



Software upgrade for a short-circuit test up to 14 kA peak

Document:

Report

Author:

Renzo Nicolás Cáceda Peña

Director – Co-director:

Jordi Roger Riba Ruiz

Manuel Moreno Eguílaz

Degree:

Degree in Industrial Electronic and Automatic
Engineering

Call:

Prorogation/2022

TREBALL DE FI D'ESTUDIS



ABSTRACT	2
RESUMEN	2
RESUM	3
INDEX OF FIGURES	3
1 INTRODUCTION	6
1.1 OBJECTIVES.....	6
1.2 SCOPE.....	6
1.3 REQUIREMENTS.....	6
1.4 JUSTIFICATION.....	7
1.5 PLANNING.....	8
2 THE SHORT-CIRCUIT MACHINE	9
2.1 DESCRIPTION OF THE SHORT-CIRCUIT MACHINE.....	9
2.2 HARDWARE.....	9
2.3 SOFTWARE.....	11
2.4 RUNNING THE SHORTCIRCUIT MACHINE.....	12
3 HISTORY AND CURRENT STATE OF THE SHORTCIRCUIT MACHINE	13
3.1 PROJECT BY JOSÉ BAILÓN.....	13
3.2 PROJECT BY POL MONREAL.....	14
3.3 CURRENT BUGS AND PROBLEMS.....	19
4 TOOLS AND DEVICES	25
4.1 DEVICES	25
4.1.1 PIC MICROCONTROLLER.....	25
4.1.2 DAQ OMEGA.....	26
4.1.3 DAQ NI.....	26
4.2 PROGRAMMING LANGUAGES.....	27
4.2.1 VISUAL BASIC.NET	27
4.2.2 C LANGUAGE	29
4.3 COMMUNICATIONS.....	30
4.4 SOFTWARE PACKAGES.....	31
4.4.1 VISUAL STUDIO 2022.....	31
4.4.2 PROTEUS DESIGN SUITE.....	31
4.4.3 CONFIGURE VIRTUAL SERIAL PORT DRIVER.....	32
4.4.4 MPLAB IDE V8.63.....	33
5 DEVELOPMENT	34
5.1 METHODOLOGY.....	34
5.2 SOLUTIONS.....	35
6 RESULTS	47
6.1 TEST AND VALIDATION.....	47
7 ENVIRONMENTAL IMPACT	58
8 CONCLUSIONS	59
9 FUTURE WORK	59

10 BIBLIOGRAPHY.....60

Abstract

The objective of this project is to improve an existing graphical interface software written in Visual Basic.NET for the control and configuration of a transformer used in short-circuit tests. The improvement will make it easier to perform tests because the original interface of a previous project contained efficiency errors that made it impossible to carry out several tests continuously and was not very intuitive to use.

The interface software sends the indicated parameters to a PIC18F2580 microcontroller which is in charge of controlling the operation of the short-circuit machine.

The main functions performed are the calculation of phase shift measurements indicated by the interface between the voltage and current through the transformer generated by pulses and making a short circuit with a duration and firing angle also indicated by the interface.

The development of the project had two parts, the first was part of the development of a first interface design and using the Proteus Design Suite Simulator to simulate operation and MPLAB IDE environment because it was necessary to make a small modification and other environments to verify communication and data transfer with the microcontroller.

The development of the second part has consisted in the adaptation of this design, in the original project found in the AMBER laboratory, that previously its communication has been tested.

Due to it is part of a larger project, a study and annotations had to be made to understand the operation of this project, it was also necessary to gather knowledge regarding the VB.NET programming language.

As will be seen in the following points, it was possible to carry out tests with the new design and save the recorded data in a much more efficient and intuitive way.

Resumen

El objetivo de este proyecto es mejorar un software de interfaz gráfica existente escrito en Visual Basic.NET para el control y configuración de un transformador utilizado en pruebas de cortocircuito. La mejora facilitará la realización de pruebas ya que la interfaz original de un proyecto anterior contenía errores de eficiencia que hacían imposible realizar varias pruebas de forma continua y no era muy intuitivo de usar.

El programa de interfaz envía los parámetros indicados a un microcontrolador PIC18F2580 el cual se encarga de controlar el funcionamiento de la máquina de cortocircuito.

Las principales funciones que realiza son el cálculo de las medidas de desfase indicadas por la interfaz entre la tensión y la corriente a través del transformador y la realización de un cortocircuito con una duración y ángulo de disparo también indicados por la interfaz.

El desarrollo del proyecto tuvo dos partes, la primera fue parte del desarrollo de un primer diseño de interfaz, utilizando el simulador Proteus Design Suite para simular el funcionamiento y el entorno MPLAB IDE porque era necesario hacer una pequeña modificación y otros entornos para verificar la comunicación y transferencia de datos. con el microcontrolador.

El desarrollo de la segunda parte ha consistido en la adaptación de este diseño, en el proyecto original encontrado en el laboratorio AMBER, que previamente ha sido probado su comunicación.

Debido a que es parte de un proyecto más grande, se tuvo que hacer un estudio y anotaciones para entender el funcionamiento de este proyecto, también fue necesario recopilar conocimientos respecto al lenguaje de programación VB.NET.

Como se verá en los siguientes puntos, fue posible realizar pruebas con el nuevo diseño y guardar los datos recogidos de una manera mucho más eficiente e intuitiva.

Resum

L'objectiu d'aquest projecte és millorar un software d'interfície gràfica existent escrit a Visual Basic.NET per controlar i configurar un transformador utilitzat en proves de curtcircuit. La millora facilitarà la realització de proves, ja que la interfície original d'un projecte anterior contenia errors d'eficiència que feien impossible fer diverses proves de forma contínua i no era gaire intuïtiu de fer servir.

El software d'interfície envia els paràmetres indicats a un microcontrolador PIC18F2580 el qual s'encarrega de controlar el funcionament de la màquina de curtcircuit.

Les funcions principals que realitza són el càlcul de les mesures de desfasament indicades per la interfície entre la tensió i el corrent a través del transformador i la realització d'un curtcircuit amb una durada i un angle de tret també indicats per la interfície.

El desenvolupament del projecte va tenir dues parts, la primera va ser part del desenvolupament d'un primer disseny d'interfície, utilitzant el simulador Proteus Design Suite per simular el funcionament i l'entorn MPLAB IDE perquè calia fer una petita modificació i altres entorns per verificar la comunicació i transferència de dades amb el microcontrolador.

El desenvolupament de la segona part ha consistit en l'adaptació d'aquest disseny, al projecte original trobat al laboratori AMBER, que prèviament ha estat provat la seva comunicació.

Com que és part d'un projecte més gran, es va haver de fer un estudi i anotacions per entendre el funcionament d'aquest projecte, també va caldre recopilar coneixements respecte al llenguatge de programació VB.NET.

Com es veurà en els punts següents, va ser possible fer proves amb el nou disseny i guardar les dades recollides d'una manera molt més eficient i intuïtiva.

Index of figures

FIGURE 1. GANTT DIAGRAM OF THIS END OF STUDIES PROJECT SOURCE: ORIGINAL PROJECT	8
FIGURE 2. SHORT CIRCUIT MACHINE ASSEMBLY DIAGRAM SOURCE: ORIGINAL PROJECT	9
FIGURE 3. MICROCONTROLLER CONNECTIONS SOURCE: ORIGINAL PROJECT	10
FIGURE 4. TRIGGER PULSE CIRCU SOURCE: ORIGINAL PROJECT	10
FIGURE 5. TRANSFORMER EQUIVALENT CIRCUIT SOURCE: ORIGINAL PROJECT.....	11
FIGURE 6. PROGRAM VISUAL STUDIO SOURCE: ORIGINAL PROJECT.....	11
FIGURE 7. ASSEMBLING THE CHARGE ADDED TO THE AUTOTRANSFORMER SOURCE: ORIGINAL PROJECT.....	13
FIGURE 8. VOLTAGE AND CURRENT SIGNALS SOURCE: ORIGINAL PROJECT	13

FIGURE 9. TRIGGER PULSE OFFSET SOURCE: ORIGINAL PROJECT.....	14
FIGURE 10. SHORT-CIRCUIT DURATION SOURCE: ORIGINAL PROJECT.....	14
FIGURE 11. GENERAL SCHEME PROTEUS SOURCE: POL PROJECT.....	ERROR! BOOKMARK NOT DEFINED.5
FIGURE 12. PULSE GENERATOR ICON SOURCE: NEW PROJECT.....	15
FIGURE 13. PULSE GENERATOR PROPERTIES SOURCE: NEW PROJECT.....	15
FIGURE 14. INCLUDES DECLARATION SOURCE: POL PROJECT.....	16
FIGURE 15. VARIABLE DECLARATION SOURCE: POL PROJECT.....	16
FIGURE 16. INTERRUPT INITIALIZATION SOURCE: POL PROJECT.....	16
FIGURE 17. INTERRUPT INITIALIZATION SOURCE: POL PROJECT.....	16
FIGURE 18. VARIABLE INITIALIZATION SOURCE: POL PROJECT.....	16
FIGURE 19. VARIABLE INITIALIZATION SOURCE: POL PROJECT.....	16
FIGURE 20. HIGH_ISR SUBPROGRAM SOURCE: POL PROJECT.....	17
FIGURE 21. HIGH_ISR SUBPROGRAM SOURCE: POL PROJECT.....	17
FIGURE 22. FLUX DIAGRAM SWITCH DECODER SOURCE: POL PROJECT.....	18
FIGURE 23. PROGRAM VISUAL STUDIO SOURCE: ORIGINAL PROJECT.....	19
FIGURE 24. OMEGA CONFIGURATION SOURCE: ORIGINAL PROJECT.....	20
FIGURE 25. NI CONFIGURATION SOURCE: ORIGINAL PROJECT.....	20
FIGURE 26. AUTOGENERATED PART OF CODE SOURCE: ORIGINAL PROJECT.....	20
FIGURE 27. INITIALIZECOMPONENT FUNCTION OF CODE SOURCE: ORIGINAL PROJECT.....	21
FIGURE 28. VARIABLE INITIALIZATION SOURCE: ORIGINAL PROJECT.....	21
FIGURE 29. FORM1_LOAD SUBPROCESS SOURCE: ORIGINAL PROJECT.....	22
FIGURE 30. SUBPROCESS TO OPEN SERIAL PORT SOURCE: ORIGINAL PROJECT.....	22
FIGURE 31. SUBPROCESS TO SEND DATA SOURCE: ORIGINAL PROJECT.....	23
FIGURE 32. SUBPROCESS TO RECEIVED DATA SOURCE: ORIGINAL PROJECT.....	23
FIGURE 33. SUBPROCESS TO RECEIVED DATA SOURCE: ORIGINAL PROJECT.....	23
FIGURE 34. SUBPROCESS TO CHOOSE DIRECTORY SOURCE: ORIGINAL PROJECT.....	24
FIGURE 35. BUTTON TO DETERMINATE MEAN DATA SOURCE: ORIGINAL PROJECT.....	25
FIGURE 36. PIC18F2580 MICROCONTROLLER. SOURCE: HTTPS://WWW.MICROCHIP.COM/EN-US/PRODUCT/PIC18F2580	25
FIGURE 37. DAQ OMEGA. SOURCE: HTTPS://ES.OMEGA.COM/PPTST/OM-DAQ-USB-2400.HTML	26
FIGURE 38. DAQ USB 6000. SOURCE: HTTPS://WWW.NI.COM/ES-ES/SUPPORT/MODEL.USB-6000.HTML	26
FIGURE 39. VISUAL BASIC.NET LOGO SOURCE: HTTPS://ES.WIKIPEDIA.ORG/WIKI/VISUAL_BASIC_.NET	27
FIGURE 40. CLASS EXAMPLE. SOURCE: VISUAL STUDIO.....	27
FIGURE 41. VOID FORM. SOURCE: VISUAL STUDIO.....	28
FIGURE 42. EXAMPLE FORM. SOURCE: VISUAL STUDIO.....	28
FIGURE 43. EXAMPLE EXECUTION FORM. SOURCE: VISUAL STUDIO.....	28
FIGURE 44. INCLUDE EXAMPLE INITIALIZATION. SOURCE: MPLAB IDE.....	29
FIGURE 45. VARIABLE EXAMPLE INITIALIZATION. SOURCE: MPLAB IDE.....	29
FIGURE 46. EXAMPLE MAIN FUNCTION. SOURCE: MPLAB IDE.....	30
FIGURE 47. EXAMPLE IF CONDITION. SOURCE: MPLAB IDE.....	30
FIGURE 48. EXAMPLE FOR LOOP. SOURCE: MPLAB IDE.....	30
FIGURE 49. VISUAL STUDIO LOGO. SOURCE: GOOGLE IMAGES.....	31
FIGURE 50. PROTEUS SIMULATOR LOGO. SOURCE: GOOGLE IMAGES.....	32
FIGURE 51. INTERFACE CONF. V.SERIAL PORT. SOURCE: GOOGLE IMAGES.....	32
FIGURE 52. MPLAB IDE LOGO. SOURCE: GOOGLE.....	33
FIGURE 53. MODIFICATION CODE. SOURCE: POL PROJECT.....	35
FIGURE 54. SCHEMATIC SKETCHES.....	36
FIGURE 55. NEW INTERFACE. SOURCE: NEW PROJECT.....	36
FIGURE 56. NEW INTERFACE SOURCE: NEW PROJECT.....	36
FIGURE 57. CHECK AND PICTURE BOX. SOURCE: NEW PROJECT.....	37

FIGURE 58. SEND DATA SUBPROCESS. <i>SOURCE: NEW PROJECT</i>	38
FIGURE 59. MESSAGE BOX FOR OFFSET TEST. <i>SOURCE: NEW PROJECT</i>	38
FIGURE 60. MESSAGE BOX OF SHORT-CIRCUIT TEST. <i>SOURCE: NEW PROJECT</i>	38
FIGURE 61. SUBPROCESS TO ENTER, CALL CREATE FILE AND SEND DATA. <i>SOURCE: NEW PROJECT</i>	39
FIGURE 62. NEW MESSAGE BOX. <i>SOURCE: NEW PROJECT</i>	39
FIGURE 63. MESSAGE BOX CODE. <i>SOURCE: NEW PROJECT</i>	39
FIGURE 64. CLEAN VARIABLES SUBPROCESS. <i>SOURCE: NEW PROJECT</i>	40
FIGURE 65. DATA RECEPTION FLUX. <i>SOURCE: NEW PROJECT</i>	41
FIGURE 66. DATA RECEPTION FLUX. <i>SOURCE: NEW PROJECT</i>	42
FIGURE 67. CREATE FILE SUBPROCESS. <i>SOURCE: NEW PROJECT</i>	43
FIGURE 68. NEW LOAD SUBPROCESS. <i>SOURCE: NEW PROJECT</i>	44
FIGURE 69. NEW INTERFACE. <i>SOURCE: NEW PROJECT</i>	44
FIGURE 70. SUBPROCESS TO CREATE THREADS. <i>SOURCE: POL PROJECT</i>	45
FIGURE 71. VARIABLE INITIALIZATION. <i>SOURCE: POL PROJECT</i>	45
FIGURE 72. NEW INTERFACE. <i>SOURCE: VISUAL STUDIO</i>	46
FIGURE 73. SHORT-CIRCUIT SIMULATED TEST. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	48
FIGURE 74. OFFSET SCREENSHOT. <i>SOURCE: PROTEUS SCREENSHOT</i>	48
FIGURE 75. DURATION SCREENSHOT. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	49
FIGURE 76. BIT FRAME. <i>SOURCE: PROTEUS SCREENSHOT</i>	50
FIGURE 77. OFFSET MEASUREMENT TEST. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	50
FIGURE 78. BIT FRAME. <i>SOURCE: PROTEUS SCREENSHOT</i>	51
FIGURE 79. SHORT-CIRCUIT FILE NAME. <i>SOURCE: NEW PROJECT</i>	51
FIGURE 80. DATA SAVED. <i>SOURCE: NEW PROJECT</i>	52
FIGURE 81. MEAN DATA SAVED. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	52
FIGURE 82. DATA SAVED. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	52
FIGURE 83. MEAN DATA SAVED. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	52
FIGURE 84. OFFSET MEASUREMENT REAL TEST. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	53
FIGURE 85. OFFSET MEASUREMENT REAL TEST. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	53
FIGURE 86. NEW MESSAGEBOX. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	54
FIGURE 87. STATE TRAMA PIC RECIBIDA. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	49
FIGURE 88. OFFSET TEST FILE NAME. <i>SOURCE: NEW PROJECT</i>	54
FIGURE 89. DATA SAVED. <i>SOURCE: NEW PROJECT</i>	55
FIGURE 90. REAL SHORT-CIRCUIT TEST. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	55
FIGURE 91. NEW MESSAGEBOX. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	56
FIGURE 92. SHORT-CIRCUIT TEST. <i>SOURCE: VISUAL STUDIO SCREENSHOT</i>	56
FIGURE 93. SHORT-CIRCUIT VOLTAGE-CURRENT FILE NAME. <i>SOURCE: NEW PROJECT</i>	57
FIGURE 94. SHORT-CIRCUIT TEMPERATURE FILE NAME. <i>SOURCE: NEW PROJECT</i>	57
FIGURE 95. SHORT-CIRCUIT VOLTAGE-CURRENT DATA. <i>SOURCE: NEW PROJECT</i>	57
FIGURE 96. SHORT-CIRCUIT TEMPERATURE DATA. <i>SOURCE: NEW PROJECT</i>	57

1 Introduction

1.1 Objectives

The objective of this Bachelor's Degree Final Project is to improve an existing software, programmed in Visual Basic, for the control of a high current low voltage transformer used in short-circuit tests of up to 14 kA peak. The software must allow the user to set the parameters of the test, control the test and obtain the data generated during the test.

1.2 Scope

The scope of this Bachelor's Degree Final Project includes:

- Analysis of the existing code for the microcontroller for a better understanding of the operation of the project.
- Review of the existing code for PC to understand how it works and detect its bugs, with the aim of improving the structure of the new code.
- Carry out an improvement of PC software to control and establish various parameters of the microcontroller in charge of applying the functions of phase shift measurement, triggering of the short-circuit test and reading of the different sensors.
- It will be required to perform a search on the syntax of the Microsoft Visual Basic programming language.
- Some tests will be carried out in an emulated environment, that is, without the short-circuit machine or the real hardware, using the Proteus Design Suite software.
- Once the code in the emulated environment works correctly, the tests will be carried out in the laboratory of the AMBER-UPC High Voltage Research and Testing Center, to check the correct operation of the software in a real environment.

1.3 Requirements

In this section the requirements for the realization of this Bachelor's Degree Final Project are listed:

- Knowledge of C programming language for microcontrollers.
- Knowledge of Microchip's MPLAB IDE development environment.
- Knowledge of Microsoft Visual Basic programming language.
- Knowledge of Microsoft Visual Studio development environment
- Knowledge of the Proteus Design Suite emulation environment.
- Knowledge of Virtual Serial Port Drive serial port emulation software.
- Knowledge of the short-circuit machine installed in the laboratory of the High Voltage Research and Test Center AMBER-UPC.

1.4 Justification

The realization of this project arises since the control software of the short-circuit machine installed in the laboratory of the High Voltage Research and Test Center AMBER-UPC presents serious operating problems, preventing systematic short-circuit tests. In effect, the software exhibits anomalous behavior almost randomly. Sometimes the short circuit is not activated. At other times, the captured data is not saved correctly.

The short-circuit machine control software was developed in 2016, thanks to a technician from the AMBER laboratory, who developed both the hardware (electronic board based on an 8-bit microcontroller) and the system software (microcontroller and PC).

This microcontroller has two general functions:

- To measure the phase between the voltage and current caused by the impedance of the load located on the secondary of the transformer of the short-circuit machine.
- To generate a pulse capable of activating an SCR type thyristor, which connects the primary of the transformer to the grid for a pre-established period of time.

Both original codes have significant design and programming errors (bugs), which make it impossible for the short-circuit machine to work correctly.

For all of the above, in 2020 the AMBER laboratory published an offer to improve the microcontroller code in C language, so that a student based his Bachelor's Degree Final Project on making this improvement. The code worked fine in an emulated environment, but it could not be tested in a real environment due to the Coronavirus SARS-CoV-2 (Covid-19) pandemic.

At the beginning of the year 2022, an offer was made to make an improvement, in this case, of the PC code based on Visual Basic, due to the aforementioned problems that it presented. Consequently, my Bachelor's Degree Final Project consists of improving this code for the PC.

The PC software allows entering the parameters and giving the commands to the microcontroller, while receiving the results (voltages, currents and temperatures) and processing them.

The Proteus Design Suite emulation software is used to simulate the operation of the microcontroller, without having to be physically in the AMBER laboratory. Once the correct operation is verified, the appropriate tests are carried out to confirm the correct behaviour in a real physical environment.

Finally, I believe that developing a desktop application that will be used for future tests within the AMBER laboratory can provide me with knowledge. Although it is true that this type of development has not been studied in the Degree in Industrial Electronic Engineering and Automation, certain programming subjects have given me enough knowledge to be able to carry out this project without great difficulties.

1.5 Planning

1. Study of the microcontroller code written in C language

As a first step to be able to start with the project, it is necessary to review the code in C language in order to understand the behaviour.

2. Study of the PC code written in Microsoft Visual Basic

On the other hand, it is also necessary to know how the PC code works and what relationship of variables it has with the microcontroller code, so that code will be analyzed.

3. Propose the methodology to follow

Once you have knowledge of the operation of both codes, it is necessary to consider what steps will be taken into account and how to proceed to improve the code in Visual Basic.

4. Make improvements of the PC code

This process will be the longest, since it will contain the development of code improvements in terms of efficiency and functionality.

5. Carry out tests in the AMBER laboratory

Once the code has been improved and its operation tested in the emulated environment with the simulator, we will go to the laboratory to carry out different tests and check that there are no errors or, if there are, solve them.

6. Write the corresponding report

Finally, it will be necessary to finish writing the report with all the information collected from the different tests carried out and the improvements made to the code.

Num.	Apartado	Duración	Febrero	Marzo	Abril	Mayo	Junio
1.	Estudio del código en C	1					
2.	Estudio del código en visual basic	2					
3.	Plantear metodología a seguir	1					
4.	Realizar mejoras en código de Visual Basic	6					
5.	Realizar ensayos en el laboratorio AMBER	5					
6.	Finalizar memoria	2					

Figure 1. Gantt diagram of this End of Studies Project.

You can see in Fig. 1 the Gantt chart with the prior planning to follow during the completion of this Bachelor's Degree Final Project. This is an initial estimation regarding the duration of each section, which may lead to changes during the development.

2 The short-circuit machine

2.1 Description of the short-circuit Machine

The short-circuit machine is located in the AMBER laboratory for the Electrical Department of the Polytechnic University of Catalonia. The parts that make up the machine are the following:

- Hardware, components and other devices that allow the physical test to be carried out, as well as those that allow the generation of the desired voltage and/or the acquisition of data. In the next point they will be named in detail.
- And software, composed of the programs that allow communication, data transfer and which allows the control and operation of the test.

2.2 Hardware

The design of the short-circuit machine is made up of various parts and components. In Figure 2 you can see the components dedicated to the control system, power, the transformer used and two of the DAQ's for data acquisition, both for voltage-current and temperature.

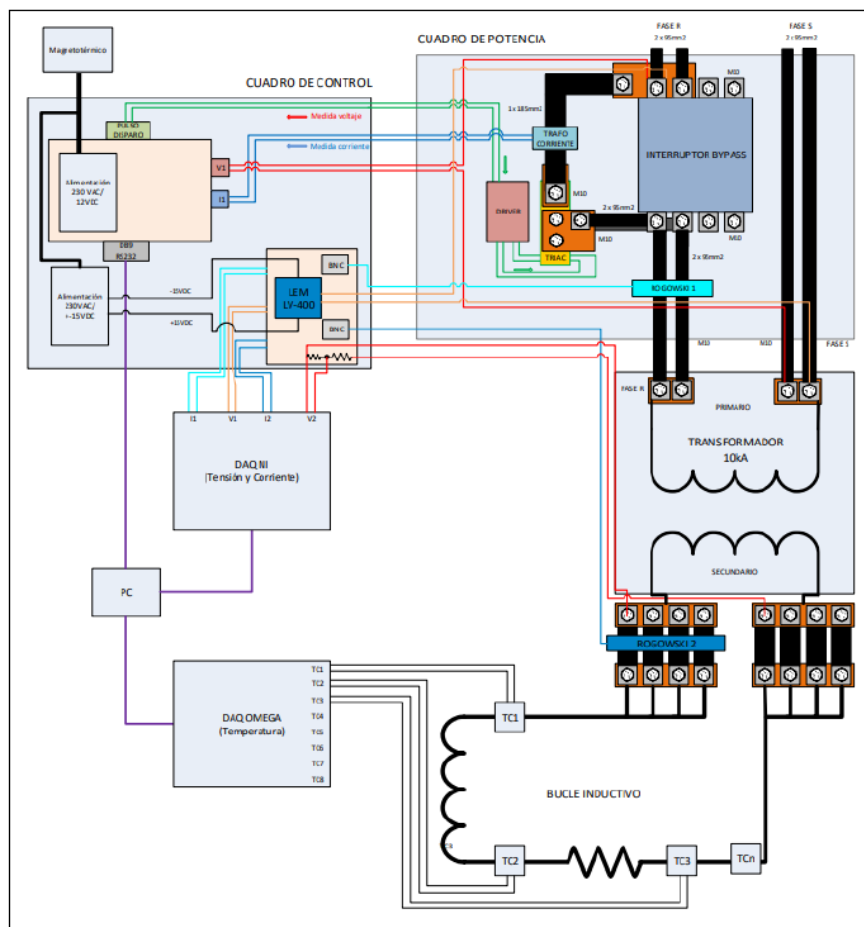


Figure 2. Short circuit machine assembly diagram. Source: Original project

The subsystem dedicated to control the short-circuit machine consists of a PIC18F2580 microcontroller and several integrated circuits and small power converters, there are also two pins (21 and 22) that receive the pulses for the voltage and current signals obtained by the power supply from the electrical grid, which feeds the primary of the transformer. Figures 3 and 4 belong to the design implemented for the simulator [1].

The power part consists of a high current low voltage transformer and the SCR model MCC162-16IO1 (see Figure 4), capable of connecting the primary of the transformer with the grid and a previously mentioned autotransformer that allows us to vary the voltage and current that circulates through the primary, fed by the grid at 230V [2].

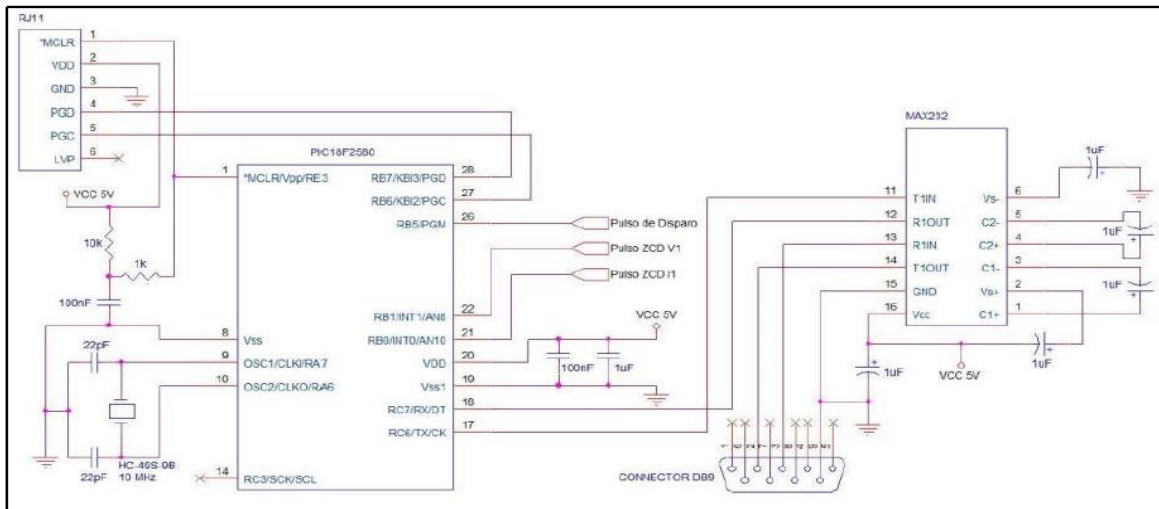


Figure 3. Microcontroller connections. Source: Original project

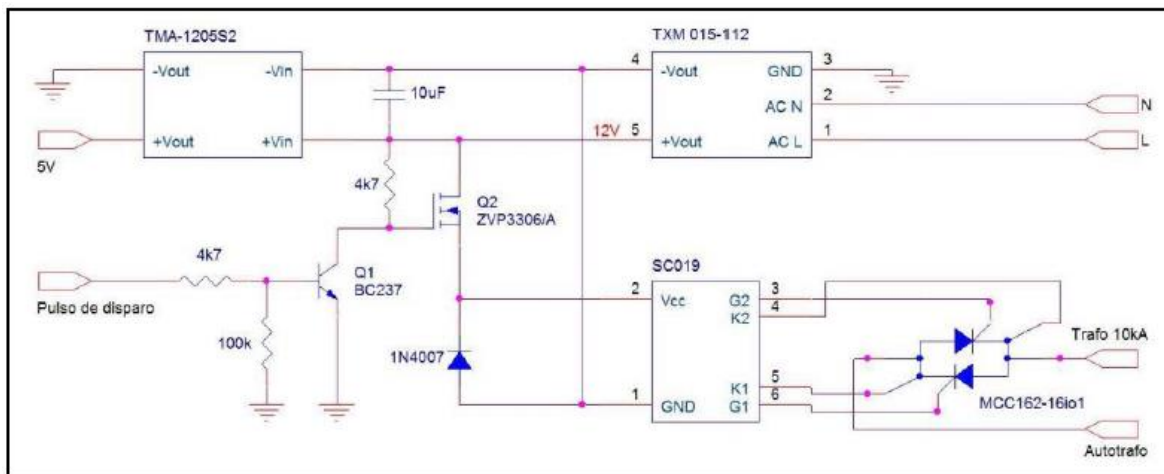


Figure 4. Trigger pulse circuit. Source: Original Project.

The transformer used for the tests is 10 kA (RMS) and resistive and inductive loads are connected to the secondary so that the operation for the short-circuit tests varies. The equivalent circuit is as follows:

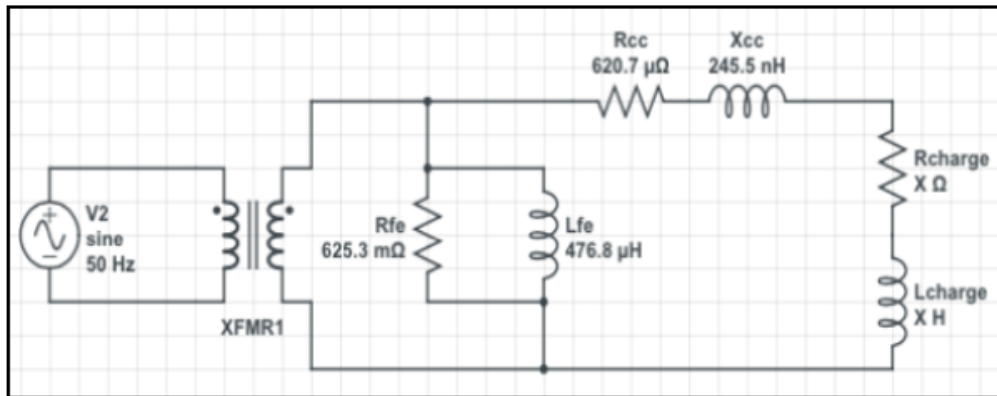


Figure 5 Transformer equivalent circuit. Source: Original Project.

Finally, there are the measurement devices, which are the DAQ to acquire the voltage-current data from the primary and the temperature DAQ connected with thermocouples to the secondary at different points to record and save the recorded values [3,4].

2.3 Software

The software design was implemented jointly with the development of the short circuit machine. On the one hand, the software for the control of the SCR is implemented in the PIC18F2580 microcontroller. The operation of this code basically has two main functions to perform: launching the pulse signal that initiates the short circuit, indicating the duration and angle of the shot and calculate the measured phase shift between the voltage and current signals. The development will be explained in detail in the next sections.

On the other hand, the graphical interface software was designed to be able to send and configure different parameters required in the test. Through it, the parameters referring to the short-circuit test are introduced.

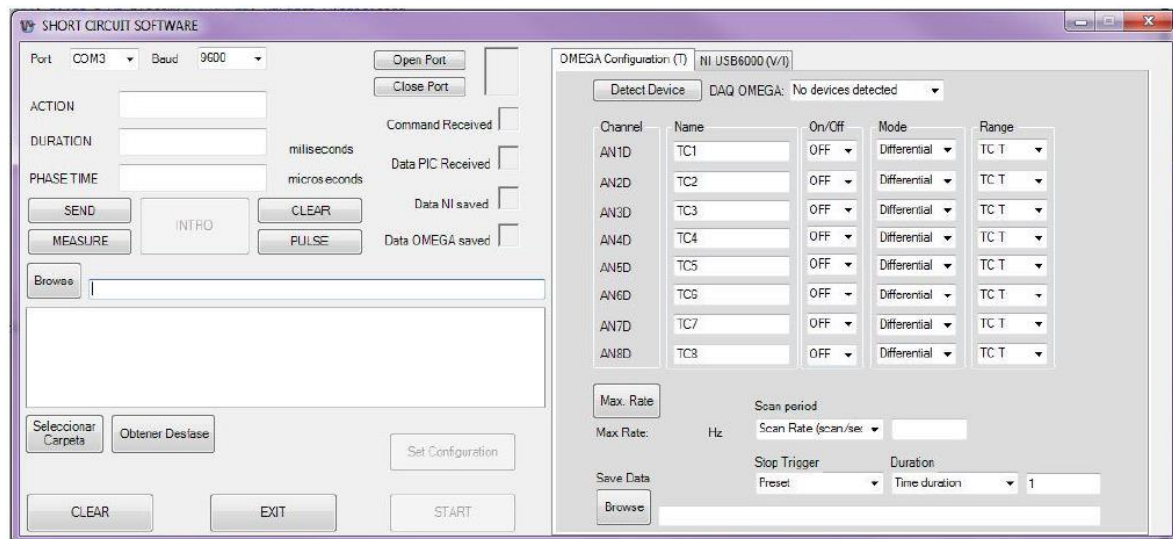


Figure 6. Program Visual Studio. Source: Original Project.

In Figure 6 the layout of the GUI is shown. On the right side the buttons for data acquisition are configured and, on the left, the parameters to be entered required for the test, both the measurements and the trigger pulse to activate the short circuit.

2.4 Running the short-circuit machine

The way to execute both tests is detailed in the following points.

1- The Visual Studio project is executed, so the application will open. The COM port and the appropriate communications speed (COM2 and 9600) are selected manually. Finally, the Open Port button is pressed to open the serial port connection.

2- Next, the Proteus project is executed to connect with the microcontroller. We will see that the Serial Port State indicator will turn green, confirming the connection.

It should be noted that the Visual Studio project must be executed first and then, the Proteus project, because the microcontroller sends the byte 0xEE to confirm the connection. Thus, when you first run Visual Studio, the application is put into standby mode to receive that data. Otherwise, the data will be sent and cannot be read by the application, as previously mentioned. The status icon would turn green, but if you do not follow this order, you will not be able to due to the code structure.

3- Once the connection between the microcontroller and the PC has been established, either of the two tests can be carried out. We start with the measurements, so we click on the "Measure" button and the data that gives the order to the microcontroller will be sent. Continue entering the value of the number of offset measurements to be calculated in the "DURATION" field. Continue clicking on the "Browse" button to determine in which folder and file the received offset data will be saved. As the last steps, click on "INTRO" and a message appears to confirm and finally, it is sent by pressing the "SEND" button.

4- So that once the frame corresponding to the calculation of measurements has been sent, the icons of "Command to PIC Received" and "Data PIC Received" will change color to green, indicating that the microcontroller has confirmed the reception of the frame correctly and the offset data has been received.

5- In the event that you want to know the average value of the test, press the "Browse" button to be able to choose the folder and view the list of files where the data is saved and select the one you want. Once the file is selected, the "Obtener Desfase" button will be pressed and the average value of the offsets will be seen in the "PHASE TIME" text field.

6- In order to carry out a short-circuit test and since a test was previously carried out, the "CLEAR" button must be pressed to initialize the variables used to default values again, since they were modified in the previous test. Continue pressing the "PULSE" button to indicate that you want to perform a pulse for the short circuit test.

7- Proceed by entering in the "DURATION" and "PHASE TIME" fields, the duration of the short-circuit in hundredths of a second and the phase angle in seconds, respectively. Like the previous test, "INTRO" was pressed, we confirmed and the "SEND" button was pressed.

8- In this case the only icon that changes color will be "COMMAND TO PIC RECEIVED", to confirm the reception of the parameters by the microcontroller.

3 History and current state of the short-circuit machine

3.1 Project by José Bailón

The description of the short-circuit machine previously mentioned was part of the project carried out by José Bailón. Beyond the hardware and software design of the project, he carried out different tests to verify the operation of the system. So, for the phase shift measurement test, he implemented an equivalent circuit with a low power autotransformer. Using three inductive loads and one resistive load several phase measurements were done [5].

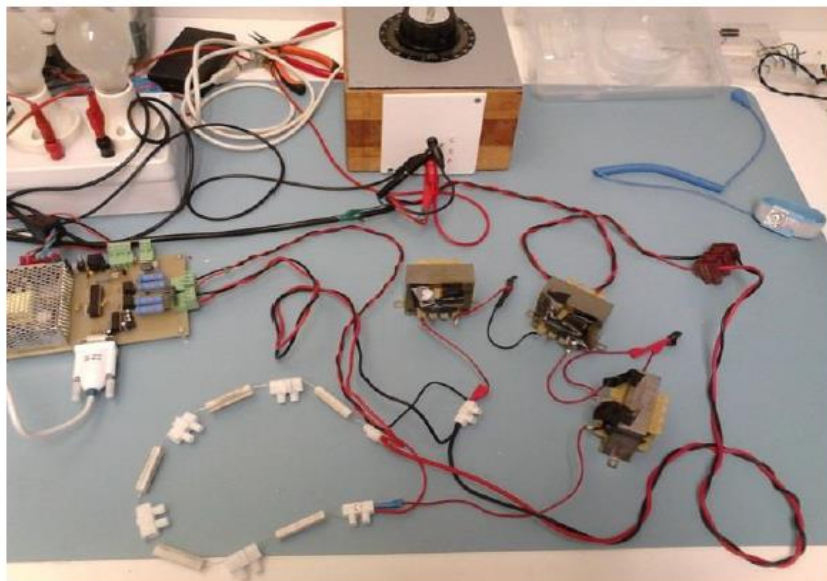


Figure 7. Assembling the charge added to the autotransformer. Source: Original Project.

Figure 7 shows the test assembly implemented with the autotransformer to be able to carry out the mentioned tests. Applying voltage to the RL load, the two signals were obtained at the input pins of the board.

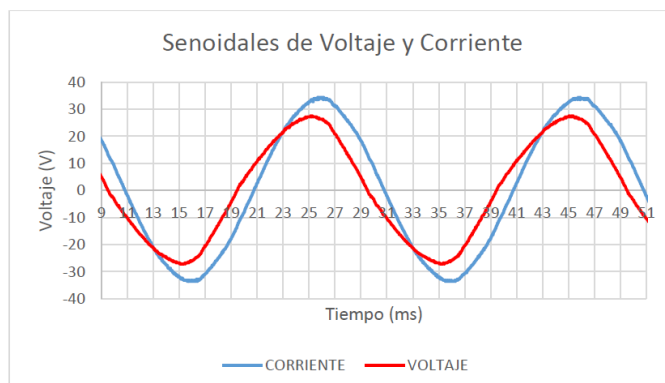


Figure 8. Voltage and current signals. Source: Original Project.

It can be verified that the signals are out of phase between 1 and 2 milliseconds, before passing through the square wave converter circuit.

To check the trigger pulse, the phase was configured from the interface with a 2 ms delay, as can be seen in Figure 9.

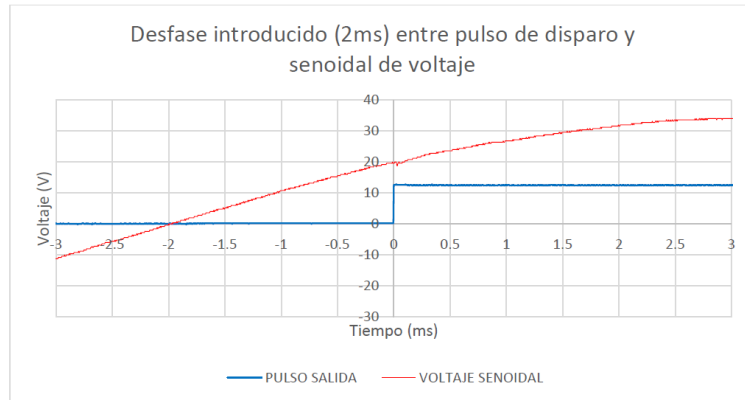


Figure 9. Trigger pulse offset. Source: Original Project.

To verify that the duration is also the one previously introduced in the interface, another test was carried out, so that it could be seen that it worked correctly. The duration of 100ms is correct.

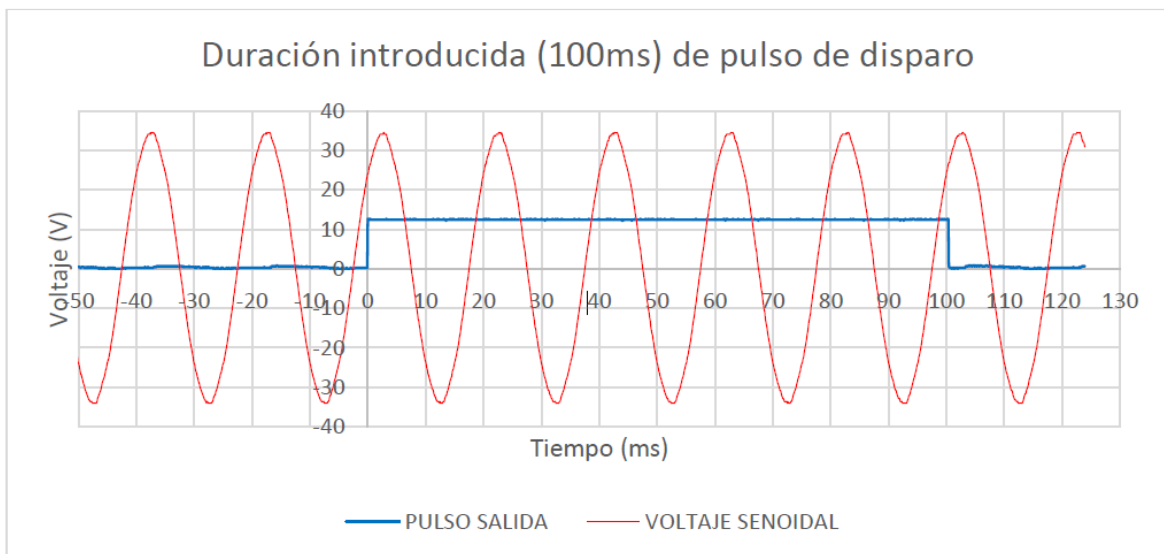


Figure 10. Short-circuit duration. Source: Original Project.

3.2 Project by Pol Monreal

Due to the fact that during the development of the project the state of alarm and its consequent quarantine was declared, the engineering student Pol Monreal was unable to carry out real tests in the AMBER laboratory. Therefore, he tested all the new design

of the microcontroller code written in C language, making use of a simulator Proteus Suite Design [6].

In order to check that the code worked properly, a project was used in the Proteus simulator. Since it was not possible to go to the laboratory to carry out tests, the project only worked when it was simulated and, on the other hand, the data acquisition implemented in the laboratory was not verified.

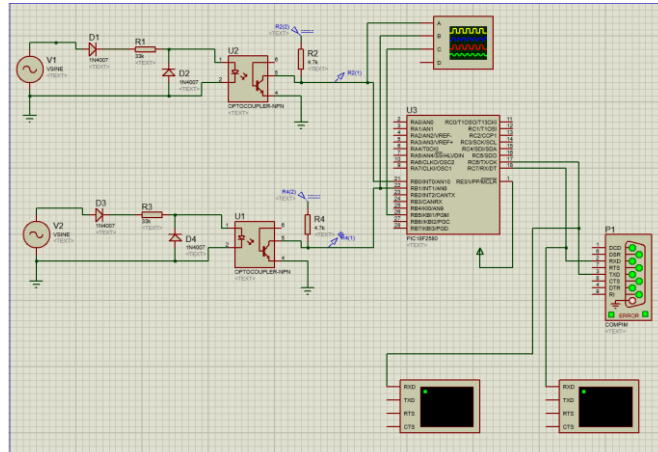


Figure 11. General scheme Proteus. Source: Pol Project.

As can be seen in Figure 11, the schematic is made up of a set of components that generate the square signal of voltage and current, emulating in some way the real components, the PIC microcontroller with its respective input-output pins and finally, on the right side, you can see the component that simulates serial port communication. In addition, it includes an oscilloscope at the output of the generator to view the square signal and a couple of windows in which to view the data that is sent and received [7].

This design was part of my project at the beginning, but due to software version issues, some components of the signal generator stopped working. Therefore, the whole set was replaced by a tool called Pulse Generator, which, as its name indicates, generates a square signal with parameters adjustable by the user.

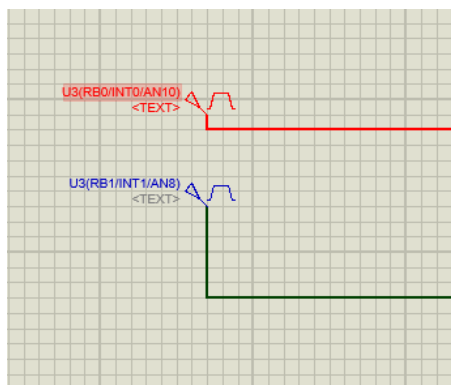


Figure 12. Pulse generator icon. Source: New Project.

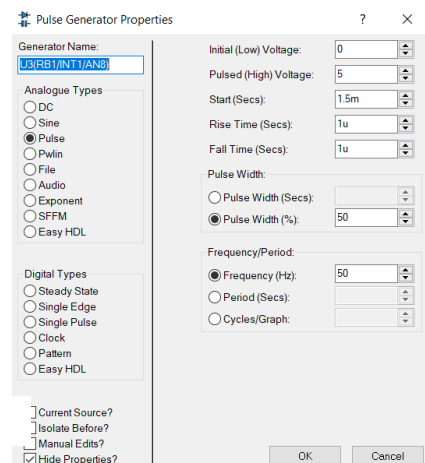


Figure 13. Pulse generator properties Source: New Project.

The original code for microcontroller control had a messy structure and was unintelligible to someone outside the project. Therefore, the design of the new code will be described in more depth than the previous project because my project starts from it.

First of all, it is necessary to include the necessary libraries and declare the variables that it will use during its execution.

```

/** I N C L U D E S ****
#include <pl0f2500.h>
#include <timers.h>
#include <delays.h>
#include <usart.h>
#include "ConfigBits.h"

//*****Declaracio de les variables*****
unsigned char j, LSB_P, LSB_D, param, timer_FB, timer_LB, data; // variables 8 bits
unsigned int timer_act, MSB_P, MSB_D, a; // variables de 16 bits
unsigned long paquets_seg, preload_des, preload_dur, durada, desfase; // variables de 32 bits

```

Figure 15. Variable declaration Source: Pol Project.

Figure 14. Includes declaration Source: Pol Project.

It is worth mentioning that the entire code will not be shown due to its length, but it is included in the annexes of this project.

Next, you can see part of the configuration of some interrupts provided by the microcontroller and its initializations, which allows us to launch the interrupts when certain conditions are met.

```

Flags.NewMeasure = 0;
Flags.ZeroCrossing = 0;

TRISCbits.TRISC6 = 0; // TX UART
LATBbits.LATB5=0;
LATBbits.LATB4=0;
TRISBbits.TRISB5 = 0;
TRISBbits.TRISB4 = 0;

```

Figure 16. Interrupt initialization Source: Pol Project.

```

INTCONbits.INT0IF = 0;
INTCONbits.INT0IE = 1;
INTCON2bits.INTEDG0 = 0; // Interruption on FALLING edge

INTCON3bits.INT1IF = 0;
INTCON3bits.INT1IE = 1;
INTCON2bits.INTEDG1 = 0; // Interruption on FALLING edge

```

Figure 17. Interrupt initialization Source: Pol Project.

All program variables are initialized to 0.

```

j = 0;
MSB_P = 0;
LSB_P = 0;
MSB_D = 0;
LSB_D = 0;
param = 0;
desfase = 0;
durada = 0;
preload_des = 0;
preload_dur = 0;

```

Figure 18. Variable initialization Source: Pol Project.

```

paquets_seg = 0;
a = 0;
timer_act = 0;
timer_FB = 0;
timer_LB = 0;
state_usart = INICI_TRAMA;
state_function = ESPERA;

Flags.NewMeasure = 0;
Flags.ZeroCrossing = 0;

```

Figure 19. Variable initialization Source: Pol Project.

After the declaration of variables, an infinite loop is included, a typical *while* (1), because the execution of the program will always be in operation waiting for a new data that arrives through the serial port. Inside this loop there is a finite state machine for the control of the different tests.

The last part of the code includes an interrupt vector, where the CPU jumps to each time an interrupt occurs, which in turn calls the *high_isr* interrupt service routine. This interrupt service routine has two main functions: the first is when an external interrupt occurs, so it will initialize Timer1 to 0 and activate a variable because there has been a zero crossing. Otherwise, if a current interruption is detected, it will save the value of timer1 in another variable and activate another variable indicating that a new offset has been calculated.

In this way, the zero crossing of the voltage signal will indicate the beginning of the shift and when the zero crossing of the current is detected, the time value will be saved to be sent as a phase shift parameter.

```
void high_isr (void)
{
  if(INTCON3bits.INT1IF) //Det
  {
    WriteTimer1(0x00); //Pos
    INTCON3bits.INT1IF = 0;
    Flags.ZeroCrossing = 1;
  }
}
```

Figure 20. *High_isr* subprogram
 Source: Pol Project.

```
if(INTCONbits.INT0IF) //Detec
{
  timer_act = ReadTimer1();
  INTCONbits.INT0IF = 0;
  Flags.NewMeasure = 1; //E
}
} // FI de high_isr
```

Figure 21. *High_isr* subprogram
 Source: Pol Project.

As previously mentioned, during the execution of the loop, the measurement and short-circuit test is carried out. The loop consists of a selection structure, switch-case type, divided into 3 cases: the first consists of waiting and receiving data from the serial port, which, depending on the value received, force one or another test.

The second case will be executed if it has been so indicated by means of a reception frame decoded in the previous case, which consists of performing the pertinent calculations to calculate the phase shift value between the voltage and current zero crossing, in turn sending the value of each offset through the serial port to the application.

The third, on the other hand, makes the pertinent calculations to carry out the trip that allows activating the SCR and carrying out the short-circuit test.

After the calculation of the short-circuit test, all the variables used during the execution are initialized to zero so that there are no problems in the next test.

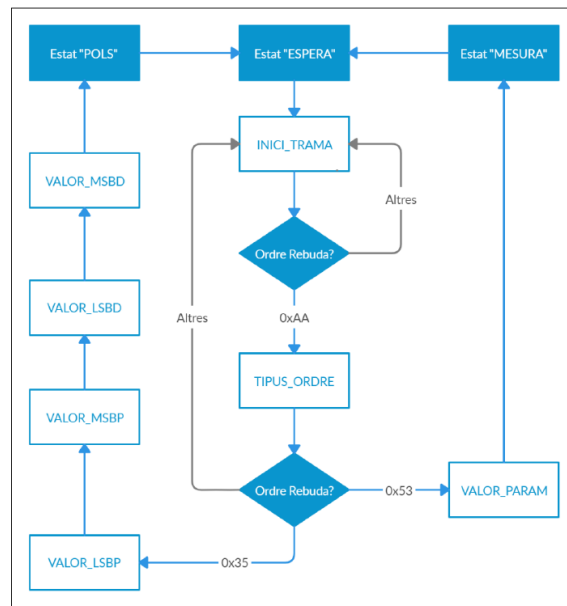


Figure 22. Flux Diagram Switch Decoder Source: Pol Project.

Figure 22 represents the structure implemented in the development done by Pol Montreal. Regarding the communication between the graphical interface and the microcontroller, it is possible to establish it through a serial port and a series of frames coded specifically for this project.

The frame sent by the microcontroller to the PC consists of the following fields:

- The first byte sent during a frame transfer is 0xEE, whose value indicates that communication has been established correctly through the serial port.
- The 0xBB byte is sent just after receiving the last byte of the frame corresponding to the parameter that indicates the number of measurements in the case of a measurement test or to the duration and phase angle parameters of the short-circuit test.
- 0x55 is the byte corresponding to the beginning of the frame sending of the offset data generated by the measurement test.
- Finally, the offset data from performing the measurement test is sent, because the data is 16 bits, but the number of bits allowed in each packet sent through the serial port only allows 8 bits, for so the data is sent in two parts. First, the 8 most significant bits (MSB) and then, the 8 least significant bit (LSB).

The frame sent by the PC consists of the following fields:

- The first sent byte, which indicates the start of data transmission, is 0xAA. This byte will be sent each time a test is performed, even if it is continuous.
- After sending the previous byte and depending on the type of test to be carried out, one byte or another is sent. In the case of carrying out the measurement test, it will be byte 0x53. It should be noted that this value, encoded in ASCII, corresponds to the character S, and it will be used during the development of the application for its good understanding with the microcontroller.

- As a consequence of carrying out the measurement test and sending its corresponding byte, the number of measurements to be carried out (Nmeasurements in the VB code) is sent. Because the number of bits is limited to 8, only 255 measurements can be made per test.
- In case of wanting to carry out the short-circuit test, after sending the frame start byte, the corresponding byte to carry out the short-circuit test is sent, 0x35 (character 5 in ASCII code).
- Once the short-circuit byte is sent, the transmission of data bytes of both duration and phase shift begins. First, the 8 most significant bits of the offset duration are sent. Then, the remaining 8 and lastly the offset angle bits, first the 8 most significant bits (MSB) and then, the last 8 least significant bits.

3.3 Current bugs and problems

The original graphical interface software presented a chaotic mess on the visual side, which made it very difficult to deduce the right order of the buttons for configuration. On the other hand, the code had several software bugs, unused variables, unused functions, etc. Many of the buttons were not well conditioned since, during the tests, if you clicked on a button by mistake, it allowed you to do it when you should not and at the same time, it completely conditioned the operation of the application, the data acquisition did not have a consistent relationship. Sometimes it worked a few times in a row and suddenly it stopped acquiring, so the application had to be restarted as many times as necessary to be able to carry out tests. Below there are some important points that reflect these errors.

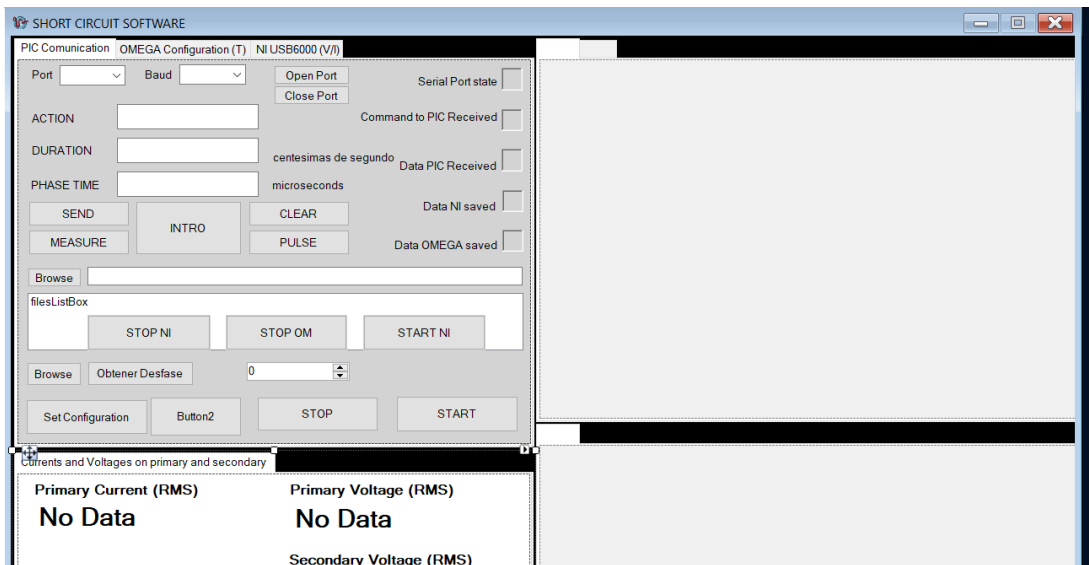


Figure 23. Program Visual Studio. Source: Original Project.

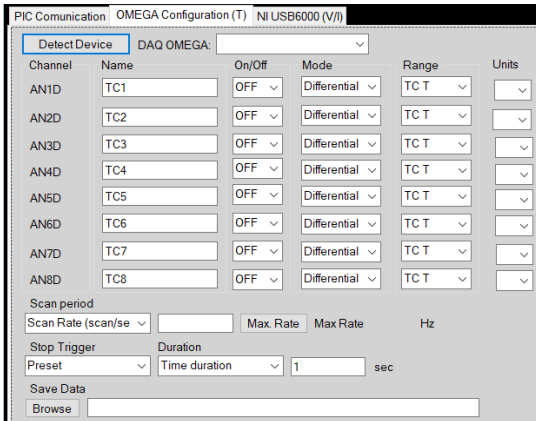


Figure 24. Omega configuration. Source: Original Project.

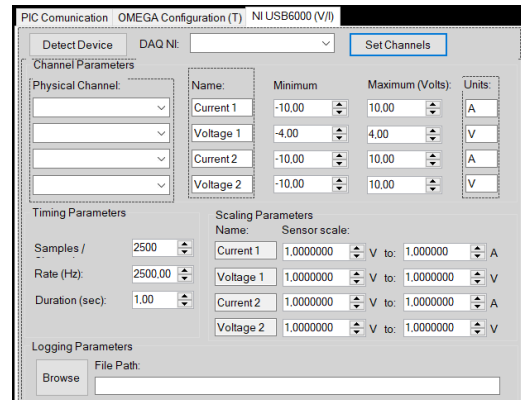


Figure 25. NI configuration. Source: Original Project.

As you can see in Figure 23, the code has 3 panels in this version. The lower left panel contains the reading values of the voltage-current DAQ and the two on the right, even if there is nothing, would contain the graphs obtained from the temperature DAQ. In later sections we will see the results.

To begin with, the used form contains useless buttons, which only make it difficult to understand how to configure the tests. As for the code, it is divided into four different parts. The first contains the Imports of the different libraries necessary for the execution of the code.

The other three remaining ones are inside the MainForm Class. The first begins with code auto-generated by the project itself when the main form is designed, in it the class constructor and the dispose function are defined, which allows freeing memory that has been allocated. In addition, all components used in the design of the form are declared and initialized.

```

Public Class MainForm
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    0 referencias
    Public Sub New()
        MyBase.New()

        Application.EnableVisualStyles()
        Application.DoEvents()

        ' Required for Windows Form Designer support
        InitializeComponent()

        ' TODO: Add any constructor code after Initiali
        ' dataTable = New DataTable
    End Sub
    
```

Figure 26. Autogenerated part of code. Source: Original Project.

```

Friend WithEvents TabControl3 As System.Windows.Forms.TabControl
Friend WithEvents TabPage8 As System.Windows.Forms.TabPage
Friend WithEvents lbl_jouleInt As System.Windows.Forms.Label
Friend WithEvents Label34 As System.Windows.Forms.Label
Friend WithEvents Label36 As System.Windows.Forms.Label
Friend WithEvents lbl_i2RMS_T As System.Windows.Forms.Label

Friend WithEvents Button3 As System.Windows.Forms.Button

1 referencia
<System.Diagnostics.DebuggerStepThrough(> Private Sub InitializeComponent()
    Me.components = New System.ComponentModel.Container()
    Dim resources As System.ComponentModel.ComponentResourceManager = New System.Co
    Me.writeToFileSaveFileDialog_NI = New System.Windows.Forms.SaveFileDialog()
    Me.fileToolTip = New System.Windows.Forms.ToolTip(Me.components)
    Me.SerialPort1 = New System.IO.Ports.SerialPort(Me.components)
    Me.Label_port = New System.Windows.Forms.Label()
    Me.tbRx = New System.Windows.Forms.TextBox()
    Me.tbTx = New System.Windows.Forms.TextBox()
    Me.btnSend = New System.Windows.Forms.Button()
    Me.btnClear = New System.Windows.Forms.Button()

```

Figure 27. InitializeComponent function of code. Source: Original Project.

As a third part of the code, the declaration of the variables that will be used during execution appears. Some variables are initialized to 0, if they are declared as Integer, True or False, if Boolean or String, because for use in the code it is necessary that they have an initial value before execution.

```

'Serial Communication and Offset calculation VARIABLES
Private readBuffer As String = String.Empty
Private short_c As Boolean
Private Bytenumber As Integer
Private ByteToRead As Integer
Private byteEnd(2) As Char
Private comOpen As Boolean
Private useTextFileWrite As Boolean
Private CharToRead As Char
Private desfase As Integer = 0
Private desfase_media As Integer = 0
Private dato As Integer = 0
Private param As Integer = 0
Private accion As String = ""
Private duracion As Integer = 0
Private MSB_Desfase As Integer = 0
Private LSB_Desfase As Integer = 0
Private MSB_Pulso As Integer = 0
Private LSB_Pulso As Integer = 0
Private pulso As Integer = 0
Private fileMeasuresNameWrite As String
Private confirm As Integer = 0
Private bit_USB As Integer = 0
Private dispositivos(3) As String
Private NI_detect As Boolean = False
Private OMEGA_detect As Boolean = False

```

Figure 28. Variable initialization. Source: Original Project.

Finally, the different functions and procedures relevant to certain events or conditions are declared.

The functionalities are divided into 3 regions, which is a way of packaging the type of functions that each section of lines will perform. Before starting with the regions and their functions, the functions *Form1_FormClosed* and *Form1_Load* are declared, which are the functions referring to when the application is first loaded and when it is closed, respectively. Both the variable declaration lines and these two functions are correct, as they are required to initialize variables and initialize certain components for proper operation.

```
Public Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    'rs.FindAllControls(Me)
    'rs.ResizeAllControls(Me)
    btnIntro.Enabled = False
    btnSetconfig.Enabled = False
    btnStart.Enabled = False
    btnStop.Enabled = False
    btnComOpen.Enabled = True

    ' read available COM Ports:.
    Dim Portnames As String() = System.IO.Ports.SerialPort.GetPortNames
    If Portnames Is "" Then
        MsgBox("There are no Com Ports detected!")
        Me.Close()
    End If
End Sub
```

Figure 29. *Form1_Load* subprocess. Source: Original Project.

Next, the region that contains the functions related to the communication configuration and produced events appears. In it, the open port button is initially declared, which configures the serial port and its respective parameters, such as the length of data bits, the parity check protocol, among others, in addition to establishing values in components. It continues with functions such as the button to close serial port communication or clear variables after each test. In this case, the first would be well defined, but the second in terms of efficiency could be improved, making it automatic after the test.

```
Private Sub btnComOpen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnComOpen.Click
    'device params
    With SerialPort1
        ParityReplace = 6H3B ' replace ";" when parity error occurs
        If cboComPort.Text = "" Then
            MsgBox("There is no Serial Port.")
            btnSend.Enabled = False
            btnClear.Enabled = False
            btnPulse.Enabled = False
            btnMeasure.Enabled = False
        Else
            PortName = cboComPort.Text
            btnClear.Enabled = True
        End If
        If cboBaudRate.Text = "" Then
            MsgBox("There is no Baud Rate.")
        Else
            BaudRate = CInt(cboBaudRate.Text)
        End If
        Parity = IO.Ports.Parity.None
        DataBits = 8
        StopBits = IO.Ports.StopBits.One
        Handshake = IO.Ports.Handshake.None
        RtsEnable = False
        ReceivedBytesThreshold = 1 'threshold: one byte in buffer > event is fired
        NewLine = vbCrLf ' CR must be the last char in frame. This terminates the SerialPort.readLine
        ReadTimeout = 10000
        Encoding = System.Text.Encoding.GetEncoding(1252)
    End With
End Sub
```

Figure 30. Subprocess to open serial port. Source: Original Project.

Although up to now there have been no major bugs, it is in this function that it contains one: it is the button that enables sending the pertinent data, both the measurement test and the short-circuit test.

```
Private Sub button_send_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnSend.Click
    If short_c = True Then
        Thread.Sleep(500)
    End If

    If comOpen And accion = "S" And bit_USB = 0 Then
        SerialPort1.WriteLine("y" & "" & accion & Chr(param))
        btnSend.Enabled = False
        btnClear.Enabled = True
    End If
    If comOpen And accion = "S" And bit_USB = 1 Then
        SerialPort1.WriteLine("y" & "" & accion & Chr(param))
        btnSend.Enabled = False
        btnClear.Enabled = True
        bit_USB = 0
    End If
    If comOpen And accion = "5" And bit_USB = 0 Then
        SerialPort1.WriteLine("y" & "" & accion & Chr(MSB_Pulso) & Chr(LSB_Pulso) & Chr(MSB_Desfase) & Chr(LSB_Desfase))
        btnSend.Enabled = False
        btnClear.Enabled = True
    End If
    If comOpen And accion = "5" And bit_USB = 1 Then
        SerialPort1.WriteLine("y" & "" & accion & Chr(MSB_Pulso) & Chr(LSB_Pulso) & Chr(MSB_Desfase) & Chr(LSB_Desfase))
        btnSend.Enabled = False
        btnClear.Enabled = True
        bit_USB = 0
    End If
    confirm = 0
End Sub
```

Figure 31. Subprocess to send data. Source: Original Project.

You can see that for each of the tests it has the same condition twice, with the difference of having a different value in the variable called *bit_USB*. Throughout the code, this variable has no effect other than making execution inefficient. The function *timer1* is continued, which disables the timer, but since it is not called again, it has no use.

The problems continue when it comes to entering the parameter values of both tests in the variables that save said parameters, that is, the problem arises because once the test has been chosen, the parameter is entered and the file is selected, when confirming the data, if you want to cancel or modify the parameter, you cannot because the code has not been programmed for the cancel option, even if the close tab cross is marked. It cannot be that an application has these errors. Furthermore, it is not correct to use the same variable in which to save the parameter of the two tests, as is the case of the variable *param*, which stores the number of lag measurements and the duration of the test of short circuit. Up to here the first region and continuing with the next one, in charge of storing the function that receives the data coming from the micro. In this case, the *SerialPort1_DataReceived* function is executed each time it detects that new data has arrived through the serial port.

```
ByteNumber = SerialPort1.BytesToRead
ByteToRead = SerialPort1.ReadByte()

Debug.Print(Hex(ByteToRead))

If ByteToRead = 238 Then
    picOpen.BackColor = Color.Green
    comOpen = True
End If

If ByteToRead = 187 And confirm = 0 Then
    picCommandReceived.BackColor = Color.Green
    confirm = 1
End If

If ByteToRead = 204 Then
    MsgBox("No zero crossing detected")
End If
```

Figure 32. Subprocess to received data. Source: Original Project.

```
If accion = "S" Then
    picDataReceived.BackColor = Color.Green
    For counter As Integer = 1 To (param * 3)
        ByteToRead = SerialPort1.ReadByte()
        dato = CInt(ByteToRead)
        If state = 0 And dato = 85 Then
            state = 1
        Else
            If state = 1 And dato <> 0 Then
                MSB = CInt(dato)
                state = 2
            Else
                LSB = CInt(dato)
                desfase = CInt((MSB * 256 + LSB) * 0.025 * 4)
                My.Computer.FileSystem.WriteAllText(fileMeasuresNameWrite & fechaHora & ".txt", CStr(desfase) & vbCrLf, True)
                state = 0
            End If
            If state = 2 And dato = 0 Then
                state = 0 'state cero si LSB es 0
            End If
        End If
    Next counter
End If
```

Figure 33. Subprocess to received data. Source: Original Project.

This was one of the big bugs, because there were absurd conditions that conditioned the operation of acquiring data, as can be seen in Figure 32.

The main problem in this function is about what can be seen in Figure 32, the values 238,187 and 204 are the data in decimal that are transmitted between the interface and the microcontroller in the test of shift measurement for the previously detailed communication, correspond to 0xEE, 0xBB and 0xCC, respectively. Since as long as one of the data is read, it enters the condition, that is the problem, since it may be that during the reception of the measurement values there is some value that just matches these values and it should not enter the condition during that reception but only when it should.

Another serious bug in the sequence can be seen in Figure 33, since it enters in a For loop as long as the test type is that of offset measurement that corresponds to 0x53 and in ASCII code, to "S". Because the communication process by the microcontroller during the reception of the values is continuous, first one value after another, it cannot be in a loop because it is more prone to errors during reception. It can be seen that the sequence is defined by states and conditions. However, the conditions are erroneous since it prevents it from entering as long as the data is 0x00, but it is very likely that some data read is, so it would lead to an error in the transmission.

Certain inefficient and little-used functions were removed for testing in the lab.

```
Private Sub btnFolder_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnFolder.Click
    If FolderBrowserDialog1.ShowDialog() = DialogResult.OK Then
        ' List files in the folder.
        ListFiles(FolderBrowserDialog1.SelectedPath)
    End If
End Sub

1 referencia
Private Sub ListFiles(ByVal folderPath As String)
    filesListBox.Items.Clear()

    'Dim fileNames = My.Computer.FileSystem.GetFiles(
    ' folderPath, FileIO.SearchOption.SearchTopLevelOnly, "*.txt")
    Dim fileNames = My.Computer.FileSystem.GetFiles(folderPath, FileIO.SearchOption.SearchTopLevelOnly, "*.txt")
    For Each fileName As String In fileNames
        filesListBox.Items.Add(fileName)
    Next
End Sub
```

Figure 34. Subprocess to choose directory. Source: Pol Project.

The functions shown in Figure 34 simply create a list of the different files in which the offset measurement data have been saved and display them in a component called ListBox. It certainly is a little useless because is also related to the function in Figure 35, since when selecting a file from this list and pressing the btnDesfase button, it generated the average of the values recorded in the file and displayed it in a TextBox component in a way that was too inefficient. As will be seen in the next sections, this will be corrected more efficiently.

```
Private Sub btnDesfase_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnDesfase.Click
    If filesListBox.SelectedItem Is Nothing Then
        MessageBox.Show("Escoja un archivo de texto.")
        Exit Sub
    End If

    ' Obtain the file path from the list box selection.
    Dim filePath = filesListBox.SelectedItem.ToString

    ' Verify that the file was not removed since the
    ' Browse button was clicked.
    If My.Computer.FileSystem.FileExists(filePath) = False Then
        MessageBox.Show("Archivo no encontrado: " & filePath)
        Exit Sub
    End If
End Sub
```

Figure 35. Button to determinate mean data. Source: Pol Project.

It is possible to see the complete code of the function as well as of the interface in the project annexes.

There have been certain features not mentioned because they did not have a notable bug. However, some were kept in future versions of the code while others were not.

Due to certain problems in sending the measurement data, the automaton in charge of receiving the data, at certain times, did not receive the data in the correct way. It was believed that it could be due to the speed of sending, the application of the interface may have received them slower, consequently, there was an incorrect reception.

4 Tools and devices

4.1 Devices

4.1.1 PIC Microcontroller

Microcontrollers are integrated circuits, in which their operation is based on the execution of orders stored in the program memory. The microcontroller used to control the short-circuit machine is the PIC18F2580 from the Microchip company.



Figure 36. PIC18F2580 Microcontroller. Source: <https://www.microchip.com/en-us/product/PIC18F2580#>

4.1.2 DAQ Omega

They are data acquisition devices, which have several input channels through which they collect data in memory, in order to be able to process the information in a computer. The Omega module is an 8-channel DAQ and allows the user to use it as a voltage input or thermocouple type J, K, T, E, R, S, B, N. It also has 4 cold junction compensation temperature sensors and open thermocouple detection. In this case it is used to collect 8 temperature channels.

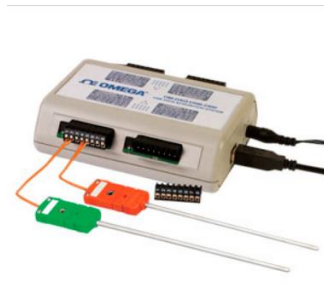


Figure 37. DAQ Omega. Source: <https://es.omega.com/pptst/OM-DAQ-USB-2400.html>

4.1.3 DAQ NI

The DAQ used for the acquisition of voltage and current is the USB-6000 module, it allows us basic functions for data logging, portable measurements and academic use because it has a low price. It is easy to connect sensors and signals with screw connectivity.



Figure 38. DAQ USB 6000. Source: <https://www.ni.com/es-es/support/model.usb-6000.html>

4.2 Programming Languages

4.2.1 Visual Basic.NET

It is a programming language, whose paradigm and/or style in terms of its way of being developed, is called object-oriented programming (OOP), since the code is developed as if it were real-life objects with their functionalities [8]. With a simple example its meaning will be better understood:

Let's say that you want to create an application to introduce information of a user and to be able to represent all this information, the User class is created, which in OOP is called in this way (Class) to a template in which its attributes are defined such as the user's name, surname, age, email, etc., and the methods that the user can perform or functionalities that they have, such as logging in, editing profile, changing password, etc.



Figure 39. Visual Basic.NET logo. Source: https://es.wikipedia.org/wiki/Visual_Basic_.NET

```

0 referencias
1 Public Class Usuario
2
3     Public Nombre As String
4     Public Apellido As String
5     Public edad As Integer
6     Public correo As String
7
8     0 referencias
9     Private Sub Iniciar_sesion()
10    End Sub
11
12    0 referencias
13    Private Sub Editar_perfil()
14    End Sub
15
16    0 referencias
17    Private Sub Cambiar_contraseña()
18    End Sub
19
20 End Class
  
```

Figure 40. Class example. Source: Visual Studio

The object is the representation of the so-called *Class*, which contains each of its attributes and methods, so that many objects of the User class can be created and each of them will contain different information values. In this way, the relationship of the created system has the different objects or classes separated and they can communicate with each other without the need for the code to be designed sequentially, for example, the User class communicates with a Product daemon class, in which user 'x' wants to buy 'y' product.

This language is the next evolution, so to speak, of the Visual Basic language, but with the difference that it is implemented on the .NET framework.

This framework allows us to create desktop applications for Windows, that is, design and program its graphical user interface (GUI). This interface allows better communication between the program and the user, integrating a group of different components that make up the graphic and visual part of the application.

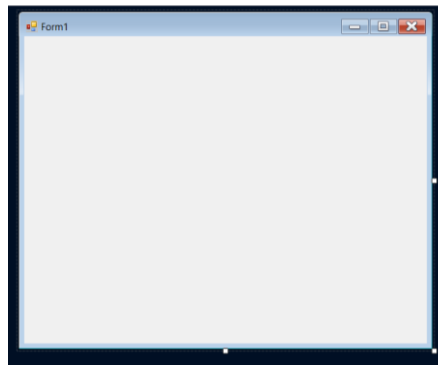


Figure 41. Void Form. Source: Visual Studio

To design these interfaces, a window called form is used, which contains these components. These forms are also objects and we can modify their properties, such as their size, color and font, among others.

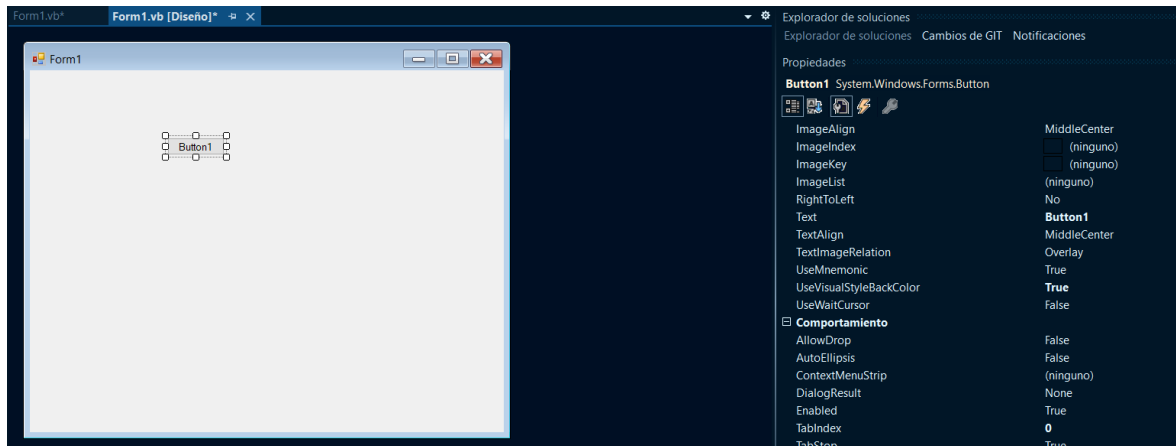


Figure 42. Example Form. Source: Visual Studio

It is important to note that these components, such as buttons, generate events that can be associated with the execution of certain instructions to develop the application.

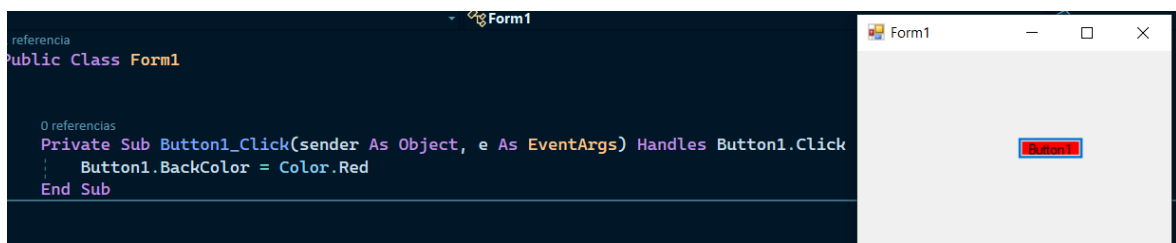


Figure 43. Example execution Form. Source: Visual Studio

4.2.2 C Language

The C language is a programming language developed in the 70s by Dennis Ritchie and evolved from the B programming language, which is usually taught when starting to program, since it is a very versatile language that allows us to develop various functionalities and/or applications, at the same time that it is efficient and uses the structural paradigm, that is, when executing the multiple instructions they are done sequentially, so it works step by step, a fact that facilitates teaching and understanding when you start from scratch. Moreover, that many other languages were based on it for its creation [9].

In the following sections, the specific development of the code in C language for the microcontroller is detailed.

In the case of programming in C, libraries are needed to offer certain functionalities, such as making use of timers, programming in communication types and in registers of a specific microcontroller. Making use of `#include <name of the library>` you get what is described.

```
#include <pl18f2580.h>
#include <timers.h>
#include <delays.h>
#include <usart.h>
#include "ConfigBits.h"
```

Figure 44. Include example initialization. Source: MPLAB IDE

To be able to create variables and give them an initial value, the following code is needed.

```
unsigned char j, LSB_F, LSB_D, param, ti;
unsigned int timer_act, MSB_F, MSB_D, a;
unsigned long paquets_seg, preload_des;

/**Declaració de les comunicacions**/
#define INI_TRAMA 0xAA
#define FER_MESURA 0x53
#define FER_POLS 0x35
#define CONFIRMACIO_RECEPCIO 0xBB
```

Figure 45. Variable example initialization. Source: MPLAB IDE

The `#define` statement is used to give a value to a representative word in the execution of the code.

In C language, the project executes the function called "main" at first and consequently, everything it contains inside.

```
void main(void) // Inici MAIN
{
    j = 0;
    MSB_P = 0;
    LSB_P = 0;
    MSB_D = 0;
}
```

Figure 46. Example main function. Source: MPLAB IDE

There are also IF ELSE conditions, SWITCH and FOR WHILE loops to be able to condition which instructions are executed depending on the given condition and how many executions to repeat the same instructions while the condition is fulfilled.

```
#include <iostream.h>
int main()
{
    int numero;
    cout << "teclea un número: ";
    cin >> numero;
    if(numero % 2 == 0)
        cout << "número par";
    else
        cout << "número impar";
    return 0;
}
```

Figure 47. Example if condition. Source: MPLAB IDE

```
void main() {
    int i;

    for (i=0;i<101;i+=5) {
        cout << i << endl;
    }
}
```

Figure 48. Example for loop. Source: MPLAB IDE

In this case, the message “numero par” will appear on the console as long as the number entered is even. In the case of the loop, it will be executed as many times as the variable “i” is less than 101 and for each execution, it will be increased by 5 and will show the value in the console in each of the executions, it would be type 0,5,10, etc.

4.3 Communications

The microcontroller and the PC that control the short-circuit machines are communicated by means of a serial link (USB-RS232).

The operation of this communication consists of a pair of conductors in which the transmission of information goes in opposite directions since one sends and the other receives in each device. The transmission is simple since it is done sequentially with a bit-by-bit sending and asynchronously. In fact, in an established communication, at first an initial bit is sent that indicates the beginning of its transmission, after sending the corresponding coded frame, a stop bit is sent. Both the start and stop bits serve to prepare the receiving mechanism, so that it is warned that there will be a frame to receive next and that there will be a break before the next frame, respectively.

As the data transmission is delimited in 8 bits and taking into account that both the microcontroller and the interface send variables that occupy 16 bits, these data are sent in two frames of 8 bits, respectively. Therefore, in both codes the division of the frame is defined.

4.4 Software packages

4.4.1 Visual Studio 2022

As for the tool used to develop the software improvement, the Visual Studio 2022 development environment has been used, which provides us with the necessary tools to develop the new code and many more properties that facilitate software development [10].



Figure 49. Visual Studio logo. Source: <https://visualstudio.microsoft.com/es/vs/features/net-development/>

Visual Studio is an integrated development environment or IDE (Integrated Development Environment), developed by Microsoft for various operating systems, in order to be able to carry out various types of applications such as creating graphic designs, web applications, among others.

The given utility of this environment for this Bachelor's Final Project will be to allow us to develop a significant improvement of the graphical user interface, both in its efficiency and in its visual aspect.

4.4.2. Proteus Design Suite

This Bachelor's Final Project is part of a project in which the Proteus Design Suite simulator was used to emulate the phase shift and short circuit measurement tests during the pandemic. Given its usefulness, it was decided to use it for enhancements to VB.NET in a visual environment.

Consequently, the Proteus Design Suite is the software used to carry out communication simulations between the microcontroller and PC application in a virtual environment. The Proteus Design Suite, from the company Labcenter Electronics Ltd, is an electronic design program that has a built-in virtual modeling system that allows us to carry out real-time simulations in a totally virtual environment, since only a PC is required.

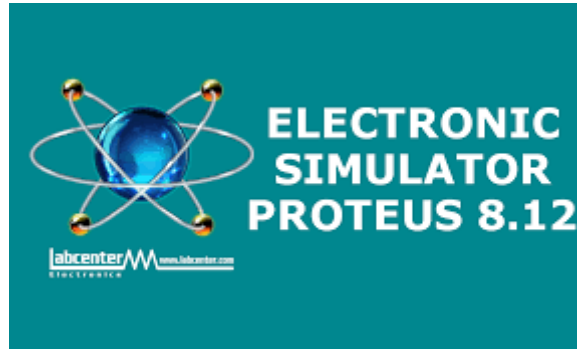


Figure 50. Proteus simulator logo. Source: <https://www.labcenter.com/downloads/>

The fact that it has incorporated a wide library with a variety of electronic components, measurement devices, communication systems and the possibility of simulating the operation of a microcontroller with its respective input-output pins, allows us to use the control code designed on it with the microcontroller that controls the short-circuit test.

4.4.3. Configure Virtual Serial Port Driver

In the same way that we use a simulator to carry out tests, this software is necessary so that it allows us to establish a virtual serial port communication. The Virtual Serial Port Driver software allows you to create as many virtual COM ports as necessary and allows you to simulate with great precision the real operation of a serial port communication. It is specifically designed for creating, debugging software and hardware with serial port [11].

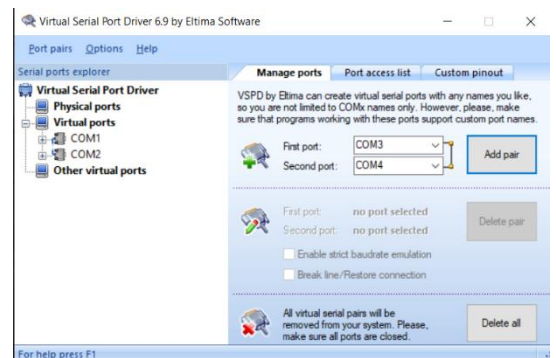


Figure 51. Configure Virtual Serial Port. Source: <https://www.virtual-serial-port.org/>

4.4.4 MPLAB IDE v8.63

MPLAB IDE is a development environment software for PIC microcontrollers from Microchip company. It allows to develop and design the code that will be able to control the behavior of the microcontrollers. It is a perfect environment since the microcontroller used in the short-circuit tests is a PIC. This software allows us to compile the code to be able to check for errors in the written code, as well as being able to enter the code written in the microcontroller and be able to see how it works during its execution [12].

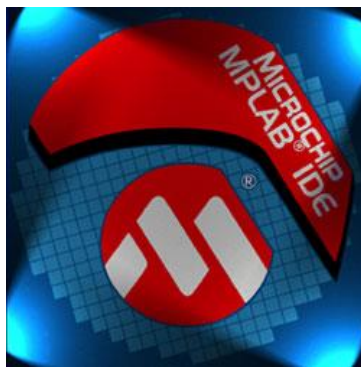


Figure 52. MPLAB IDE logo. Source: <https://www.microchip.com/en-us/tools-resources/archives/mplab-ecosystem>

5 Development

5.1 Methodology

Before starting with the redesign of the graphical interface and programming, it is decided to collect information regarding the project and this new programming language, Visual Basic.

In the first place, it begins with the reading of both the report made by the AMBER-UPC laboratory technician and by the previous TFG student. In this way, it is possible to understand both the objective initially proposed and the design of both parts, that is, the initial design of the interface and the redesign of the code for the control of the microcontroller. The previous code written in C language by José Bailón is not taken into account, because it contained structural errors and was no longer functional for the project.

Once you have read the previous project and have a more global idea of the work previously carried out both in 2016 and 2020 by the technician José Bailón and the student Pol Monreal Mira, respectively, you begin with the reading and analysis of the codes of both the microcontroller and the of the PC.

Starting from the improvement of the C language code of the microcontroller, it is necessary to understand the correct operation. Once the code in C language has been analyzed, we proceed to analyze the operation of the previous version of the graphical interface. Different tests are performed to understand the observed behavior and the different errors that may arise due to different button configuration. Errors produced or bad behavior of the program are noted down, to be clear about what problems to solve and improve.

It begins with the design of a new graphic structure and its button configuration after a first sketch, in which both essays are encapsulated so that they contain their respective buttons and save their parameters in different variables.

As mentioned in previous sections on code structure errors, the methodology consists of modifying and designing a new graphical interface so that the objects behave in the desired way, so that the buttons are conditioned to being pressed. Therefore, in this way there can be no failures. In addition, after each change, an execution of the application is carried out to check if the change made behaves as expected. Many times, it was not done, since there were inconsistencies in the design, so that it did not perform its function.

Once the operation is clear, checks are made on the communication via serial port and it is ensured that the transmission of bits is correct. Therefore, the operation of the code is verified using the Proteus program, in which it can be verified that the short circuit is made by means of an oscilloscope, the duration and the firing angle, etc. In the case of phase shift measurements, it is verified that the phase shift between the voltage and current signal is the same between the one displayed by the oscilloscope and the one stored in the files.

On the other hand, once the verification with the simulator is finished, it is a question of using the new graphical interface in the AMBER laboratory, where the devices, short-circuit machine and transformer, among others, are located. As mentioned above, it is based on a different version than the one used in the laboratory, so the version used in the laboratory has to be adapted. Modifications have to be made in the code so that it is compatible with measurement and data acquisition devices.

As a step after the adaptation, various short-circuit tests are carried out to verify that it works correctly. The changes and checks will be explained in the next sections.

5.2 Solutions

As commented in the current problems of the code, when receiving the data offset measurement test. It was solved by adding 300 milliseconds of delay between the start byte of the measurement frame, the first and second byte of the measurement data. In this way, the automaton for receiving data, which will be detailed later, works correctly.

```

while (BusyUSART());
WriteUSART(INICI_ENVIAMENT); // Enviem byte d'inici de trama
Delay1KTCYx(100);
Delay1KTCYx(100);
Delay1KTCYx(100);
while (BusyUSART());
WriteUSART(timer_FB); // Enviem 8 primers bits
Delay1KTCYx(100);
Delay1KTCYx(100);
Delay1KTCYx(100);
while (BusyUSART());
WriteUSART(timer_LB); //Enviem 8 ultims bits
Delay1KTCYx(100);
Delay1KTCYx(100);
Delay1KTCYx(100);
j++;

```

Figure 53. Modification code. Source: Pol Project.

Bearing in mind that the interface version was different from the one adapted for the simulator, a brief description of this version will be made.

To begin with, the code had three divided regions, the previously mentioned one about communication with the PIC microcontroller and two new ones. DAQ Omega, which is the temperature data acquisition module and the NI USB DAQ, a module that acquires voltage and current data. Both have very similar functions, although with differences in the form of development. They have, for example, functions in charge of making a reading of the device so that it is detected by the program, those in charge of enabling the input channels through which the value read is acquired, both by the thermocouple in the case of temperature and by the voltage-current reading pins. Also, those dedicated to the creation of files and writing in them with the acquired values, so that a graph is generated with these values.

It should be noted that the most important function is generated in the Omega region and communicates the three regions, because, with the push of a button, it generates and creates 3 execution threads, for the acquisition and visualization of data collected during the test of short circuit. The first generated thread configures the NI USB DAQ functions to create the files in which to save the read values, as the second generated thread is from the following DAQ, it also configures the corresponding functions to create files. In this way, the configuration of the parameters to carry out the short-circuit is finally left.

To start with the new design, both of the graphic part and the programming part, it is initially considered how the distribution is desired in terms of button configuration, making a couple of very schematic sketches for it. So, it is decided to make it as simple as possible so that

once it is finished and functional, any user can perform a short-circuit test by clicking on three buttons. In this way, many functions and buttons that were in the previous design are eliminated. Once the graphic structure to be used has been decided, the buttons and other components are placed within their respective blocks, as can be seen in Figure 51.

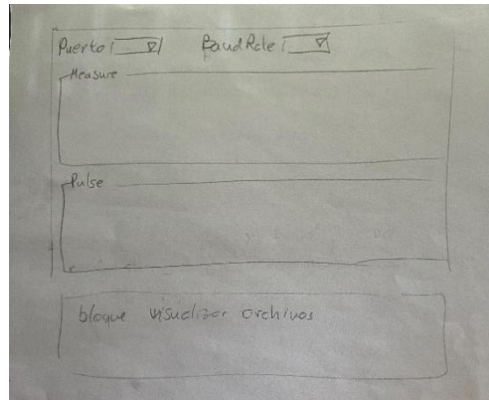


Figure 54. Schematic sketches.

Regarding the new graphic design, as can be seen in Figure 52 and 53, to change the configuration between PIC, Omega and NI, previously you clicked on the windows of a tool called TabControl, which contained the components for the configuration of each test, so it was decided to change this design to one in which it contained buttons. In this way, a format is designed that contains a main form with buttons on one side and in the rest of the space, there are the child forms, which are the ones mentioned above.

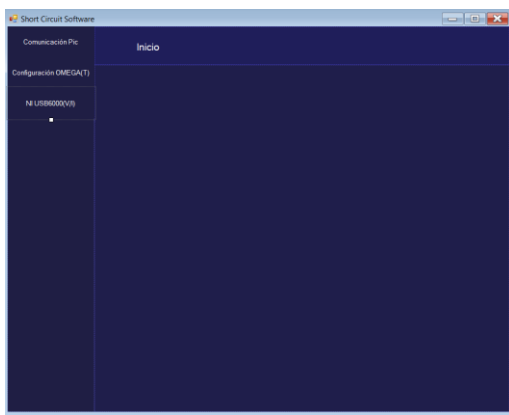


Figure 55. New interface. Source: New Project.

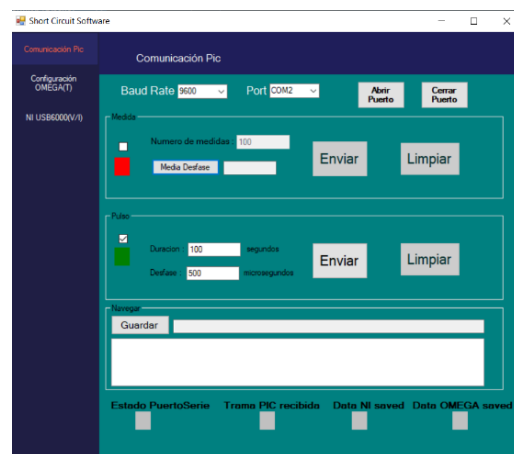


Figure 56. New interface. Source: New Project.

It is decided to leave the components pertinent to the selection of communications speed, the communication port and the lower icons, except for Data Pic Received. It should be noted that the checkboxes were added in a later version, as will be discussed shortly. In Figure 56 you can see the result of what would be the first design initially, but as will be seen later, there are certain changes to make it even simpler.

To increase the ease of use, and as it is not possible to press buttons referring to another test, it is decided to add the checkbox component with its respective PictureBox and to be able to differentiate which test is being carried out at that moment.

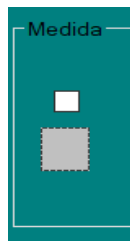


Figure 57. Check and picture box. Source: New Project.

Next, the changes and improvements within the code and the consequences that cause new changes in the graphic part are detailed.

As for the charging function, opening and closing the port, there are only changes in enabling certain buttons that were not in the previous version. As mentioned above, the functions that are executed when a click event occurs in one of the two checkboxes, cause the buttons relevant to the test to be performed to be enabled.

Consequently, by reducing the use of buttons to just enter parameters and send them as was done in the previous version, it is decided to unify both in a button and call different functions to enter the parameters in variables and then send them.

```

Private Sub enviarMedidasbtn_Click(sender As Object, e As EventArgs) Handles enviarMedidasbtn.Click
    If guardarOK = True Then
        If Nmedidasbox.Text <> "" Then
            Nmedidas = Nmedidasbox.Text
            IntroducirDatos()
            EnviarDatos()
        End If
        guardarOK = False
    Else
        MsgBox("Crea o escoja archivo para guardar datos")
    End If
End Sub

0 referencias
Private Sub enviarPulsobtn_Click(sender As Object, e As EventArgs) Handles enviarPulsobtn.Click
    If duracionbox.Text <> "" And pulsebox.Text <> "" Then
        duracionpulso = CInt(duracionbox.Text * 100)
        Mdesfase = CInt(desfasebox.Text)
        IntroducirDatos()
        EnviarDatos()
    End If
End Sub

```

Figure 58. Send data subprocess. Source: New Project.

As shown in Figure 56, both tests have their respective send button functions, so it would only be executed if the fields where the value of the parameter is entered is not empty, since if there is nothing written it will not allow press and nothing will be sent.

The first call to the IntroducerData function and the second SendData function is common to both, since depending on the test to be carried out, it will enter one condition or another, so after saving the values and before they are sent, a message will appear to confirm the data. As for the SubmitData function, it contains the instructions that the submit button contained in the previous version, without the repetition of instructions mentioned above.

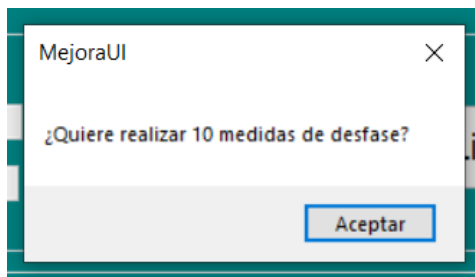


Figure 59. Message box for offset test. Source: New Project.

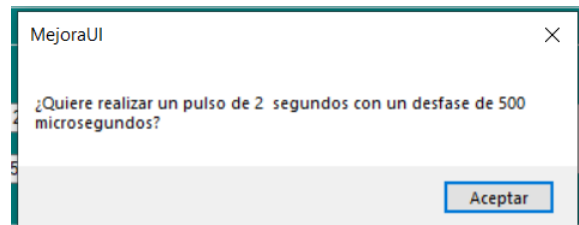


Figure 60. Message box of short-circuit test. Source: New Project.

To solve the error mentioned on previous sections and allow the confirmation message to cancel if the parameters entered are wrong or if you want to not carry out that test, extra lines are added in the IntroducirDatos function that allows us this option and not as in the previous version that it showed the confirmation message without being able to cancel.

```
Private Sub IntroducirDatos(ByVal sender As System.Object, ByVal e As System.EventArgs)

    If tipoEnsayo = "S" Then
        Dim respuesta As Integer
        respuesta = MsgBox("¿Quiere realizar " & Nmedidas & " medidas de desfase?", vbYesNo, "Advertencia")
        If respuesta = vbYes Then
            ArchivoPordefectoMedidas()
            ' CheckBox2.Enabled = False
            ' FormConfigOMEGA.ConfigEnsayo()
            EnviarDatos()
        Else
            If respuesta = vbNo Then
                MsgBox("Envio Cancelado")
            End If
        End If

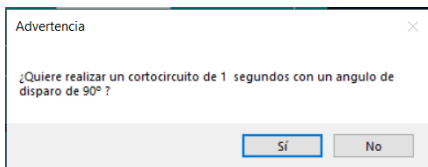
        ' enviarPulsobtn.Enabled = True
    End Sub
```

Figure 61. Subprocess to enter, call create file and send data. Source: New Project.

Another problem is related to the pulse in the signal that allowed activating the SCR in the previous version with a phase angle measured in time, more specifically in microseconds. The units do not add up, since the angle is measured in degrees, but the previous version used microseconds, since that was the magnitude and unit that the microcontroller understands. To modify this and be able to simulate with Proteus, it is necessary to do a conversion from degrees to microseconds. The frequency of the pulse generator used in the simulation is 50 Hz, which is equivalent to a period of 0.02 s. Knowing that the period of any signal is equal to 360°, it is only necessary to apply a rule of 3 and multiply by 10⁶ to go from seconds to microseconds.

$$Desfase(\mu s) = \frac{Angulo\ introducido(^{\circ}) * 0,02\ (s)}{360(^{\circ})} * 10^6$$

In this way, it allows us to enter the firing angle that we want and that the microcontroller understands it.



```
Dim respuesta As Integer
respuesta = MsgBox("¿Quiere realizar " & Nmedidas & " medidas de desfase?", vbYesNo, "Advertencia")
If respuesta = vbYes Then
    EnviarDatos()
End If
```

Figure 63. Message box code. Source: New Project.

Figure 62. New message box. Source: New Project.

As a later modification regarding the clear button in Figure 56, it is decided to eliminate the button and introduce the instructions of the function just after the data has been sent, so that in addition to clearing variables, the fields where the parameters are entered are also initialized.

```
Sub LimpiarDatos()
    iconCommandPic.BackColor = Color.Silver
    iconDataNI.BackColor = Color.Silver
    iconDataOMEGA.BackColor = Color.Silver
    If comOpen Then
        SerialPort1.DiscardInBuffer()
        SerialPort1.DiscardOutBuffer()
        Nmedidasbox.Text = "180"
        duracionbox.Text = "1"
        desfasebox.Text = "90"
        MSB_desfase = 0
        LSB_desfase = 0
        MSB_duracion = 0
        LSB_duracion = 0
    End If
End Sub
```

Figure 64. Clean variables subprocess. Source: New Project.

To continue with the design of the version, it is necessary to modify the function in charge of receiving the data through the serial port, a function called SerialPort1_DataReceived.

As a consequence of the previous bad functionality, a finite state machine (FSM) is designed to change the state every time it receives a different piece of data, based on the Select Case structure.

If the expected data is not received, the FSM returns to the previous state or, in extreme cases, to the initial state.

To begin with, status 1 indicates that correct communication has been established through the serial port, since the byte read corresponds to 0xEE. As it is known, this byte is only received once since the interface is opened and it is reflected in the FSM since the sequence only passes through state 0 once. When passing to state 1, it waits to receive a confirmation, value of 0xBB, from the microcontroller that the frame corresponding to the test parameters has been received correctly. So that in each state the color icon will change to indicate what has been explained.

State 2 only corresponds to the shift measurement test because data reception is required. Therefore, when starting the reception of data sent by the microcontroller, for each data to be sent it begins with a byte, 0x55, and then the two bytes of the offset, so that the first byte of offset corresponds to state 3 and the second to state 4.

In this last state, the collected data is written to a file and a counter variable is increased by 1 so that, if this variable is equal to the number of measurements, it writes the average value at the end of the same file, exits from state 4 and returns to state 1, to wait for the confirmation of a new test, otherwise it returns to state 2 to receive the next data.

```

Select Case state
Case 0
    If dato = &HFE Then
        iconEstadoPuerto.BackColor = Color.Green
        state = 1
        placaConectada = True
        cont = 0
    Else
        state = 0
    End If
Case 1
    If dato = &HBB Then
        iconCommandPic.BackColor = Color.Green

        state = 2
    Else
        state = 1
    End If
Case 2
    If dato = &H55 Then
        state = 3
    Else
        state = 1
    End If
Case 3
    MSB = CInt(dato)
    state = 4

Case 4
    Debug.Print("contador")
    Debug.Print(cont)
    LSB = CInt(dato)
    desfase = CInt((MSB * 256 + LSB) * 0.1) ' 0.025 * 4)
    cont += 1

    archivoDefectoNmedidas.WriteLine(CStr(desfase) & vbCrLf)
    ' archivoDefectoNmedidas.Close()
    suma += desfase

    If cont = Nmedidas Then

        media = suma / Nmedidas
        ' If Primertxt = True Then
        archivoDefectoNmedidas.WriteLine(CStr(media) & vbCrLf)
        archivoDefectoNmedidas.Close()

        ' End If
        globalmedia = media
        cont = 0
        suma = 0
        media = 0
        state = 1
    Else
        state = 2
    End If
End Select

```

Figure 65. Data reception flux. Source: New Project.

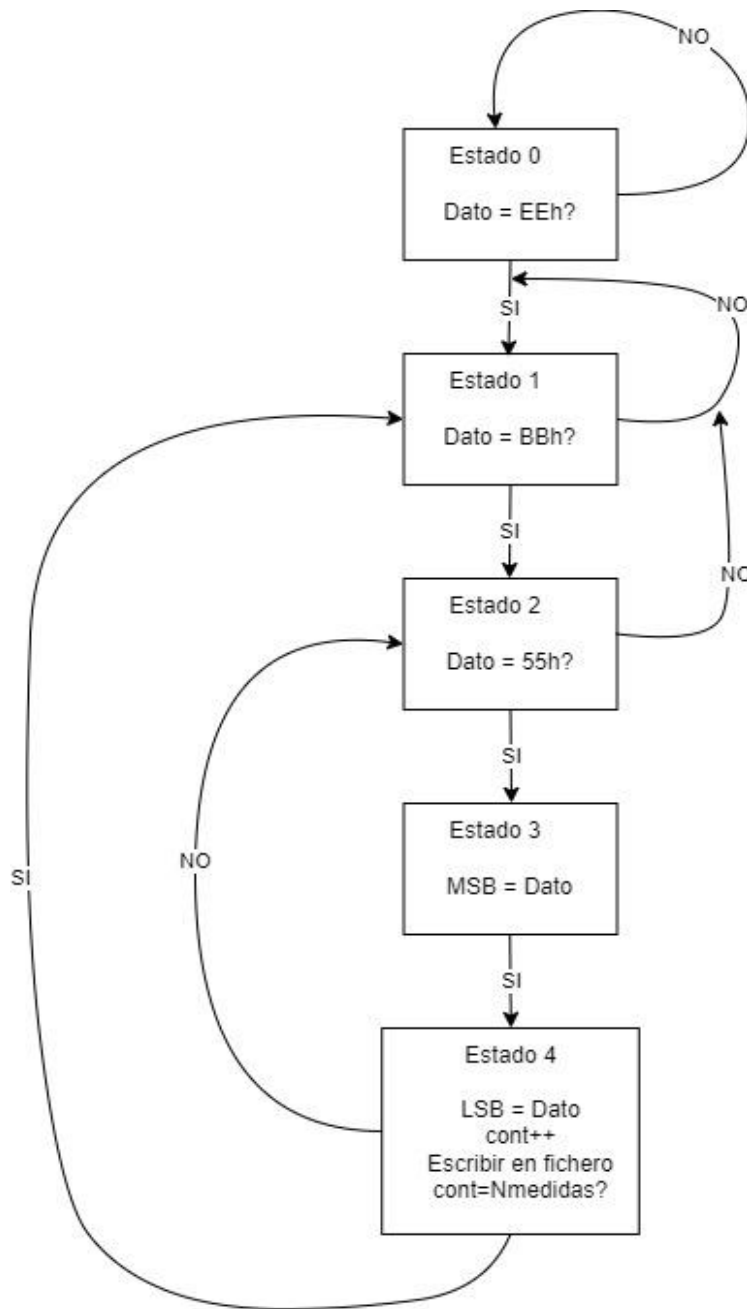


Figure 66. Data reception flux. Source: New Project.

Finishing the improvements in the code, the creation of a file in which to save the measurements has been automated. Since the default test is short-circuit, the fact of changing to the measurement test by selecting the corresponding checkbox automatically creates a file in which the phase shift measurement data is saved. It is also added in the function that refers to the click event, on the save button, which automatically generates a file name for each measurement test that you want to perform. In this way, the name would also change, because a counter is implemented that indicates the number of tests carried out.

```

Private Sub abrirNavegadorbtn_Click(sender As Object, e As EventArgs) Handles abrirNavegadorbtn.Click
    If CheckBox1.Checked = True And Primertxt = False Then
        enviarMedidasbtn.Enabled = True
        guardarOK = True
        'GuardarArchivo.DefaultExt = "*.txt"
        GuardarArchivo.FileName = "Medidas" & i
        GuardarArchivo.Filter = "Text Files|*.txt|All Files|*.*"

        If GuardarArchivo.ShowDialog() = DialogResult.OK Then
            i = i + 1
            archivosMedidaPATHTextBox.Text = GuardarArchivo.FileName
        End If
        iconCommandPic.BackColor = Color.Silver
    Else
        MsgBox("Seleccione ensayo de medida para poder guardar ")
    End If
End Sub

```

Figure 67. Create file subprocess. Source: New Project.

So that each time a new file is saved, the next one has a different name from the previous one, by default the following nomenclature is used: the name "Measures1", the number 1 refers to the first file after the execution of the interface. In case you want to put another name, it would also be possible, although the counter would increase anyway.

As a last important modification, the permissions in the fields in which the parameters are entered are limited. In this way, it is only allowed to enter numeric characters, control characters and as a decimal indicator, the comma “,”.

These changes were during the development of the improvement using the Proteus simulator. While when it was wanted to adapt to the version of the AMBER laboratory, there were changes in the interface and code.

Since there were certain problems related to the version used in the laboratory, such as the project references that allow using the functions and libraries to generate graphs, ZedGraph or the one that allowed configuring daq Omega, it was decided to use again the design in which the button configuration was inside the TabControl component and reuse the design to change the configuration between PIC, Omega or NI, so that there will no longer be child forms as in the simulated version and instead of generating a new project for the laboratory, the original was used, changing parts of the code and modifying the graphic part to the one explained above.

Taking this into account, the Main Form_Load function had certain changes since now it would have to initialize variables and call functions referring to the three mentioned configurations.

```

Main_omega2(dispositivo)
cboDAQ1.Items.Add(dispositivo)

If dispositivo = "N0208262015" Then
  OMEGA_detect = True
ElseIf dispositivo Is "Ningun dispositivo detectado" Then
  MsgBox("El dispositivo de medida de temperatura OMEGA no se ha detectado, cierre la aplicacion y vuelva a conect")
End If
cboDAQ1.Text = dispositivo
cboOnoff1.SelectedItem = "ON"
cboOnoff2.SelectedItem = "ON"
cboOnoff3.SelectedItem = "ON"
  ArchivoPordefecto2()
tbScan.Text = "500"

setchannels_ni()
  
```

Figure 68. New Load subprocess. Source: New Project.

Figure 68 corresponds to a fragment of the Load function, in which the call to the Main_omega2() and setchannels_ni() functions is shown. These functions communicate with the data acquisition devices and confirm if the PC has established communication with these devices and can acquire the log data.

The graphic part of the design had some changes to simplify its use.

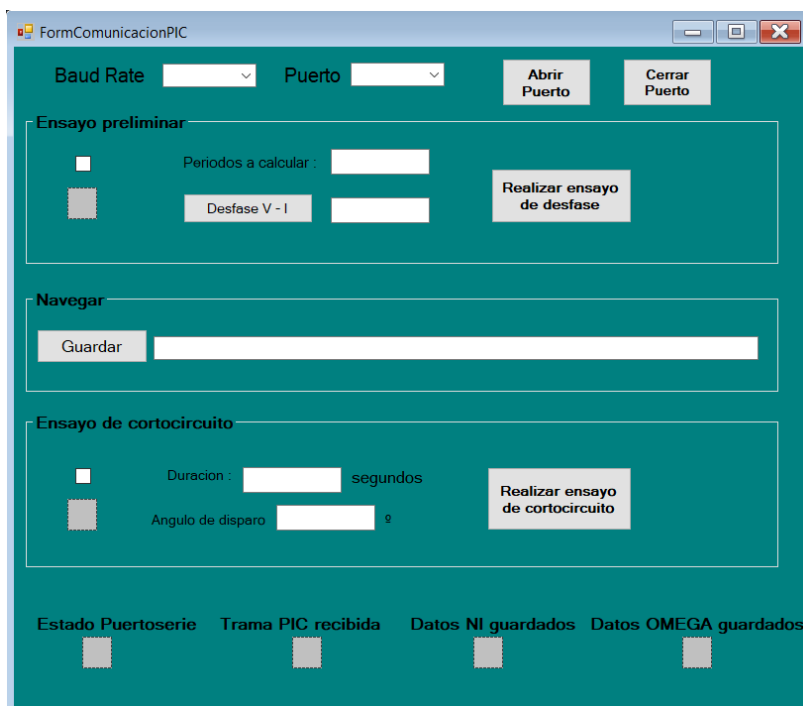


Figure 69. New interface. Source: New Project.

Regarding the development of the version adapted to the laboratory, the most important function of the project will be the following.

```

1 referencia
Public Sub generaHilos(ByVal sender As System.Object, ByVal e As System.EventArgs)
    If tipoEnsayo = "5" Then
        Dim Thread1 As New System.Threading.Thread(AddressOf startButton_Click_1)
        Dim Thread2 As New System.Threading.Thread(AddressOf runOmega)

        zedGraphT.GraphPane.CurveList.Clear()
        zedGraphT.GraphPane.GraphObjList.Clear()

        Thread1.Start()
        Thread2.Start()
        'short_c = True
    End If
  
```

Figure 70. Subprocess to create threads. Source: Pol Project.

As a version used in the laboratory, when carrying out the short-circuit test, the short-circuit button should be able to generate, when carrying out the test itself, the acquisition of data and its graph.

In order to simplify and make this process more efficient, it was decided to automate the generation of the files for both tests. A function capable of creating the files with the time and day as the file name was designed, so that each test would have its generated file and it would be unique.

Figure 68 shows the generation of the file for the offset measurement test, as for the short-circuit test, it is practically identical.

```

1 referencia
Private Function GeneraArchivosMedidas() As String
    Dim fecha_hora As String = DateTime.Now.ToString("dd_MM_yyyy__HH_mm_ss")
    pathmedidas = rutaNmedidas & fecha_hora & ".txt"

    Return pathmedidas
End Function

1 referencia
Private Sub ArchivoPordefectoMedidas()

    GeneraArchivosMedidas()
    archivoDefectoNmedidas = New StreamWriter(pathmedidas)
    ' ' archivoDefecto = New StreamWriter(datapath)
    'archivoDefectoNmedidas.Close()
    fileOmegaPathWriteTextBox.Text = pathmedidas
End Sub
  
```

Figure 71. Variable initialization. Source: Pol Project.

The generaHilos() function creates two threads of execution almost simultaneously. The first generated thread is the one corresponding to the acquisition and generation of a graph

on the data collected on voltage and current during the short-circuit test. The second thread corresponds to the temperature data collected during the test.

Each of these threads has had some superficial modification since it corresponded to the generation of the graphics of the original project and they already worked correctly.

During the start of the first thread, it makes a call to the function `startButton_Click_1`, which corresponds to the writing of the file in which to save the voltage and current data. In parallel to this, this same function makes the call to the `EnviarDatos()` function. In parallel, the thread corresponding to `runOmega()`, configures the channels of the temperature DAQ, generates the graph and saves in the corresponding file.

In order to simplify the use of buttons, the COM port selection buttons, baudrate, open and close port have been suppressed. Since the port is opened automatically once the graphical interface application is opened, this was achieved knowing that the port through which it communicates is COM3. Some code instructions were written that would always connect to that port, set the baudrate to 9600, and once these two parameters were set, the port communicated correctly. Only waiting to connect the microcontroller board since, as previously mentioned, the interface had to be opened first and then the micro connected. So, the "Estado Puertoserie" icon changed color to green.

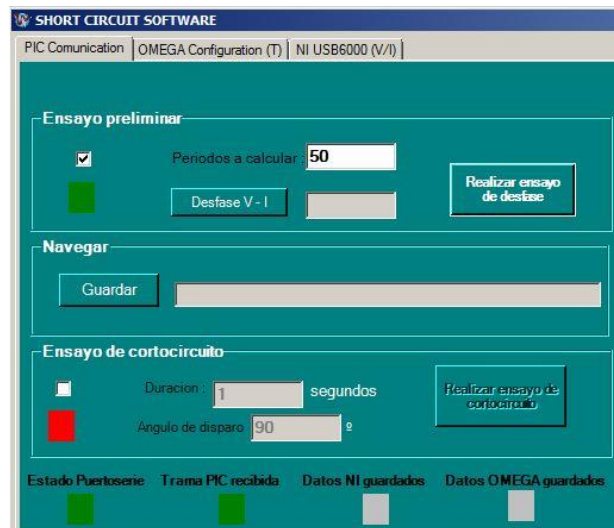


Figure 72. New interface. Source: Visual Studio screenshot.

6 Results

6.1 Test and validation

Before starting with the tests carried out, I will begin to detail the process of carrying out both tests.

So, when selecting any test and clicking on the button to carry out the shift measurement test or short circuit test, the sequence in which it is carried out is as follows:

It begins with the configuration of the autotransformer until the desired voltage is reached, in the case of the tests 158V was used.

We start with the offset measurements test, we select the corresponding checkbox so that it allows us to click on the button to perform that test.

By default, it has the value of calculating 100 measurements, being possible to change the value.

As a consequence, the "Realizar ensayo de desfases" button is pressed and confirm the test.

The machine will make a noise so that it is doing the test during the time it takes to calculate 100 shifts or whatever.

The values will be saved in a file with the recorded data and the calculated average offset value.

As for wanting to carry out the short-circuit test, it is very similar. It begins by changing the checkbox corresponding to this test.

By default, it is established that the short circuit lasts 1 second with an angle of 90°, being also possible to change it.

Once the test parameters have been established, continue pressing the "Realizar ensayo de cortocircuito" button and confirm the test.

It will be possible to listen to the short circuit during the period that duration was entered.

Both voltage-current and temperature data collected by the DAQs will be saved in files, each file for its corresponding DAQ.

For the verification of both tests in the simulator, a test of a short-circuit test of 2 seconds duration and 100 microsecond of angle was carried out, as can be seen, it was before making changes and improvements for the version adapted to the laboratory. However, the operation of sending and receiving data is the same. In this case, instead of pressing the "Realizar ensayo de desfases" button, the "Enviar" button of the Measurement or Pulse block would be pressed and then its corresponding confirmation.

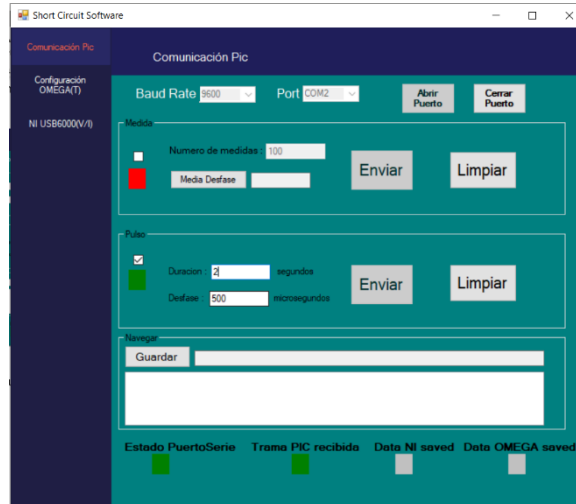


Figure 73. Short-circuit simulated test. Source: Visual Studio screenshot.

The proteus simulator allowed us to verify that the 500 μ s lag is reflected, $372.01 \text{ ms} - 371.50 \text{ ms} = 0.51 \text{ ms} = 500 \mu\text{s}$, let's ignore the 0.01 ms, because it was not possible to place the cursor exactly in the simulator.

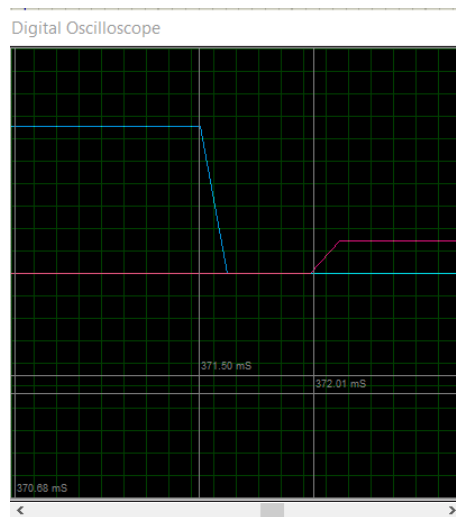


Figure 74. Offset screenshot. Source: Proteus screenshot.

As we can see, the duration of the short circuit was also 2 s.
 $410 \text{ ms} = 0.41 + 1.59 \text{ s} = 2 \text{ s}$.

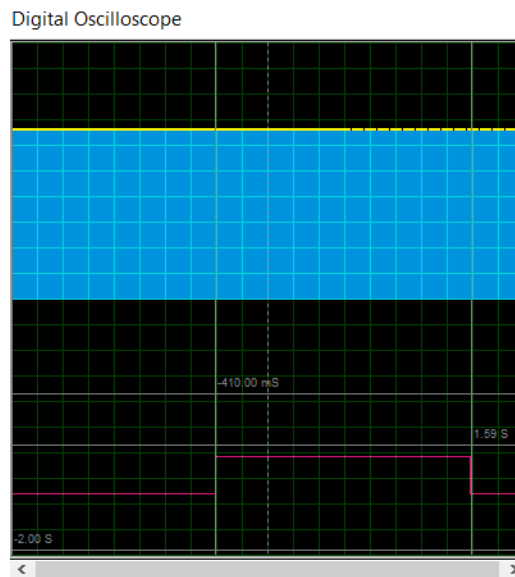


Figure 75. Duration screenshot. Source: Proteus screenshot.

As it can be verified in the following capture, the upper terminal corresponds to the frame sent by the interface to the microcontroller.

- The value 0xFF corresponds to the frame start byte for the microcontroller itself
- The value 0xAA is the frame start byte, but it corresponds to the communication logic between the devices.
- Byte 0x35 corresponds to the short-circuit test as explained above.
- The next two bytes, 0x00 and 0xC8, correspond to the most and least significant byte of the duration in hundredths of a second (200) because the logic programmed for the control of the microcontroller so established.
- Bytes 0x01 and 0xF4 correspond to the most and least significant byte of the phase shift angle in microseconds, 500.
- The last byte 0x0D, corresponds to the final byte of the frame for the microcontroller.

The lower terminal corresponds to the frame sent by the microcontroller.

- 0xEE, corresponds to the byte that established communication.
- 0xBB, corresponds to the byte to confirm that the parameters have been sent correctly to the microcontroller.

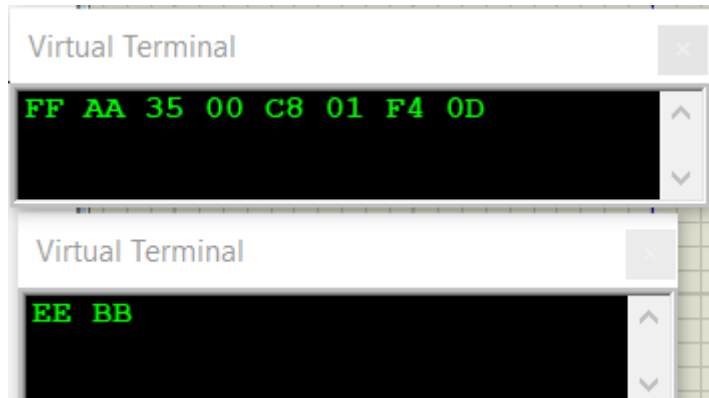


Figure 76. Bit frame. Source: Proteus screenshot.

To verify the operation of the offset measurement test, it was checked with a 5 measurement.

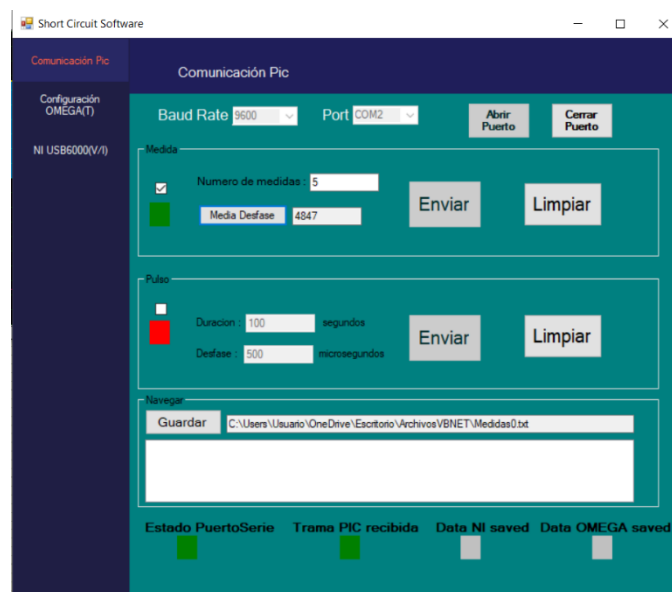


Figure 77. Offset measurement test. Source: Visual Studio screenshot.

- The value 0xFF corresponds to the frame start byte for the microcontroller itself
- The value 0xAA is the frame start byte, but it corresponds to the communication logic between the devices.
- Byte 0x53 corresponds to the offset measurement test as explained above.
- Byte 0x05, corresponds to the number of measurements to perform.

- The last byte 0x0D, corresponds to the final byte of the frame for the microcontroller.

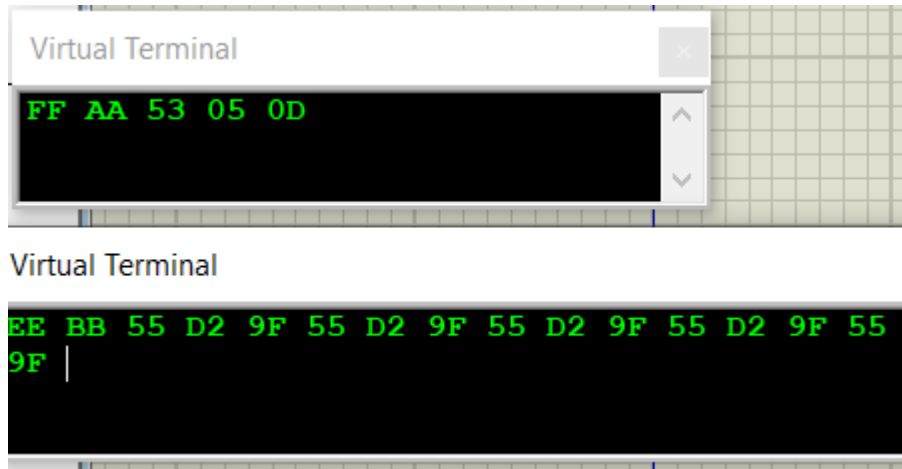


Figure 78. Bit frame. Source: Proteus screenshot.

The data is saved in a file with a format previous of the laboratory.

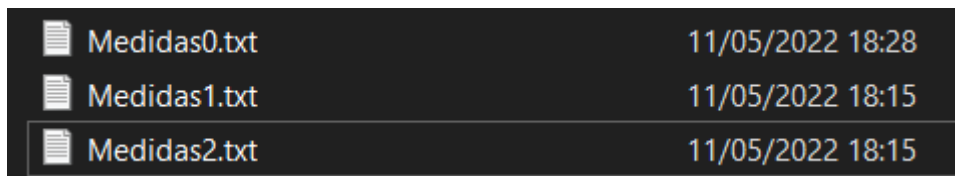



Figure 79. Short-circuit file name. Source: New Project.

The data is saved inside the file with the following composition. We can see that there are 6 data, the first 5 correspond to the measured data and the last one to the average value, which since they all coincide.

Pressing on the “Media Desfase” button, the value appears in its text field.

 Medidas
 Archivo Edi
 5392
 5392
 5392
 5392
 5392
 5392
 5392

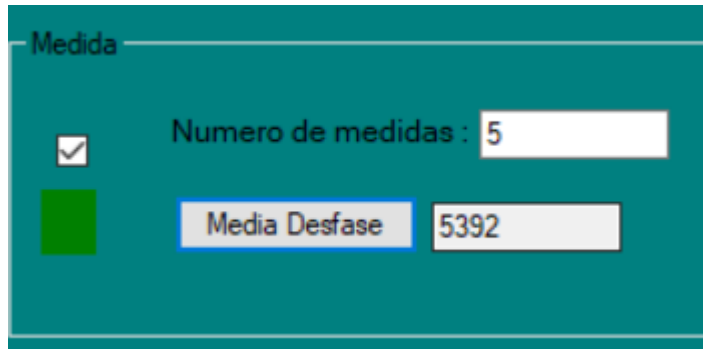



Figure 81. Mean data saved. Source: Visual Studio screenshot.

Figure 80. Data saved. Source: New Project.

When performing the test with more measurements to calculate, it provides us with the different stored values. The following test is with 10 measurements.

 Medidas
 Archivo E
 2665
 5392
 5392
 5392
 5392
 5392
 5392
 5392
 5392
 5392
 5392
 5392
 5392
 5119

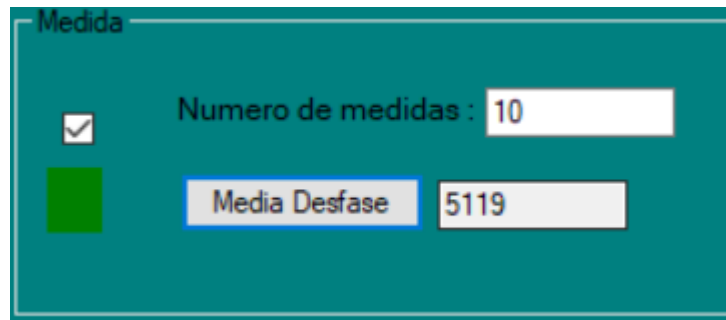


Figure 83. Mean data saved. Source: Visual Studio screenshot.

Figure 82. Data saved. Source: Pol Project.

We continue with tests carried out in the laboratory. Offset measurement test for 10 periods to be calculated.

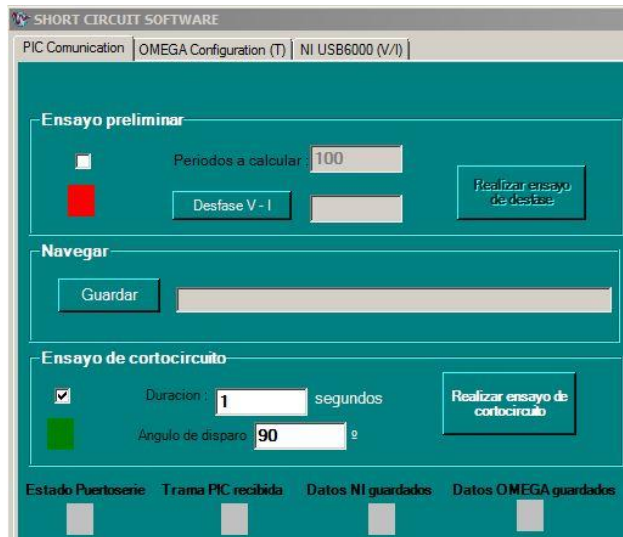


Figure 84. Offset measure real test. Source: Visual Studio screenshot.

1 – It begins by opening the interface application, once it has been fully loaded, the microcontroller is executed and it will be seen how the “Estado Puertoserie” icon changes to green.

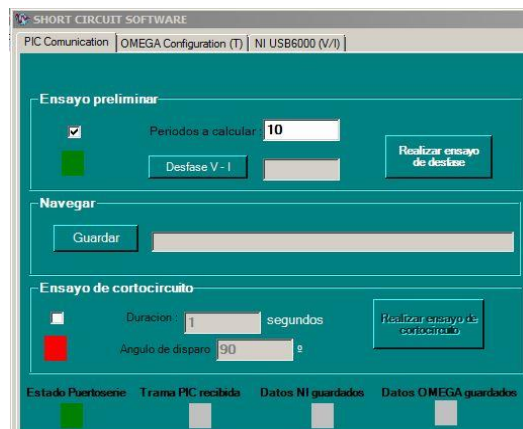


Figure 85. Offset measure real test. Source: Visual Studio screenshot.

- 2 – Enter 10 in the “Periodos a calcular” field, to indicate that there are 10 units of measurement that you want to record.
- 3 – Proceed to press the button “Realizar ensayo de desfase”.
- 4 – A message appears to confirm that these are the parameters with which you want to perform the test.

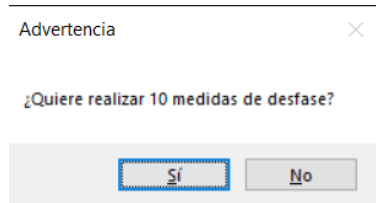


Figure 86. New MessageBox. Source: Visual Studio screenshot.

- 5 – It is confirmed by clicking Yes and then the color of the icon. “Trama PIC recibida” changes.

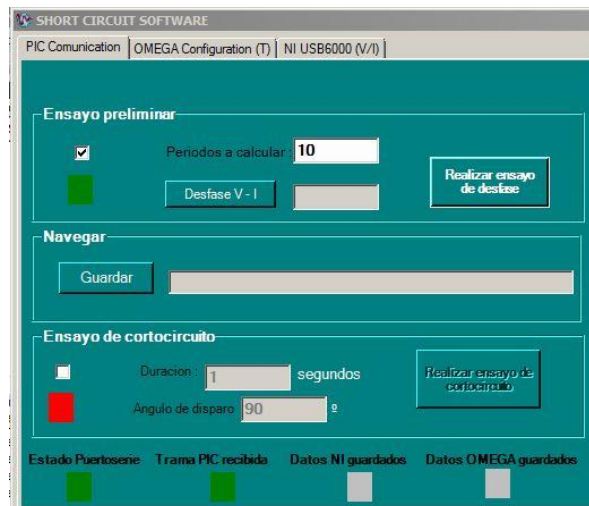


Figure 87. State Trama PIC recibida . Source: Visual Studio screenshot.

- 6 – The data collected during the shift measurement are saved in files with this type of name explained above.

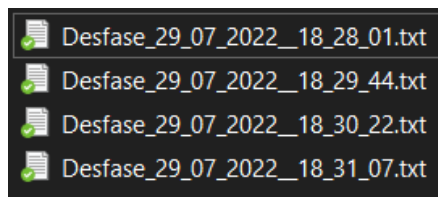


Figure 88. offset test file name. Source: New Project.

7 – Data is saved into these files with this layout, in microseconds. As you can see there are 11 values, the last one represents the arithmetic mean of the 10 registered values.

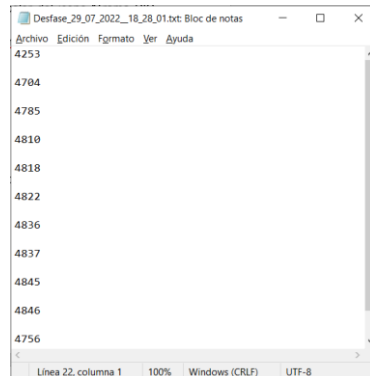


Figure 89. Data saved. Source: New Project.

Short circuit test for 1 second and 90 degrees firing angle.

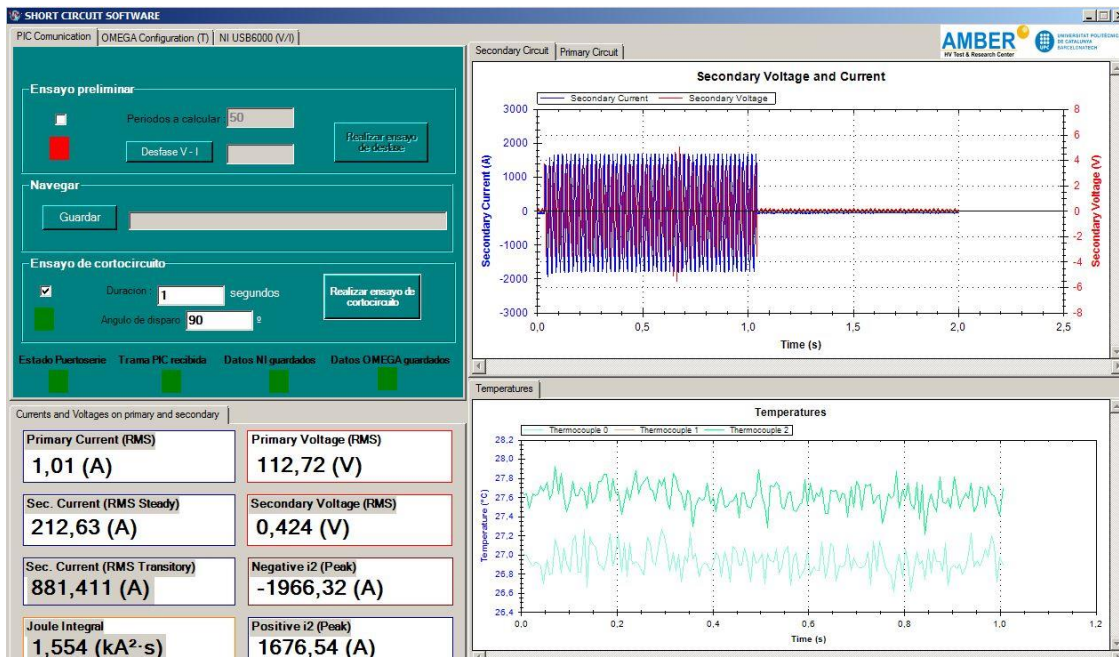


Figure 90. Real Short-circuit test. Source: Visual Studio screenshot.

1 – In the event that the short-circuit test is carried out at first, the first point 1 explained above would have to be followed. In the event of carrying out the short-circuit after a measurement test, it would only be necessary to select the checkbox of the corresponding test.

2 – Enter 1 in the "Duracion" field to indicate that the duration to be recorded is 1 second and 90 in the "Angulo de disparo" field.

3 – Proceed to press the button “Realizar ensayo de cortocircuito”.

4 – A message appears to confirm that these are the parameters with which you want to perform the test.

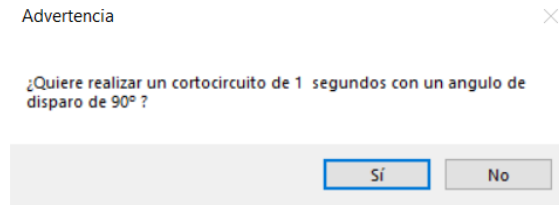


Figure 91. New MessageBox. Source: Visual Studio screenshot.

5 – It is confirmed by clicking Yes and then the color of the “PIC frame received” icon changes, as in the previous case.

6 - As can be seen in Figure 92, the test has generated voltage for 1 second and has a value of 158 V RMS limited by the autotransformer. A slight delayed onset is observed due to the 90°.

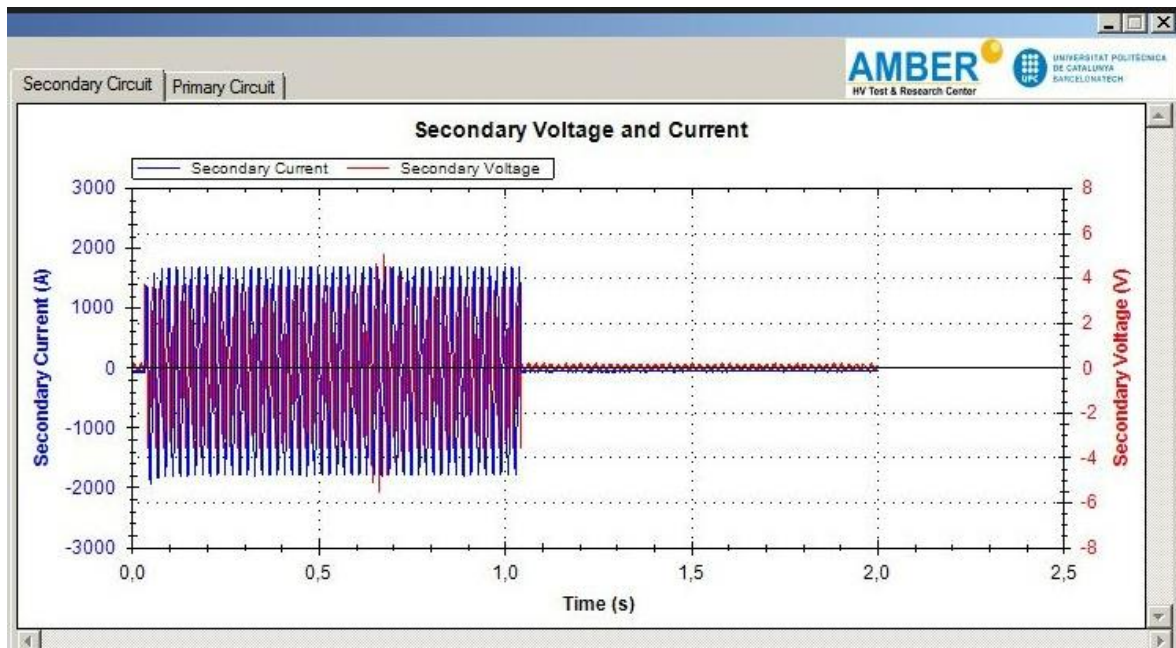


Figure 92. Short-circuit test. Source: Visual Studio screenshot.

7 – The data collected during the test are stored in this way and in this arrangement.

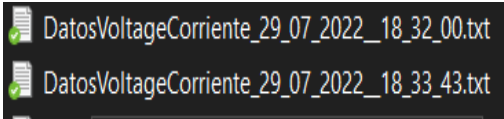


Figure 93. Short-circuit voltage-current file name. Source: New Project.

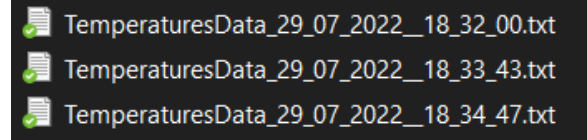


Figure 94. Short-circuit temperature file name. Source: New Project.

8 – Values come inside files with this layout.



Figure 95. Short-circuit voltage-current data. Source: New Project.

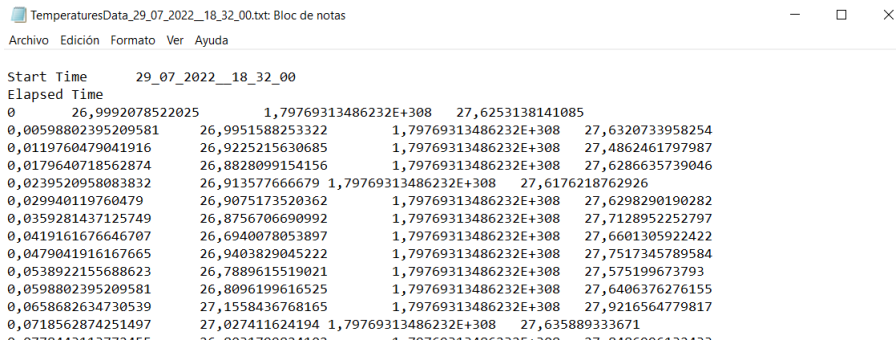


Figure 96. Short-circuit temperature data. Source: New Project.

7 Environmental impact

Due to the short-circuit test, it will not only be useful to collect data from the test itself or to be able to specify the values of the equivalent circuit, but during the test the voltage and current generated will produce heat through the circuit and its respective components, which it will allow to determine what types of materials such as steel, iron, etc. of the different components used for the charge will reach more or less temperature. Also how the temperature of the circuit will be reflected to the variation of the magnitude value of a resistance, for example.

Not only thermal tests can be extrapolated during the short circuit, but also the well-known aging tests to be able to study the behavior over time and the way that affects on the different materials that make up the cables of the circuit or the components.

However, it is possible to carry out many more types of tests on the circuit itself or its components than those mentioned above, such as insulation tests on the transformer itself to verify the behavior of the insulation after several short-circuit tests. And related to this test, the measurement of dielectric rigidity to see the resistance capacity of these insulators.

Each of these extrapolated tests can be used to determine how external agents such as excessive heating, aging, humidity, etc. affect them.

Having the possibility of being able to carry out more than one test with the same short-circuit machine, the environmental impact will be reduced since the use of materials will consequently be considerably less. In turn, the economic cost will also be reflected, more materials can be used efficiently.

As mentioned before in the budget, the energy cost will also be reflected because the use of the laboratory PC for more than one type of test at the same time instead of one for each system dedicated to the mentioned tests, will make even more reduce environmental impact.

8 Conclusions

Once the tests are finished and we see that it has worked correctly, we come to the following conclusions.

Due to possible problems that could be generated during the reception of the data in the measurement test, the designed finite state machine was solid and being able to check the correct operation in the simulated version assured us a high probability in the work environment. Fact that was also validated using the short-circuit machine.

The main objective has been achieved, improving the software used for the graphical interface in a much more efficient and simple way and correctly performing the two types of tests that the short-circuit machine allowed us.

The new design allows us to carry out tests by pressing the minimum number of buttons, just 2 or 3 buttons, so that as many tests as desired can be carried out.

It was possible to automate the creation and saving of data in files in such a way that pressing the button to perform tests is generated with a specific name automatically, without the need to search for the location or name to give the file.

We tried to give it a more visual design using the buttons for the three configurations, but due to complications in the laboratory related to the version of the work environment, the original was used, although it is also very practical.

After all that has been done and problems that occurred during the adaptation in the laboratory, it is worth highlighting the effort invested in this work and being able to verify that it will be useful in the future in future tests that are required for the electric department.

9 Future work

Due to problems regarding the adaptation of the simulated interface design used for the three configurations, it could be proposed to be able to develop, as a future improvement, a new version of the one installed in the AMBER laboratory with the commented design, since the structure organized in buttons is visually more pleasant. , in addition to being much more practical to be able to switch between configuration panels with buttons of a considerable size.

However, the functions developed for the configuration of the two data acquisition modules were functional, they had certain efficiency errors that could be simplified.

Taking these two points into account, a much more robust version could be used in the laboratory.

10 Bibliography

- [1] PIC18F2580 Microcontroller. A: *Microchip* [online]. Microchip Technology Inc., 2022. [Consultation: June 2022]. Available to: <<https://www.microchip.com/en-us/product/PIC18F2580#>>.
- [2] Modul SCR MCC162-16IO1. A: *RS* [online]. *RS Components*, 2022. [Consultation: June 2022]. Available to: <<https://es.rs-online.com/web/p/tiristores/0194041>>.
- [3] Omega DAQ Module. A: *Omega* [online]. OMEGA®, 2022. [Consultation: August 2020]. Available to: <<https://es.omega.com/pptst/OM-DAQ-USB-2400.html>>.
- [4] DAQ USB-6000. A: *National Instruments* [online]. NATIONAL INSTRUMENTS CORP., 2022. [Consultation: August 2022]. Available to: <<https://www.ni.com/es-es/support/model.usb-6000.html>>.
- [5] José Bailón. PROYECTO DE CORTOCIRCUITO. Diseño de sistema de medición de desfase y disparo de cortocircuito, UPC, Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa. Departament d'Enginyeria Elèctrica, 2016 [Consultation: February 2020]. Not available.
- [6] Pol Monreal i Mira. MILLORA D'UN SOFTWARE DE CONTROL D'UNA MÀQUINA PER FER ASSAIGS DE CURT CIRCUIT, Final degree project, UPC, Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa. Departament d'Enginyeria Elèctrica, 2020 [Consultation: February 2020] Available to : <<https://upcommons.upc.edu/handle/2117/331170>>.
- [7] Proteus Software. A: Proteus [online]. Labcenter Electronic, 2022. [Consulta: February 2022]. Available to: <<https://www.labcenter.com/downloads/>>.
- [8] Visual Basic.NET. A: Wikipedia [online]. Wikimedia Foundation, 2022. [Consultation: February 2020]. Available to: <https://es.wikipedia.org/wiki/Visual_Basic_.NET>.
- [9] Language C. A: Wikipedia [online]. Wikimedia Foundation, 2022. [Consultation: February 2022]. Available to: <[https://es.wikipedia.org/wiki/C_\(lenguaje_de_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n))>
- [10] Visual Studio 2022. A: Microsoft [en línia]. Microsoft Corporation, 2022. [Consulta: February 2020]. Available to: <<https://visualstudio.microsoft.com/es/vs/features/net-development/>>.
- [11] Virtual Serial Port Driver. A: Virtual Serial Port [online]. Electronic Team, Inc. 2022. [Consultation: February 2020]. Available to: <<https://www.virtual-serial-port.org/>>.
- [12] MPLAB IDE. A: Microchip [online]. Microchip Technology Inc., 2022. [Consultation: February 2022]. Available to: <<https://www.microchip.com/en-us/tools-resources/archives/mplab-ecosystem>>.