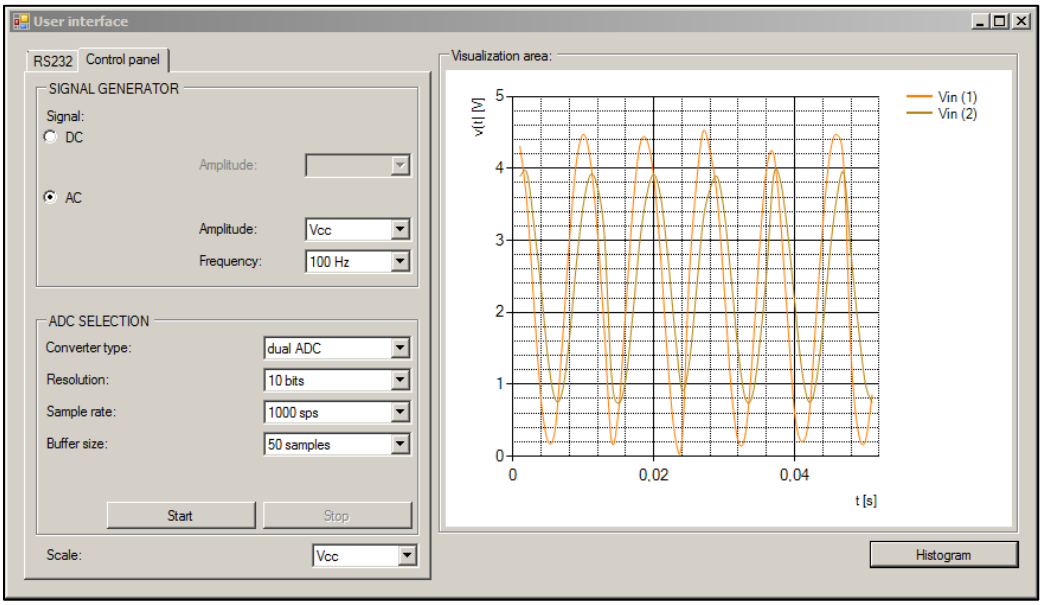


Fig. 4.8 Signals input in the dual ADC. $V_{in}(1)$ from the DAC and $V_{in}(2)$ from the filter output.

As can be seen, the filter output signal is dephased near to the expected 45º and the maximum voltage is decreased from 4,5 V to 3,8 V, which corresponds to 84 % of attenuation. These values are not exactly the expected due to mainly the ADC internal impedance which hasn't been taken into account.

5. CONCLUSIONS

The project main objective has been achieved; the system has the capabilities of acquiring a signal allowing a variety of configurations including the ADC type selection. Also generating AC and DC signals is provided. The interface allows interacting with the system and showing the results. Also, PSoC advantages and some limitations have been detected.

With the PSoC families of chip, Cypress provides a powerful tool in order to implement a high variety of devices; although the main advantage is the chip capability to be reconfigured into several preloaded systems. This capability is the key to implement the DAS system, not only for switching between peripherals if not for reusing the global resources. Several configurations which need for the same resources can be activated or not from the same firmware and in real time.

Despite of this, the global resources are the principal limitation detected in the DAS implementation. Concretely, the three variable clocks must be specifically distributed for the user modules in order to be available for the rest of the user modules. This makes that the entire variety of ADC sample rates can't be achieved. Also by this reason, the generated AC signals have an important restriction in their generated frequencies, which are limited to some hundreds of hertz. A possible future work could be to study and design a better reuse of the global resources in order to obtain more performance from the user modules implemented.

Another point to be considered is the CPU use for the expected tasks. Some difficulties to sample the input signal properly have been found when using the LCD, generating a DAC AC signal and transmitting the sample one by one at highest sample rates. This presents a future work in order to optimize the PSoC firmware using the assembler language for programming it instead of using the C++ libraries provided by Cypress for each user module.

In terms of environment, it is important to say that it has not been a point to consider in this project due to its educational objectives. In the expected conditions the main power line can be present, so it is not necessary to maximize the DAS system autonomy, a feature that would be a trade off with the system performance, for example limiting the sample rates.

BIBLIOGRAPHY

[1] Bertran Albertí, E., *Procesado digital de señales,* Edicions UPC, Barcelona (2006)

[2] Halámek, J., Viscor, I., Kasal, M., “Dynamic range and acquisition system”, *Measurement science review,* Institute of Scientific Instruments, Královopolská (2001)

[3] Drake Moyano, J. M., “Ruidos e Interferencias: Técnicas de reducción”, *Instrumentación electrónica de comunicaciones,* Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación, Cantabria (2005)

[4] Garcimartín Montero, A., *Sistemas de medida y adquisición de datos,* DPTO. De Física y Matemática Aplicada, Navarra.

[5] Gómez, J., *¿Cómo controlar el puerto serie usando C#?,* Escuela Técnica Superior de Telecomunicaciones, Bilbao.

[6] Mićaković, P., *Architecture and Programming of PSoC Microcontrollers,* Free online book mikroElektronika, http://www.easypsoc.com/book/

[7] Saravanja, Z., *Fundamentals of PSoC GPIO*, psocdeveloper.com (2004), http://www.psocdeveloper.com/articles/fundamentals-of-psoc-gpio/introduction.html

[8] Cypress Semiconductor Corporation, *PSoC Designer IDE Guide,* Document # 001-42655 Rev *E, San Jose, www.cypress.com/?docID=16790

[9] Cypress Semiconductor Corporation, *PSoC Programmable System-on-Chip,* PSoC TRM, Document No. 001-14463 Rev. *D, San Jose, www.cypress.com/?docID=29893

[10] Cypress Semiconductor Corporation, *CY3210-PSoCEVAL1 PSoC 1 Evaluation Kit Guide,* Doc. #: 001-66768 Rev. *B, San Jose, www.cypress.com/?**docI**D=32028

ANNEX A. HOW ADCINCVR MAKES THE CONVERSION

The ADCINCVR user module is implemented with three digital blocks apart from the analog one used to acquire the external signal. These digital blocks are an 8-bits counter and a 16-bits PWM composed by two 8-bits PWM.

The counter is used to accumulate the number of cycles that the analog block comparator output is positive. The counter is able to provide only the LSB of the digital code due to its 8-bits size. In order to generate a digital code greater than 8-bits, an interruption is needed. When the hardware counter overflows, the interruption is generated in order to increment a software counter with the digital code MSB. This design is needed in order to save a Counter8 digital block, which will correspond to the digital code MSB.

The PWM is used to time the entire process. It times the integrate time, which corresponds to 2^{Bits+2} times the DataClock. During this time, the PWM outputs a high value that enables the counter to measure the integration. When the integrate time ends the PWM outputs a low value to disable the counter. At this moment the integrator is reset, the counter is read and the result is calculated, then the counter is reenabled to make the next measure. The time needed to do these steps is named CalcTime. It corresponds to a minimum of 180 CPU cycles expressed in terms of the DataClock, but can be increased in order to adjust the sample rate.

ANNEX B. DATA FORMAT USED BY THE USER MODULES

The DAC and the ADCs analog signal is referenced to an analog ground, thereby this signal may have a polarity.

Depending on the reference used to make the digital code, it can express the signal polarity or not. Therefore, these codes can be signed or unsigned. Also, there are different formats to express the signed one.

The way to express this code can vary from one user module to another. The format used by the user modules are explained below.

SIGN AND MAGNITUDE
This format expresses the magnitude as a normal binary and uses the MSB to express the signal polarity. It uses 0 for positive numbers and 1 for negative numbers. Its range is from -2^{N-1} -1 to 2^{N-1} -1.

The main disadvantage is the difficult to operate them algebraically. The magnitude is symmetric respect to the ground but the zero value is expressed by two codes.

2'S COMPLEMENT
It is the native 2's complement format of the microcontroller. The positive values are expressed as natural binary and the negatives are given by the positive complement plus '1'. Its range is from -2^{N-1} to 2^{N-1} -1.

The advantage of this format is the facility to operate them, because any number is expressed with its own code and the sum of them make the proper code to the decimal result.

OFFSET BINARY
In this format, the lowest analog value is represented by zero and the highest by the bits totally set. The code range is from 0 to 2^N -1. So, the signal polarity isn't represented.

Some user modules don't use the sign and magnitude format, in these case 2's complement format is called *signed* and the offset binary is called *unsigned*.

ANNEX C. USER MODULES LIBRARY FUNCTIONS USED

Cypress provides each user module with a library of C++ programming language functions in order to implement the projects based in this language. The ones used to implement this project are listed and explained below.

UART

UART_CmdReset(void);
Resets Rx command buffer.

UART_IntCntl(BYTE bMask);
Selectively enables/disables RX and TX interrupts.

UART_Start(BYTE bParity);
Enables user module and set parity.

UART_bReadRxData(void);
Returns data in RX Data register without checking status of character is valid.

UART_PutSHexInt(int iValue);
Sends a four character hex representation of iValue to the TX port.

UART_PutCRLF(void);
Sends a carriage return and a line feed to the TX port.

UART_CPutString(const char *azStr);
Sends NULL terminated constant (ROM) string out TX port.

LCD

LCD_Start(void);
Initializes LCD to use the multi-line, 4-bit interface.

LCD_InitBG(BYTE bBGType);
Initializes the LCD to display the specified type of horizontal bar graph.

LCD_Init(void);
Initializes LCD to use the multi-line, 4-bit interface.

LCD_Position(BYTE bRow, BYTE bCol);
Moves the cursor to a location specified by the parameters.

LCD_PrCString(const char *sRomString);
Prints a null terminated ROM-based character string to the LCD at the present cursor location.

LCD_PrHexInt(INT iValue);

Prints an integer as a four-character hex string at the present LCD cursor position.

LCD_DrawBG(BYTE bRow, BYTE bCol, BYTE bLen, BYTE bPixelColEnd);
Draws the horizontal bar graph starting at character location with a character length to a column position.

Counter8
Counter8_EnableInt(void);
Enables interrupt mode operation.

Counter8_Start(void);
Starts the counter operation.

Counter8_Stop(void);
Stops counter operation.

Counter8_WritePeriod(BYTE bPeriod);
Writes the Period register with the period value. It is transferred to the Counter register immediately, if the counter is stopped or when the counter reaches the zero count.

DAC6
DAC6_Start(BYTE bPowerSetting);
Performs all required initialization for this user module and sets the power level for the analog block.

DAC6_WriteBlind(CHAR cOutputValue);
Immediately updates the output voltage to the indicated value.

PGA
PGA_Start(BYTE bPowerSetting);
Performs all required initialization for the user module and sets the power level for the analog block.

PGA_SetGain(BYTE bGainSetting);
Sets the gain for the user module.

ADCINCVR
ADCINCVR_Start(BYTE bPower);
Performs all required initialization for the user module and sets the power level for the analog block.

ADCINCVR_SetResolution(BYTE bResolution);
Sets the resolution of the A/D converter.

ADCINCVR_GetSamples(BYTE bNumSamples);

Initializes and starts the ADC algorithm to collect the specified number of samples.

ADCINCVR_fIsDataAvailable(void);
Returns non-zero data when a data conversion is completed and data is available.

ADCINCVR_iGetData(void);
Returns last converted data.

ADCINCVR_ClearFlag(void);
Clears Data Available flag.

DUALADC
DUALADC_Start(BYTE bPowerSetting);
Initializes the user module and sets the power level for the analog block.

DUALADC_GetSamples(BYTE bNumSamples);
Initializes and starts the ADC algorithm to collect the specified number of samples.

DUALADC_fIsDataAvailable(void);
Returns non-zero when a data conversion is complete and data is available.

DUALADC_iGetData1(void);
Returns last converted data for ADC input1.

DUALADC_iGetData2(void);
Returns last converted data for ADC input2.

DUALADC_ClearFlag(void);
Clear Data Available flag.

SAR6
SAR6_Start(BYTE bPowerSetting);
Performs all required initialization for the user module and sets the power level for the analog block.

SAR6_cGetSample(void);
Performs a conversion, returning a 2's complement value.

Dynamic Reconfiguration
LoadConfig_*config*(void);
Executes code that configures the device to implement the named configuration.

UnLoadConfig_*config*(void);
Executes code that configures the device to undo the settings of a loaded configuration.

ANNEX D. TEST OF DAC GENERATION

This annex aims to complete the DAC generation test showing some screenshots. In order to acquire the tested signals, the integrating ADC type is selected, with a resolution of 7 bits and a sample rate of 2 000 sps. The number of adquired samples is set to 1000 samples.

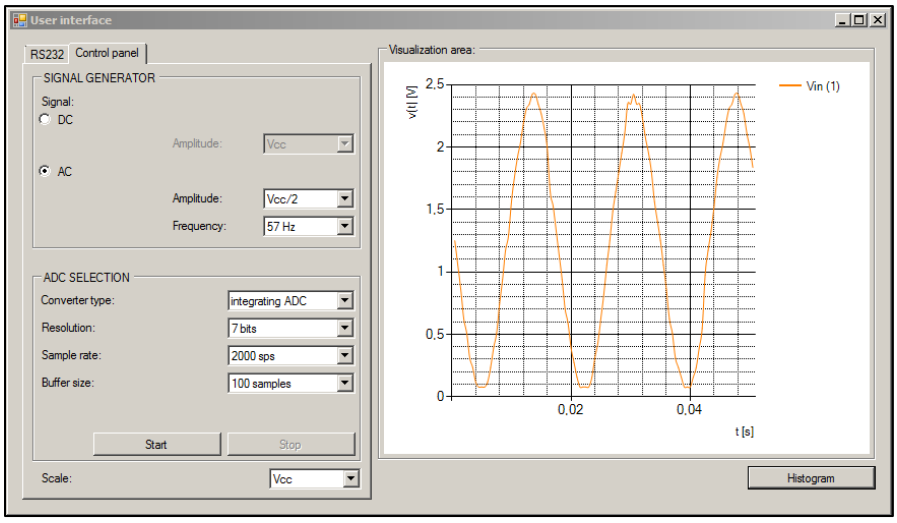


Fig. 5.1 57 Hz AC signal generation with 2,5 V amplitude.

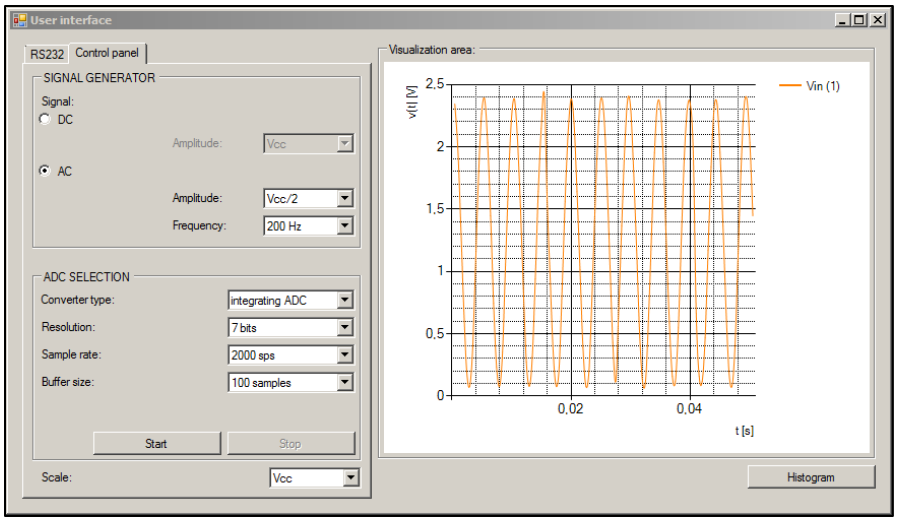


Fig. 5.2 100 Hz AC signal generation with 2,5 V amplitude.

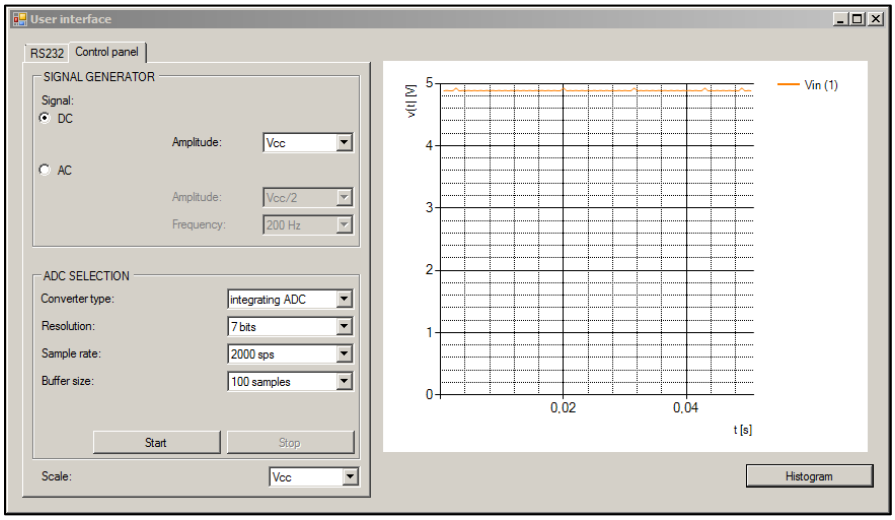


Fig. 5.3 DC signal generation with 5 V amplitude.

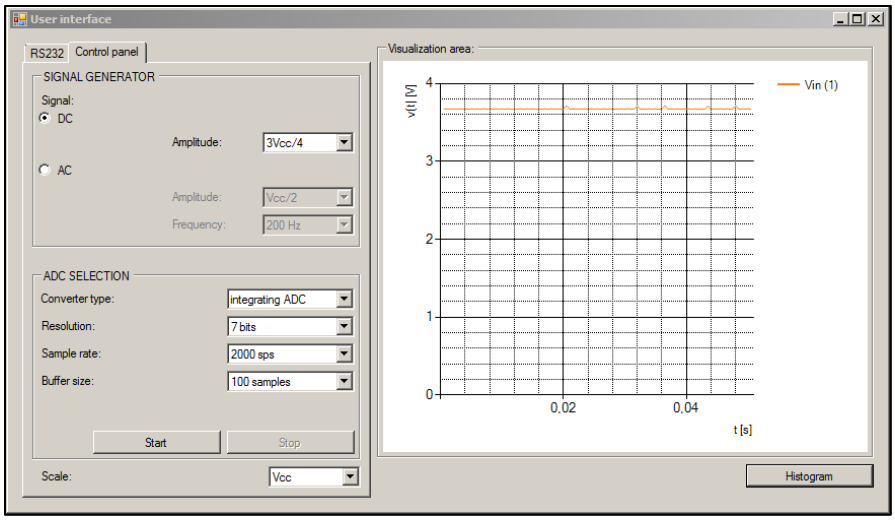


Fig. 5.4 DC signal generation with 3,75 V amplitude.

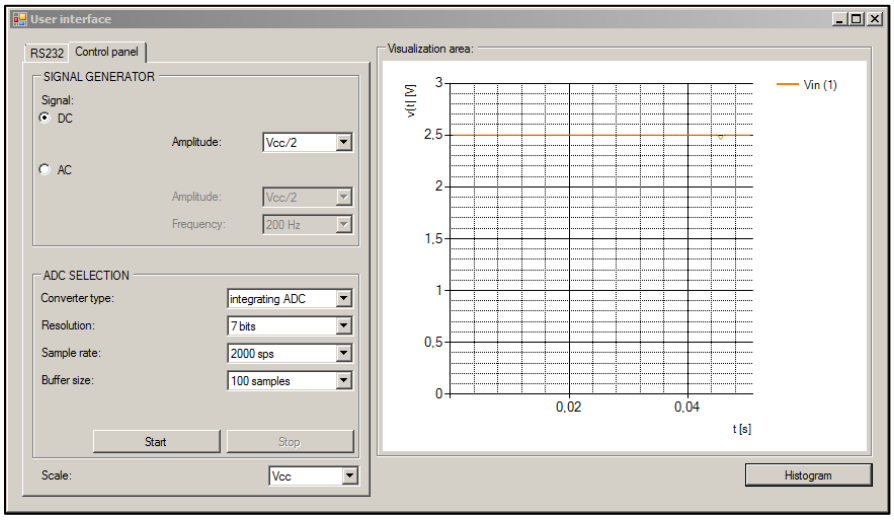


Fig. 5.5 DC signal generation with 2,5 V amplitude.

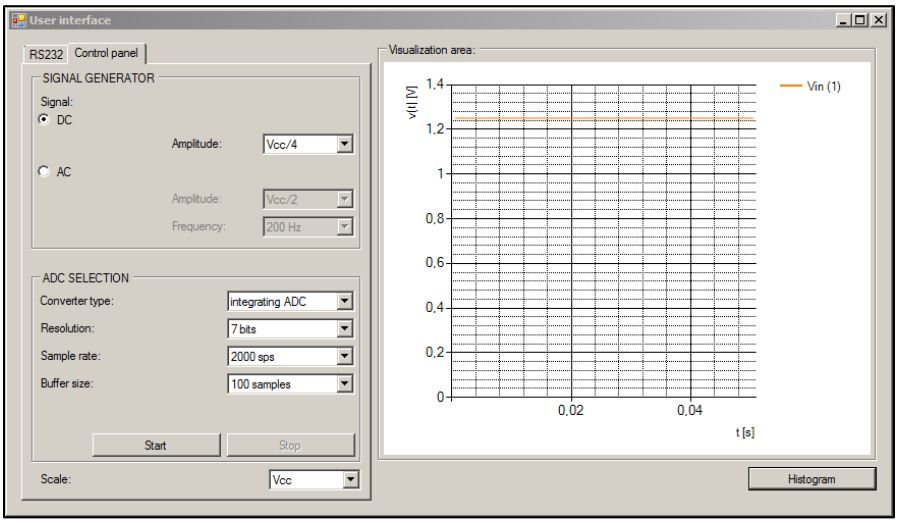


Fig. 5.6 DC signal generation with 1,25 V amplitude.

ANNEX E. PSOC AND USER INTERFACE SOURCE CODES

This annex shows the PSoC and the PC user interface source codes.

PSoC source code

```
#include <m8c.h>                        /* part specific constants and
macros */
#include <PSoCDynamic.h>                        // Dynamic
reconfiguration API
#include "PSoCAPI.h"                    /* PSoC API definitions for all
User Modules */

#define RESOLUTION_ADC1 7                    /* ADC resolution */
#define RESOLUTION_ADC2 7
#define RESOLUTION_ADC3 7
#define RESOLUTION_ADC4 7
#define RESOLUTION_ADC5 10
#define RESOLUTION_SAR 6
#define SIN_PERIOD 64

#define SCALE_BG_ADC1  (( 1 << RESOLUTION_ADC1)/55) /* BarGraph scale
factor */
#define SCALE_BG_ADC2  (( 1 << RESOLUTION_ADC2)/55)
#define SCALE_BG_ADC3  (( 1 << RESOLUTION_ADC3)/55)
#define SCALE_BG_ADC4  (( 1 << RESOLUTION_ADC4)/55)
#define SCALE_BG_ADC5  (( 1 << RESOLUTION_ADC5)/55)
#define SCALE_BG_SAR   (( 1 << RESOLUTION_SAR)/55)

//variables
BYTE rx;                                                // Variable con el
contenido del búffer de recepción del UART
BYTE bgPos;                             // BarGraph position
BYTE BG_ADC1;
int  iResult;                           // ADC result variable /DUALADC
output signal
int iResult_source;                                     // DUALADC source
signal
char iResult_SAR;                               // SAR result variable
int res1;
int res2;
int res3;

int x = 100;
int w = 0;
int resultado[100];

int pos_SIN = 1;
int SINtable[SIN_PERIOD] = {31, 33, 36, 39, 41, 44, 46, 49, 51, 53, 55,
56, 58, 59, 59, 60, 60, 60, 59, 59, 58, 56, 55, 53, 51, 49, 47, 44, 42,
39, 36, 33, 31, 28, 25, 22, 19, 16, 13, 11, 9, 7, 5, 3, 2, 1, 0, 0, 0, 0,
1, 2, 3, 4, 6, 7, 10, 12, 14, 17, 20, 23, 26, 29};
int SIN_vout[SIN_PERIOD];
```

```
int v = 1;
int i;

void config1(void)
{
	LCD_1_Init(); LCD_1_Position(0,0); LCD_1_PrCString("CONFIG 1: ADC
1");

	UnloadConfig_Config2();
	UnloadConfig_Config3();
	UnloadConfig_Config4();
	UnloadConfig_Config5();
	UnloadConfig_Config6();
	LoadConfig_Config1();

	PGA_1_Start(PGA_1_HIGHPOWER);		// Turn on PGA power
	ADCINCVR_1_Start(ADCINCVR_1_HIGHPOWER);
	ADCINCVR_1_GetSamples(0);

	while(1)
	{
		if(ADCINCVR_1_fIsDataAvailable() != 0)	// Check if ADC
conversion is complete
		{
			iResult = ADCINCVR_1_iGetData(); /* Get result, convert
to unsigned (sumando el valor correspondiente a n bits - 1) and clear
flag */
		ADCINCVR_1_ClearFlag();

		LCD_1_Position(1,0);		/* display result on LCD in
hex and as a bar graph */
		LCD_1_PrHexInt(iResult);
		bgPos = (BYTE)(iResult/BG_ADC1);
		LCD_1_DrawBG(1, 5, 11, bgPos);

			UART_1_PutSHexInt(iResult);	/* Print result to UART
*/
		UART_1_PutCRLF();		/* Tack on a CR and LF */
		}

		rx = UART_1_bReadRxData();
		switch(rx)
		{
			case 'x':
			{
				LCD_1_Init();
				LCD_1_Position(0,0);
				LCD_1_PrCString("PSoC 1");
				LCD_1_Position(1,0);
				LCD_1_PrCString("Salir de ADC 1");
				return;
			}
			case '1':   // Cambiamos resolución
			{
				if(res1)
				{
```

```
                                    res1 = 0;
                                    res2 = 1;
                                    res3 = 1;
                                    LCD_1_Init();
                                    LCD_1_Position(0,0);
                                    LCD_1_PrCString("SetRes.: 7bits");
                                    BG_ADC1 = (( 1 << RESOLUTION_ADC1)/55);

                                    ADCINCVR_1_SetResolution(7);
                                    ADCINCVR_1_Start(ADCINCVR_1_HIGHPOWER);
                                    ADCINCVR_1_GetSamples(0);
                              }
                              break;
                        }
                        case '2':   // Cambiamos resolución
                        {
                              if(res2)
                              {
                                    res2 = 0;
                                    res1 = 1;
                                    res3 = 1;
                                    LCD_1_Init();
                                    LCD_1_Position(0,0);
                                    LCD_1_PrCString("SetRes.: 10bits");
                                    BG_ADC1 = (( 1 << 10)/55);

                                    ADCINCVR_1_SetResolution(10);
                                    ADCINCVR_1_Start(ADCINCVR_1_HIGHPOWER);
                                    ADCINCVR_1_GetSamples(0);
                              }
                              break;
                        }
                        case '3':   // Cambiamos resolución
                        {
                              if(res3)
                              {
                                    res3 = 0;
                                    res1 = 1;
                                    res2 = 1;
                                    LCD_1_Init();
                                    LCD_1_Position(0,0);

                                    LCD_1_PrCString("SetRes.: 13bits");
                                    BG_ADC1 = (( 1 << 13)/55);

                                    ADCINCVR_1_SetResolution(13);
                                    ADCINCVR_1_Start(ADCINCVR_1_HIGHPOWER);
                                    ADCINCVR_1_GetSamples(0);

                              }
                              break;
                        }
                        default:
                              break;
                  }
            }
      }
```

```
void config2(void)
{
	LCD_1_Init(); LCD_1_Position(0,0); LCD_1_PrCString("CONFIG 2: ADC
2");

	UnloadConfig_Config1();
	UnloadConfig_Config3();
	UnloadConfig_Config4();
	UnloadConfig_Config5();
	UnloadConfig_Config6();
	LoadConfig_Config2();

	PGA_1_Start(PGA_1_HIGHPOWER);		// Turn on PGA power
	ADCINCVR_2_Start(ADCINCVR_2_HIGHPOWER);
	ADCINCVR_2_GetSamples(0);

	while(1)
	{
		if(ADCINCVR_2_fIsDataAvailable() != 0)	// Check if ADC
conversion is complete
		{
			//iResult = ADCINCVR_2_iGetData() + 2048; /* Get
result, convert to unsigned and clear flag */
			iResult = ADCINCVR_2_iGetData(); /* Get result, convert
to unsigned and clear flag */
		ADCINCVR_2_ClearFlag();

		//LCD_1_Position(1,0);		/* display result on LCD in
hex and as a bar graph */
		//LCD_1_PrHexInt(iResult);
		//bgPos = (BYTE)(iResult/SCALE_BG_ADC2);
		//LCD_1_DrawBG(1, 5, 11, bgPos);

			UART_1_PutSHexInt(iResult);	/* Print result to UART
*/
		UART_1_PutCRLF();		/* Tack on a CR and LF */
		}

		rx = UART_1_bReadRxData();
		if(rx == 'x')
		{
			LCD_1_Init();
			LCD_1_Position(0,0);
			LCD_1_PrCString("PSoC 1");
			LCD_1_Position(1,0);
			LCD_1_PrCString("Salir de ADC 2");
			rx = 0;
			return;
		}
		rx = 0;
	}
}

void config3(void)
{
```

```
	LCD_1_Init(); LCD_1_Position(0,0); LCD_1_PrCString("CONFIG 3: ADC
3");

	UnloadConfig_Config1();
	UnloadConfig_Config2();
	UnloadConfig_Config4();
	UnloadConfig_Config5();
	UnloadConfig_Config6();
	LoadConfig_Config3();

	PGA_1_Start(PGA_1_HIGHPOWER);		// Turn on PGA power
	ADCINCVR_3_Start(ADCINCVR_3_HIGHPOWER);
	ADCINCVR_3_GetSamples(0);

	while(1)
	{
		w = 0;
		while(w < x)
		{
			if(ADCINCVR_3_fIsDataAvailable() != 0)	// Check if
ADC conversion is complete
			{
				resultado[w] = ADCINCVR_3_iGetData();
				ADCINCVR_3_ClearFlag();
				w++;
			}
		}
		w = 0;
		while(w < x)
		{
			UART_1_PutSHexInt(resultado[w]);UART_1_PutCRLF();
			w++;
		}
		rx = UART_1_bReadRxData();
		if(rx == 'x')
		{
			LCD_1_Init();LCD_1_Position(0,0);LCD_1_PrCString("PSoC
1");LCD_1_Position(1,0);LCD_1_PrCString("Salir de ADC 3");
			rx = 0;
			return;
		}
	}
}

void config4(void)
{
	LCD_1_Init(); LCD_1_Position(0,0); LCD_1_PrCString("CONFIG 4: ADC
4");

	UnloadConfig_Config1();
	UnloadConfig_Config2();
	UnloadConfig_Config3();
	UnloadConfig_Config5();
	UnloadConfig_Config6();
	LoadConfig_Config4();

	PGA_1_Start(PGA_1_HIGHPOWER);		// Turn on PGA power
```

```
	ADCINCVR_4_Start(ADCINCVR_4_HIGHPOWER);
	ADCINCVR_4_GetSamples(0);

	while(1)
	{
		w = 0;
		while(w < x)
		{
			if(ADCINCVR_4_fIsDataAvailable() != 0)    // Check if
ADC conversion is complete
			{
				resultado[w] = ADCINCVR_4_iGetData();
				ADCINCVR_4_ClearFlag();
				w++;
			}
		}
		w = 0;
		while(w < x)
		{
			UART_1_PutSHexInt(resultado[w]);UART_1_PutCRLF();
			w++;
		}
		rx = UART_1_bReadRxData();
		if(rx == 'x')
		{
			LCD_1_Init();LCD_1_Position(0,0);LCD_1_PrCString("PSoC
1");LCD_1_Position(1,0);LCD_1_PrCString("Salir de ADC 4");
			rx = 0;
			return;
		}
	}
}

void config5(void)
{
	LCD_1_Init(); LCD_1_Position(0,0); LCD_1_PrCString("CONFIG 5:
DUALADC");

	UnloadConfig_Config1();
	UnloadConfig_Config2();
	UnloadConfig_Config3();
	UnloadConfig_Config4();
	UnloadConfig_Config6();
	LoadConfig_Config5();

	PGA_1_Start(PGA_1_HIGHPOWER);              // Turn on PGA1 power
	PGA_2_Start(PGA_2_HIGHPOWER);              // Turn on PGA2 power
	DUALADC_1_Start(DUALADC_1_HIGHPOWER);
	DUALADC_1_GetSamples(0);

	while(1)
	{
		if(DUALADC_1_fIsDataAvailable() != 0)     // Check if ADC
conversion is complete
		{
```

```
                  // Muestreamos la señal de entrada en el circuito, que
proviene del p01 y el PGA2 y la señal de salida, que proviene del p00
                  iResult_source = DUALADC_1_iGetData1();
                  iResult = DUALADC_1_iGetData2();
                  DUALADC_1_ClearFlag();

                  // Mostramos en el LCD SÓLO la señal de salida
            //LCD_1_Position(1,0);
            //LCD_1_PrHexInt(iResult);
            //bgPos = (BYTE)(iResult/SCALE_BG_ADC5);
            //LCD_1_DrawBG(1, 5, 11, bgPos);

                  // Mandamos por el UART ambas señales
                  UART_1_PutSHexInt(iResult);                     //
Enviamos la muestra del segundo canal (output signal)
            UART_1_CPutString("src");                        // Enviamos
un código para reconocer el primer canal
                  UART_1_PutSHexInt(iResult_source);        // Enviamos
la muestra del primer canal (source signal)
                  UART_1_PutCRLF();
            }

            rx = UART_1_bReadRxData();
            if(rx == 'x')
            {
                  LCD_1_Init();LCD_1_Position(0,0);LCD_1_PrCString("PSoC
1");LCD_1_Position(1,0);LCD_1_PrCString("Salir de DUALADC");
                  rx = 0;
                  return;
            }
            rx = 0;
      }
}

void config6(void)
{
      LCD_1_Init(); LCD_1_Position(0,0); LCD_1_PrCString("CONFIG 6: SAR
6");

      UnloadConfig_Config1();
      UnloadConfig_Config2();
      UnloadConfig_Config3();
      UnloadConfig_Config4();
      UnloadConfig_Config5();
      LoadConfig_Config6();

      PGA_1_Start(PGA_1_HIGHPOWER); // Start PGA in HIGH power mode
      SAR6_1_Start(SAR6_1_HIGHPOWER); // Start ADC in HIGH power mode

      while(1)
      {
            //PRT0DR = 0x04; //Se coloca en nivel bajo el Port_0_2
            // ver datasheet: AN2094 (io pin-port config)
            w = 0;
            while(w < x)
            {
                  resultado[w] = SAR6_1_cGetSample() + 32;
```

```
                  w++;
            }
            w = 0;
            while(w < x)
            {
                  UART_1_PutSHexInt(resultado[w]);UART_1_PutCRLF();
                  w++;
            }
            rx = UART_1_bReadRxData();
            if(rx == 'x')
            {
                  LCD_1_Init();LCD_1_Position(0,0);LCD_1_PrCString("PSoC
1");LCD_1_Position(1,0);LCD_1_PrCString("Salir de SAR 6");
                  rx = 0;
                  return;
            }
            rx = 0;

            //PRT0DR = 0x00; // se coloca en nivel alto el Port_0_2
      }
}

void genera_SIN(void)
{
      // Cada vez que se llama a la función se coloca un valor del vector
del SENO en el DAC, generando así la señal a lo largo del tiempo.
      pos_SIN--;
      if(pos_SIN != 0)
            DAC6_1_WriteBlind(SIN_vout[pos_SIN]);
      else
            pos_SIN = 64;
}

void main(void)
{
      Counter8_1_EnableInt();
      BG_ADC1 = (( 1 << RESOLUTION_ADC1)/55);
      DAC6_1_Start(DAC6_1_HIGHPOWER);
    UART_1_Start(UART_PARITY_NONE);      /* Enable UART */
      LCD_1_Start();                       /* Init the LCD */
    LCD_1_InitBG(LCD_1_SOLID_BG);
      M8C_EnableGInt;                      /* Enable Global interrupts */

      LCD_1_Init(); LCD_1_Position(0,0); LCD_1_PrCString("Select a
config:");

      while (1)
      {
            res1 = res2 = res3 = 1;
            rx = UART_1_bReadRxData();

            switch(rx)
            {
            // DAC modo contínuo
                  case 'a':
                  {
```

```
                    // Paramos el contador regresivo. El SENO no se puede
generar y el DAC mantiene su valor al valor establecido por el resto de
funciones.
                          Counter8_1_Stop();
                          LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC DC mode");
                          rx = 0; break;
                    }
              // Modo configuración de amplitud
                    case 'c':
                    {
                          DAC6_1_WriteBlind(62);
                          LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC DC mode"); LCD_1_Position(1,0);
LCD_1_PrCString("Amplitude: 5 V");
                          v = 0; rx = 0; break;
                    }
                    case 'd':
                    {
                          DAC6_1_WriteBlind(46);
                          LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC DC mode"); LCD_1_Position(1,0);
LCD_1_PrCString("Amplitude: 3.75V");
                          v = 0; rx = 0; break;
                    }
                    case 'e':
                    {
                          DAC6_1_WriteBlind(31);
                          LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC DC mode"); LCD_1_Position(1,0);
LCD_1_PrCString("Amplitude: 2.5 V");
                          v = 0; rx = 0; break;
                    }
                    case 'f':
                    {
                          DAC6_1_WriteBlind(15);
                          LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC DC mode"); LCD_1_Position(1,0);
LCD_1_PrCString("Amplitude: 1.25V");
                          v = 0; rx = 0; break;
                    }
              // DAC modo variable (SENO)
                    case 'g':
                    {
                    // Encendemos el contador regresivo. Cuando éste
alcanza el 0 la interrupción asociada llama a la función que genera el
SENO y vuelve a empezar.
                          Counter8_1_Start();
                          LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC AC mode");
                          rx = 0; break;
                    }
              // Modo configuración de amplitud
                    case 'i':
                    {
                          for(i = 0; i<= SIN_PERIOD; i++)
                                SIN_vout[i] = SINtable[i];
```

```
                        LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC AC mode"); LCD_1_Position(1,0);
LCD_1_PrCString("Amplitude: 5 V");
                        v = 0; rx = 0; break;
                  }
                  case 'j':
                  {
                        for(i = 0; i<= SIN_PERIOD; i++)
                              SIN_vout[i] = (SINtable[i] *3) /4;
                        LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC AC mode"); LCD_1_Position(1,0);
LCD_1_PrCString("Amplitude: 3.75V");
                        v = 0; rx = 0; break;
                  }
                  case 'k':
                  {
                        for(i = 0; i<= SIN_PERIOD; i++)
                              SIN_vout[i] = SINtable[i] /2;
                        LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC AC mode"); LCD_1_Position(1,0);
LCD_1_PrCString("Amplitude: 2.5 V");
                        v = 0; rx = 0; break;
                  }
                  case 'l':
                  {
                        for(i = 0; i<= SIN_PERIOD; i++)
                              SIN_vout[i] = SINtable[i] /4;
                        LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC AC mode"); LCD_1_Position(1,0);
LCD_1_PrCString("Amplitude: 1.25V");
                        v = 0; rx = 0; break;
                  }
            // Modo configuración de frecuencia
                  case 'n':
                  {
                        Counter8_1_WritePeriod(255);
                        LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC AC mode"); LCD_1_Position(1,0);
LCD_1_PrCString("Frequency: 57Hz");
                        v = 0; rx = 0; break;
                  }
                  case 'o':
                  {
                        Counter8_1_WritePeriod(0x90);
                        LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC AC mode"); LCD_1_Position(1,0);
LCD_1_PrCString("Frequency: 100Hz");
                        v = 0; rx = 0; break;
                  }
                  case 'p':
                  {
                        Counter8_1_WritePeriod(0x48);
                        LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("DAC AC mode"); LCD_1_Position(1,0);
LCD_1_PrCString("Frequency: 200Hz");
                        v = 0; rx = 0; break;
```

```
                }
            // Modo selección de capa config (Selector de ADC)
                case '1':
                    config1(); rx = 0; break;
                case '2':
                    config2(); rx = 0; break;
                case '3':
                    config3(); rx = 0; break;
                case '4':
                    config4(); rx = 0; break;
                case '5':
                    config5(); rx = 0; break;
                case '6':
                    config6(); rx = 0; break;
            // Selección del fondo de escala del PGA
                case '7':
                {
                    LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("PGA scale"); LCD_1_Position(1,0); LCD_1_PrCString("Gain:
1");
                    PGA_1_SetGain(PGA_1_G1_00); rx = 0; break;
                }
                case '8':
                {
                // Acepta entradas de 0 a 625 mV
                    LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("PGA scale"); LCD_1_Position(1,0); LCD_1_PrCString("Gain:
2");
                    PGA_1_SetGain(PGA_1_G2_00); rx = 0; break;
                }
                case '9':
                {
                // Acepta entradas de 0 a 104 mV
                    LCD_1_Init(); LCD_1_Position(0,0);
LCD_1_PrCString("PGA scale"); LCD_1_Position(1,0); LCD_1_PrCString("Gain:
8");
                    PGA_1_SetGain(PGA_1_G8_00); rx = 0; break;
                }
                default:
                    break;
            }
        }
}
```

User interface source code

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.IO.Ports;
using System.Windows.Forms.DataVisualization.Charting;

namespace interfaz_project
{
    public partial class Form1 : Form
    {
        SerialPort sp = new SerialPort();
        bool sp_abierto = false;
        string ruta_config = @"C:\config";
        StreamReader sr_config_r;
        StreamWriter sr_config_w;

        string port_config;
        string[] tasas = new string[18] { "75", "110", "134", "150",
"300", "600", "1200", "1800", "2400", "4800", "7200", "9600", "14400",
"19200", "38400", "57600", "115200", "128000" };
        string tasa_config;
        string[] bits_datos_s = new string[5] { "4", "5", "6", "7", "8"
};
        string bits_datos_config;
        string[] paridades = new string[5] { "Even", "Odd", "None",
"Mark", "Space" };
        string paridad_config;
        string[] bits_stop_s = new string[4] { "0", "1", "1,5", "2" };
        string bits_stop_config;
        string[] handshake_s = new string[4] { "None", "Hardware",
"Hardware & Software", "Software" };
        string handshake_config;

        string[] adc_types = new string[3] { "integrating ADC", "SAR
ADC", "dual ADC" };
        string[] sample_rates = new string[4] { "400 sps", "2000 sps",
"5000 sps", "7400 sps" };
        string[] resolutions = new string[3] { "7 bits", "10 bits", "13
bits" };
        string[] dac_voltage = new string[4] { "Vcc", "3Vcc/4", "Vcc/2",
"Vcc/4" };
        string[] sin_freq = new string[3] { "57 Hz", "100 Hz", "200 Hz"
};
        string[] scale = new string[3] { "Vcc", "Vcc /2", "Vcc /8" };

        string[] buffer = new string[5] { "25 samples", "50 samples",
"100 samples", "1000 samples", "10000 samples" };
        int buffer_size = 0;
```

```
        int device_selected = 0;
        int res_selected = 0;
        int i;

        double vcc = 5;
        double vfs;                 // Vref, Tensión fondo de escala
        double res;                 // Resolución actual
        int code;                   // Código generado por el ADC
        double power;               // Potencia de la resolución
        double value;               // Valor medido (código traducido a
valor de tensión)

        char erre = (char)13;
        char ene = (char)10;
        string s1_string;
        Series s1 = new Series();
        Series s2 = new Series();
        Series histo = new Series();
        bool histogram_mode = true;
        int histo_pos;
        DataPoint dp; double dp_y;
        double f_sample; double t; double t_sample;


        public Form1()
        {
            InitializeComponent();

            this.comboBox1.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox2.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox3.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox4.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox5.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox6.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox7.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox8.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox9.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox10.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox11.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox12.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox13.DropDownStyle = ComboBoxStyle.DropDownList;
            this.comboBox14.DropDownStyle = ComboBoxStyle.DropDownList;

            comboBox1.Items.AddRange(SerialPort.GetPortNames());
            comboBox2.Items.AddRange(tasas);
            comboBox3.Items.AddRange(bits_datos_s);
            comboBox4.Items.AddRange(paridades);
            comboBox5.Items.AddRange(bits_stop_s);
            comboBox6.Items.AddRange(handshake_s);
            comboBox7.Items.AddRange(adc_types);
            comboBox10.Items.AddRange(buffer);
            comboBox11.Items.AddRange(dac_voltage);
            comboBox12.Items.AddRange(dac_voltage);
            comboBox13.Items.AddRange(sin_freq);
            comboBox14.Items.AddRange(scale);
            button3.Enabled = false; button4.Enabled = false;
button5.Enabled = false;
```

```
            comboBox11.Enabled = false;
            comboBox12.Enabled = false;
            comboBox13.Enabled = false;

            s1.ChartType = SeriesChartType.Spline;
            s1.LegendText = "Vin (1)";
            s2.ChartType = SeriesChartType.Spline;
            s2.LegendText = "Vin (2)";
            histo.LegendText = "Vin (1)";
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            CheckForIllegalCrossThreadCalls = false;

            try
            {
                sr_config_r = new StreamReader(ruta_config);
            }
            catch (FileNotFoundException ex)
            {
                MessageBox.Show("Configuration file still set.");
                return;
            }

            //cargamos configuración del fichero en los combobox y en las
variables
            port_config = sr_config_r.ReadLine();
            for (i = 0; i < SerialPort.GetPortNames().Length; i++)
                if (SerialPort.GetPortNames()[i] == port_config)
                    comboBox1.SelectedIndex = i;

            tasa_config = sr_config_r.ReadLine();
            for (i = 0; i < tasas.Length; i++)
                if (tasas[i] == tasa_config)
                    comboBox2.SelectedIndex = i;

            bits_datos_config = sr_config_r.ReadLine();
            for (i = 0; i < bits_datos_s.Length; i++)
                if (bits_datos_s[i] == bits_datos_config)
                    comboBox3.SelectedIndex = i;

            paridad_config = sr_config_r.ReadLine();
            for (i = 0; i < paridades.Length; i++)
                if (paridades[i] == paridad_config)
                    comboBox4.SelectedIndex = i;

            bits_stop_config = sr_config_r.ReadLine();
            for (i = 0; i < bits_stop_s.Length; i++)
                if (bits_stop_s[i] == bits_stop_config)
                    comboBox5.SelectedIndex = i;

            handshake_config = sr_config_r.ReadLine();
            for (i = 0; i < handshake_s.Length; i++)
                if (handshake_s[i] == handshake_config)
```

```
                    comboBox6.SelectedIndex = i;

            sr_config_r.Close();
        }

        // Pestaña configuración
        private void button1_Click(object sender, EventArgs e)
        {
            if (sp_abierto)
            {
                sp.Close();
                comboBox1.Enabled = true;
                comboBox2.Enabled = true;
                comboBox3.Enabled = true;
                comboBox4.Enabled = true;
                comboBox5.Enabled = true;
                comboBox6.Enabled = true;
                button1.Text = "Open port";
                sp_abierto = false;
                return;
            }
            else
            {
                try
                {
                    sp.PortName = comboBox1.Text;
                }
                catch (ArgumentException)
                {
                    MessageBox.Show("No COM device founded.");
                    return;
                }
                sp.BaudRate = Convert.ToInt32(comboBox2.Text);
                sp.DataBits = Convert.ToInt32(comboBox3.Text);

                switch(comboBox4.Text)
                {
                    case "Even": sp.Parity = Parity.Even; break;
                    case "Odd": sp.Parity = Parity.Odd; break;
                    case "None": sp.Parity = Parity.None; break;
                    case "Mark": sp.Parity = Parity.Mark; break;
                    case "Space": sp.Parity = Parity.Space; break;
                    default: break;
                }
                switch(comboBox5.Text)
                {
                    case "0": sp.StopBits = StopBits.None; break;
                    case "1": sp.StopBits = StopBits.One; break;
                    case "1,5": sp.StopBits = StopBits.OnePointFive;
break;
                    case "2": sp.StopBits = StopBits.Two; break;
                    default: break;
                }
                switch(comboBox6.Text)
                {
                    case "None":  sp.Handshake  = Handshake.None; break;
```

```
                    case "Hardware": sp.Handshake =
Handshake.RequestToSend; break;
                    case "Hardware & Software": sp.Handshake =
Handshake.RequestToSendXOnXOff; break;
                    case "Software": sp.Handshake = Handshake.XOnXOff;
break;
                    default: break;
                }
                sp.Open();
                comboBox1.Enabled = false;
                comboBox2.Enabled = false;
                comboBox3.Enabled = false;
                comboBox4.Enabled = false;
                comboBox5.Enabled = false;
                comboBox6.Enabled = false;
                sp_abierto = true;
                button1.Text = "Close port";
                sp.DataReceived += new
SerialDataReceivedEventHandler(DataReceivedHandler);
                return;
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            //comprobar que todos los combobox están llenos
            while (comboBox1.Text == "" | comboBox2.Text == "" |
comboBox3.Text == "" | comboBox4.Text == "" |
                comboBox5.Text == "" | comboBox6.Text == "")
            {
                MessageBox.Show("Fill in the gaps.");
                return;
            }

            //guardar configuración de los combobox en el fichero
            //en el caso de que no exista el fichero primero lo crea y
luego sobreescribe la configuración en él
            sr_config_w = new StreamWriter(ruta_config);
            sr_config_w.Close();
            sr_config_w = new StreamWriter(ruta_config);

            port_config = comboBox1.Text;
            tasa_config = comboBox2.Text;
            bits_datos_config = comboBox3.Text;
            paridad_config = comboBox4.Text;
            bits_stop_config = comboBox5.Text;
            handshake_config = comboBox6.Text;

            sr_config_w.WriteLine(comboBox1.Text);
            sr_config_w.WriteLine(comboBox2.Text);
            sr_config_w.WriteLine(comboBox3.Text);
            sr_config_w.WriteLine(comboBox4.Text);
            sr_config_w.WriteLine(comboBox5.Text);
            sr_config_w.WriteLine(comboBox6.Text);

            sr_config_w.Close();
```

```
        }
        //

        private void tabControl1_Selecting(object sender,
TabControlCancelEventArgs e)
        {
            if (tabControl1.SelectedTab == tabPage1)
            {
                if (button4.Enabled == true)
                {
                    MessageBox.Show("First, stop the ADC.");
                    tabControl1.SelectedTab = tabPage2;
                    return;
                }
            }

            if (tabControl1.SelectedTab == tabPage2)
            {
                if (sp_abierto == false)
                {
                    MessageBox.Show("Open COM port before.");
                    tabControl1.SelectedTab = tabPage1;
                    return;
                }
            }
        }

        // Pestaña generador de señal
        private void radioButton1_CheckedChanged(object sender, EventArgs
e)
        {
            if (radioButton1.Checked == true)
            {
                // Seleccionamos el DAC continuo y habilitamos las
opciones
                sp.Write("a"); sp.Write("x");
                label13.Enabled = true; comboBox11.Enabled = true;
                label14.Enabled = false; comboBox12.Enabled = false;
                label15.Enabled = false; comboBox13.Enabled = false;
            }

            else
            {
                // Seleccionamos el DAC variable y habilitamos las
opciones
                sp.Write("g"); sp.Write("x");
                label13.Enabled = false; comboBox11.Enabled = false;
                label14.Enabled = true; comboBox12.Enabled = true;
                label15.Enabled = true; comboBox13.Enabled = true;
            }
        }

        private void radioButton2_CheckedChanged(object sender, EventArgs
e)
        {
            if (radioButton1.Checked == true)
            {
```

```
                // Seleccionamos el DAC continuo y habilitamos las
opciones
                sp.Write("a"); sp.Write("x");
                label13.Enabled = true; comboBox11.Enabled = true;
                label14.Enabled = false; comboBox12.Enabled = false;
                label15.Enabled = false; comboBox13.Enabled = false;
            }

            else
            {
                // Seleccionamos el DAC variable y habilitamos las
opciones
                sp.Write("g"); sp.Write("x");
                label13.Enabled = false; comboBox11.Enabled = false;
                label14.Enabled = true; comboBox12.Enabled = true;
                label15.Enabled = true; comboBox13.Enabled = true;
            }
        }

        private void comboBox11_SelectedIndexChanged(object sender,
EventArgs e)
        {
            // Seleccionamos la amplitud
            switch (comboBox11.SelectedIndex.ToString())
            {
                case "0": sp.Write("c"); sp.Write("x"); break;
                case "1": sp.Write("d"); sp.Write("x"); break;
                case "2": sp.Write("e"); sp.Write("x"); break;
                case "3": sp.Write("f"); sp.Write("x"); break;
                default: break;
            }
        }

        private void comboBox12_SelectedIndexChanged(object sender,
EventArgs e)
        {
            // Seleccionamos la amplitud
            switch (comboBox12.SelectedIndex.ToString())
            {
                case "0": sp.Write("i"); sp.Write("x"); break;
                case "1": sp.Write("j"); sp.Write("x"); break;
                case "2": sp.Write("k"); sp.Write("x"); break;
                case "3": sp.Write("l"); sp.Write("x"); break;
                default: break;
            }
        }

        private void comboBox13_SelectedIndexChanged(object sender,
EventArgs e)
        {
            // Seleccionamos la frecuencia
            switch (comboBox13.SelectedIndex.ToString())
            {
                case "0": sp.Write("n"); sp.Write("x"); break;
                case "1": sp.Write("o"); sp.Write("x"); break;
                case "2": sp.Write("p"); sp.Write("x"); break;
```

```
                default: break;
            }
        }

        private void comboBox14_SelectedIndexChanged(object sender,
EventArgs e)
        {
            // Seleccionamos la escala de entrada (Ganancia del PGA)
            switch (comboBox14.SelectedIndex.ToString())
            {
                case "0": vfs = vcc; sp.Write("7"); sp.Write("x"); break;
                case "1": vfs = vcc/2; sp.Write("8"); sp.Write("x");
break;
                case "2": vfs = vcc/8; sp.Write("9"); sp.Write("x");
break;
                default: break;
            }
        }
        //

        // Pestaña selector de ADC
        private void comboBox7_SelectedIndexChanged(object sender,
EventArgs e)
        {
            button3.Enabled = true;
            comboBox14.SelectedIndex = 0;

            if (comboBox7.SelectedIndex == 0)
            {
                // ADCINC conversor
                comboBox8.Items.Clear();
comboBox8.Items.AddRange(resolutions); comboBox8.SelectedIndex = 0;
                comboBox9.Items.Clear();
comboBox9.Items.AddRange(sample_rates); comboBox9.SelectedIndex = 0;
            }
            else if (comboBox7.SelectedIndex == 1)
            {
                // SAR conversor
                comboBox8.Items.Clear(); comboBox8.Items.Add("6 bits");
comboBox8.SelectedIndex = 0;
                comboBox9.Items.Clear(); comboBox9.Items.Add("3900 sps");
comboBox9.SelectedIndex = 0;
                device_selected = 6;
                res = 6;
                f_sample = 3900;
            }
            else if (comboBox7.SelectedIndex == 2)
            {
                // DUALADC conversor
                comboBox8.Items.Clear(); comboBox8.Items.Add("10 bits");
comboBox8.SelectedIndex = 0;
                comboBox9.Items.Clear(); comboBox9.Items.Add("1000 sps");
comboBox9.SelectedIndex = 0;
                device_selected = 5;
                res = 10;
                f_sample = 1000;
            }
```

```
            comboBox10.SelectedIndex = 0;
        }

        private void comboBox8_SelectedIndexChanged(object sender,
EventArgs e)
        {
            if (comboBox7.SelectedIndex == 0)
            {
                // Conversor ADCINC de 7, 10 ó 13 bits
                if (comboBox8.SelectedIndex == 0)
                {
                    comboBox9.Items.Clear();
comboBox9.Items.AddRange(sample_rates); comboBox9.SelectedIndex = 0;
                    res = 7; res_selected = 1;
                }
                else if (comboBox8.SelectedIndex == 1)
                {
                    comboBox9.Items.Clear(); comboBox9.Items.Add("50
sps"); comboBox9.SelectedIndex = 0;
                    device_selected = 1;
                    res = 10; res_selected = 2;
                    f_sample = 50;
                }
                else if (comboBox8.SelectedIndex == 2)
                {
                    comboBox9.Items.Clear(); comboBox9.Items.Add("6
sps"); comboBox9.SelectedIndex = 0;
                    device_selected = 1;
                    res = 13; res_selected = 3;
                    f_sample = 6;
                }
            }
        }

        private void comboBox9_SelectedIndexChanged(object sender,
EventArgs e)
        {
            // It selects the ADC function mode depending on the combobox
selection
            if (comboBox7.SelectedIndex == 0)
                if(comboBox8.SelectedIndex == 0)
                    switch (comboBox9.SelectedIndex)
                    {
                        case 0: device_selected = 1; f_sample = 400;
break;
                        case 1: device_selected = 2; f_sample = 2000;
break;
                        case 2: device_selected = 3; f_sample = 5000;
break;
                        case 3: device_selected = 4; f_sample = 7400;
break;
                        default: break;
                    }
        }

        private void button3_Click(object sender, EventArgs e)
```

```
        {
            button3.Enabled = false; button4.Enabled = true;
button5.Enabled = false;
            groupBox1.Visible = false;
            comboBox14.Enabled = false;

            switch (comboBox10.SelectedIndex)
            {
                case 0: buffer_size = 25; break;
                case 1: buffer_size = 50; break;
                case 2: buffer_size = 100; break;
                case 3: buffer_size = 1000; break;
                case 4: buffer_size = 10000; break;
                default: break;
            }

            s1_string = "";
            sp.Write(device_selected.ToString());
            sp.Write(res_selected.ToString());
        }

        private void button4_Click(object sender, EventArgs e)
        {
            // Enviamos una X para salir del ADC seleccionado
            sp.Write("x");
            procesa_Datos();
            sp.DiscardInBuffer();
        }
        //

        // Procesado de datos
        private void procesa_Datos()
        {
            // Deshabilitamos el botón "Stop" -> Se inicia el procesado
de los datos
            button4.Enabled = false;

            t = 0;
            t_sample = 1/f_sample;
            power = Math.Pow(2, res);
            chart1.Series.Clear();
            s1.Points.Clear(); s2.Points.Clear();

            foreach (string s in s1_string.Split(erre))
                if (s != "")
                {
                    t = t + t_sample;
                    if (device_selected == 5)
                    {
                        code = Int32.Parse(s.Substring(0, 4),
System.Globalization.NumberStyles.HexNumber);
                        value = (code) * vfs / power;
                        s1.Points.AddXY(t, value);
                        code = Int32.Parse(s.Substring(7, 4),
System.Globalization.NumberStyles.HexNumber);
                        value = (code) * vfs / power;
                        s2.Points.AddXY(t, value);
```

```
                    }
                    else
                    {
                        code = Int32.Parse(s,
System.Globalization.NumberStyles.HexNumber);
                        value = (code) * vfs / power;
                        s1.Points.AddXY(t, value);
                    }
                }

            // Calculamos el histograma
            calcula_Histograma();
            añade_Timeline();

            // Habilitamos el botón "Start" -> Ya se han procesado los
datos y se ha mostrado el gráfico
            button3.Enabled = true; button5.Enabled = true;
            groupBox1.Visible = true;
            comboBox14.Enabled = true;
        }

        private void calcula_Histograma()
        {
            histo.Points.Clear();
            histo.Points.AddXY(0, 0);
            foreach (DataPoint p in s1.Points)
            {
                dp_y = p.YValues[0];
                dp = histo.Points.FindByValue(dp_y, "X");
                if (dp == null)
                {
                    histo.Points.AddXY(dp_y, 1);
                    dp = histo.Points.FindByValue(dp_y, "X");
                    histo_pos = histo.Points.IndexOf(dp);
                    foreach (DataPoint p2 in
s1.Points.FindAllByValue(dp_y))
                        histo.Points[histo_pos].YValues[0]++;
                }
            }
            histo.Points.RemoveAt(0);
        }

        private void añade_Timeline()
        {
            chart1.Series.Clear();
            chart1.Series.Add(s1);
            if (device_selected == 5)
                chart1.Series.Add(s2);

            chart1.ChartAreas[0].AxisX.Title = "t [s]";
chart1.ChartAreas[0].AxisX.TitleAlignment = StringAlignment.Far;
            chart1.ChartAreas[0].AxisY.Title = "v(t) [V]";
chart1.ChartAreas[0].AxisY.TitleAlignment = StringAlignment.Far;
            button5.Text = "Histogram"; histogram_mode = false;
        }
```

```
        private void añade_Histogram()
        {
            chart1.Series.Clear();
            chart1.Series.Add(histo);

            chart1.ChartAreas[0].AxisX.Title = "v(t) [V]";
chart1.ChartAreas[0].AxisX.TitleAlignment = StringAlignment.Far;
            chart1.ChartAreas[0].AxisY.Title = "Samples";
chart1.ChartAreas[0].AxisY.TitleAlignment = StringAlignment.Far;
            button5.Text = "Timeline"; histogram_mode = true;
        }

        private void button5_Click(object sender, EventArgs e)
        {
            // It switchs between histogram or timeline modes
            if (histogram_mode)
                añade_Timeline();
            else
                añade_Histogram();
        }
        //

        // Otros eventos
        private void DataReceivedHandler(object sender,
SerialDataReceivedEventArgs e)
        {
            while(sp.BytesToRead>0)
            {
                s1_string += sp.ReadLine();
                if (buffer_size-- == 0)
                {
                    // Enviamos una X para salir del ADC seleccionado
                    sp.Write("x");
                    procesa_Datos();
                    sp.DiscardInBuffer();
                }
            }
        }

        private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
        {
            if (sp_abierto)
            {
                sp.Close();
                sp.Dispose();
            }
        }
        //
    }
}
```