07-007

Study, design and development of a MODBUS master that evaluates the MODBUS communication between equipment

José Simões; Eurico Augusto Rodrigues De Seabra; André Fernandes

Universidade do Minho;

The objective of this work is to study, design and develop a device that allows the user to simulate, evaluate and verify the Modbus communication between the Platform Management System (PMS) of the ship and the main equipment on board. This work will focus in particular on the ship's generators, where the available signals will be assessed. The signals, stipulated in the specification documents, are transmitted directly from the chosen equipment, the incoming signal should then be made available to the reception of the PMS. It is intended that the communication data between the PMS and the generator be simulated by developing a device that serves the purpose of verifying that each communication code corresponds to a sensor. This device is created using an Arduino Uno acting as a Modbus RTU slave and is programmed with a sample sensor/signal list from the generator sensor/signal list

The objective of such equipment will be to fix in advance the possible programming errors and / or assembly errors that otherwise could only be detected at a later stage of the installation, and could therefore lead to increased costs and encompass delays to the final date for implementation.

Keywords: Modbus Master; Modbus Slave; RTU; Serial; Arduino slave; Generator Set

Estudo, projeto e desenvolvimento de um programa que avalia a comunicação modbus entre equipamentos

O objetivo deste trabalho é desenvolver, estudar e projetar um dispositivo que permite simular, avaliar e verificar a comunicação Modbus entre o Sistema de Gestão da Plataforma (PMS) do navio e os principais equipamentos a bordo. Este trabalho incidirá, em particular, os geradores do navio, em que os sinais disponíveis serão avaliados, vindo diretamente do equipamento escolhido, tendo em conta os sinais exigidos pelos documentos de especificação que deve ser posto à disposição dos PMS. Pretende-se que os dados de comunicação entre o PMS e o gerador ser simulado através do desenvolvimento de um dispositivo que serve a finalidade de verificar que cada código de comunicação corresponde a um determinado sensor.

A finalidade deste tipo de equipamento vai ser o de fixar previamente os possíveis erros de programação e / ou a instalação que de outra forma só poderiam ser detetados numa fase posterior da instalação, e poderia, portanto, levar ao aumento dos custos e englobam atrasos à data limite de execução.

Palabras clave: *Modbus Master; Modbus Slave; RTU; Serial; Arduíno escravo; Conjunto gerador*

Correspondencia: José Simões A65378@alunos.uminho.pt

1 Introduction

Data communication is essential in any mechatronic system or product. It allows for devices to communicate with one another, in a more efficient and self-regulating system. To effectively rely on the communication between devices, the communication must firsts be tested and evaluated for errors and to ensure correct data transmission. Sensors and sensor data addresses can easily be switched and when a device is communicating with another, it is imperative that the correct sensor data is transmitted when a slave device receives a signal to do so. If this condition is not verified, the master device could be receiving wrong signals and coincidently sending alerts or warning messages to the user. In more extreme cases the device can enter an emergency shutdown situation and fail to restart.

2 Objectives

The objective of this work is to study, design and develop a device that allows the shipyard to simulate, evaluate and verify the Modbus communication between the Platform Management System (PMS) of the ship and the main equipment on board. The purpose of such program will be to fix in advance the possible programming errors and/or assembly errors that otherwise could only be detected at a later stage of the installation, and could, therefore, lead to increased costs and encompass delays toward the final date for implementation. The program will also be used to monitor equipment operation while testing and to record sensor data to later send to the equipment's manufacturer.

3 State of the Art

Currently, there are numerous Modbus master programs available on the market, both paid and free/open source. Although, none of them completely deliver on the functionality and available features expected for use by the Shipyard. Some available programs include:

- Modbus Poll (modbus tools, 2016)
- Simply Modbus Master (Simply Modbus, 2015)
- ICC Modbus Master Tool (Industrial Control Communications, Inc, 2016)
- QModMaster (elbar, 2016)
- Modscan32 (Continental Control Systems)

Of these, Modbus Poll is the most used, it is a paid program and it allows the user to log the read data and to define the display formats for each coil or register. The downside to this program is that the user must define the register and coil addresses and associated data variables each time a session is started. (modbus tools, 2016) Qmodmaster is one of the most used free Modbus master programs and as such it is quite limited, nonetheless, it is still used to test and debug slave communication through the serial monitor. The shipyard has used this program to test equipment, but because of the tedious and time-consuming task of testing each coil or register one by one. As well as constantly having to convert the received values to the correct data format, the usage was discouraged and only applied when necessary.

4 Contextualization

In this section, the background information for the project is presented. The Modbus protocol is briefly explained to allow correct understanding of how the protocol. The LabVIEW© model is also

explained as well as ways to facilitate the establishment of the Modbus connection and how to create a session from a low-level programming.

4.1 Modbus

Modbus is an industrial protocol that was developed in 1979 by Modicon, Incorporated, to make communication possible between industrial automation systems and Modicon programmable controllers. (Cyburt, n.d.) Originally implemented as an application-level protocol intended to transfer data over a serial layer, Modbus has expanded to include implementations over serial, TCP/IP, and the user datagram protocol (UDP). (National Instruments, 2014) Thus making it a standard communications protocol for electronic devices. (McCrohan, 2011)

The MODBUS protocol follows a client/server (master/slave) architecture where a client requests data from a server. The client can also send data to the server. The client initiates a process by sending a function code that denotes the type of operation to execute. The operation performed by the MODBUS protocol defines the process a controller uses to request access to another device, how it will respond to requests from other devices, and how errors will be detected and reported. The MODBUS protocol establishes a common format for the layout and contents of message fields (MODBUS, 2012) (Modicon, 1996) (Real Time Automation, n.d.).

The Modbus-accessible data is kept, in general, in one of four data banks or address ranges in the slave device (Minrui Fei, 2014). These four data banks are holding registers, input registers discrete inputs and coils. (Hsiang-Chuan Liu, 2013) (James J. (Jong Hyuk) Park, 2015) The behavior of each block is described in ,

Table 1 (National Instruments, 2013),

Table 1: Modbus Data Model Blocks

Memory Block	Data Type	Master Access	Slave Access
Coils	Boolean	Read/Write	Read/Write
Discrete Inputs	Boolean	Read-only	Read/Write
Holding Registers	Unsigned Word	Read/Write	Read/Write
Input Registers	Unsigned Word	Read-only	Read/Write

The Modbus protocol describes each block as encompassing “an address space of as many as 65,536 (2^{16}) elements.” Modbus describes the address of each data element ranging from 0 to 65,535. Conversely, each data element is numbered starting at 1 to 65,536. (National Instruments, 2014) In the Modbus protocol, it is directly stated that “The quantity of registers to be read, combined with all the other fields in the expected response, must not exceed the allowable length of a Modbus messages of 256 bytes." (Courvelle, 2002)

4.2 LabVIEW

LabVIEW© is an integrated development environment designed specifically for engineers and scientists. The core of LabVIEW© is its graphical programming language that uses a dataflow model instead of sequential lines of text code, thus empowering the user to write functional code

using a visual layout that resembles the thought process. Meaning that the user spends less time worrying about semicolons and syntax and more time completing the program. (National Instruments, n.d.)

With LabVIEW© there are 4 possible ways to create the Modbus RTU instance, these are listed below:

- Modbus Library “Modbus Library”;
- Modbus Library “Modbus”;
- Modbus I/O Server;
- VISA Communications.

The first two items are Modbus libraries and are fairly similar to one another. These are not used because they limit the possible number of stop bits to 2. Thus, also limiting the possible slaves for connection to the slaves that are configured to have 2 stop bits in the communication packet. The next possible choice is the I/O server, this solution was not used in the project because the I/O server is ideal when a Modbus master communicates with a static group of slaves. In the case of this project, the master is to communicate with a dynamic group of slaves where a coil/slave address is not always used and the same coil/register can be used as a different signal in various slaves. The last solution was to create the Modbus RTU instance using low-level programming. The low-level serial communication programming is performed with the LabVIEW® VISA serial communications pallet located in the “Data Communications” pallet in the toolbar, in the “Protocols” section, under the “Serial” icon.

5 Modbus Master Program

NAVAL MB Master is the name of the program created for this work. Initially, the program was planned where all the available functions and program functionalities were discussed. The program was then created in the LabVIEW© platform and more functions were added as the individual modules or sub-VIs were tested and debugged. The program was exposed to rigorous testing to verify that all functions were operating as previously planned.

5.1 Brainstorming Program Functions

When deciding on program functionalities, a meeting was held with the West SEA Shipyard engineers where various ideas were laid out and the program functionalities were decided upon. The defined functions that should be available to the user include:

- Connect/disconnect from slave;
- Perform read, write and continuous read(scan) commands;
- Erase read data;
- Visualize # of packages sent and # of error messages;
- Reset package and error counters;
- Monitor COM transmissions;
- View history of COM transmissions;
- Change Modbus and program settings;
- Access Modbus documents, program info, and exit program;
- Save scanned values to text file;
- Graph at least 10 Coils/Registers in real time;
- Graph at least 10 Coils/Registers from saved file.

While programming and debugging the different modules that make up NAVAL MB Master, it came to the attention that various functions or settings should also be implemented in the program. These functions or settings include:

- Limit package size with slave buffer size in mind;
- Create delay between TX and RX transmissions;
- Allow for later development of ASCII and TCP/IP transmissions;
- Save setting for next session startup;
- Not allow user to change settings when a connection is established;
- read coils/registers from a text file that will be read or written to (Auto Mode);
- Allow the user to manually insert which coils/registers to read/write to;
- In auto mode, allow user to choose between reading coils/registers in groups or reading each one separately;
- Allow user to choose data formatting in auto mode, and allow user to define the data type for the read data;
- Depending on the function code, limit which events/buttons are available to use;
- Display any Modbus error message to the user.

5.2 Programming

The NAVAL MB Master program was fully programmed in LabVIEW®, some components like manufacturer Excel® sheets to be imported to the program were programmed using VBA to create Excel Macro routines.

5.2.1 LabVIEW

The LabVIEW® programming platform was used to program and debug the NAVAL MB Master program. The program was used to create the Modbus communication protocol from a low level and to create subprograms that could be easily implemented into the main program (National Instruments, 2009).

The fundamental component of the NAVAL MB Master program is the communication component that sends a query to a slave device and then reads the slave device's buffer for the answer. This component was the most time consuming to program and debug because the communication had to correspond perfectly to the Modbus communication protocol, otherwise, the query would not be correctly sent to the slave and the slave, in turn, would not respond due to the incorrect message. The core communication flow can be viewed below, where Figure 1 shows the front panel of the code and Figure 2 shows the block diagram of the communication flow.

The code contains only two input for simplification, the VISA resource name which is where the COM port is chosen and the TX data which is where the query message needs to be typed in hexadecimal to be directly sent. Other inputs that need to be connected to the "VISA configure serial port" block include the communication baud rate, the number of bits in the message, the parity bit, the number of stop bits and other settings that pertain to the port configuration. The "VISA Write" block is connected to the port configuration block and is also connected to the TX data control from the front panel and sends Tx Data message to the slave. Next, having processed the wait time, the program proceeds to execute the "Bytes at Port" block which finds out how many bytes are to be read from the buffer. This value is then connected to the "VISA Read" block, which reads all filled slave buffer bytes and displays them as HEX in the RX data indicator. The "RX data in bytes" indicator gives the read value in bytes. The code then proceeds to close the communication port by using the "VISA Close" block and displays any errors using the "error out" block.

Figure 1: Front panel input and output data

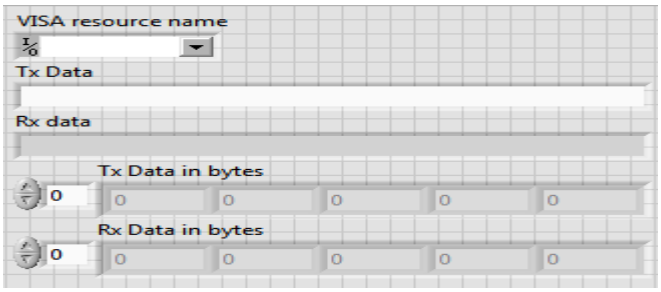
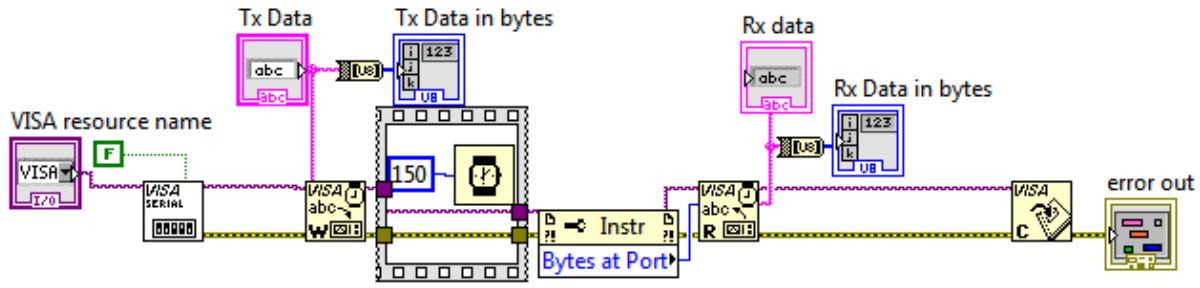


Figure 2: Core communication blocks in program



SubVIs are small functions or programs created separately that allow for code simplification and allow the programmer to use a block to represent repetitive code. A Sub VI can also be used to divide a program into smaller modules that make up the program when connected. All blocks listed below are SubVIs created to simplify the program coding and to better organize the code for later comprehension:

- Analyze Live Data: that allows for a better visualization of the coil and register scanned data on gauges and graphs. The block also allows for saving of the read data for later visualization;
- Analyze Logged Data: allows for data visualization of a saved text file from the “Analyze Live Data” block;
- Max Address: calculator program that calculates the maximum number of addresses that can be read or written to;
- Serial Settings [Updater]: used to update the serial settings changed by the user;
- Slave Settings [Updater]: is used to update the slave settings changed by the user;
- The “Open Modbus Manual” block is programmed to open a Modbus PDF file included in the program installation using the installed PDF viewer on the computer and open the Modbus website using the installed computer web browser;
- Select [Slave] Manufacturer: opens a dialog window where the user selects the manufacturer and manufacturer text file. The program proceeds by dissecting all the information from the text file and organizes it into columns to be easily used later in the program;
- Separate into Register Types: inputs the register and coil array created in the “Select Manufacturer” block and separates the register and coil addresses by memory type, coils, discrete inputs, input registers and holding registers respectfully. This separation into sub-

arrays allows for a better program flow since the Modbus functions are also separated by address memory type;
- Modbus RTU Communication Block: is the SubVI that controls the communication to and from the slave device. A simplified version of the code was displayed in Figure 2;
- Automatic Data Treatment: it formats the received data from the slave when executing the program in automatic mode;
- Manual Data [Treatment]: it formats the received data from the slave when executing the program in manual mode. Excel Macro.

When importing manufacturer data to the program, a uniform data formatting needed to be established so the program consistently knows what data is in each column of the manufacturer data. A detailed data analysis was executed to find out the most relevant data to be used from the manufacturer's document. The most necessary data and settings from the manufacturer document are as follows:

- Coil/Register start address;
- # of coils/registers to read/write to;
- Bit;
- Sensor/Signal Name;
- Unit of measurement;
- Data type;
- Scale;
- Minimum Value;
- Maximum Value.

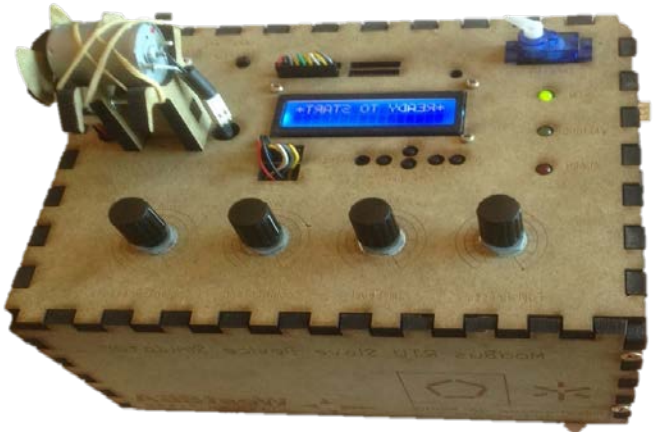
Next, various routines were created using Excel® Macro which would treat each manufacturer data document to output only the required fields listed above for later importing into the program. An additional macro was created to save any other manufacturer data. The Excel® workbook is named "Other", this workbook allows for the manual input by the user of all the necessary fields. Since the user input the data directly in the proper format, the macro only saves the file as a comma separated text file. When the user has completed the data input, the macro must be run by clicking the "Format to Text File" button, which will proceed to save the document as a comma separated text file.

5.3 Testing

To test the NAVAL MB Master program, a Modbus RTU Slave Device Simulator, as shown in Figure 3, was used. The shipyard did not possess any onboard equipment yet, so a simulator needed to be used for the test. The slave device simulator was constructed using an Arduino Uno (Arduino, n.d.) running a slave Modbus library (Arduino, n.d.), and was programmed to function similarly to the generator being implemented onboard the ship (Kunal Yogeshkumar Parikh, 2016), (Peng, Zhang, Yang, & Li, 2008). The simulator was composed of several potentiometers to simulate liquid levels and temperatures, a dc motor and encoder to simulate generator rotation and speed readings and various push buttons to start and stop the "generator" (electronhacks, 2014).

Both program operating modes had to be tested (auto and manual mode) as well as all other implemented program functions. All Modbus function codes needed to be tested for correct transmission and execution as each were programmed from low level programming and therefor prone to programming errors. Error messages also needed testing so that it can auto diagnose connection problems and display each to the user.

Figure 3: Modbus RTU slave generator simulator



6 Conclusions

This project involved creating a program to test equipment communication over Modbus before installing it onboard the ships. The issue was that there was no facilitated way to test the equipment's communication before it was installed onboard the ship and connected to the ship's remote monitoring system. Therefore, verifying the communication and sensor addresses before terminating the equipment's warranty was discussed along with the major objectives of the project.

To proceed with the project, first, the Modbus protocol had to be well understood so that when replicating, the communication packet's errors would not be sent. The protocol explained what each component of the packet represented and how it is assembled. LabVIEW® libraries were researched and tested but did not function with slaves that used 1 stop bit in the communication packet. As a last alternative, the LabVIEW® VISA pallet was used so that communication could be established with a slave device using low-level programming blocks.

From the start, the company stated that equipment to test the program would not be available to test the master program, therefore, a slave simulator needed to be created. The device was created with the purpose of testing the Naval MB Master program and was created to simulate the functionality of various sensors and actuators available on a generator set to be installed on the ships.

Initially, when first starting the program, the program functions went through a brainstorming session where the best and most important function were selected to be implemented into the program. The program was created in the LabVIEW® platform using the VISA pallet functions to establish the low-level serial communication. The program was created in modules which were individually tested and later interconnected to form the final program. Various Excel® macros also needed to be programmed to create a uniform manufacturer sheet that the program could easily import. Finally, the program was completely tested with the Arduino© generator simulator module and all functions responded as expected.

6.1 Program Outcome

The program when tested and presented to the shipyard was a success. All the functions and functionalities implemented into the program were working as expected. According to Alexandre, a West SEA electrical engineer, "The demonstration shown, seemed to display that the program had the potential to meet our expectations, but real field tests could not be completed due to the

unavailability of the equipment." The program can potentially communicate to other devices using CAN bus protocol or Profibus using protocol converters. (Umesh Goyal, 2013)

It was a very gratifying and rewarding experience to be able to communicate with a "homemade" Modbus slave device over the Modbus protocol and have information flow in both directions. This idea could be expanded to control everyday appliances at home, a home security system, backyard sprinklers or even a greenhouse. For better integration, the user should also create the Modbus master device in a way to allow it to autonomously control all the slaves connected to it.

6.2 Faced Problems

Like in all projects, various difficulties were encountered. Some of the encountered problems include:

- Learning process of Modbus protocol;
- Find a working Modbus library to use in the LabVIEW© program;
- Writing data in text input instead of hexadecimal input in VISA write block.

Early on in the project, the biggest difficulty was getting to know the Modbus communication protocol. Various manufacturer manuals provided by the shipyard engineers along with the Modbus website assisted in theoretically understanding the protocol. Building the Modbus slave device along with testing and debugging using the QModMaster program allowed to generate a practical understanding of the protocol and to solidify the previously theoretical understanding.

Finding a Modbus library in the LabVIEW© platform also proved to be a challenge. When initially searching for the libraries, various Modbus libraries were easily found that had the potentiality to be used. After trying to connect it to the Arduino© slave device without success, it was perceived that the libraries did not support a 1-stop bit transmission, only a 2-stop bit. Most slaves transmit using a 1-stop bit transmission since the libraries do not contain any texts to explain how to change the setting, an alternative had to be used. The alternative was to create the communication using low-level settings to open the port with the correct slave settings that allow for a flawless transmission.

7 References

Arduino. (n.d.). *Arduino - ArduinoBoardUno*. (Arduino) Retrieved from https://www.arduino.cc/en/main/arduinoBoardUno

Arduino. (n.d.). *Libraries*. (Arduino) Retrieved from https://www.arduino.cc/en/Reference/Libraries

Continental Control Systems. (n.d.). *Modbus Software.* Retrieved from Continental Control Systems: https://ctlsys.com/support/modbus_software/

Courvelle, M. (2002, 11 10). *Maximum Amount of Holding Registers per request*. (Control.com) Retrieved from http://control.com/thread/1026161502#1026161502

Cyburt, B. (n.d.). *Introduction to Modbus*. (Acromag, Inc) Retrieved from http://www.automation.com/library/articles-white-papers/fieldbus-serial-bus-io-networks/introduction-to-modbus

elbar. (2016). QModMaster. Sourceforge. Retrieved from https://sourceforge.net/projects/qmodmaster/files/?source=navbar

electronhacks. (2014). Arduino Modbus PLC / RTU. Electron Hacks. Retrieved from http://www.electronhacks.com/2014/04/arduino-modbus-plc-rtu/

Hsiang-Chuan Liu, W.-P. S. (2013). *Information Technology and Computer Application Engineering.* New York: CRC Press.

Industrial Control Communications, Inc. (2016). *ICC Modbus Master Tool.* Middleton, WI : Industrial Control Communications, Inc.

James J. (Jong Hyuk) Park, I. S. (2015). *Computer Science and its Applications.* Springer.

Kunal Yogeshkumar Parikh, M. J. (2016). MODBUS Protocol for Reading Parameter of AC Drive. *International Journal on Recent and Innovation Trends in Computing and Communication* , 185-187.

McCrohan, J. (2011). Arduino Modbus RTU ADC. Dublin, Ireland: dereenigne.org. Retrieved from http://dereenigne.org/arduino/arduino-modbus-rtu-adc

Minrui Fei, C. P. (2014). *Computational Intelligence, Networked Systems and Their Applications.* Springer.

MODBUS. (2012). Modbus over Serial Line Specification & Implementation guide. MODBUS.

modbus tools. (2016). *Modbus Poll*. Retrieved from modbus tools: http://www.modbustools.com/modbus_poll.html

Modicon. (1996, June). *Modbus Protocol Reference Guide.* North Andover, Massachusetts.

National Instruments. (2009). *Introduction to MODBUS.*

National Instruments. (2013). Serial Quick Reference Guide. National Instruments.

National Instruments. (2014, 08 01). *The Modbus Protocol In-Depth - National Instruments.*

National Instruments. (n.d.). *LabVIEW System Design Software*. (National Instruments) Retrieved from http://www.ni.com/labview/

Peng, D.-g., Zhang, H., Yang, L., & Li, H. (2008). Design and Realization of Modbus Protocol Based on Embedded Linux System. *International Conference on Embedded Software and Systems Symposia, 2008. ICESS Symposia '08* .

Real Time Automation. (n.d.). *Modbus RTU Protocol Overview*. (Real Time Automation) Retrieved from http://www.rtaautomation.com/technologies/modbus-rtu/

Simply Modbus. (2015). *Simply Modbus Master 6.4.1.* Simply Modbus.

Umesh Goyal, G. K. (2013). Implementing MOD bus and CAN bus Protocol. *International Journal of Engineering Trends and Technology (IJETT)*.