IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 11 October 2022, accepted 19 November 2022, date of publication 24 November 2022, date of current version 30 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3224720

RESEARCH ARTICLE

Modbus Protocol Performance Analysis in a Variable Configuration of the Physical Fieldbus Architecture

VASILE GHEORGHIŢĂ GĂITAN, (Member, IEEE), AND IONEL ZAGAN, (Member, IEEE)
[1]Faculty of Electrical Engineering and Computer Science, Stefan cel Mare University, 720229 Suceava, Romania
[2]Integrated Center for Research, Development and Innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for Fabrication and Control (MANSiD), Stefan cel Mare University, 720229 Suceava, Romania

Corresponding authors: Ionel Zagan (zagan@usm.ro) and Vasile Gheorghiţă Găitan (vgaitan@usm.ro)

This work was supported in part by the Integrated Center for Research, Development and Innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for Fabrication and Control (MANSiD), Stefan cel Mare University, Suceava, Romania; in part by the Project 119722/Centru Pentru Transferal de Cunoştinţe Către întreprinderi Din Domeniul ICT—CENTRIC—Contract Subsidiary 21773/04.10.2022/DIGI-TOUCH/Fragar Trading through the Infrastructure from the Project "Integrated Center for Research, Development and Innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for Fabrication and Control" under Contract 5/AXA 1/1.2.3/G/13.06.2018 and Contract SMIS 2014+119722 (ID P_40_305); and in part by the Sectoral Operational Program for Increase of the Economic Competitiveness co-funded from the European Regional Development Fund under Contract 671/09.04.2015.

ABSTRACT Few developments have changed the evolution of the automation process as substantially as the development of industrial local area networks and communication protocols. Especially fieldbus systems, networks created for the lowest levels of the automation hierarchy, have an enormous influence on the flexibility and performance of modern automation systems in all application areas. The technique of updating data on Modbus type devices involves a procedure of reading data at regular time intervals called polling. Some automation processes include interdependence between different control variables belonging to different hardware devices. This paper presents and analyzes various scenarios related to a variable architecture configuration of physical modules based on Modbus communication protocol. The proposed Modbus Extension (ModbusE) concept is presented by defining the new optimized message format, and the structure of the acquisition cycle to obtain a deterministic temporal behavior, solutions for describing devices at the Modbus protocol level being presented. The status update of each Modbus module is done according to the address of the device, but also the number of registers per device. The paper analyses the worst-case scenario of communication involving Modbus devices on the same network and exchanging data corresponding to one or more server registers.

INDEX TERMS Modbus, acquisition cycle, remote terminal unit (RTU), communication.

I. INTRODUCTION

The need to solve the wiring problems in the field of automation, that has become a problem in large industrial processes, was certainly the main constraint for the development of fieldbus systems. Other obvious and attractive advantages of the fieldbus concept are the modularity, the possibility of easily expanding the installations and the option of having much more intelligent field devices that can communicate not only for process data transfer, but also for maintenance, security, and configuration [1]. Another point of view that led to different design approaches was to consider network systems in industrial process control as constituent entities of real-time distributed systems. While cabling optimization concepts were in many cases simple bottom-up approaches, these real-time distributed ideas led to sophisticated and usually well-researched designs.

Fieldbus systems must be seen as an integrated part of a complex automation concepts and not as stand-alone solutions. Considering the large dimensions of process automation installations, the fieldbus system benefits are particularly obvious. However, the concept was not uncontested when

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

it was introduced. The fieldbus approach was an ambitious concept: a step towards decentralization, including data preprocessing in field devices, security [2], [3], which together increases the quality of the control process and reduces the computational limit of centralized controllers. With the development of implementation technologies, the ability to configure and parameterize field devices remotely via the network was also added.

This advanced concept, on the other hand, required increased communication between devices that goes far beyond simple data exchange [4]. A more detailed definition of fieldbus is given by the Fieldbus Foundation, organization that supports one of the major fieldbus systems [1]: Fieldbus is a digital, bidirectional, multi-drop communication link between intelligent measurement and control devices. It serves as a Local Area Network (LAN) for advanced process control, remote input/output applications, and secure high-speed factory automation applications [5], [6]. This is somewhat restrictive though, as it limits the application to advanced processes and factory automation. Thus, the generic term fieldbus defines cable networks or industrial buses, in this context "field" meaning relatively far from the "center". Fieldbus systems were not the result of an invalidated ideas, they appeared in a continuous and often difficult process of evolution based on existing technology.

Today, numerous application fields are unthinkable without them: factory automation [7], distributed process control [8], building automation and energy distribution in general, but also the automotive industry, railway applications and aeronautical field. All these sectors rely heavily on the availability of suitable networks that meet special requirements of individual applications [9], [10]. Current modern anomaly detection and classification methods for industrial control systems are based on network traffic data of industrial field protocols such as Modbus TCP and S7 Communication [11].

Based on research related to the Modbus communication protocol, the contributions of this paper are summarized as follows:

1) Decomposition of a typical industrial protocol functionality;
2) Analysing, testing, and validation of various scenarios related to variable architecture configuration of physical modules based on Modbus protocol.
3) Defining the structure of an acquisition cycle (ModbusE) corresponding to incompletely defined networks;
4) Development and definitions of the ModbusE protocol basic aspects.

As a main contribution in the field of industrial communication protocols, the structure of an acquisition cycle for incompletely defined networks has been implemented by adding a timestamp to achieve wide time consistency. Thus, the Base Station Gateway (BSG) can add a time stamp to a unit of information or a set of information. The main advantage of adding a timestamp to a simple protocol, such as Modbus, is to achieve time consistency (a common view of the controlled and/or monitored process). Time consistency is very important in the implementation of industrial process control algorithms. By introducing this additional information we can add a low-cost feature specific to sophisticated protocols. This option is implemented at fieldbus level for Modbus (the time stamp is added by the BSG) and on both the controller and the ModbusE fieldbus. These options are enabled or disabled at the installation stage of the BSG and in stations that have ModbusE functionality.

The rest of this paper is organized as follows. Chapter II reveals the fieldbus systems general characteristics and chapter III describes the mathematical model for Modbus communication performance analysis. Chapter IV validates the experimental results and chapter V reviews the previous Modbus based designs. Chapters VI and VII presents the discussions and conclusions.

II. GENERAL CHARACTERISTICS OF FIELDBUS SYSTEMS

Networks data transmission was possible only with special investments in specialized hardware and software solutions. The purpose of Open Systems Interconnection (OSI) model was to control this development. The International Organization for Standardization (ISO) introduced and supports open system concepts [12]. These systems integrate hardware and software components that adhere to a specific set of standards. These standards guarantee that systems from different manufacturers are compatible and can communicate easily. Within fieldbus systems, like all modern communication systems, fieldbus protocols are essentially modeled based on ISO/OSI model. However, in most cases only layers 1, 2 and 7 are used [13]. This is actually the result due to the Manufacturing Automation Protocol (MAP), where a full seven-level stack was found to be too resource-intensive and not allow for efficient implementation. For this reason, the MiniMAP approach (a version containing only layers 1, 2 and 7) and based on it the IEC fieldbus (International Electrotechnical Commission) standard explicitly prescribes a three-layer structure, which consists of the physical layer, data link and the application layer [1]. The classification of Fieldbus traffic is divided into two types, namely: automation-oriented traffic and parallel traffic (Figure 1). Two categories can be distinguished for automation-oriented traffic, namely the first real-time (process data) and the second non-real-time (management data). In terms of process data, this type is sometimes also referred to as cyclic traffic or identified traffic because once the request is specified the communication relationships must be known. Depending on the process characteristics, this data can be periodic or aperiodic. Periodic traffic is mostly about the state of a process and is usually handled by time-slotted communication strategy, where each variable is allocated a dedicated bandwidth slot.

Network bandwidth is improved depending on the sampling time or generation rate of the data source. Aperiodic, acyclic, or spontaneous traffic is generated on demand in an

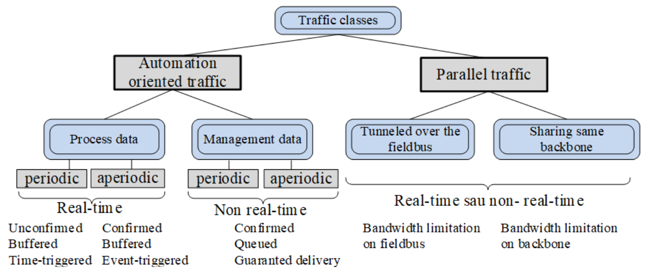
Traffic classes
Automation oriented traffic
Parallel traffic
Process data
Management data
Tunneled over the fieldbus
Sharing same backbone
periodic
aperiodic
periodic
aperiodic
Real-time
Non real-time
Real-time sau non- real-time
Unconfirmed Buffered Time-triggered
Confirmed Buffered Event-triggered
Confirmed Queued Guaranted delivery
Bandwidth limitation on fieldbus
Bandwidth limitation on backbone

FIGURE 1. Traffic classes within fieldbus systems.

event-driven manner and transmitted based on the availability of free communication bandwidth. Management data is typically aperiodic because it usually occurs depending on the system state. Traditionally, they are often transmitted in certain dedicated communication channels for parameters that use a small percentage of the bandwidth. The parallel traffic type does not belong to application processes that refer to the actual control of technical process [1]. Rather, it is generated by parallel independent processes sharing a common communication medium. Modbus is a communication protocol developed by Gould Modicon (now Schneider Electric) for process control systems. This is still considered a public protocol and has become the de facto standard in multi-vendor integration [14].

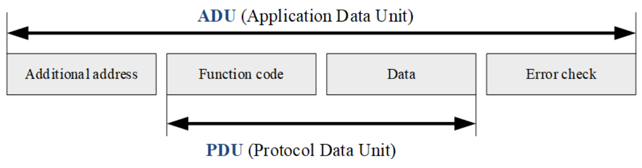

FIGURE 2. General MODBUS framework including ADU and PDU.

III. THE MATHEMATICAL MODEL FOR MODBUS COMMUNICATION PERFORMANCE ANALYSIS

A. DESCRIPTION OF MODBUS PROTOCOL FUNCTIONS

The Modbus protocol defines the Protocol Data Unit (PDU) as the elementary protocol data unit, regardless the underlying communication levels [15]. Modbus protocol mapping for a specific bus or network is done at the Application Data Unit (ADU) level by introducing additional fields (Figure 2). Considering the interdependence between the client-server model and Media access control (MAC) strategies, it should be noted that, in principle, client-server communication can be implemented in both single-master and multi-master systems. In the latter case, which can be based on Carrier-sense multiple access (CSMA), Time Division Multiple Access (TDMA) or Transport Protocol (TP), each master can take over the role of a client, whereas in single-master systems this position is reserved for the bus master. Thus, the client-server paradigm is mainly used for master-slave systems, represented by PROFIBUS, ASi, MODBUS, P-NET, BITbus and INTERBUS, and for reliable application-level data transfer (file transfer, network and device management). Especially for management functions, the client-server model is also widely used in fieldbus systems that organise regular data transfer according to the publisher-subscriber model, such as WorldFIP, EIB, CANopen, DeviceNet, ControlNet or LonWorks. Within this project, the register update times are tested and analyzed on a Modbus device that has a maximum number of 50 holding registers, these being at consecutive or dispersed addresses.

In Figure 3, a client has been defined, in this case an application (SMARTConvert), which runs on a PC-type computing system. The main features and development stimuli of fieldbus systems are:

- Focused solutions: Fieldbus systems had no stated general purpose. They have always been developed with a concrete scope and designed to meet the respective boundary conditions (such as not only time behavior, efficiency and reliability, but also costs) in the best possible way.
- Smart devices: A key goal of embedded systems and fieldbuses is to bring more intelligence to the domain, i.e. the end devices. As with embedded systems, fieldbus developers also used technology building blocks available at the time, such as off-the-shelf microcontrollers, to keep costs down. However, dedicated solutions have also been developed to meet special fieldbus needs.
- Limited resources: Embedded applications and fieldbus embedding systems share the fundamental problem of limited resources. Regardless of the state of the art in microelectronics, embedded devices are less powerful than standard computers. Communication subsystems typically have less bandwidth than computer networks, and power consumption is an issue.
- Comprehensive concepts: Fieldbus systems are not just networks. Communication is only one part of a distributed automation concept with comprehensive software applications and numerous other tools. In some advanced cases, fieldbuses have been incorporated into special frameworks that exhibit many features of distributed operating systems.
- Distribution: A distributed network is a prerequisite of real-time distributed systems. Many data processing tasks can be removed from a central controller and placed directly in field devices if the interface can handle sufficiently complex communication modes.
- Flexibility and modularity: A fieldbus installation like any other network can be expanded much more easily than a centralized system, if the limits of the address space or cable length are not exceeded. For the special case of fieldbuses, system-commissioning process is more advantageous due to a more simplified parameterization and configuration of complex field devices.
- Maintenance: Monitoring devices, applying updates and other maintenance tasks are easier if possible over a network.

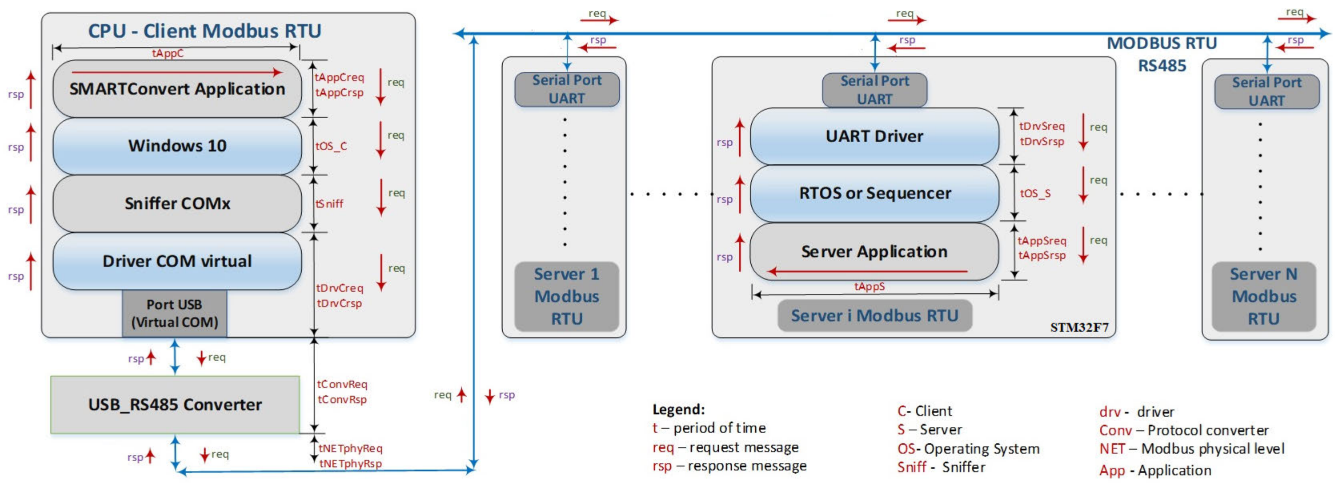

FIGURE 3. The support architecture for mathematical model description and Modbus acquisition cycles times definition.

The concept was to create a transparent, multi-level network as a basis for Computer-Integrated Manufacturing (CIM). Usually, this model comprises up to five levels, sometimes even more.

While networks for the upper layers already existed when defining the automation pyramid, the field level was still governed by point-to-point connections. Fieldbus systems have also been developed to bridge this gap. The actual integration of field-level networks into the rest of the hierarchy was, in fact, considered in the original standardization [1]. However, for most proprietary developments the actual integration of networks into the automation hierarchy was never a primary intention.

A device implementing the Modbus client service can initiate Modbus messaging requests to another device defining a specific Modbus server. These requests allow the client to transmit data and/or send commands to a remote device. Regarding the Modbus server services, a device implementing the Modbus server can respond to requests from any Modbus client. In this paper, we will use the term server for slave, and client for master, in accordance with the new Modbus specifications. The Modbus server service allows a device to make all of its internal and input/output data available to remote devices for reading and writing, and allows other commands to be executed. The communication speed for which the tests were done is 9600 bps. For this, the mathematical model will be defined to facilitate the definition and implementation of tests necessary to determine update times of the values from/to a device on MODBUS RTU (Remote Terminal Unit) bus. Figure 3 illustrates the message communication diagram, highlighting the various components that could produce measurable delays. For unicast addresses, Modbus defines a transaction (T) that consists of a client request and the related response issued by server that received the request uniquely. In this context, four components were highlighted:

- SMARTConvert application;
- Operating system (in this case WINDOWS 10).
- Sniffer that monitors messages on virtual COMx.
- USB driver;
- USB RS485 converter;
- Server i Modbus RTU.

TABLE 1. Modbus acquisition cycle notations.

Abbreviation	Significance
t	Time period
TFxx	Modbus transaction for specific function xx
TF03OK	Transaction with correct response for FC03
TF03ERR	Transaction with error response for FC03
TreqFxx	Transaction request side time
TrspFxx	Transaction response part time
TrspOkFxx	Response part without error
TrspErrFxx	Response part with error
Ttout	Timeout per transaction (server not responding, or client lost the message)
n	Maximum number of servers
tbit	Time period of one bit transmitted on physical layer
ch	Data char
tframe	UART frame time period (10 \time tbit, no parity 1 start+8 data+1 stop)
lframe	Frame bit length (usually 10, 11 bits)
tch	Time period of one character (payload 8b)
h	Header
data	Data Area (Registers)
nreg	Number of records
Si	An i slot at the application level
SiReq(FCxx,nreg)	Slot request part
SiRsp(FCxx,nreg)	Slot response part.
AC(adr, FCxx)	Acquisition cycle

The only place we can monitor activity on the client device is at the sniffer level. This message path may not be very precise. The notations used in this paper are presented in Table 1. Therefore, we consider the functions that work with this type of registers. From the analysis of the control function codes, the possible functions are:

1) FC03 (0×03): Read holding register;

2) FC06 (0 × 06): Write single register;
3) FC16 (0 × 10): Write Multiple registers;
4) FC22 (0 × 16): Mask write register;
5) FC23 (0 × 17): Read/Write multiple registers.

Function FC03 (0×03) read memory registers and store the data. This function code is used to read the contents of a contiguous block of memory registers from a remotely accessed device.

Of these functions, we will only use FC03 and FC16. Thus, registers numbered from one to 16 are addressed with addresses from zero to 15. The PDU request specifies the address of the start register and the number of registers.In the PDU, registers are addressed starting at zero address. Thus, registers are located with addresses 0 through 15. Register data in the response message is packed as two bytes for each register, with the binary contents right aligned for each byte. The FC16 function organizational chart is shown in Figure 4. For each data, the first byte contains the most significant bits and the second byte contains the 7-0 bits. Memory allocation is done at the byte level using fc03_registers area. The acquisition is accomplished with a specialized task, then it will access the fc03_registers area to write, and the Modbus function implementation task will read. In order to eliminate race conditions a mutex must be provided on the access to this memory area which will eliminate this inconvenience. The transaction time in terms of Modbus message length for function FC03 is calculated using the following equation. First, the number of bits of the transaction is obtained. Now we can calculate the transaction time on the physical communication medium.

$$TF03 = TreqF03 + TrspF03 \tag{1}$$

$$\begin{aligned} TreqF03 &= h + data + crc = (1\ addr + 1\ function\ code \\ &\quad + 2 start\ address + 2\ amount\ \ of\ \ registers) \\ &\quad + 0 + 2 = 8\ frame \end{aligned} \tag{2}$$

$$\begin{aligned} TrspF03 &= TrspOkF03 = (1\ addr + 1\ function\ code \\ &\quad + 1\ amount\ of\ \ registers) + nreg \times 2 + CRC \\ &= 3 + nreg \times 2 + 2 = 5 + nreg \times 2 \end{aligned} \tag{3}$$

$$TF03OK = 13 + nreg \times 2\ (frames) \tag{4}$$

$$TF03OK = (13 + nreg \times 2) \times lframe\ (bits) \tag{5}$$

$$tTF03OK = (13 + nreg \times 2) \times lframe \times tbit \tag{6}$$

If the server response is an error then:

$$\begin{aligned} TrspF03 &= TrspErrF03 = (1\ addr + 1\ error\ code \\ &\quad + 1\ exception\ code) + 0 + CRC \\ &= 3 + 0 + 2 = 5 \end{aligned} \tag{7}$$

$$tTF03ERR = 13 \times lframe \times tbit \tag{8}$$

Function FC16 (0 × 10) is used to write a continuous block of registers (from 1 to 123 registers) to a remotely accessed device. The values to be written are specified in the data field of the Modbus request. Data is packed as two bytes per register. A normal response returns the function code, starting address, and the amount of registers written. The example in Table 2 shows a request to write two registers starting at address 1 with the values 0 × 00AA and 0 × 5502. Transaction timing in terms of Modbus message length for FC16 is calculated using the equation (9).

$$TF16 = TreqF16 + TrspF16 \tag{9}$$

The transaction time slot on physical medium is given by the following equation:

$$\mathrm{tTF16OK} = (17 + \mathrm{nreg} \times 2) \times \mathrm{lframe} \times \mathrm{tbit} \tag{10}$$

For these formulas, we must take into account the Modbus specifications, which say that between two consecutive characters the maximum time, can be 1.5 characters, and between two consecutive messages, there must be a time distance of at least 3.5 characters. This timing actually specifies the

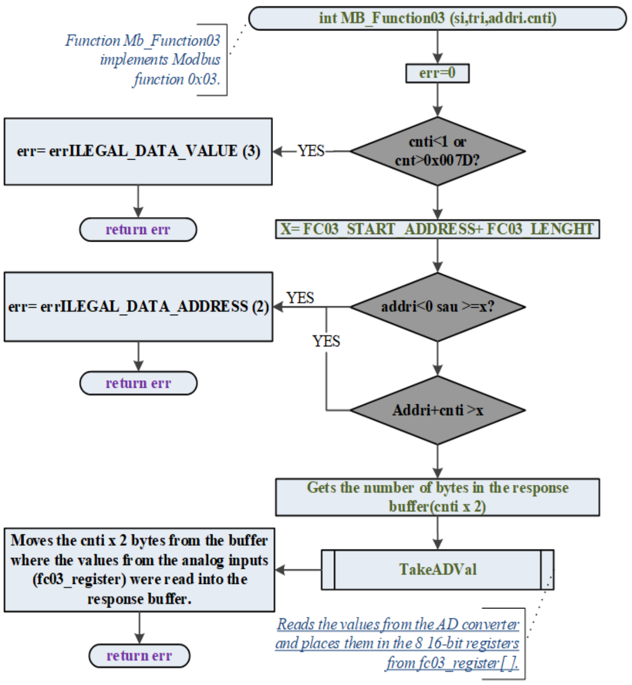
int MB_Function03 (si,tri,addri.cnti)
Function Mb_Function03 implements Modbus function 0x03.
err=0
cnti<1 or cnt>0x007D?
YES
err= errILEGAL_DATA_VALUE (3)
return err
X= FC03_START_ADDRESS+ FC03_LENGHT
addri<0 sau >=x?
YES
err= errILEGAL_DATA_ADDRESS (2)
return err
YES
Addri+cnti >x
Gets the number of bytes in the response buffer(cnti x 2)
TakeADVal
Moves the cnti x 2 bytes from the buffer where the values from the analog inputs (fc03_register) were read into the response buffer.
return err
Reads the values from the AD converter and places them in the 8 16-bit registers from fc03_register[ ].

FIGURE 4. MB_Function03 function flowchart.

TABLE 2. Example of request to write two registers.

Request		Response	
Field name	(hex)	Field name	(hex)
Function	10	Function	10
High start address (Hi)	00	High start address (Hi)	00
Lower Start Address (Lo)	01	Lower Start Address (Lo)	01
Registers number (Hi)	00	Registers number (Hi)	00
Registers number (Lo)	02	Registers number (Lo)	02
Byte counter	04		
Hi register value	00		
Lo register value	AA		
Hi register value	55		
Lo register value	02		

end of a Modbus message. As a result, an addition of times occurs between zero and the frames maximum number of the message minus one (after the last character follows the pause between messages of 3.5 characters).

Si acquisition slot encompasses all operations on the Modbus transaction from sending a request, until receiving the response from server or a timeout. Based on Figure 3, the slot i timing is defined by equation (11):

$$tSi(adr, FCxx, nreg) = tSiReq\,(adr, FCxx, nreg) + tSiRsp\,(adr, FCxx, nreg) \quad (11)$$

Request and response times have been highlighted separately because Modbus transaction messages are highly asymmetric.

Thus, in the Modbus protocol, the asymmetry depends on the control function and nreg. The equations presented defined the Modbus transaction that takes place on Modbus physical layer. Based on this, an acquisition slot has been defined at the level of a client running on an advanced operating system (Windows 10). This slot encapsulates a single Modbus transaction, and next we will define a complete acquisition cycle.

$$tAC\,(AC\_S) = \sum\nolimits_{i}^{i\in MCA} tSi\,(adr, FCxx, nreg) + \sum\nolimits_{i}^{i\in MCA} tadjust \quad (12)$$

The tadjusti parameter represent the time to introduce a specific delay between slots when some Modbus RTU servers have high jitter behavior when responding to requests. Usually tadjusti parameter is zero.

B. MATHEMATICAL EQUATIONS PARTICULARIZATION BASED ON PROJECT CONFIGURATION

An acquisition cycle, which only runs on the client, consists of any set of acquisition slots (AC_S) that can refer to:

1) Consecutive addresses (no gaps of Modbus RTU server type devices);
2) Addresses scattered, but also with or without consecutive address sequences;
3) Addresses that are repeated interspersed with different control functions or the same control function;
4) Performing operations such as reading from a server and sending the value to another server on the same network or on a network connected to a virtual COM or Modbus TCP/IP gateway client that sends remotely to a TCP/IP gateway, which is the client of a local virtual COM etc.

Depending on the character transmission using the Modbus protocol α parameter can take values in the range 0 to 1. If there is no delay between transmitted characters then α is 0, and if a delay is introduced between the transmissions of two characters on Modbus α parameter is different from 0. If the distance between two successive transmitted characters is 1.5 characters, as specified in the Modbus standard, the value for α is 1 (maximum value). α1and α2 have been introduced for Modbus functions FC03 and FC16, as there are different averages in message transmission. Modbus is an incompletely defined protocol because it does not determine how and in what order data acquisition is performed on the communication bus. Thus, the ModbusE concept proposes an extension of the Modbus protocol, defined by an acquisition cycle on the client-server bus, resulting in a deterministic communication protocol. Additionally, the ModbusE protocol incorporates the option to add a time stamp to data acquisition messages that are part of the acquisition cycle.

In this subchapter we calculate the AC_S acquisition cycle time with the equation (13).

$$\begin{aligned} \boldsymbol{tAC}\,(\boldsymbol{AC\_S}) = {} & \sum\nolimits_{i}^{i\in MCA} (tClienti(adri, FCxx, nregi) \\ & + tServeri(adri, FCxx, nregi) \\ & + tTFxxi(FCxx, \alpha i, erri, tout,\ nregi) \\ & + tHWneti) + \sum_{i}^{i\in MCA} tadjusti \end{aligned} \quad (13)$$

We assume that tadjusti is zero because there is no significant jitter for servers requests, so we consider the following case. A single read for all 10 registers from all addresses (1-246, without address 247 assigned to the fan coil server), the thermostat having the address in the range 1-246, and a single write for the 10 registers to the server with address 247 (the worst case regarding the fan coil address). For this case, we need to read 10 registers from 246 server stations and write 10 registers to address 247. The AC_S set for the read can be expressed by the equation (14).

$$\begin{aligned} & \boldsymbol{AC\_S}\,(adr, FCxx, adrReg, nrReg) \\ & \quad = \{(1, 3, 5, 10), \\ & \qquad (2, 3, 5, 10), \ldots, (246, 3, 5, 10), (247, 16, 5, 10) \end{aligned} \quad (14)$$

From the communication time point of view, the tTFxxi function is not affected by the address value or the register address. In this test, we send 246 time-identical messages on the Modbus. The acquisition cycle time for AC_S can be calculated by means of the equation (15).

$$\begin{aligned} \boldsymbol{tAC} = {} & 246\ x\ tAll1(1, FC03, 10) \\ & + 246\ x\ tTF031(FC03, \alpha 1, err1, tout, 10) \\ & + tAll247(247, FC16, 10) \\ & + tTF16\_247(FC16, \alpha 2, err2, tout, 10) \end{aligned} \quad (15)$$

In this scenario, we can have multiple instances at the slot level. If the Modbus message is transmitted correctly, the following equations is defined:

$$\boldsymbol{tTF03\_i}\,(FC03, \alpha 1, 0, 0, 10) = 41.668 + 50.0016 \times \alpha 1\ ms \quad (16)$$

$$\boldsymbol{tTF16\_247}\,(FC16, \alpha 2, 0, 0, 10) = 45.8348 + 56.2518 \times \alpha 2\ ms \quad (17)$$

The time for the Modbus acquisition cycle is defined by equation (17):

$$\boldsymbol{tAC} = 246\ x\ tAlli\,(i, FC03, 10) + tAll247(247, FC16, 10) + 10250,328\ ms + 12300,3936 \times \alpha 1\ ms + 45,8348\ ms + 56,2518 \times \alpha 2\ ms \qquad (18)$$

If $\alpha 1 = 1, \alpha 2 = 1, \mathrm{err}1 = 0, \mathrm{err}2 = 0$ and $\mathrm{tout} = 0$ then the acquisition cycle time is calculated as follows.

$$\boldsymbol{tAC} = 246\ x\ tAlli(i, FC03, 10) + tAll247(247, FC16, 10) + 22652,8082\ ms \qquad (19)$$

If $\alpha 1 = 0, \alpha 2 = 0, \mathrm{err}1 = 0, \mathrm{err}2 = 0$ and $\mathrm{tout} = 0$, then tAC is calculated according to equation (19).

$$\boldsymbol{tAC} = 246\ x\ tAlli(i, FC03, 10) + tAll247(247, FC16, 10) + 10296.1628\ ms \qquad (20)$$

If the message is transmitted with errors, then tTF03i is calculated with the following equations.

$$\boldsymbol{tTF03i}(FC03, \alpha, 1, 0, x) = tTF03ERR = 20834 + 18750,6{\times}\alpha 1\ us \qquad (21)$$

$$\boldsymbol{tTF16\_247}(FC16, \alpha 2, 1, 0, x) = tTF16ERR = 42709.7 + 51564.15 \times \alpha 2 \qquad (22)$$

If $\alpha 1 = 1, \alpha 2 = 1, \mathrm{err}1 = 1, \mathrm{err}2 = 1$ and $\mathrm{tout} = 0$ then tTF031(FC03,1,1,0,x) = 39.5846 ms and tTF162(FC16, 1,1,0,x) = 94.27385 ms. The time for acquisition cycle is:

$$\boldsymbol{tAC} = 246\ x\ tAlli((i, FC03, 10) + tAll247(247, FC16, 10) + 9832.07385\ ms \qquad (23)$$

If $\alpha 1 = 0, \alpha 2 = 0, \mathrm{err}1 = 1, \mathrm{err}2 = 1$, and $\mathrm{tout} = 0$ then tTF031(FC03,0,1,0,x) = 20.834 ms and tTF162(FC16,0,1,0,x) = 42.7097 ms. In this case, the acquisition cycle time is calculated according to equation (23).

$$\boldsymbol{tAC} = 246\ x\ tAlli((i, FC03, 10) + tAll247(247, FC16, 10) + 5167.8737\ ms \qquad (24)$$

As there are more transactions, it is difficult to highlight the timeout. Any timeout adds about 50 ms to the communication delay.

IV. EXPERIMENTAL RESULTS

A. MESSAGE TRANSMISSION TIMING BETWEEN DEVICES BELONGING TO THE SAME MODBUS COMMUNICATION PROTOCOL

In order to test the communication platform regarding the exchange of messages between devices belonging to the same Modbus type communication protocol, an application was made on STM32F7-Discovery development kit on which 247 stations each with 50 registers were simulated. Thus, using a device, the tests related to the project can be carried out, namely the use of all server addresses, from one to 247. For these 50 registers available at the level of each station (server), the required Modbus functions are available, namely FC03 for reading and FC16 for writing.

The STM32F7-Discovery is a development platform that includes the STM32F746NG MCU microcontroller. The development kit allows users to develop and share applications with the STM32F7 series of microcontrollers based on the ARM Cortex M7 core. Arduino connectivity provides unlimited expansion facilities with a wide range of specialized boards. The experiments in chapter V and the shortcomings found on this occasion led to the choice of this design kit. In addition, different editions of STM32 series microcontrollers have different hardware integrated Modbus protocol facilities. Only the STM32F7 implements them all in one chip. The facilities offered are the following:

- 7, 8, 16 and 32-bit cyclic redundancy check (CRC) calculation hardware unit with the possibility to freely choose the polynomial;
- Programmable sampling, which allows obtaining serial communication speeds of up to 27 Mbps;
- Hardware direction control for the RS485 driver with programmable delay times both when changing direction and returning from it;
- Fast transfer through Direct memory access (DMA);
- Time-out facilities (which allows the easy implementation of measuring time distances of 1.5 characters between characters and 3.5 characters between messages);
- USART 7 was used for the tests.

Connection between the PC and the development kit was made with a USB – 3.3V converter identified as Prolific USB-to-serial - Comm Port. In the following, the times for exchanging values from a thermostat to a fan coil are validated for the worst-case scenario, (the thermostat has address 1, and the fan coil has address 225 (247), and there are no free addresses between them). In this experiment, it will be taken into account that all devices have 10 consecutive registers that need to be updated. Tests will include update times for 9600 bps communication speed. In terms of device numbers, the application has been designed to use all user addresses from 1 to 247. ID 0 is used for Broadcast commands and device IDs 248 to 255 are reserved and are generally used for vendor-specific gateway functions. For practical laboratory tests, using an STM32F7 microcontroller on which Modbus devices were emulated, the PC did not have enough resources to communicate with more than 39 devices. Thus, it was concluded that the operating system in this case should only be used for a specific application. Using Modbus Poll software and Modbus server stations, PDU request/response messages with associated time stamps were monitored and analyzed. In the case of the Modbus request message "03 00 00 00 00 00 01 84 0A", related to function FC03, the time stamp "001833-10:32:18.939-01" is associated.

In case of the 39-station tAC acquisition cycle measurement, the times for 9600 bps communication rate are as follows: 4.526 s, 4.527 s, 4.533 s, and 4.554 s. As we

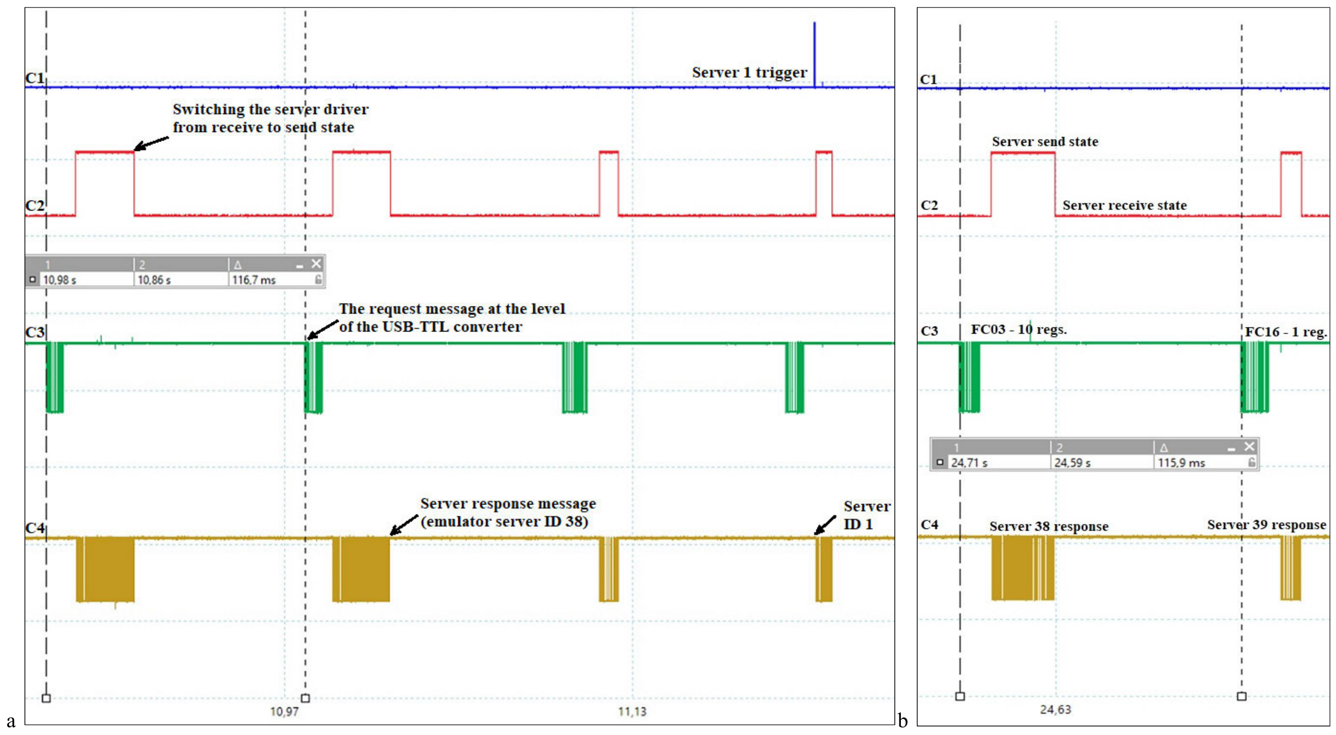

FIGURE 5. FC03 and FC16 Modbus messages interprettion. a) The time period between two requests sent by the client to the servers 37 - 38; b) Time between two consecutive requests with FC03 and 10 registers – FC16, 1 register (experimental values: 115.8 ms, 116.3 ms, and 116.2 ms).

can see in Figure 5, C1 cursor marks the trigger on server 1 and C2 cursor indicates the driver transition to the queried server from receive to transmit state. Stations with IDs 1 and 39 have shorter transmission periods because only one data register is read. The red (C2) pulses are commands for the server driver to switch to broadcast. Longer C3 pulses correspond to requests of type FC03, when reading 10 registers, and the shortest are F16 when writing a register (fan coil – server address 39) and FC03 when reading a register (thermostat – server address 1).

Thus, between two consecutive requests a period of 116.7 ms was measured (Figure 5.a). It includes querying the server with ID 37, switching the driver in the broadcast, the time for processing at the server level that includes the request analysis, sending the response to the client with 10 registers (C4) as the message payload, respectively the period between two messages according to Modbus specifications. The practical data from the oscilloscope capture illustrates the time between two requests sent by the client to the servers 37 and 38. The experimental data obtained in Figure 5.a validate the theoretical notions presented in chapter IV, so that for two consecutive requests of FC03 type function with 10 registers the following periods were obtained 115.5 ms, 115.8 ms, 116.7 ms and 116.7 ms. The time between two FC03 – FC16 requests is 116.3 ms and 115.9 ms (Figure5.b).

For the case where there are 257 servers, specifying the equations for the project purpose with $\alpha1 = 1$, $\alpha2 = 1$, err1 = 0, err2 = 0, and tout = 0 gives a period of 246 \time tAlli(i,FC03,10) + tAll247 (247,FC16,10) + 22652.8082 ms for tAC, so the practical results obtained validate the presented mathematical model. For $\alpha1 = 0$, $\alpha2 = 0$, err1 = 0, err2 = 0 and tout = 0 a maximum of 246 \time tAlli(i,FC03,10) + tAll247(247,FC16,10) + 10296.1628 ms can be obtained.

The acquisition cycle defined in this paper applies to protocols based on the time-triggering paradigm. This is not essential when using the IEC 61131-3 standard, which is the dominant standard for Programmable Logic Controller (PLC) programming, as it uses a centralized model to control cyclic program execution. Conversely, it can be a disadvantage for the IEC 61499 standard, as it implements function blocks event-based execution activation. The execution semantics of a function block is not clearly defined in this standard, so two major aspects related to the notion of time can be identified. The first of them relates to the lifetime of an event and the second to notions of block communication, i.e. if there is a command to execute the block. However, for this standard, increasing the determinism of the functional blocks behavior [16] involves the use of the synchronized execution [17], the cyclic execution [18], [19] and the ISaGRAF model [20]. These patterns are close to the cyclic execution of function blocks.

Analyzing, for example, the Modbus specification as well as other non-standard extensions [21], we can argue that only Modbus RTU is suitable for real-time communications. Modbus ASCII allows a maximum of 6 seconds between characters, which is unacceptable for any real-time application [22], and Modbus TCP/IP uses the TCP/IP stack, which is of the best-effort type.

A Modbus RTU message can be up to 256 bytes [14]. A Modbus network allows 247 addresses for the server station, address zero is used for broadcast messages, and addresses from 248 to 255 are reserved. An Application Protocol Data Unit (APDU) message contains a station address byte, a function code byte, and the data area containing the request or response parameters and a checksum. Data is always 11 bits long, of which one start bit, 8 data bits, one parity bit and one stop bit, or two stop bits and no parity. With Modbus RTU, each frame must be separated from the other by a minimum of 3.5 characters of inactivity. If there is more than 1.5 characters of inactivity between two consecutive characters, the framework is considered incomplete and is rejected. Based on these characteristics, we can conclude that maximum length of a message is $(256 + 3.5 + 255 \times 1.5)$ \time $11 = 7062$ bits, where 256 is the maximum number of characters in a message and 255 is the maximum number of spaces in the message which is 256 characters long.

Modbus messages, except broadcast messages, are sent in command-response pairs. Consequently, we need to determine the maximum length of the command or response, given that the request or response is of maximum length. Most requests and responses are five characters, but there are exceptions when the length can reach the maximum value.

The proposed acquisition cycle wants to transform non-deterministic protocols (Modbus, M-Bus, and ASCII-DCON) into deterministic protocols. This acquisition cycle is time-triggered because a low-cost microcontroller implementation using synchronous serial communication is desired.

This concept does not allow bus arbitration via the protocols mentioned above. However, implementation on serial port microcontroller systems is cheaper, allowing actual implementation and validation of the Modbus extension for affordable distributed embedded systems. This timing should be specified, but unfortunately, only a few manufacturers guarantee a value for this parameter, and the protocol does not specify anything about this issue. A solution could be to develop, at the BSG level, a command-processing-response time evaluation function for all requests used in application, with a reasonable timing for different operating modes corresponding to the tested devices. An alternative solution could be to approximate the processing time and attach four counters to each slot. We need a counter for messages correctly received, a counter for errors generated by exceeding the 1.5-character idle time between two consecutive characters in a Modbus frame, a counter for CRC errors, and a counter for time-out errors.

The experimental data obtained from the laboratory tests validate the mathematical model for the analysis and testing imposed on this project, thus for the response message time period of the server with ID 1 and function Modbus FC03 with 1 register a time period between 7.116 ms and 7.221 ms can be obtained (Figure 6). The measured times for processing on server one are as follows: 5.087 ms, 5.306 ms, 5.635 ms, and 5.689 ms, with an average of 5.429 ms. Thus, the query message time period is not greater than 8.26 ms.

If there are errors generated by exceeding the 1.5 character idle time between two consecutive characters in a Modbus frame, it is desirable to replace the station because it does not comply with the Modbus specification. If significant time-out errors are generated, then the estimated time to process the commands is wrong and the slot jitter should be increased. This operation can be performed dynamically by the BSG or statically by the designer configuring the network. Increasing the slot time is performed until the generated errors are acceptable, which are a few percent of all messages, depending on the application requirements.

If there are many CRC errors without accompanying other errors, such as 1.5 characters or time-out, then the network may be affected by noise or there are physical faults in the network or at the server station. For each message, the Round-trip time (RTT) is calculated and the maximum value is considered. These values will be taken into account when the acquisition cycle is set up.

It is desirable that the difference between the minimum and maximum RTT values does not increase in value. Each slot may have a margin of safety determined by the performance requirements of the acquisition cycle (its time).

A slot must have a transmission time multiple of three and less than or equal to the maximum value of the RTT plus the safety margin, which mainly includes the processing time of the BSG. The slot allocated to Service Data Object (SDO) messages should not exceed 10-15% of the acquisition cycle, and if there are longer orders, then they should be segmented.

Most Modbus commands allow segmentation, so one or more commands can be attached to a slot. It would be preferable for a command to be attached to a single slot, and in case of critical situations, the second message could be a station status query, thus checking whether critical events have occurred or not.

B. MODBUS EXTENSION PROPOSED SOLUTION

Fieldbuses implement only layers 1, 2, and 7 of the OSI protocol stack, with the current approach-addressing layer 2. For shared bus networks, addressing is handled by the Media Access Controller subsystem. Usually, incompletely defined protocols do not include time variable specification, so at the server station level, the time variable is not explicitly used. So a master station called BSG is required to manage the time variable. This station allows access to the fieldbus via the Internet, via direct connection or via a host computer. BSG can add a timestamp to a unit of information or a set of information. In this case, we can achieve broad temporal coherence.

The time interval is dictated, on the one hand, by the broadcast command protocol specification, such as the start of data acquisition, and, on the other hand, by the acquisition cycle structure at the level of each station (usually the binary variables are written and read faster than analog variables, which require more time for data conversion). We must not neglect

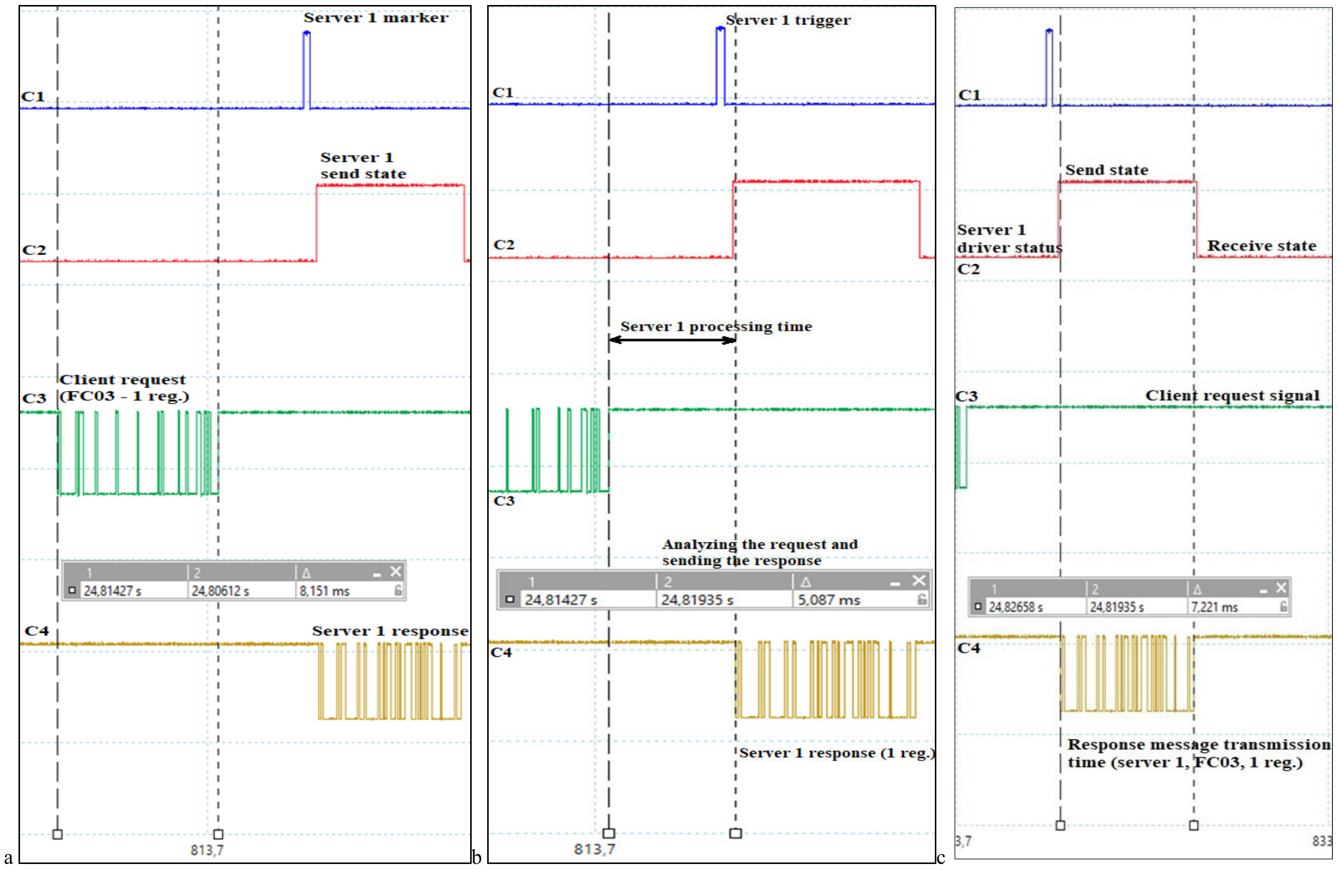

FIGURE 6. Response times corresponding to the client-server devices. a) Time period of the request message corresponding to the server ID 1, FC03, 1 register (experimental values: 8.26 ms, 8.206 ms, 8.3 ms); b) Server processing time (request analysis and response sending, forced delay = 1ms) ID 1, FC03, 1 register, (experimental values: 5.635 ms, 5.689 ms, 5.306 ms); c) Time period required to issue the server response message ID 1, FC03, 1 register (experimental values: 7.212 ms, 7.166 ms, 7.18 ms).

the time required for signal processing in such a network, the stations being totally synchronized. Therefore, the implementation effort is only on the BSG station. For this acquisition cycle, as a result of the research carried out we present a practical example for the Modbus protocol, which is widely used in SCADA systems. Analyzing the practical results obtained with the Modbus protocol, we establish that its performance can be improved. To achieve this goal, we propose an original extension called ModbusExtension (ModbusE). A ModbusE acquisition cycle consists of n slots. As a result, the time period for a tAC acquisition cycle is given by the following equation:

$$t_{CA} = \sum_{i=0}^{n-1} ts_i \qquad (25)$$

The time period for slot i (tSi) is made up of the times required for software processing at the client, respectively at the server (tswi, tswMi, tswSi). To these are added the hardware delay times at the client and at the server (thwi, thwMi, thwSi, tlinei), the times required for the characters issued by the client through the request message and those sent by the server in response to these requests (tcommi, tcommMi, tcommSi) and the slot adjustment time (tadjusti).

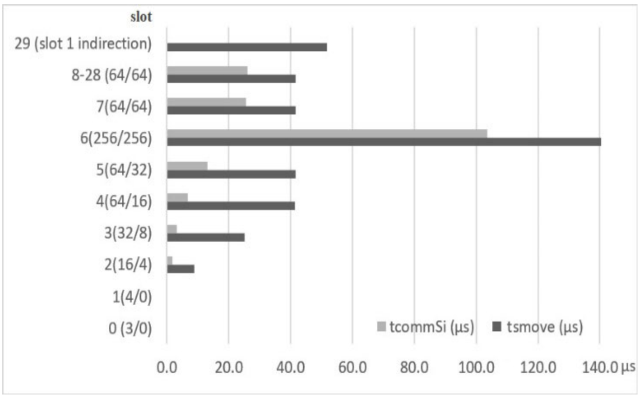

FIGURE 7. Measured times for acquisition cycle slots at the server.

Figure 7 and Table 3 show the times measured for client and server acquisition cycle slots when ModbusE is used. Thus, tsmove represents the time required to transmit the message from the server's reception buffer to the final buffer. In the proposed ModbusE approach, research has led to a main contribution, namely, the structure of an acquisition cycle corresponding to incompletely defined networks to add a time stamp and achieve temporal coherence.

TABLE 3. Times measured for acquisition cycle slots.

Slot	tmCRC (µs)	tmswitchi (µs)	tmpsnsli (µs)	tmfosli (µs)	tSi (µs)
0 (3/0)	0	6.145	4.285	4.285	46.96
1(4/0)	0	6.11	4.341	4.168	49.02
2(16/4)	0	6.145	4.324	3.869	86.04
3(32/8)	2.467	6.078	4.317	8.281	98
4(64/16)	2.819	6.167	4.317	8.715	122
5(64/32)	3.348	6.519	4.317	9.25	128.2
6(256/256)	4.228	6.579	4.521	10.28	342.2
7(64/64)	18.59	6.519	4.317	34.44	141
8-28 (64/64)	6.255	6.167	4.317	12.16	141
29 (slot1 indirection)	6.167	6.343	4.317	12.69	141

Based on the Table 3 experimental data, tmCRC represent the CRC calculation timing for message received in the previous slot, and tmswitchi is the time required to switch mbeThreadCycleRTU task (the most priority in the system). The symbol tmpsnsli defines the time to prepare the emission of the command for the new slot (the client prepares the emission of the new slot i), and tmfosli represents the time required to finish the operations related to the old slot (the client finishes the old slot). The solutions proposed in this paper are simple and inexpensive, as they allow the integration of existing acquisition modules without hardware or software changes.

V. RELATED WORK

Researchers in the literature have proposed various extensions and architectures to address the main Modbus protocol challenges and fieldbus requirements. From a general perspective, it is not enough to define new fieldbus proposals that are not based on decentralized transmission technology or Ethernet. Developing an extension of already available solutions is perfectly acceptable, as long as full backward compatibility is ensured and the extension offers tangible advantages in the perspective of one or more application areas. In [23], the authors describe a case study of the Protege language in an industrial setting. The authors implemented the Modbus protocol over TCP/IP and the serial line and tested it using an industrial gateway. The implementation described in [23] demonstrates Protege's advantages for software productivity, maintenance, and code reuse so that it achieves many desirable properties of industrial integrated networking software. The authors' main technical contribution is exemplifying how to decompose the functionality of a typical industrial protocol to improve code modularity and reusability. In the test presented by the authors, 128 bytes were reserved for time variables used by packet processing or protocol logic. Since Modbus does not support retransmission, no buffer has been reserved for storing temporal packets.

The information in [24] highlights and validates a secure command and control channel over the Modbus/TCP protocol, which is the logical next step for attackers, and evaluates its suitability. The channel stores information in the least significant bits of the working registers to carry information using Modbus read and write methods. This provides an explicit trade-off between bandwidth and channel stealth that can be set by the attacker. As the security posture improves, attackers will likely move to more stringent approaches such as secured channels. To test the validity of the approach for implementing a secured communication channel, the Modbus protocol and the control channel were implemented. The implementation was accomplished in Python 2.7 using the Modbus library. The instrumented code is then used to run multiple transfers of a compressed file from the server, or to the client, the equivalent of ex filtering a compressed data archive. This action is selected as the most suitable for the benchmark test because it maximizes bandwidth channel utilization. In [25], researchers suggest an innovative Modbus extension, in a backward compatible way, to cover deficiencies such as address space size, guaranteed bandwidth allocation, and shared transfer between multiple client stations. The proposed extension does not require any additional hardware, so it is suitable for low-cost distributed embedded systems. The concept proposed by the authors in [25] was implemented with a low development effort, starting from the available Commercial Off-The-Shelf (COTS) protocol stack.

In [26], Al-Dalky et al. propose an automated tool for generating malicious SCADA traffic. Recently, the connectivity of SCADA systems to the Internet has introduced new vulnerabilities, thus becoming the target of a huge number of attacks. For this implementation, traffic generation of the popular SCADA Modbus protocol is considered. The generated traffic characteristics are derived from the Modbus protocol based on Snort Network Intrusion Detection System (NIDS). The proposed tool uses Scapy to generate packets considering extracted traffic characteristics. The authors successful validates tool results, which is used to read a Snort rule file containing Modbus rules to extract the required traffic characteristics. The paper [26] describe and highlights how to develop an automated tool for generating malicious Modbus traffic from the NIDS. Thus, in order to secure SCADA systems, Scapy is used to evaluate the effectiveness of the proposed solutions. The proposed tool successfully generates malicious SCADA traffic triggering Snort NIDS rules. However, is useful and necessary to extend proposed tool [26] to generate other types of SCADA protocols like those of DNP3. In [27], the over-control and jitter of Modbus TCP communication are well evaluated by theoretical analysis and experimental data. The results presented by the authors validates that TCP recognition mechanism is mainly responsible for RTT jitter, which in turn can affect the system security and reliability. In contrast, the related implementation expenses are much more affordable for typical embedded applications. The experimental results analysis is supplemented with theoretical aspects. The TCP Acknowledgment (ACK) segment mechanism, which is an elementary component of the TCP/IP protocol used by receivers to acknowledge receipt of data from senders, has been tested to introduce a significant amount of jitter into Modbus TCP communication.

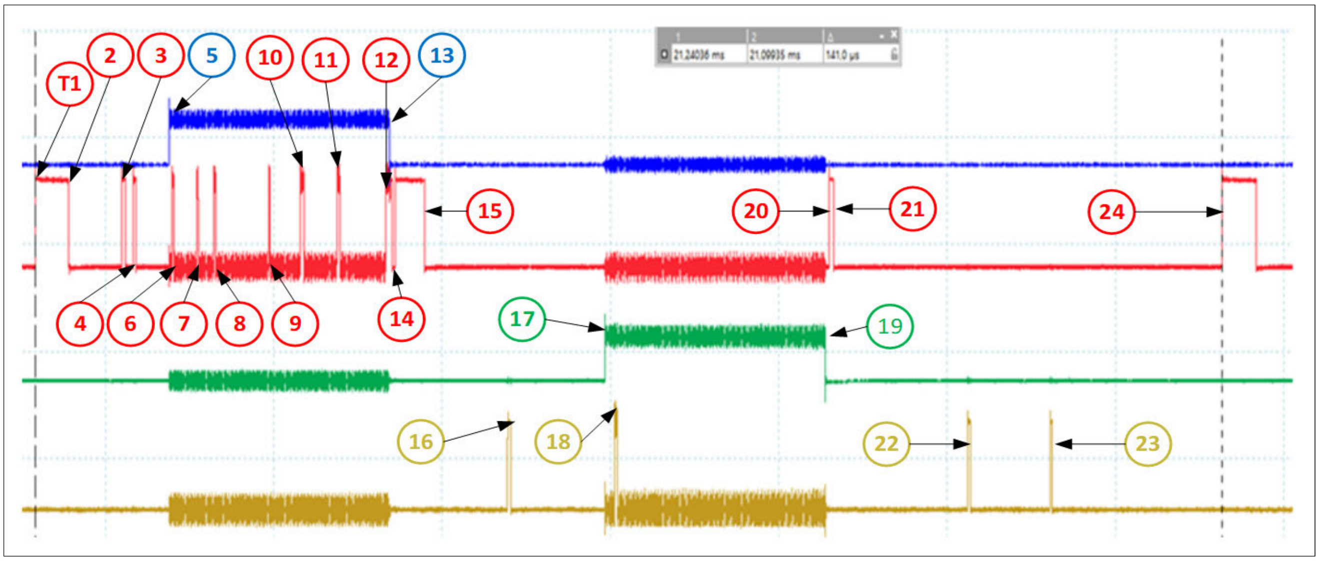

FIGURE 8. Modbus acquisition timings within a tSi slot (141 μs).

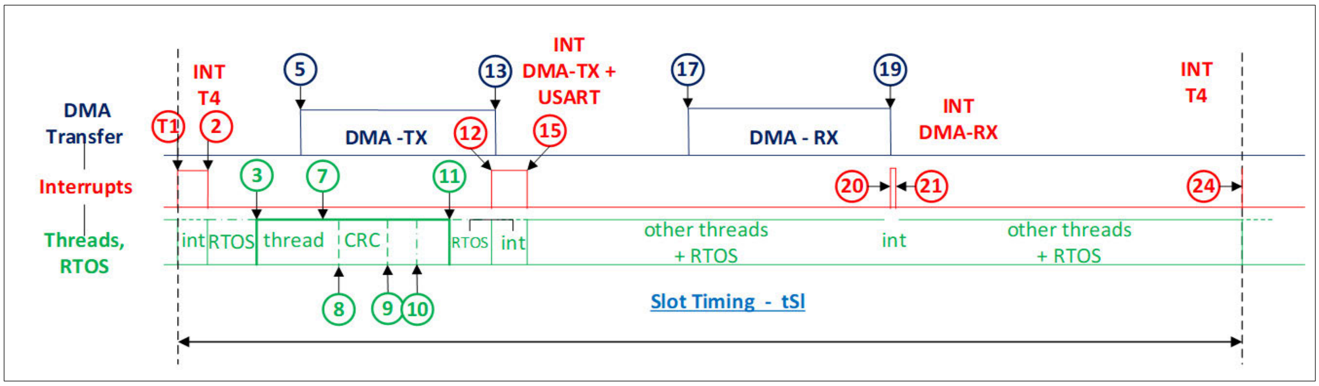

FIGURE 9. The degree of operations overlap during a slot corresponding to the oscilloscope capture shown in Figure 8.

In [28], the authors have addressed the transmission quality improvement in industrial communications based on Modbus RTU, using relay devices with the ability to detect and correct errors. This method respects the Modbus RTU protocol parameters, so that the enhanced bus with relay devices can be used together with common devices. Arguably, this feature has not been found in previous approaches. Tests performed on repeaters/error correctors validate the fulfillment of design requirements and demonstrate their effectiveness in improving transmission quality for industrial Modbus RTU data transmission networks. The authors in [28] present the design and implementation of error detection and correction scheme in a Modbus RTU network based on the REED-SOLOMON code. Parity bits transmission is done following Modbus protocol parameters, so the bus can be used by common devices without error detection/correction scheme. The detection, respectively the correction, of errors is carried out by means of specialized repeater type devices in the receiver and transmitter. This method respects the parameters of the Modbus RTU protocol, so the enhanced bus with relay devices can be used together with tested devices. At a transfer rate of 19.200bps, relay devices enable full compatibility with equipment that is directly connected to the communication bus, if response times to Modbus requests are not less than 5ms. Connecting with equipment that has lower response times must be done through repeaters to avoid data collisions. Thus, the implementation of error correction methods using the Modbus RTU protocol, such as the one presented in the work [28], proved to be efficient and up-to-date.

VI. DISCUSSION

Evaluating the wired and wireless technologies in the computer world available today, regarding their applicability in automation is a first step in this direction. Ultimately, we can expect completely new optimized automation networks where the communication network segments are seamlessly integrated and perhaps only the application protocol remains compatible with traditional wired solutions to integrate.

From Modbus protocol analyzes, we determine that its performance can be improved. To achieve this goal, we propose an original extension called ModbusE. The innovative solutions are simple and inexpensive, as they allow the integration of existing acquisition modules without hardware or software changes. To achieve time consistency, the solution described in this paper for the acquisition cycle uses the Data Link Layer (DLL). This solution does not deal with the application layer (AL) and user layer (such as IEC61499), but can serve as support for these layers. A data acquisition cycle at the BSG level that satisfies time coherence has been defined and evaluated. The important points at the slot level for implementation are presented in Figure 8. Thus, the degree of operations overlap during the time of an acquisition slot can be drawn. The meaning of these points and the times they delimit are described in the following. The start and end position T1/T24 mark the timing of acquisition slot i. Figure 9 indicates the degree of acquisition slot overlap operations.

Time moment T1 represents the time at which the acquisition slot begins. The interrupt period given by timer 4, which measures the slot timing, is indicated by the period T1-T2. The time to execute mbeThreadCycleRTU task is marked by T2-T3. At the time period T4-T6 the client prepares the emission of the new slot i, and during the period T6-T7 mbeThreadCycleRTU deletes the event sent by the interrupt given by timer 4, when the operations related to the old slot (T10) are finished. At time moment T11 ThreadCycleRTU task switches to waiting state for an event from timer 4, which indicates the start of a new slot. T12-T14 represents the time period of the handler for interrupt generated by DMA with the emission of the entire message, and at time T16 the client has finished the emission and the server begins the CRC calculation. The time period of sending the message by DMA from the server to the client is given by T17-T19, and the handler jitter for the interrupt generated by DMA with the entire message reception from server is given by T20-T21 period.

Testing the SMARTConvert platform regarding the messages exchange between devices belonging to the same Modbus type communication protocol in the worst case scenario (the thermostat has address 1 and the fan coil has address 247, and there are no free addresses between them) involved the following practical considerations. The tAC acquisition cycle for 39 stations was measured and the timing measured with the PicoScope 6404B oscilloscope by Pico Technology (St Neots, UK) are as follows: 4.526 s, 4.527 s, 4.533 s and 4.554 s. Customizing the equations for the project purpose with $\alpha 1 = 1$, $\alpha 2 = 1$, $\mathrm{err1} = 0$, $\mathrm{err2} = 0$ and $\mathrm{tout} = 0$ indicate a time period of 246 \time tAlli(i,FC03,10) + tAll247(247,FC16,10) + 22652.8082 ms for tAC, thus that the practical results obtained validate the presented mathematical model.

ModbusE brings improvements in Modbus message merging, thus reducing communication times and improving data flow on the RS485 network. Thus, the ModbusE project is implemented at the application level and was initially defined in [14]. Here the communication message structure, the BSG as the Client device and the data acquisition cycle were defined. The microcontroller validation of the ModbusE concept is due to the new communication message structure, a deterministic temporal behaviour due to the acquisition cycle, the proposal of Modbus devices in an industrial Internet of things (IIoT) integrated architecture.

VII. CONCLUSION AND FUTURE WORK

This research report based on the Modbus protocol tests the update times of values from a thermostat to a fan coil for the worst-case scenario where the thermostat has address 1 and the fan coil has address 247, and there are no free addresses in between. Measurements were made for the communication speed is 9600 bps, along with Modbus Poll and related time stamps. The customization of the mathematical equations for the purpose of the project based on the proposed mathematical model consisted of two cases, namely AC_S (adr, FCxx, adrReg, nrReg) = {(1,3,7,10), (2,16,7,10)} respectively AC_S (adr, FCxx, addrReg, nrReg) = $\{(1, 3, 5, 10), (2, 3, 5, 10), \ldots, (247, 16, 5, 10)\}$. For both situations, the cases of error, timeout and correctly transmitted Modbus message were considered.

In the ModbusE protocol, only the slot number, data and CRC cyclic redundancy check are transmitted during a slot, thus increasing the bandwidth of the communication channel. ModbusE messages do not have function code fields and function parameters in either the request message or the response message, and the required bandwidth is clearly lower than in Modbus RTU. For the same useful amount of data, the ModbusE message is shorter than the Modbus RTU message. The data structure and length information is either configured offline with simple Modbus configuration tools or done online by BSG. Devices that can integrate the ModbusE concept are those embedded in systems with medium and high response times, industrial data acquisition processes and IoT applications. In this paper, we measured the delays corresponding to the Modbus communication protocol in a PC (client) - STM32F7 (servers) configuration by emulating on the microcontroller several configurations of Server stations, and at the same time sending on Modbus different messages each with a different number of registers.

The performance of ModbusE regarding the acquisition cycle time efficiency is a useful support for IoT gateway (IIoT_MBE_Gateway) implementation. For this, in this paper, mathematical relationships have been defined for determining the jitter within Modbus RTU acquisition cycle slot. By using DMA channels, it led to improved communication channel utilization by parallelizing software operations whose control was provided by DMA. For future work, a deterministic temporal behavior will be obtained for the Industrial IoT smart gateway based on ModbusE. An application instance of BSG, named IIoT_MBE_System, can be successfully used for industrial process control management.

At the same time, the description of the Modbus and ModbusE devices will be carried out, as well as the definition of an architecture for the integration in IIoT.

REFERENCES

[1] R. Zurawski, *The Industrial Communication Technology Handbook*. Boca Raton, FL, USA: CRC Press, 2015.

[2] Z. Li, H. Zhao, J. Shi, Y. Huang, and J. Xiong, "An intelligent fuzzing data generation method based on deep adversarial learning," *IEEE Access*, vol. 7, pp. 49327–49340, 2019, doi: 10.1109/ACCESS.2019.2911121.

[3] J. Qian, X. Du, B. Chen, B. Qu, K. Zeng, and J. Liu, "Cyber-physical integrated intrusion detection scheme in SCADA system of process manufacturing industry," *IEEE Access*, vol. 8, pp. 147471–147481, 2020, doi: 10.1109/ACCESS.2020.3015900.

[4] V. G. Găitan and I. Zagan, "Reţele industriale locale—Modbus extins," Dept. Comput. Sci., Editura Universităţii Ştefan cel Mare din Suceava, Bucharest, Romania, Tech. Rep., 2019.

[5] J. Johnson, B. Fox, K. Kaur, and J. Anandan, "Evaluation of interoperable distributed energy resources to IEEE 1547.1 using SunSpec modbus, IEEE 1815, and IEEE 2030.5," *IEEE Access*, vol. 9, pp. 142129–142146, 2021, doi: 10.1109/ACCESS.2021.3120304.

[6] R. Sánchez-Herrera, M. A. Márquez, and J. M. Andújar, "Easy and secure handling of sensors and actuators as cloud-based service," *IEEE Access*, vol. 8, pp. 10433–10442, 2020, doi: 10.1109/ACCESS.2020.2965639.

[7] L. Rosa, M. Freitas, S. Mazo, E. Monteiro, T. Cruz, and P. Simões, "A comprehensive security analysis of a SCADA protocol: From OSINT to mitigation," *IEEE Access*, vol. 7, pp. 42156–42168, 2019, doi: 10.1109/ACCESS.2019.2906926.

[8] V. G. Găitan and I. Zagan, "Experimental implementation and performance evaluation of an IoT access gateway for the modbus extension," *Sensors*, vol. 21, no. 1, p. 246, Jan. 2021, doi: 10.3390/s21010246.

[9] B.-W. Park, S.-J. Park, and F.-S. Kang, "A novel communication method using PWM and capture function of DSP for parallel controlled power electronics systems," *IEEE Access*, vol. 10, pp. 68266–68280, 2022, doi: 10.1109/ACCESS.2022.3186690.

[10] M. Urbina, A. Astarloa, J. Lázaro, U. Bidarte, I. Villalta, and M. Rodriguez, "Cyber-physical production system gateway based on a programmable SoC platform," *IEEE Access*, vol. 5, p. 20408–20417, 2017, doi: 10.1109/ACCESS.2017.2757048.

[11] J.-R. Jiang and Y.-T. Chen, "Industrial control system anomaly detection and classification based on network traffic," *IEEE Access*, vol. 10, pp. 41874–41888, 2022, doi: 10.1109/ACCESS.2022.3167814.

[12] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 8th ed. London, U.K.: Pearson, 2020.

[13] P. Pleinevaux and J.-D. Decotignie, "Time critical communication networks: Field buses," *IEEE Netw.*, vol. 2, no. 3, pp. 55–63, May 1988.

[14] V.-G. Gaitan, N.-C. Găitan, and I. Ungurean, "A flexible acquisition cycle for incompletely defined fieldbus protocols," *ISA Trans.*, vol. 53, no. 3, pp. 776–786, May 2014, doi: 10.1016/j.isatra.2014.02.006.

[15] *MODBUS Messaging on TCP/IP Implementation Guide*. Accessed: Jul. 2022. [Online]. Available: https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

[16] V. Vyatkin, "IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review," *IEEE Trans. Ind. Inf.*, vol. 7, no. 4, pp. 768–781, Nov. 2011, doi: 10.1109/TII.2011.2166785.

[17] L. H. Yoong, P. S. Roop, V. Vyatkin, and Z. Salcic, "A synchronous approach for IEC 61499 function block implementation," *IEEE Trans. Comput.*, vol. 58, no. 12, pp. 1599–1614, Dec. 2009, doi: 10.1109/TC.2009.128.

[18] P. Tata and V. Vyatkin, "Proposing a novel IEC61499 runtime framework implementing the cyclic execution semantics," in *Proc. 7th IEEE Int. Conf. Ind. Informat. (INDIN)*, Jun. 2009, pp. 416–421, doi: 10.1109/INDIN.2009.5195840.

[19] J. Lastra, A. Lobov, and L. Godinho, "Closed loop control using an IEC 61499 application generator for scan-based controllers," in *Proc. 10th IEEE Conf. Emerg. Technol. Factory Autom. (ETF)*, vol. 1, 2005, pp. 323–330, doi: 10.1109/ETFA.2005.1612541.

[20] V. Vyatkin and J. Chouinard, "On comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations," in *Proc. 6th IEEE Int. Conf. Ind. Informat. (INDIN)*, Jul. 2008, pp. 289–294, doi: 10.1109/INDIN.2008.4618111.

[21] Daniel Flow Products. (Nov. 1992). *Modbus Communications Model 2500, Part Number: 3-9000-545 REVISION D*. [Online]. Available: https://www.emerson.com/documents/automation/daniel-modbus-communications-model-2500-manual-en-43890.pdf

[22] Z. Wang, X. Shen, J. Chen, Y. Song, T. Wang, and Y. Sun, "Real-time performance evaluation of urgent aperiodic messages in FF communication and its improvement," *Comput. Standards Interfaces*, vol. 27, no. 2, pp. 105–115, Jan. 2005, doi: 10.1016/j.csi.2004.05.001.

[23] Y. Wang and V. Gaspes, "A compositional implementation of modbus in protege," in *Proc. 6th IEEE Int. Symp. Ind. Embedded Syst.*, Jun. 2011, pp. 123–131, doi: 10.1109/SIES.2011.5953654.

[24] A. Lemay, J. M. Fernandez, and S. Knight, "A modbus command and control channel," in *Proc. Annu. IEEE Syst. Conf. (SysCon)*, Orlando, FL, USA, Apr. 2016, pp. 1–6, doi: 10.1109/SYSCON.2016.7490631.

[25] G. Cena, M. Cereia, I. Cibrario Bertolotti, and S. Scanzio, "A MODBUS extension for inexpensive distributed embedded systems," in *Proc. IEEE Int. Workshop Factory Commun. Syst.*, Nancy, France, May 2010, pp. 251–260, doi: 10.1109/WFCS.2010.5548625.

[26] R. Al-Dalky, O. Abduljaleel, K. Salah, H. Otrok, and M. Al-Qutayri, "A modbus traffic generator for evaluating the security of SCADA systems," in *Proc. 9th Int. Symp. Commun. Syst., Netw. Digit. Sign (CSNDSP)*, Manchester, U.K., Jul. 2014, pp. 809–814, doi: 10.1109/CSNDSP.2014.6923938.

[27] T. Hu and I. C. Bertolotti, "Overhead and ACK-induced jitter in modbus TCP communication," in *Proc. IEEE 1st Int. Forum Res. Technol. Soc. Ind. Leveraging Better Tomorrow (RTSI)*, Turin, Italy, Sep. 2015, pp. 392–397, doi: 10.1109/RTSI.2015.7325130.

[28] C. Urrea, C. Morales, and R. Muñoz, "Design and implementation of an error detection and correction method compatible with MODBUS-RTU by means of systematic codes," *Measurement*, vol. 91, pp. 266–275, Sep. 2016, doi: 10.1016/j.measurement.2016.05.055.

VASILE GHEORGHIŢĂ GĂITAN (Member, IEEE) received the M.Sc. and Ph.D. degrees from the Gheorghe Asachi Technical University of Iasi, Iasi, Romania, in 1984 and 1997, respectively. He is currently a Professor with the Department of Computers, Stefan cel Mare University of Suceava, Suceava, Romania. His research interests include real-time scheduling, embedded middleware, digital systems design with field-programmable gate arrays, fieldbuses, and embedded system applications.

Mr. Găitan is a member of the IEEE Computer Society.

IONEL ZAGAN (Member, IEEE) received the M.Sc. degree in computer science from the Stefan cel Mare University of Suceava, Suceava, Romania, in 2005, where he is currently pursuing the Ph.D. degree in engineering. He is also a Lecturer with the Department of Computers, Stefan cel Mare University of Suceava. His research interests include real-time systems, field-programmable gate arrays, microcontrollers, and pipeline processors with parallel execution of tasks.

Mr. Zagan is a member of the IEEE Computer Society.