IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 20, 2021, accepted February 5, 2021, date of publication February 8, 2021, date of current version February 17, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3058059*

# FPGA Prototyping Using the STEMlab Board With Application on Frequency Response Analysis of Electric Machinery

**BRUNO R. GAMA[1,2], WILSON C. SANT'ANA[1,2], (Member, IEEE), GERMANO LAMBERT-TORRES[1], (Fellow, IEEE), CAMILA P. SALOMON[2], (Member, IEEE), ERIK L. BONALDI[1], LUIZ E. BORGES-DA-SILVA[2], (Senior Member, IEEE), RAFAEL B. B. CARVALHO[3], AND FABIO M. STEINER[3]**

[1]Instituto Gnarus, Itajuba 37500-052, Brazil
[2]Instituto de Engenharia de Sistemas e Tecnologia da Informação, Universidade Federal de Itajuba, Itajuba 37500-903, Brazil
[3]EDF Norte Fluminense, Macae 27910-970, Brazil

Corresponding author: Bruno R. Gama (brrgama@gmail.com)

This work was supported by the Brazilian Research Agencies CNPq, CAPES, FAPEMIG, and ANEEL Research and Development.

**ABSTRACT** This paper presents a complete procedure on prototyping using the FPGA of the STEMlab board and is intended to serve as a guide for developers, students and researchers interested in speeding up their projects and experiments. Due to the reconfigurability of its internal circuitry, being as simple as a code modification (using hardware description language), FPGA technology allows testing of several controller topologies and/or parameters without the need of any physical change at hardware. This feature allows a much faster development cycle of either commercial products or academic experiments. Besides the reconfigurability of the FPGA, the STEMlab board also offers the advantage of several peripheral already available, which includes, among others, high speed analog-to-digital and digital-to-analog converters, Ethernet communication and a dual-core ARM processor capable of running a Linux operating system. In this paper, a didactic method of use of this board is presented, from getting started to a complete academic and industrial application: detection of early damage on an induction motor using frequency response analysis.

**INDEX TERMS** Electronic engineering education, field programmable gate arrays, system-on-a-chip.

## I. INTRODUCTION

Developers and academics in the engineering field have to face an increasing pressure to either reduce the time-to-market of commercial products [1] or to ethically deal with the so called ''Publish or Perish'' mindset of the academia [2]. In both cases, the expected results are required to be achieved faster and faster. In this sense, any tool that helps the speeding-up of collection of results is of great value.

Field-Programmable Gate Array (FPGA) technology allows hardware synthesis and reconfigurability into a tiny chip device. This characteristic is a tremendous advantage to any student/researcher/developer as an entire board of digital circuitry now can fit into a tiny device and, even better, the entire circuit can be modified as desired by altering some lines on its HDL (Hardware Description Language) code.

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

Also, due to its parallel processing capabilities, FPGAs can shorten the computational time of a given application (FPGAs can be several times faster than a conventional CPU, with the same data precision [3] - although comparisons must not generalize performance without consideration of the technical specifications [4]), resulting in lower control delay and better dynamic performance [5]. Due to this characteristic, FPGAs have been employed to perform co-simulation in Real-Time Simulators. A co-simulation can be defined as the division of a complex mathematical model to be simulated synchronously in different hardwares [6]. Real-time simulators are any machine capable of solving the model equations at the same time-step of the real-world system being simulated [7]. Considering systems with slow dynamics, such as mechanical systems, a regular digital processor could be employed. However, when considering systems with fast dynamics, such as power systems, power electronics, electric machinery and drives and etc., powerful and fast processors

must be employed and here is also a niche where the parallel nature of FPGAs offers many advantages [8].

The literature presents several applications of FPGAs, either as real-time simulators or implementing real-time controllers. However, these papers are focused on describing their application itself and not give any sort of hint on how a beginner student could start his own project. This current paper is in an instructional form, in order to provide the basic concepts of FPGA programming. A bottom-up approach is taken, where the reader is introduced to the FPGA workflow by performing simple tasks such as reading/writing from/to I/O pins and evolves into more complexes tasks such as Ethernet communication and the use of the analog peripherals of the board. It is important to notice that the examples presented here are based on simple digital electronics concepts - as these can be useful for the majority of students/researchers, without expanding too much the scope of the paper. Concerning specific topics, such as signal processing and filtering, textbooks are a good starting point - although, some issues, such as the accumulating roundoff errors can be better solved using the approach proposed in [9].

The introductory concepts are summed up into a practical academic and industrial project, which is the detection of failures in electric machinery through Frequency Response Analysis (FRA). In the FRA technique [10], a high speed digital-to-analog converter (DAC) is employed to inject a signal at a given frequency at the machine windings and two high speed analog-to-digital converters (ADC) measure the voltage response. For each injected frequency, an impedance is calculated based on the two measured voltages - forming an impedance spectrum. The failure detection is achieved by comparison of the spectra obtained at different stages of the machine lifetime. The FRA is a technique similar to Electrochemical Impedance Spectroscopy (EIS), with some differences in the frequency range of the injected signals and the mathematical analysis tools. Literature shows that the EIS technique has already been developed using a low cost FPGA board in [11]. However, that paper (alongside with the literature) was also focused on describing the application itself and did not provided the resources needed for a beginner student to start his own development. The contribution of this current paper is twofold: the first is to provide the necessary means for the students/researchers in order to develop their own projects on an intimidating platform such as FPGAs (especially those with ARM processors integrated); the second is to provide an academic example that makes full use of the resources available at the board.

Section II presents an overview of the low cost STEMlab board, manufactured with a Zynq-7000 device (a Xilinx FPGA with a dual core ARM processor at the same chip) and high speed analog peripherals. The Xilinx workflow is presented in subsection II-A (including a simple example of read/write in I/O pins) and the basics of TCP/IP communication on this board are presented in subsection II-B. Section III presents the basics required to control the high speed digital-to-analog converter available at the board (including an example of analog voltage synthesis controlled from TCP/IP commands). Section IV presents the basics required to control the high speed analog-to-digital converter available at the board (including an example of analog acquisition also controlled from TCP/IP commands). Section V presents a practical application of the previous concepts, performing detection of failures on an induction motor using the FRA technique.

**TABLE 1. STEMLab boards comparison.**

| | STEMLab 125-10 | STEMLab 125-14 |
|---|---|---|
| Processor | Dual core ARM Cortex A9 | Dual core ARM Cortex A9 |
| FPGA | Xilinx Zynq 7010 SOC | Xilinx Zynq 7010 SOC |
| RAM | 256 MB | 512 MB |
| System memory | Micro SD (32 GB) | Micro SD (32 GB) |
| Power consumption | 5 V; 1.5 A | 5 V; 2 A |
| RF input/output channels | 2 | 2 |
| Sample rate | 125 MS/s | 125 MS/s |
| ADC/DAC resolution | 10 bit | 14 bit |
| Input impedance | 1 MΩ; 10 pF | 1 MΩ/; 10 pF |
| Load impedance | 50 Ω | 50 Ω |

## II. OVERVIEW OF THE STEMLAB BOARD AND GETTING STARTED

The two most important players on the programmable logic market are, noticeably, Xilinx and Altera/Intel - both with a large range of devices, from the less resourceful CPLDs (Complex Programmable Logic Devices) to the more powerful FPGAs with integrated ARM cores, capable of running complete Linux-based operating systems. Due to the versatility and high connectivity of the Linux systems, the choice for a development board with an integrated ARM core has been made priority. This would allow the students on the research group to easily change parameters and to remotely control their applications. The next choice has been made on the specific board, among the many available options. Here, the intended applications are the most important criteria: for example, for an application on multilevel power electronics converters, where an elevated number of digital I/O pins are required, one of the best options on the market would be the Snickerdoodle board [12], which provides access to more than 100 pins of the FPGA. On the other hand, regarding the FRA application (to be discussed on Section V), it is required to have a board with at least two high speed ADCs and at least one high speed DAC. Based on these requirements, it has been chosen the STEMLab board [13], which provides two ADCs and two DAC of both 125 MSample/s.

There are two different versions of the STEMLab board. The main difference between them is related to the resolution of the ADC and DAC used in each version. Table 1 compares basic technical specifications of the two boards: one of them has 10 bit resolution converters, while the other one has 14 bit resolution converters. Fig.1 presents a platform overview, with some main components highlighted.

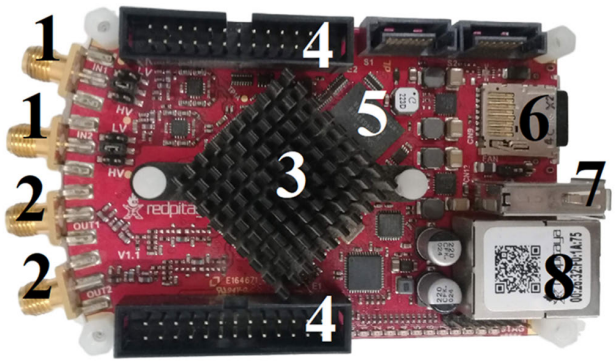


**FIGURE 1.** Board general components. 1: ADC inputs; 2: DAC outputs; 3: SoC Zynq-7000; 4: Extension connectors; 5: RAM; 6: micro SD slot; 7: USB; 8: Ethernet.

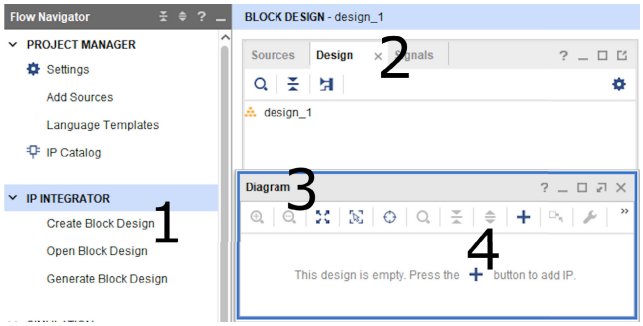


**FIGURE 2.** Creating a Block Design.

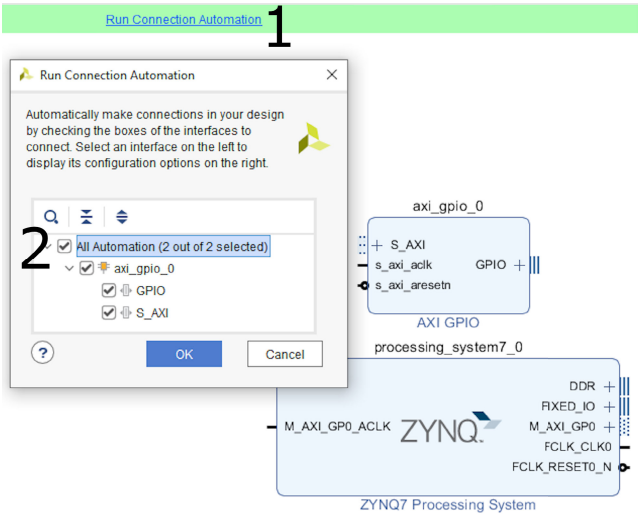


**FIGURE 3.** Adding IP blocks and connecting them.

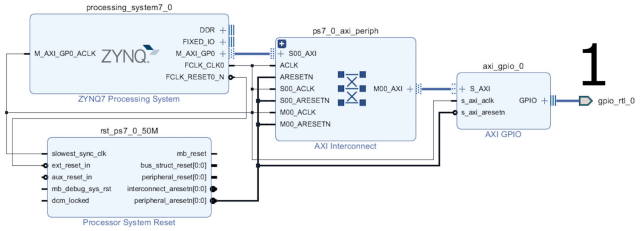


**FIGURE 4.** Regenerating Design.

The core of the models is a Zynq-7000 device, comprising a Xilinx FPGA (hereafter defined as PL - programmable logic) and an integrated dual-core ARM cortex-A9 processor (hereafter defined as PS - processing system) [14].

### A. XILINX WORKFLOW AND GPIO EXAMPLE

Aiming an understanding of the Xilinx workflow, here it is presented an example of GPIO read/write. From this example more complex systems can be developed. It is used the Vivado Design Suit, a powerful tool by Xilinx for synthesis and analysis of hardware description language (HDL) designs.

Starting with a new project, a block design must be created, this is done by accessing the "Create Block Design" button, represented by number 1 in Fig. 2. The created design can be seen in tab "Design", represented by number 2.

The created diagram opens in a new screen, pointed by number 3 in Fig. 2. This is where the Intellectual Property (IP) blocs are added to compose the diagram. Those IPs are pre-implemented circuits that perform determined functions, they are used to speed the development of a circuit. The IPs can be added to the diagram by clicking on the "+" icon numbered by 4 in Fig. 2. Now, the following IPs must be added to the created design: ZYNQ7 Processing System and AXI GPIO. For that, the user can click the "+" icon and search for those IPs. It is necessary to connect the added blocks: for that, there is an option to build automatically all connections, which is performed by clicking on "Run Connection Automation", pointed as number 1 in Fig. 3.

Then, the user can select "All Automation", pointed by number 2 in Fig. 3 and, finally, press "OK". The result of this procedure is presented in Fig. 4. One can see that two more blocks were automatically added to the design to perform the connection between the ZYNQ7 Processing System and the AXI GPIO, also, all wire connections were done. This group of IPs will be from now on considered a single block (with the default name "design_1") which communicates with other blocks using the input and output ports, marked as number 1 in Fig. 4.

With the presented blocks configuration, the communication between ARM and FPGA can be performed.The main block is the ZYNQ7 Processing System, it works as a logic connection between PS and PL structures [15]. The AXI Interconnect block is responsible for the connection between one or more AXI memory-mapped master device to one or more AXI memory-mapped slave device [16]. In the presented example the connection is made between the master ZYNQ7 Processing System and the slave AXI GPIO. The AXI GPIO IP is a block with general purpose input/output interface, it provides access to the internal properties of the device and can control the behavior of external devices [17].


**FIGURE 5.** Editing Offset Address.

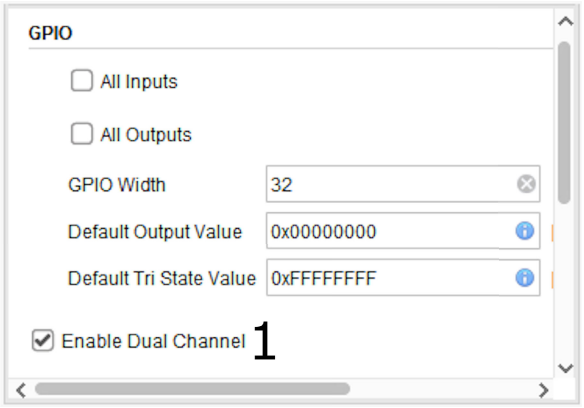


**FIGURE 6.** Enabling second channel to GPIO.


**FIGURE 7.** Creating a new source.

The GPIO IP is used to send commands from Linux to FPGA, and for reading data from FPGA by a program running on Linux.

GPIOs can be accessed by a memory mapping on a Linux running program, for that, it is necessary to set the range of memory address that is available for GPIO access. This is defined at "Address Editor" tab and "Offset Address" field, which are presented respectively by number 1 and 2 in Fig. 5.

It is also possible to add a second channel for the GPIO block, this can be done by double clicking the GPIO IP and selecting "Enable Dual Channel", pointed by number 1 in Fig. 6.

### 1) WRITE TO A LED

A first basic example is presented here. In this proposed system, the user must be able to turn on/off a LED of the board.

To begin, a new design source must be created, this can be achieved by pressing "Add Source" (number 1 in Fig. 7) and,

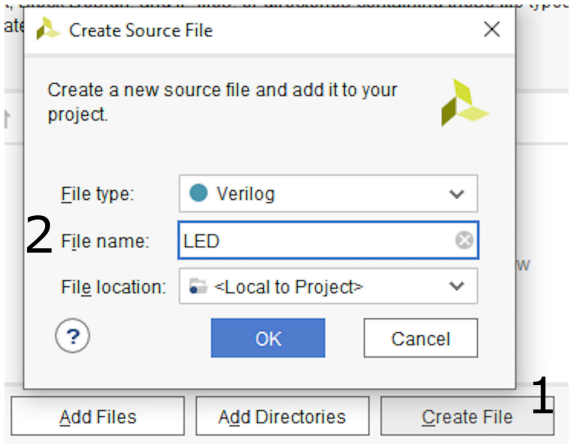


**FIGURE 8.** Selecting file name and type of the new source.

in the new screen, selecting "Add or create a design source" (number 2 in Fig. 7). Select "Next" and a new screen is open, presented by Fig. 8. In this new screen, a new file is created by selecting "Create File" (number 1 in Fig. 8), in the new small screen, it is defined the file type and the file name (number 2 in Fig. 8), as can be seen, it is selected "Verilog" for file type and "LED" for file name. Select "OK" and "Finish".In this created file, the following Verilog code must be written and saved:

```
module LED (control, led);

    input control;
    output led;
    wire [31:0] control;
    assign led = control[0:0];

endmodule
```

This block receives as input a control signal. This signal is supplied by the previous designed GPIO block (design_1). This block outputs a signal that will be used to turn on or off the LED, depending on the value of the received control signal.

A second Verilog source is created, this one must be the top module block. This block is responsible for assembling the final circuit, which makes the connection between the control block (design_1), the LED block and board's LED. This block is named "top" and has the following Verilog code:

```
    output led_o;
    wire [31:0] wire_control;
    wire wire_led;
    design_1 u1(
                .gpio_rtl_0_tri_o(
                wire_control)
                );
    LED u2(
            .control(wire_control),
            .led(wire_led)
            );
    assign led_o = wire_led;

endmodule
```

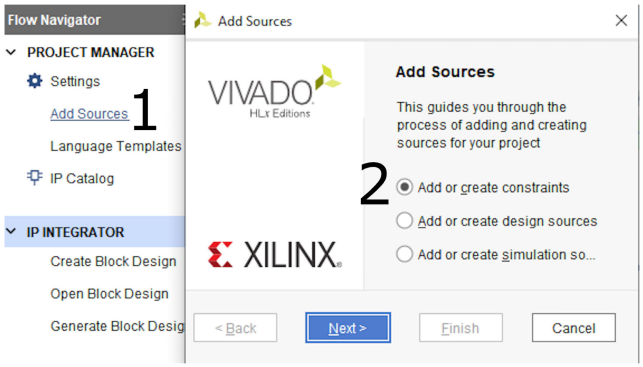


**FIGURE 9.** Creating a new constraints source.

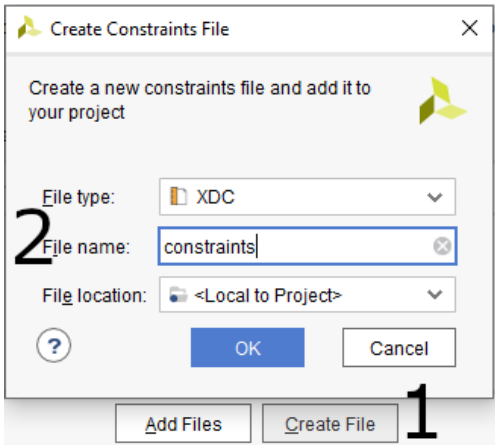


**FIGURE 10.** Selecting file name and type of the constraints source.

In the presented Verilog code, the "u1" is a block of type "design_1", and "u2" block is a block of type "LED". The "led_o" is the output signal which is attached to a FPGA's pin, this pin is connected to one of the board's LED. This attachment is made by a source file called "Constraints". To create a constraints file, one should access "Add Sources" (number 1 in Fig. 9 ), on the new screen select "Add or create constraints" (number 2 in Fig. 9), select "Next". The next screen allows the creating of the new constraints, one should click on "Create File" (number 1 in Fig. 10), select the file type and file name (number 2 in Fig. 10), then select "OK" and "Finish". In this new constraints file, it must be written a code that defines the "led_o" as an output and attaches it to the FPGA's pin F16, which is a pin connected to one of the board's available LED, according to [13]. The _i, _o and _t suffixes define the connections to the iobuffer. The constraints code is presented as follows:

```
### LED

set_property
        IOSTANDARD LVCMOS33
        [get_ports {led_o}]

set_property
        SLEW SLOW
        [get_ports {led_o}]

set_property
        DRIVE 8
        [get_ports {led_o[*]}]

set_property
        PACKAGE_PIN F16
        [get_ports {led_o}]
```

The circuit must now be synthesized, implemented and its bitstream must be generated, these procedures can be accomplish by selecting the left menu buttons "Run Synthesis", "Run Implementation", and "Generate Bitstream", which are respectively pointed in Fig. 11 by numbers 1, 2, and 3.

After all processes are finished, a bitstream file (.bit) will be available in the project's folder: "project"/runs/impl_1. This file is used to configure the FPGA.

Two codes are presented as follows: the first one defines the address and the access to the hardware GPIO by a memory mapping, and the second is the C code used to test the system.

For the next examples, the parameters.h code will be omitted, since its content is the same.

```
//-------------
//parameters.h
//-------------

#define gpio_address 0x42000000
#define gpio_mmap mmap(NULL, sysconf(_SC_PAGESIZE),
                       PROT_READ|PROT_WRITE,
                       MAP_SHARED, fd, gpio_address)
#define led_reg *((uint32_t *)(gpio_mmap))
```

```
//-------------
//WriteToLED.c
//-------------

#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdlib.h>
#include "parameters.h"

int main(int argc, char **argv)
{
  while(1)
  {
   void *cfg;
   int fd, control;

   printf(
          "Press 1 to turn on LED or
           0 to turn off LED: "
          );

   //Read user input
   scanf("%d",&control);

   char *name = "/dev/mem";
   if((fd = open(name, O_RDWR)) < 0)
   {
    perror("open");
    return 1;
   }

   led_reg = control;
   munmap(cfg, sysconf(_SC_PAGESIZE));
  }
 return 0;
}
```

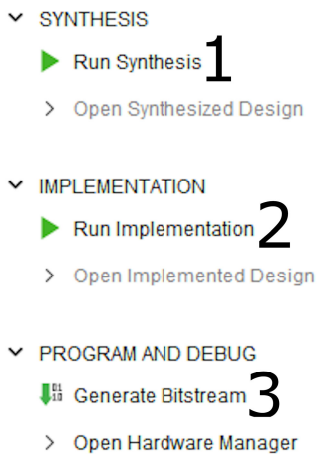


**FIGURE 11.** Synthesis, implementation and generate bitstream.

The presented code reads an input from the user. If it is the number 1, it will turn the LED on. Conversely, if it is 0, it will turn the led off. As a result of this experiment, the reader is able to develop a simple circuit that writes to a FPGA pin - in this case, a pin attached to a LED.

### 2) BLINKING LED

This second example will upgrade the previous one by adding the capability of blinking a led.

For a blinking led example, it is necessary to use a clock signal, therefore the 50 MHz clock signal supplied by the ZYNQ7 Processing System block has been used through the FCLK_CLK0 pin. The new LED module has the following Verilog code:

```
module LED (control, led, clk);

    input control;
    input clk;
    output led;

    wire [31:0] control;
    reg [26:0] counter;

    always @(posedge clk)
    begin
        counter <= counter + 1;
    end

    assign led = control? counter[26] : 1'b0;

endmodule
```

In this new module, a counter register has been used in order to produce the effect of blinking led. Therefore, at each positive edge of the clock signal, the 27 bits counter increases by 1 its counting. The most significant bit of the counter is assigned to the output of the module, turning the led on and off at fixed periods of time. Considering a clock frequency of 50 MHz and a 27 bits counter register, the frequency of the blinking led is 0.37 Hz, therefore, a square wave with a period of 2.7 seconds.

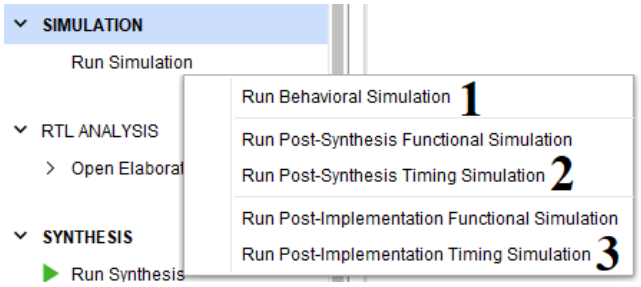


**FIGURE 12.** Timing diagram simulation.

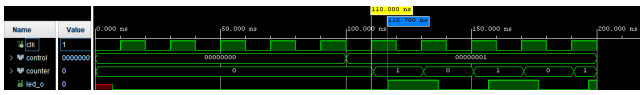

**FIGURE 13.** Post-Synthesis Timing Simulation.

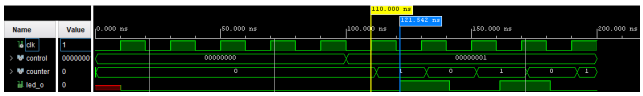

**FIGURE 14.** Post-Implementation Timing Simulation.

After designing, synthesizing and implementing a new circuit, it is a good practice to analyze its timing diagrams. Those diagrams provide information about the signals' delays. This procedure is quite simple with Vivado Software, as one should select one of the tree available simulation options: Behavioral Simulation (number 1 in Fig. 12), Post-Synthesis Timing Simulation (number 2 in Fig. 12), or Post-Implementation Timing Simulation (number 3 in Fig. 12).

Fig. 13 and Fig. 14 present, respectively, the Post-Synthesis Timing Simulation and the Post-Implementation Timing Simulation. In order to ease the visualization of the clock signal (tag clk) and the output signal (tag led_o ), it has been necessary to reduce the size of the counter from 27 bits to 1 bit, resulting in a 25 MHz frequency for the led_o signal.

The total simulation time is 200 ns, from 0 ns to 100 ns, the control input signal (tag control) is set to 0, generating a logical low output. From time 100 ns to 200 ns, the control signal is set to logical high, therefore the led should start blinking (this can be seen by analyzing the led_o signal). By the post-synthesis timing simulation, it can be seen that there is a delay of 16.75 ns between the clock positive edge (time 110.00 ns in Fig. 13), and the led_o positive edge (time 116.78 ns in Fig. 13); while, for the post-implementation timing simulation, there is a delay of 21.542 ns between those two signals. Based on those timing analysis, the designer is provided with detailed information of how the circuit will perform, validating it and identifying possible errors and bottlenecks.

### 3) READ FROM A PIN

This third example will work with the reading operation. This is done by a FPGA circuit that is constantly reading the state of a digital pin, and a software, build in C language, that

interacts with the user and prints, on screen, the pin's digital value.

To begin, a new project is created. Second, the same previously presented control block must be created ("design_1"), this block will be the interface between the C program and the FPGA. Third, the following constraints file must be created:

```
### Digital IO
set_property
        IOSTANDARD LVCMOS33
        [get_ports DIO_0]

set_property
        SLEW FAST
        [get_ports DIO_0]

set_property
        PACKAGE_PIN G18
        [get_ports DIO_0]
```

This code defines the "DIO_0" as an input and attaches it to the FPGA's pin G18, which is a pin connected to one of the board's available digital I/O, according to [13].

The Verilog code used to build the block responsible for receiving the digital signal as input and transferring it to the control block is presented bellow:

```
module ReadPin( data_i, data_o);

       input data_i;
       output data_o;
       wire data_i;
       assign data_o = data_i;

endmodule
```

The top module block is presented below. This block connects the digital input pin (DIO_0), the read block (ReadPin), and the control block (design_1). To clarify the connections made by the top module, Fig.15 presents the RTL schematic generated with the Vivado software. With this schematic, one can see the data path, which is received from the digital input pin and transferred through the read block up to the control block.

```
module top(DIO_0);

    input DIO_0;
    wire [31:0] wire_control;
    wire wire_pin;

    design_1 u1(
                .gpio_rtl_0_tri_i(
                wire_control)
                );

    ReadPin u2(
           .data_i(wire_pin),
           .data_o(wire_control)
           );

    assign wire_pin = DIO_0;

endmodule
```

Finally, the program, developed in C language, prints on user's screen the value of the digital pin each time the enter key is pressed. The C code is presented below:

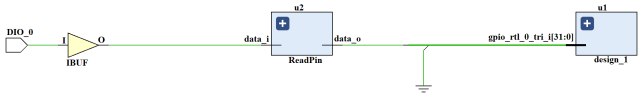


**FIGURE 15.** Read from a pin - RTL schematic.

```
//-------------
//ReadPin.c
//-------------

#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdlib.h>
#include "parameters.h"

int main(int argc, char **argv)
{
 while(1)
 {
  void *cfg;
  int fd;
  int c;
  int control;

  printf("Press enter to read
                  the digital pin: ");
  c = getchar();

  char *name = "/dev/mem";
  if((fd = open(name, O_RDWR)) < 0)
  {
   perror("open");
   return 1;
  }

  control = pin_reg;
  printf("Pin value: %d\n", control);

  munmap(cfg, sysconf(_SC_PAGESIZE));
 }
 return 0;
}
```

### B. TCP/IP COMMUNICATION

One of the greatest advantages of using the SoC (System On Chip) Zynq-7000 device is the possibility to integrate all the flexibility and reconfigurability provided by a FPGA with the software programmability and connectivity provided by a dual core ARM processor running Linux. This allows the students/researchers to remotely control their applications.

In order to improve the previously presented examples, the system now can communicate in a network using a client-server model. Therefore, the server, running in the hardware, must be able to receive, interpret, and execute commands sent by the client (which is running in a remote PC). The client-server communication uses TCP/IP based sockets. A simplified diagram of this model is presented in Fig. 16 [18].

In the presented model, the stages to accomplish a TCP communication are:

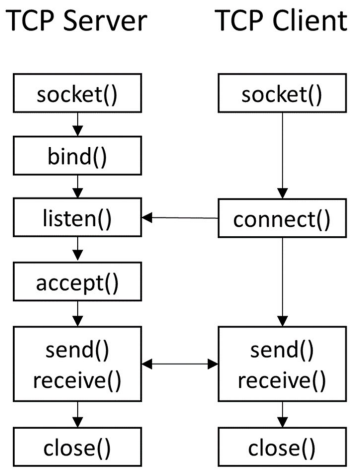


**FIGURE 16.** Simplified diagram of TCP-IP based socket communication. Adapted from [18].

- Socket(): Creation of an instance of the socket abstraction.
- Bind(): Associate the server with an address and port.
- Listen(): Listen for connections from clients.
- Connect(): Try a connection with a server (for the client).
- Accept(): Accept a client connection.
- Send/Receive(): Exchange data.
- Close(): Close connection.

Two new applications are presented. In the first one, a remote client is able to write to one of the hardware's led. The second application shows a read operation, where a remote client is able to read the state of one of the hardware's pin. In both cases, a server software must be build and executed at the ARM processor, while a client software must be build and executed in a remote machine.

### 1) REMOTELY WRITE TO A LED

The FPGA hardware circuit to remotely write to a LED is exactly the same circuit as the presented in Sec. II-A1 - therefore, no changes are needed. To develop the server software, it has been used the C language. The steps presented in Fig. 16 were followed. A base code has been developed and it is presented in sequence.

To develop the client software, it has been used the Python language, although the user could use Matlab, Visual Basic or any other high level language. The steps presented in Fig. 16 have been followed and a base code has been developed and it is presented in sequence.

### 2) REMOTELY READ FROM A PIN

The FPGA hardware circuit to remotely read from a pin is exactly the same circuit as the presented in Sec. II-A1 - therefore, no changes are needed. To develop the server software, it has been used the C language. The steps presented in Fig. 16 have been followed. A base code has been developed and it is presented in sequence.

```
//-------------
//RemotelyWriteToLED.c
//-------------

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include "parameters.h"

//Connection port
#define PORT_SERVER 3030

// Write to a LED function
int writeToLED(int newsockfd,
               volatile void *address);

int main(int argc, char **argv)
{
 while(1)
 {
  void *cfg;
  int fd;

  //Variable to receive command
  //message from client
  char buffer_word[5];
  buffer_word[4] = '\0';

  char *name = "/dev/mem";
  fd = open(name, O_RDWR);


  //Server Configuration
  int sockfd, newsockfd, portno;
  socklen_t clilen;
  struct sockaddr_in serv_addr, cli_addr;

  //socket()
  sockfd = socket(AF_INET,
                  SOCK_STREAM, 0);

  setsockopt(sockfd, SOL_SOCKET,
             SO_REUSEADDR, &(int){1},
                 sizeof(int));

  bzero((char *) &serv_addr,
                 sizeof(serv_addr));

  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons(PORT_SERVER);

  //bind()
  bind(sockfd, (struct sockaddr *)&
  serv_addr, sizeof(serv_addr));

  //listen()
  listen(sockfd,5);

  printf("Waiting for connection...\n");
  clilen = sizeof(cli_addr);

  //Accepts client connection
  newsockfd =
          accept(sockfd,
          (struct sockaddr *) &cli_addr,
          &clilen);
```

```
  bzero(buffer_word,4);

  //Receives client's command
  recv(newsockfd,buffer_word,4,0);

  //Function: write to a LED
  if (
      strncmp(buffer_word,
      "WLED", 4) == 0
     )
  {
   int e = writeToLED(newsockfd);
  }

  //Close connection
  close(newsockfd);
  close(sockfd);
 }
 return 0;
}

int writeToLED(int newsockfd)
{
 //Variable to receive control signal
 char control_char[1];

 recv(newsockfd,
 &control_char,
 sizeof(control_char), 0);

 int control_int = atoi(control_char);

 //Write to LED
 led_reg = control_int;

 return 0;
}
```

To develop the client software, it has been used the Python language. The steps presented in Fig. 16 have been followed. A base code has been developed and it is presented in sequence.

```
import socket

 #socket()
 s = socket.socket(socket.AF_INET,
                   socket.SOCK_STREAM)

 #Connect to IP and Port of the server
 s.connect(('192.168.0.35',3030))

 print("Connected to server")

 msg = "WLED"
 #Send command message to server
 s.send(bytes(msg,"utf-8"))
 modeLed = input("Enter with 1 to turn"+
                 "on LED or with 0 to"+
                 "turn off LED:\n")
 msg = modeLed
 #Send control signal to turn on/off LED
 s.send(bytes(msg,"utf-8"))
```

## III. HIGH SPEED DIGITAL-TO-ANALOG CONVERTER

To build a signal generating system, some concepts must be understood. Therefore, first, there must be an explanation about the system clock and the high speed DAC of the board. After presenting the main concepts, a signal generation system which is remotely controlled is proposed, designed and implemented, and the results obtained with the system are presented.

```
//-------------
//RemotelyReadPin.c
//-------------

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include "parameters.h"

//Connection port
#define PORT_SERVER 3030

// Read digital pin function
int readDigPin(int newsockfd,
               volatile void *address);

int main(int argc, char **argv)
{
 while(1)
 {
  void *cfg;
  int fd;

  char buffer_word[5];
  buffer_word[4] = '\0';

  char *name = "/dev/mem";
  fd = open(name, O_RDWR);

  //Server Configuration
  int sockfd, newsockfd, portno;
  socklen_t clilen;
  struct sockaddr_in serv_addr, cli_addr;

  //socket()
  sockfd = socket(AF_INET,
                  SOCK_STREAM, 0);

  setsockopt(sockfd, SOL_SOCKET,
             SO_REUSEADDR, &(int){1},
                 sizeof(int));

  bzero((char *) &serv_addr,
                 sizeof(serv_addr));

  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons(PORT_SERVER);

  //bind()
  bind(sockfd, (struct sockaddr *)&
  serv_addr, sizeof(serv_addr));

  //listen()
  listen(sockfd,5);
  printf("Waiting for connection...\n");
  clilen = sizeof(cli_addr);

  //Accepts client connection
  newsockfd =
          accept(sockfd,
          (struct sockaddr *) &cli_addr,
          &clilen);

  bzero(buffer_word,4);

  //Receives client's command
  recv(newsockfd,buffer_word,4,0);
```

### A. SYSTEM CLOCK

For the proper work of the system, it is necessary a global clock that will be used for the whole system. According with the board circuits presented in [13], a differential clock of 125MHz is supplied to the FPGA.

```
  //Function: read digital pin
  if (strncmp(buffer_word,"RPIN",4)==0)
  {
   int e = readDigPin(newsockfd);
  }

  //Close connection
  close(newsockfd);
  close(sockfd);
 }
 return 0;
}

int readDigPin(int newsockfd)
{
 //Variable to receive digital pin state
 int pinValue_int;

 //Read from the pin
 pinValue_int = pin_reg;

 char pinValue_char[1];
 sprintf(pinValue_char,
         "%d",
         pinValue_int);

 //Send to client the pin value
 send(newsockfd,
&pinValue_char,
 sizeof(pinValue_char), 0);

 return 0;
}
```

```
import socket

 #socket()
 s = socket.socket(socket.AF_INET,
                   socket.SOCK_STREAM)

 #Connect to IP and Port of the server
 s.connect(('192.168.0.35',3030))

 print("Connected to server")

 msg = "RPIN"
 #Send command message to server
 s.send(bytes(msg,"utf-8"))
 #Receive data from server
 msg = s.recv(1024)
 print("Pin state is: "
      + msg.decode("utf-8"))
```

### B. DIGITAL TO ANALOG CONVERTER

The DAC available in STEMLab board is the DAC1401D125. It is a dual port, 2-channels Complementary Metal Oxide Semiconductor (CMOS) DAC. It supports an update rate of up to 125 Msps (therefore a high speed device) and has a resolution of 14 bit. Besides the high dynamic performance, it can operates with low power dissipation. It has two operation mode, Dual Port Mode and Interleaved Mode. The operation mode is selected by MODE pin on DAC, if MODE = 0, it works on Interleaved Mode, if MODE = 1, it works on Dual Port Mode [19]. According with the board circuits presented in [13], in the STEMlab board, its DAC1401D125 has its MODE pin connected to the ground - meaning that the STEMlab board is hard-wired to use interleaved mode only.

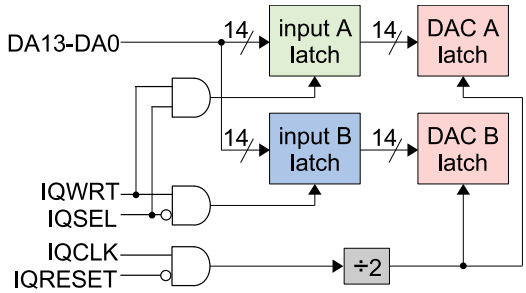


**FIGURE 17.** Interleaved mode circuit. Based on [19].

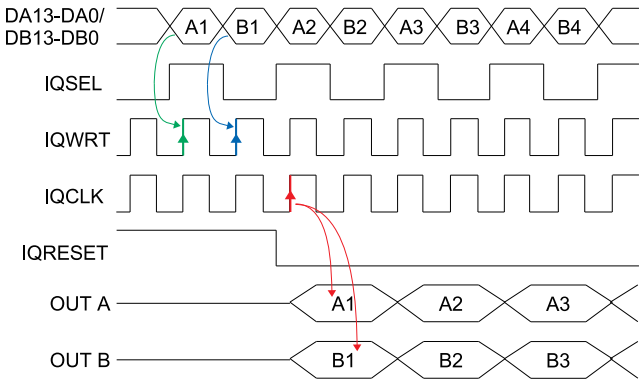


**FIGURE 18.** Interleaved mode signals. Based on [19].

Interleaved mode circuit is presented in Fig. 17. In this mode, both DACs use the same data input channel. On rising edge of IQWRT, input signals (DA13-DA0) are directed to Input Latch A or to Input Latch B. The selection of the direction of the data is made by the IQSEL signal. If IQSEL = 0, input data goes to Input Latch A, if IQSEL = 1, input data goes to Input Latch B. To operate with an update rate of 125 Msps, the system clock (IQCLK) must operate at 250 MHz. The IQCLK signal is divided by 2 to obtain the 125 Msps update rate at DAC Latch A output and DAC Latch B output. When IQRESET is enable, IQCLK is disable.

Fig. 18 presents a timing chart for the interleaved mode. When IQRESET is low, the outputs are valid. One can see the green arrow indicating a rising edge on IQWRT while IQSEL is high - on this condition, the digital data of channel A is saved by Input Latch A. Also, the blue arrow indicates a rising edge on IQWRT while IQSEL is low - on this condition, the digital data of channel B is saved by Input Latch B. The red arrows indicates the subsequent rising edge on IQCLK - on this condition, the analog outputs of both channels are update through the output latches.

### C. SIGNAL GENERATION SYSTEM

Fig. 19 presents an application example system for two channel signal generation. In the presented layout, the Clock block is a Xilinx IP Core, the Configuration block and FPGA/DAC Interface block are custom made blocks by the authors, and

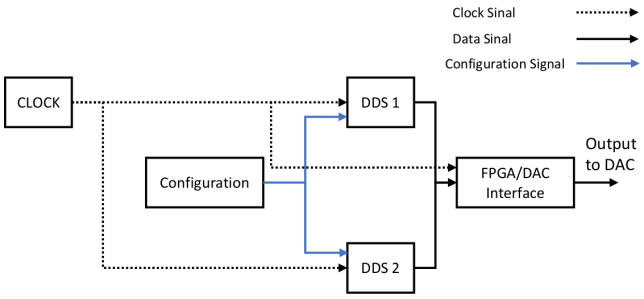


**FIGURE 19.** Signal generator schematic.

the DDS block in an open-core obtained from [20]. These blocks are fully described further in this subsection.

Four distinct blocks are presented. The clock block is responsible for supplying a reliable clock signal to the whole system. The DDS (Direct Digital Synthesis) block produces a programmable sine wave at its output. The frequency and phase of this wave are adjustable at runtime [21]. The configuration block configures the two DDSs blocks. Finally, the Interface block is responsible for correctly supplying DAC with data and signals. Each of these blocks mentioned represents an IP that is add to FPGA project, those IPs are now presented.

### 1) CLOCK BLOCK

The IP used to supply a constant and reliable clock signal to the system is the IP Clocking Wizard. This is a block designed by Xilinx that allows the creation of a clocking circuit that operates with a required frequency, phase and duty cycle, with reduced jitter. It uses mixed-mode clock manager (MMCM) or phase-locked loop (PLL) primitive [22].

In this project, the core operates as a PLL, which, as input, it receives a differential 125 MHz signal from external oscillator, and, as output, it supplies two signals, one with 125 MHz and a second with 250 MHz. Also, the IP provides a locked signal, which indicates whether the output clocks are stable and usable.

Since the available DAC in STEMLab board supports an update rate of up to 125Msps, the 125 MHz clock signal must be used as input of the DDSs blocks. On the other hand, a 250 MHz clock signal is used as input of the FPGA/DAC Interface block. This is necessary for the correct operation of the interleaved mode, as explained in section III-B.

### 2) DDS BLOCK

The IP used to produce a sine wave signal is an open source core called Direct Digital Synthesizer IP Core. With this IP, the frequency and phase of generating signal can be modified at runtime. Also, it is possible to define the wave phase and amplitude resolution [21].

As presented in [21], the output frequency $f_{DDS}$ is determined by *FTW* (Frequency Tuning Word), according to (1).

$$f_{DDS} = \frac{FTW}{(2^{14})} f_s \tag{1}$$

where $f_s$ is the sampling rate.

Also presented in [21], the initial phase ($p_{DDS}$) is determined by *PTW* (Phase Tuning Word), according to (2).

$$p_{DDS} = \frac{PTW}{(2^{14})} 2\pi \tag{2}$$

### 3) FPGA/DAC INTERFACE

This is the IP that connects FPGA with DAC, it implements all signals that allows DAC to work properly. Those signals are: DA13-DA0/DB13-DB0, IQWRT, IQSEL, IQCLK, IQRESET, and must behave as presented in Fig.18.

To clarify how this block is build, it is presented how each of the presented signal must be treated:

- IQWRT and IQCLK: Since both signals must operate at the rate of 250 MHz (Fig. 18), the interface block must replicate the 250 MHz clock signal coming from the Clock Block to those two signals.
- IQSEL: Since this signal must operate at the rate of 125 MHz (Fig. 18) it is used the 125 MHz clock signal coming from the Clock Block to generate it.
- DA13-DA0/DB13-DB0: Signals from channel one and channel two must be updated according to the state of the IQSEL signal, and must follow the rising edge of the IQCLK signal.

### 4) CONFIGURATION BLOCK

This IP controls both DACs. It provides them with the desired frequency and phase for the output signal. It is desirable to be able to modify those signals during system runtime operation, therefore the configuration IP must integrate the PS and PL sides. This way, it is possible for a user to input a desired signal frequency into a software (PS side), and this information is used to configure the DDS block (PL side).

The exchange of data between PS and PL is made by the use of the following Xilinx blocks:

- Zynq7 Processing System: Responsible for creating a logic connection between PL and PS structures.
- AXI GPIO: It is a general purpose input/output interface block. With this block, it is possible to access internal properties of the device and control external devices.
- AXI Interconnect: Responsible for connecting a AXI memory-mapped master (Zynq7 Processing System) to a AXI memory-mapped slave (AXI GPIO).

The entire process to create and connect all these blocks is explained in Section II.

### 5) EXPERIMENTAL RESULTS FOR SIGNAL GENERATION

The system presented in Fig. 19 has been implemented in the Vivado software.

The software developed for this application is available in a git repository in [23]. To validate its functionality, it has been used an oscilloscope to analyse the signals generated by the system. Fig. 20 presents the measurements made for a generated signal of 1MHz and 0.8V peak-to-peak. The yellow signal is from channel one and the cyan signal is from channel two.

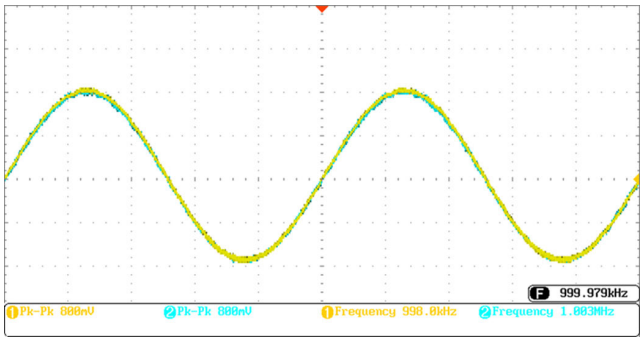


**FIGURE 20.** 1MHz signal generation.

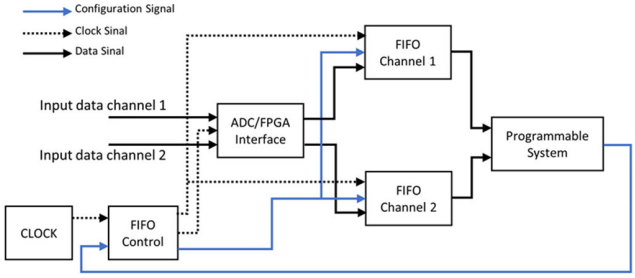


**FIGURE 21.** Signal acquisition schematic.

## IV. HIGH SPEED ANALOG-TO-DIGITAL CONVERTER

To build a signal acquisition system, it is necessary to fully understand the working of the board's ADC, therefore, this device must be explained first. After that, a signal acquisition system is proposed, designed and implemented, and the results obtained with the system are presented.

### A. ANALOG TO DIGITAL CONVERTER

The ADC incorporated to the STEMLab board is the LTC2145CUP-14. This converter is a 2-channel simultaneous sampling 14-bit device, it is designed for digitalizing high frequency and wide dynamic range signals. It works with a sampling frequency of up to 125Msps. Digital outputs can be full rate CMOS, double data rate CMOS, or double data rate Low-voltage differential signalling (LVDS) [24].

### B. SIGNAL ACQUISITION SYSTEM

The idea for the signal acquisition system is presented in Fig. 21.

Five different blocks were used to build this system. The clock block is a Xilinx IP Core and is the same used for the signal generating system, it has the purpose to supply a reliable clock signal to the whole system. The FIFO Control block is a custom made block by the authors, and is responsible to control the FIFO's operations, like writing to it or reading from it. The FIFO itself is an open-core obtained from [20]. It is a memory block, used to temporary store data from DAC. The ADC/FPGA Interface is also a custom made block by the authors. It is the block that receives data from the ADC and transfers it to the FPGA. Finally, the Programmable system block is an arrangement of Xilinx IPs, which is responsible for the communication between PS and PL sides - it controls the FIFO Control block and receives and saves data from the FIFOs. Each of these blocks mentioned represents an IP that is added to the FPGA project. Those IPs are presented as follows.

#### 1) CLOCK BLOCK

Same block presented in section III-C1.

#### 2) FIFO CONTROL

As presented in Fig. 21, the FIFO Control block has tree functions.

The first function is to supply a clock signal to the ADC/FPGA Interface block - thus it is possible to control the data acquisition rate. To provide a variable clock, it has been applied the same idea presented in Sub-subsection II-A2, where a square wave, with predefined frequency, is obtained by incrementing a counter at each rising edge of the clock signal. In the case of the FIFO Control block, based on a input configuration signal, it is possible to link any bit of the counter to the block output.

The second function is to provide a clock signal to the FIFOs blocks - thus, it is possible to control the rate of the data writing and data reading operation. For the data writing operation, it is used the same rate applied to the ADC/FPGA Interface block. Whereas for the reading operation, it is used a rate determined by the Programmable System block.

The third function is to supply a configuration signal to the FIFOs blocks - this configuration defines the operation mode of the FIFO, whether the reading or writing operating mode.

The FIFO Control block receives, as input two signals, which are: the clock signal, provided by the Clock block, and the configuration signal, provided by the Programmable System block.

#### 3) ADC/FPGA INTERFACE

The ADC/FPGA Interface block is responsible to convert the input data, from the offset binary format, to the two's complement format. This can be performed with a simple logic inversion of the most significant bit.

The interface block, as observed in Fig. 21 receives as inputs the data from the DAC's channel one and channel two. Also, it receives the clock signal provided by the FIFO Control block. As previously explained, this clock determines the rate of writing data to the FIFOs, or the rate of reading data from the FIFOs.

The block outputs the converted to two's complement data from channels one and two to the next blocks. Another function performed by this block is the conversion of the 14 bit input data into an integer size (32 bit) output data. This conversion is performed in order to better manipulate the data - since the software running on the PS treats the input data as an integer.

#### 4) FIFO

The First In First Out (FIFO) block is a memory block, where data can be stored and read. In this kind of block,

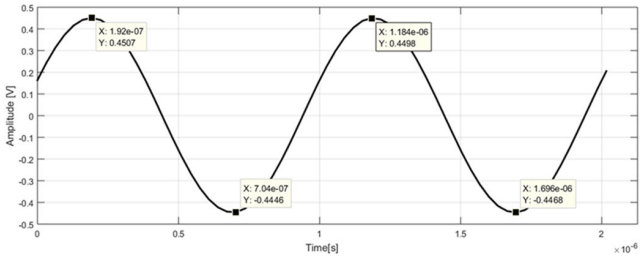


**FIGURE 22.** 1MHz signal acquisition.

the first stored data are the first to be read. This IP is also an OpenSource block, obtained from the site OpenCores [20].

It is possible to reset the FIFO using the input port RESET, that must be active high for at least two clock cycles.

Data is written to the FIFO using the input port WRITE. Data can only been written to the FIFO if it is not full. It is possible to monitor if the FIFO is complete or not complete with the output port FULL. This port is high when the FIFO is full.

Data is retrieved from the FIFO using the input port READ. The output port, DATA_PRESENT, indicates the existence of data inside the FIFO - therefore, if DATA_PRESENT is active high, the reading operation is allowed.

#### 5) PROGRAMMABLE SYSTEM

This IP integrates the PS and PL sides. This way, it is possible to use a software (running in PS size) to control the FPGA (PL size). Therefore, the block is responsible to supply the configuration signal to the FIFO Control block, as observed in Fig. 21. It is also responsible for receiving and saving the data stored in the FIFO IPs. This block is made by some Xilinx IPs, which are: Zynq7 Processing System, AXI GPIO, AXI Interconnect, all of this blocks were explained in Subsubsection III-C4. The exchange data between PS and PL follows the explanation of Section II.

### C. EXPERIMENTAL RESULTS FOR SIGNAL ACQUISITION

The system in Fig. 21 has been implemented in the Vivado software. The software developed for this application is available in a git repository in [23]. To validate it, the system has been used to acquire signals from a signal generator. In Fig. 22 it is presented the acquisition of an 1MHz and 0.9V peak-to-peak signal with the developed system. The highlighted points in Fig. 22 are the positive and negative peak values. Those values are, respectively, approximately +0.45V and −0.45V, which are in concordance with the generated signal.

## V. APPLICATION EXAMPLE - FREQUENCY RESPONSE ANALYSIS (FRA)

Using the instructions from the previous sections, the user can integrate the previous concepts in a practical application. As an application example, it is presented a system to detect early damage on the insulation of electrical machines. The system applies the FRA technique in addition to statistical indexes to obtain a fault tendency on a device under test. In this example, it is used an induction machine with taps, which gives the possibility to simulate damages to its insulation by inserting parasitic elements across the taps.

**TABLE 2.** Post-Synthesis Resources utilization.

| Resource | Estimation | Available | Utilization % |
|---|---|---|---|
| LUT | 3678 | 17600 | 20.90 |
| LUTRAM | 2048 | 6000 | 34.13 |
| FF | 381 | 35200 | 1.08 |
| IO | 71 | 100 | 71.00 |
| BUFG | 1 | 32 | 3.13 |

**TABLE 3.** Post-Implementation Resources utilization.

| Resource | Estimation | Available | Utilization % |
|---|---|---|---|
| LUT | 5945 | 17600 | 33.78 |
| LUTRAM | 2112 | 6000 | 35.20 |
| FF | 4852 | 35200 | 13.80 |
| IO | 73 | 100 | 73.00 |
| BUFG | 4 | 32 | 12.50 |
| PLL | 1 | 2 | 50.00 |

For this application, the required FPGA resources have been consulted and are presented, in Table 2, for Post-Syntheses and, in Table 3, for Post-Implementation. In both presented tables, LUT stands for lookup table, LUTRAM stands for lookup table random access memory, FF stands for flip-flop, IO stands for input-output, BUFG stands for global buffer, and PLL stands for phase-locked loop.

#### 1) FREQUENCY RESPONSE ANALYSIS

The FRA technique is a powerful diagnosis tool. A complete view of the technique is presented in [25]. The method applies a wide range of frequency signals into the machine's winding, and, for each frequency applied, the winding's impedance is calculated. After a complete frequency sweep, an impedance spectrum is obtained. This procedure must be repeated periodically during the lifetime of the analysed device, and, based on the differences between all spectrum obtained with a base spectrum (called baseline), it is possible to infer about the machine's insulation condition [26].

For proper application of the FRA technique it is necessary the use of a measurement circuit. Reference [27] presents these circuits. The one selected for this work is presented in Fig. 23.

By a analysis of the circuit in Fig. 23, it is obtained (3).

$$Z = \frac{V1}{(V1 - V2)} Rsh \tag{3}$$

where, $V1$ and $V2$ are, respectively, the measured voltage at channels 1 and 2 of the developed system. $Z$ is the impedance being measured, and $R_{sh}$ is a shunt resistor to measure the current at impedance $Z$.

#### 2) ABSOLUTE SUM OF LOGARITHMIC ERROR (ASLE)

With the use of the FRA method, the diagnosis is made by a visual comparative analysis between the baseline spectrum

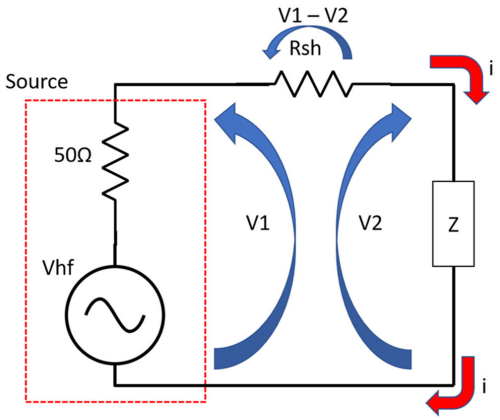


**FIGURE 23.** Measurement circuit.

and the new obtained spectra. The problem with this method it that the diagnostic is strongly dependent of subjectiveness of the analyst team. A way to enhance the analysis is by the use of statistical indexes [28], this allows a simple and practical way of analysing the machine, reducing the need of a specialist team and, also, reducing the subjectiveness component of the analysis. There are a variety of indexes in the literature of the FRA [29], however a well known index is the Absolute Sum of Logarithmic Error (ASLE), which is presented in (4). This index compares two set of data in a logarithmic scale. Two signals with high similarity are represented by a small ASLE index value, for two identical set of data the ASLE value is 0. Therefore, the higher the index value, the higher are the difference between the comparing data [30].

$$ASLE = \frac{\sum_{i=1}^{n} |20log_{10}y_i - 20log_{10}x_i|}{n} \tag{4}$$

where $n$ is the number of points of the generated spectrum, $x$ are the impedance points of the analysed spectrum, and $y$ are the impedance points of the baseline spectrum.

### 3) CONTROL SOFTWARE

A software must be used to control the developed hardware in FPGA. The interaction between this two instances is made by a client-server architecture, where the server is the hardware and the client is the control software. Thus, the server is responsible by receiving and executing commands from the client, and the client is responsible by sending a sequence of commands to perform the frequency sweep into the machine under test, and by calculating the statistical indexes. The control software has been developed in Python language, the diagram of its operation is presented in Fig. 24. For study purpose, a simplified version of the server and client software are presented in a git repository in [23].

### 4) EXPERIMENTAL SETUP

The developed system is applied to a 3HP, 460V, 4 poles squirrel-cage induction machine, presented in Fig. 25, with the developed system coupled.

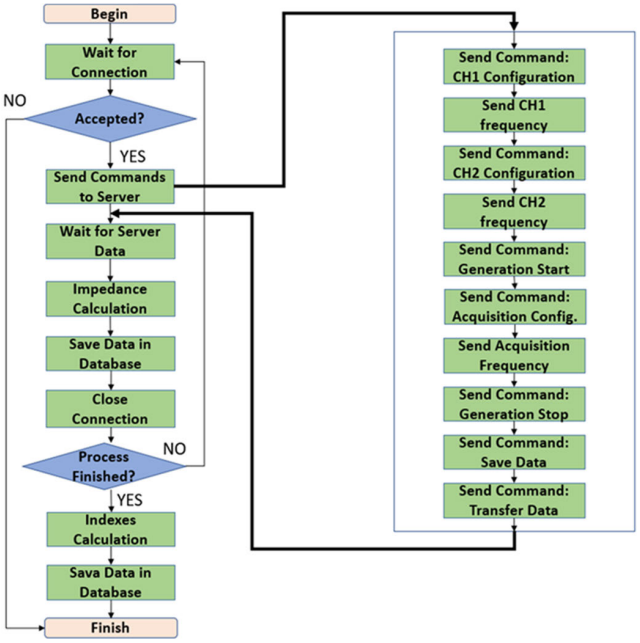


**FIGURE 24.** Software sweep frequency algorithm.

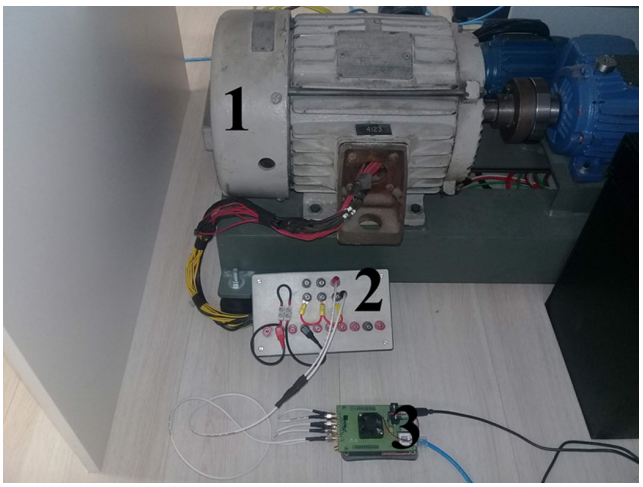


**FIGURE 25.** Developed system applied to induction machine. 1: induction machine. 2: panel to access taps of machine winding. 3: developed system.

The analysed machine is custom made, with access to some winding points (taps). According with [31], the insulation condition is related to machine winding capacitance, therefore, in this work, insulation faults are simulated by introducing different capacitors into taps of machine winding.

### 5) RESULTS ON THE INDUCTION MACHINE

For the experiments with the induction machine, six different capacitor values were used between two taps of the machine winding. These taps represent 5% of the total machine winding. The values of the capacitors were chosen to simulate a progression of the insulation failure. The system was configured to analyse the machine in range of 1kHz to 1MHz.

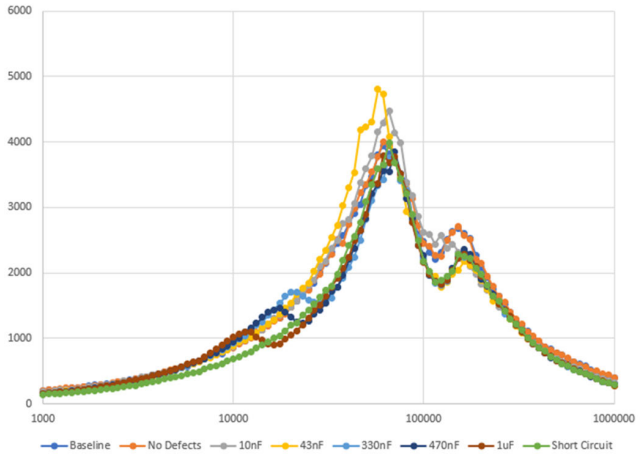


**FIGURE 26.** Impedance spectra for each insulation condition.

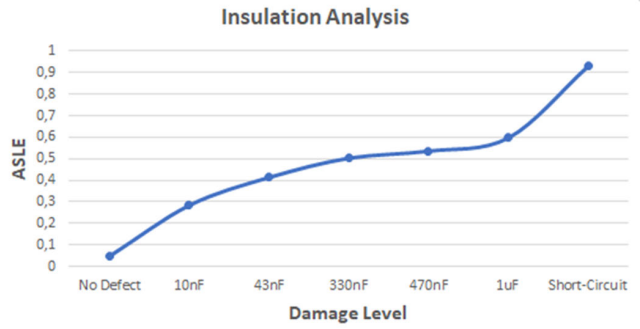


**FIGURE 27.** Induction machine statistical index curve.

For each different condition, twenty impedance spectra are obtained. The spectra are presented in Fig. 26, where the x-axis represents the analysed frequencies, and the y-axis represents the machine impedance. To avoid visual pollution, it is taken the average of the twenty spectra obtained for each condition. The capacitances inserted in the taps to simulate damage are: 10nF, 43nF, 330nF, 470nF, 1uF, and a short-circuit case.

From Fig. 26, it is seen that the quantification, based only on visual analysis, of the difference between an analysed spectrum and the baseline is not a trivial task, and can be easily misinterpreted by the analyst team. Therefore, the use of statistical indexes brings an impartial way of spectra interpretation. Fig. 27 presents the statistical index ASLE curve obtained.

Observing the results presented in Fig. 27, one can conclude about the machine's insulation damage evolution. For a condition without damage to insulation, it is obtained an index of, approximately, 0.05. This value increases as the damage to insulation becomes more severe, achieving a maximum value of 0.93 with the critical condition of short-circuit between winding's taps.

## VI. CONCLUSION

This work has presented a detailed instructional guide on FPGA prototyping using a low cost development board. The goal of this guide is to serve as an introductory material for graduate and undergraduate students developing their own research projects. Key concepts have been presented, starting with the basic workflow and read/write on I/O pins until the control of high speed analog peripherals and TCP/IP communication.

The instructions were focused on the STEMlab board, that comprises a Xilinx FPGA - although, with some modifications, they can also be applied to other boards, even the ones with FPGAs from another vendors (such as Altera/Intel).

Besides the introductory guide, this paper also has presented a complete academic and industrial application that sums-up the basic concepts introduced, presenting a system for detection of failures in electric machinery using FRA. The high speed analog peripherals included in the board have been used to perform frequency sweeps on the tested machine and the resulting impedance spectra have been used to infer about the motor's condition. Parasitic capacitances have been used to emulate early insulation damage on the machine's winding. The presented system have been able to properly detect the evolution of damage from a healthy baseline condition until a turn-insulation-short damage of 5% of the winding, passing through the intermediary stages with variation the parasitic capacitance.

As future development, some other previous research applications that were originally designed using commercial data acquisition systems are expected to be migrated to the STEMlab board (such as the detection of failures in electric machinery using electrical signature analysis [32], estimation of efficiency and torque of induction motors [33], among others). The intention is to familiarize undergraduate and masters degree students with the activities performed by the research group and also to train the students on the FPGA boards in order for them to develop their new applications.

## REFERENCES

[1] P. M. Menghal and A. J. Laxmi, "Real time simulation: Recent progress & perspective," in *Proc. Conf. Rec. Int. J. Electr. Eng. Electr. Syst.*, vol. 3, no. 1, 2010, pp. 45–59.

[2] C. A. Gross, "Dealing ethically with the publish or perish pressure," in *Proc. IEEE Power Energy Soc. Gen. Meeting - Convers. Del. Electr. Energy 21st Century*, Jul. 2008, pp. 1–2, doi: 10.1109/PES.2008.4596692.

[3] A. Ebrahimi and M. Zandsalimy, "Evaluation of FPGA hardware as a new approach for accelerating the numerical solution of CFD problems," *IEEE Access*, vol. 5, pp. 9717–9727, 2017, doi: 10.1109/ACCESS.2017.2705434.

[4] R. Inta, D. J. Bowman, and S. M. Scott, "The 'chimera': An off-the-shelf CPU/GPGPU/FPGA hybrid computing platform," *Int. J. Reconfigurable Comput.*, vol. 2012, pp. 1–10, Jan. 2012, doi: 10.1155/2012/241439.

[5] Z. Ma and X. Zhang, "FPGA implementation of sensorless sliding mode observer with a novel rotation direction detection for PMSM drives," *IEEE Access*, vol. 6, pp. 55528–55536, 2018, doi: 10.1109/ACCESS.2018.2871730.

[6] H. Palahalli, E. Ragaini, and G. Gruosso, "Smart grid simulation including communication network: A hardware in the loop approach," *IEEE Access*, vol. 7, pp. 90171–90179, 2019.

[7] M. D. O. Faruque, T. Strasser, G. Lauss, V. Jalili-Marandi, P. Forsyth, C. Dufour, V. Dinavahi, A. Monti, P. Kotsampopoulos, J. A. Martinez, K. Strunz, M. Saeedifard, X. Wang, D. Shearer, and M. Paolone, "Real-time simulation technologies for power systems design, testing, and analysis," *IEEE Power Energy Technol. Syst. J.*, vol. 2, no. 2, pp. 63–73, Jun. 2015.

[8] J. Belanger, P. Venne, and J. Paquin, "The what, where and why of real-time simulation," *Planet RT*, vol. 1, no. 1, pp. 25–29, 2010.

[9] M. J. Newman and D. G. Holmes, "Delta operator digital filters for high performance inverter applications," *IEEE Trans. Power Electron.*, vol. 18, no. 1, pp. 447–454, Jan. 2003, doi: 10.1109/TPEL.2002.807105.

[10] W. C. Sant'Ana, G. Lambert-Torres, L. E. Borges da Silva, E. L. Bonaldi, L. E. de Lacerda de Oliveira, C. P. Salomon, and J. G. Borges da Silva, "Influence of rotor position on the repeatability of frequency response analysis measurements on rotating machines and a statistical approach for more meaningful diagnostics," *Electr. Power Syst. Res.*, vol. 133, pp. 71–78, Apr. 2016, doi: 10.1016/j.epsr.2015.11.044.

[11] Y. Lee, Y. Kuo, H. Chen, and Y. Hsieh, "Field-programmable gate array-based coating impedance detector for rapid evaluation of early degradation of protective coatings," *IEEE Access*, vol. 7, pp. 20472–20478, 2019, doi: 10.1109/ACCESS.2019.2896996.

[12] (2016). *Snickerdoodle User Guide*. KRTKL Embedded Systems. [Online]. Available: https://krtkl.com/resources/docs/

[13] (2017). *STEMLab Documentation*. Red Pitaya. Red Pitaya Revision 7cf3a9a8. [Online]. Available: https://redpitaya.readthedocs.io/en/latest/

[14] *Zynq-7000 SoC Data Sheet: Overview*, Xilinx, 2018, dS190. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[15] (2017). *Processing System 7 v5.5 LogiCORE IP Product Guide*. Xilinx, pG082. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/processing_system7/v5_5/pg082-processing-system7.pdf

[16] (2017). *AXI Interconnect v2.1 LogiCORE IP Product Guide*. Xilinx, pG059. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf

[17] (2016). *AXI GPIO v2.0 LogiCORE IP Product Guide*. Xilinx. pG144. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf

[18] K. L. C. M. J. Donahoo, "Basic tcp sockets," in *TCP IP Sockets in C—A Practical Guide for Programmers*, 2nd ed., Burlington, MA, USA: Morgan Kaufmann, 2009, ch. 2, pp. 11–35.

[19] (2012). *DAC1401D125 Dual 14-bit DAC, up to 125 Msps*. IDT. rev. 03. [Online]. Available: https://www.idt.com/br/en/document/dst/dac1401d125-datasheet

[20] (2020). *The Reference Community for Free Open Source Gateware IP Cores*. OpenCores. rev. 2020. [Online]. Available: http://opencores.org

[21] (2008). *Direct Digital Synthesizer IP Core*. M. Kumm. [Online]. Available: https://opencores.org/projects/dds_synthesizer

[22] (2016). *Clocking Wizard v5.3 LogiCORE IP Product Guide*. Xilinx. pG065. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/clk_wiz/v5_3/pg065-clk-wiz.pdf

[23] (2020). *Client and Server Software Developed to Control a Signal Generation and Acquisition System Designed With FPGA*. [Online]. Available: https://github.com/brrgama/FPGA-Prototyping-Tutorial

[24] (2011). *LTC2145-14 14-bit, 125 Msps Low Power Dual ADC*. Linear Technology. lT 0712 REV A. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/21454314fa.pdf

[25] S. A. Ryder, "Diagnosing transformer faults using frequency response analysis," *IEEE Elect. Insul. Mag.*, vol. 19, no. 2, pp. 16–22, Mar. 2003, doi: 10.1109/mei.2003.1192032.

[26] W. C. Sant'Ana, C. P. Salomon, G. Lambert-Torres, L. E. B. da Silva, E. L. Bonaldi, L. E. de Lacerda de Oliveira, and J. G. B. da Silva, "Early detection of insulation failures on electric generators through online frequency response analysis," *Electr. Power Syst. Res.*, vol. 140, pp. 337–343, Nov. 2016, doi: 10.1016/j.epsr.2016.06.007.

[27] L. Lamarre and P. Picher, "Impedance characterization of hydro generator stator windings and preliminary results of FRA analysis," in *Proc. Conf. Rec. IEEE Int. Symp. Electr. Insul.*, Jun. 2008, pp. 227–230, doi: 10.1109/elinsl.2008.4570316.

[28] J.-W. Kim, B. Park, S. C. Jeong, S. W. Kim, and P. Park, "Fault diagnosis of a power transformer using an improved frequency-response analysis," *IEEE Trans. Power Del.*, vol. 20, no. 1, pp. 169–178, Jan. 2005, doi: 10.1109/tpwrd.2004.835428.

[29] W. C. Sant'Ana, C. P. Salomon, G. Lambert-Torres, L. E. B. Silva, E. L. Bonaldi, L. E. L. Oliveira, and J. G. B. Silva, "A survey on statistical indexes applied on frequency response analysis of electric machinery and a trend based approach for more reliable results," *Electr. Power Syst. Res. (Print)*, vol. 137, pp. 26–33, 2016, doi: 10.1016/j.epsr.2016.03.044.

[30] K. P. Badgujar, M. Maoyafikuddin, and S. V. Kulkarni, "Alternative statistical techniques for aiding SFRA diagnostics in transformers," *IET Gener., Transmiss. Distrib.*, vol. 6, no. 3, pp. 189–198, Mar. 2012, doi: 10.1049/iet-gtd.2011.0268.

[31] F. Perisse, P. Werynski, and D. Roger, "A new method for AC machine turn insulation diagnostic based on high frequency resonances," *IEEE Trans. Dielectr. Electr. Insul.*, vol. 14, no. 5, pp. 1308–1315, Oct. 2007, doi: 10.1109/tdei.2007.4339494.

[32] C. P. Salomon, C. Ferreira, W. C. Sant'Ana, G. Lambert-Torres, L. E. B. da Silva, E. L. Bonaldi, L. E. L. de Oliveira, and B. S. Torres, "A study of fault diagnosis based on electrical signature analysis for synchronous generators predictive maintenance in bulk electric systems," *Energies*, vol. 12, no. 8, pp. 1–16, 2019, doi: 10.3390/en12081506.

[33] C. P. Salomon, W. C. Sant'Ana, G. Lambert-Torres, L. E. B. da Silva, E. L. Bonaldi, and L. E. L. D. Oliveira, "Comparison among methods for induction motor low-intrusive efficiency evaluation including a new AGT approach with a modified stator resistance," *Energies*, vol. 11, no. 4, pp. 1–21, 2018, doi: 10.3390/en11040691.


**BRUNO R. GAMA** received the B.S. degree in automation and control engineering and the M.Sc. degree in electrical engineering from Itajuba Federal University (UNIFEI), Itajuba, Brazil, in 2017 and 2019, respectively, where he is currently pursuing the D.Sc. degree in electrical engineering.

Since 2017, he has been a Research Assistant at Gnarus Institute, Itajuba, Brazil. He has experience with development of systems based on SOC/FPGAs. His research interests include condition-based maintenance of electric machinery and digital signal processing.

**WILSON C. SANT'ANA** (Member, IEEE) received the B.S., M.Sc., and D.Sc. degrees in electrical engineering from Itajuba Federal University (UNIFEI), Itajuba, Brazil, in 2001, 2004, and 2016, respectively.

Since 2011, he has been a Researcher with the Gnarus Institute, Itajuba. He has experience with development of hardware and software for microcontrollers, DSPs, and FPGAs. His research interests include condition-based maintenance, frequency response analysis of electric machinery, power electronics, and control systems.

**GERMANO LAMBERT-TORRES** (Fellow, IEEE) received the Ph.D. degree in electrical engineering from the Ecole Polytechnique de Montreal, Canada, in 1990.

From 1983 to 2012, he was with the Electrical Engineering Department, Itajuba Federal University (UNIFEI), Itajuba, Brazil, where he was also the Dean of the Research and Graduate Studies, from 2000 to 2004. From 1995 to 1996, he was a Visiting Professor with the University of Waterloo, Waterloo, ON, Canada. Since 2010, he has been the Director of the Research and Development, PS Solucoes, Itajuba. He also serves as a Consultant for many utility companies in Brazil and South America, and he has taught numerous IEEE tutorials in USA, Europe, and Asia. He is the author/editor or coauthor of nine books, more than 30 book chapters, and 50 transactions articles on intelligent systems and nonclassical logic.

**CAMILA P. SALOMON** (Member, IEEE) received the B.S., M.Sc., and D.Sc. degrees in electrical engineering from the Itajuba Federal University (UNIFEI), Itajuba, Brazil, in 2011, 2014, and 2017, respectively.

Since 2018, she has been with the Institute of Electrical Engineering and Power Systems, UNIFEI. Her research interests include condition-based maintenance, electrical signature analysis, artificial intelligence application, and induction motor efficiency methodologies.

**ERIK L. BONALDI** received the B.S., M.Sc., and Ph.D. degrees in electrical engineering from Itajuba Federal University, Itajuba, Brazil, in 1999, 2002, and 2006, respectively.

He is currently the CEO with PS Solutions, Itajuba, and a Research Associate with the Gnarus Institute, Itajuba. His research interests include industrial electronic automation, predictive maintenance, artificial intelligence methodologies, and rough sets classifier.

**LUIZ E. BORGES-DA-SILVA** (Senior Member, IEEE) received the B.S. and M.Sc. degrees in electrical engineering from Itajuba Federal University (UNIFEI), Itajuba, Brazil, in 1977 and 1982, respectively, and the Ph.D. degree from the Ecole Polytechnique de Montreal, Montreal, QC, Canada, in 1988.

In 1998, he was a Visiting Professor with The University of Tennessee, Knoxville, TN, USA. He is currently a Professor with the Department of Electronic Engineering, UNIFEI. His research interests include power electronics, electronic power systems, power converters, and applications of adaptive and intelligent control in industrial problems.

**RAFAEL B. B. CARVALHO** received the B.S. degree in electrical engineering from Itajuba Federal University (UNIFEI), Itajuba, Brazil, in 2009.

He was a Research Assistant with the High Voltage Laboratory (LAT - Laboratorio de Alta Tensao), UNIFEI. He is currently with EDF Norte Fluminense, Brazil, as the Coordinator of electrical maintenance. His research interests include condition based maintenance and vibration analysis.

**FABIO M. STEINER** received the bachelor's degree in administration from the Candido Mendes University, Brazil, in 1991.

His specializations in Finance from IBMEC and Oil and Gas Business Management from COPPE. From 2003 to 2012, he was with the Brazilian National Electric System Operator (ONS - Operador Nacional do Sistema Eletrico). Since 2012, he has been with EDF Norte Fluminense, Brazil, as the Manager of Research and Development.