Industrial Automation Debug Message Display Over Modbus RTU Using C#

Sudip Chakraborty [1] & P. S. Aithal [2]

[1] D.Sc. Researcher, Institute of Computer Science and Information Sciences, Srinivas University, Mangalore-575 001, India,
OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in

[2] Vice Chancellor, Srinivas University, Mangalore, India,
OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

Area/Section: Computer Science.
Type of the Paper: Experimental Research.
Type of Review: Peer Reviewed as per |C|O|P|E| guidance.
Indexed in: OpenAIRE.
DOI: https://doi.org/10.5281/zenodo.8139709
Google Scholar Citation: IJMTS

How to Cite this Paper:
Chakraborty, S., & Aithal, P. S. (2023). Industrial Automation Debug Message Display Over Modbus RTU Using C#. *International Journal of Management, Technology, and Social Sciences (IJMTS), 8*(2), 305-313. DOI: https://doi.org/10.5281/zenodo.8139709

International Journal of Management, Technology, and Social Sciences (IJMTS)
A Refereed International Journal of Srinivas University, India.

CrossRef DOI: https://doi.org/10.47992/IJMTS.2581.6012.0285

Received on: 20/06/2023
Published on: 30/06/2023

Industrial Automation Debug Message Display Over Modbus RTU Using C#

Sudip Chakraborty [1] & P. S. Aithal [2]
[1] D.Sc. Researcher, Institute of Computer Science and Information Sciences, Srinivas University, Mangalore-575 001, India,
OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in
[2] Vice Chancellor, Srinivas University, Mangalore, India,
OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

ABSTRACT

Purpose: To debug any device needs to display debug message. Seeing the message, we can detect what is the issue in our program. The are several popular debug tools available. The popular is JTAG. This debug interface is not available to debug inside all embedded systems. Instead of it, the Researcher uses a serial debug terminal. However, it has several drawbacks. Its wire length is generally within the table. The message is displayed only in one terminal. Sometimes the connected system is damaged due to high voltage spike injection. Here we demonstrate a procedure to eliminate all the above drawbacks. Instead of a direct serial display, we recommend using a modified serial debugger like Modbus hooked debug display. It is safe, can display multiple arrays simultaneously using broadcast messages, and can transmit long-range data. We developed a terminal program in C#. We provide a sample Embedded side test program. The complete project code is available to download.
Design/Methodology/Approach: First, we create a Graphical user interface using C# language. To display the message, we use the "Listbox" control. We added a timer control to fetch the data availability. The timer interval is one millisecond by default. Within this time, it checks whether any data is reached. When the message/data arrives, it starts parsing. At first, it reads the Device ID. If the device ID matches, then it checks the CRC. If CRC is valid, it extracts the data from the received packets and displays it inside the list box.
Findings/Result: Various tools are needed to resolve the issue when we debug the system. We connect the debugger and try to uncover the problem. Sometimes we do less concentrate on the hardware connection. So, there are chances of a wrong connection. Further, it can lead to the cause for damage to our PC/laptop. Here is our approach to minimize the damage which is unintentionally created. We isolate the PC/Laptop from the debugging system. It is safe and reliable. The researchers, new to embedded system debugging, can get information on where they can safely debug by creating an isolated environment.
Originality/Value: This work provides some efficient view to debug message display using Modbus. It has several advantages over a simple serial debug interface. It can be long-range. The same message can be displayed in multiple terminals for team debugs season. The Researcher working with various team members on the same project can be viewed effectively. The new Researcher can be found some good reference information from this work.
Paper Type: Experimental-based Research.
Keywords: Modbus Debugger, Modbus Debug terminal, Debug terminal over Modbus.

1. INTRODUCTION :

The Debug is an essential part of any electronics product life cycle. We mainly debug at product development time. For microcontroller-based products, we generally debug with JTAG. Furthermore, we use the serial debugger for Arduino or where JTAG is unavailable. However, using a serial debugger, a couple of issues are there. Usually, serial debuggers are connected directly to hardware. One side is connected to a USB port, and the other is associated with a controller serial port. The connected laptop or system might be burnt instantly if the controller board is short with high-voltage, like 230V AC. According to our experience, some cases have already happened where we could easily protect our

laptop or PC using some procedure. Here is the cost incurred for the PC/ or system unit, and if we are not backed up properly, all our hard work might be irrecoverable. Another issue we face for serial debuggers is that the wire might not be lengthy. Sometimes we need to display the debug message from a remote place, creating a long wire, and the signal gets noisy. The third one is multiple displays. Let us say we are working with the team. The debug message needs to display on multiple terminals. Simply, it is not possible to display the debug message. We can use a modified serial debugger and RS485 based debug message displays for those drawbacks.

Hazard Injection: To eliminate this type of issue, we use isolated RS485. Several vendors, like Texas Instruments, are manufacturing the isolated RS485 chip. It has galvanic isolation. There is no physical wired connection between the input and output. Sometimes we isolate signal flow but ground pin we still made common. It is also can create issues. So we need complete isolation for full protection. So using this chip, we can protect ourselves entirely using high voltage hazards.

Long Distance: RS485 instead of RS232 can be achieved at a maximum distance of 1200 meters. It has some facilities. This signal can be directly fed to a datalogger and remote monitoring purposes.

Multicast: Rs485 signal multicast in nature. Our debug message display can be viewed by at least 32 nodes that can be connected at a time. That means 32 nodes can be seen in the debug message. If we want to activate multiple devices triggering from the broadcast signal, we can use this type of signal.

2. RELATED WORKS :

Bingol O. et al. paper focuses on implementing a web-based intelligent home automation system controlled by a programmable logic controller (PLC). It highlights the advantages of web-based control in home automation [1]. Yunzhou, Z. et al. present a design for a training and experimental platform that utilizes multi-protocol communication [2]. Bajer, M., in their paper, explores the dataflow concept in modern industrial automation systems [3]. Pfrang S. et al. emphasize advancing protocol fuzzing techniques for industrial automation and control systems 4]. Wang, H. et. al. introduces the development of a three-dimensional virtual programmable logic controller (PLC) experiment model based on Unity3D [5]. Familiar, B., in their paper, focuses on sensors, devices, and gateways in IoT and real-time business applications [6]. Mathias, C. M. focuses on the design of IoT security, specifically in the context of smart grids. It analyzes software vulnerabilities that pose security risks in innovative grid systems and proposes mitigation measures [7]. Del Carmen Curras-Francos et. al., in their paper, focuses on the cooperative development of an Arduino-compatible building automation system for practical teaching purposes [8]. Müller, M. introduces an intelligent assistance system for controlling wind-assisted ship propulsion systems. It presents the system's design and implementation, combining sensor data, control algorithms, and decision-making mechanisms to optimize performance [9]. Aguilar et. al. focuses on creating general-purpose automation software based on the Raspberry Pi platform [10]. Overall, the literature study provides various research papers covering intelligent home automation, industrial communication protocols, virtual experimentation platforms, IoT security, building automation, intelligent assistance systems, and general-purpose automation software. These papers contribute to the knowledge in the automation field and provide valuable insights into the design, implementation, and optimization of automation systems in different domains.

3. OBJECTIVES :

This work aims to provide the information to the Researcher to have the updated debugger to get better protection for the system debugging. Here we give the block diagram, procedure, and ready-to-use code to the Researcher so they can use it instantly in their research. We demonstrate the drawbacks of the technologies they are using now and provide if they update the modified system. Sometimes damages happen due to short circuits with high voltage. We can eliminate using this modified process.

4. APPROACH AND METHODOLOGY :

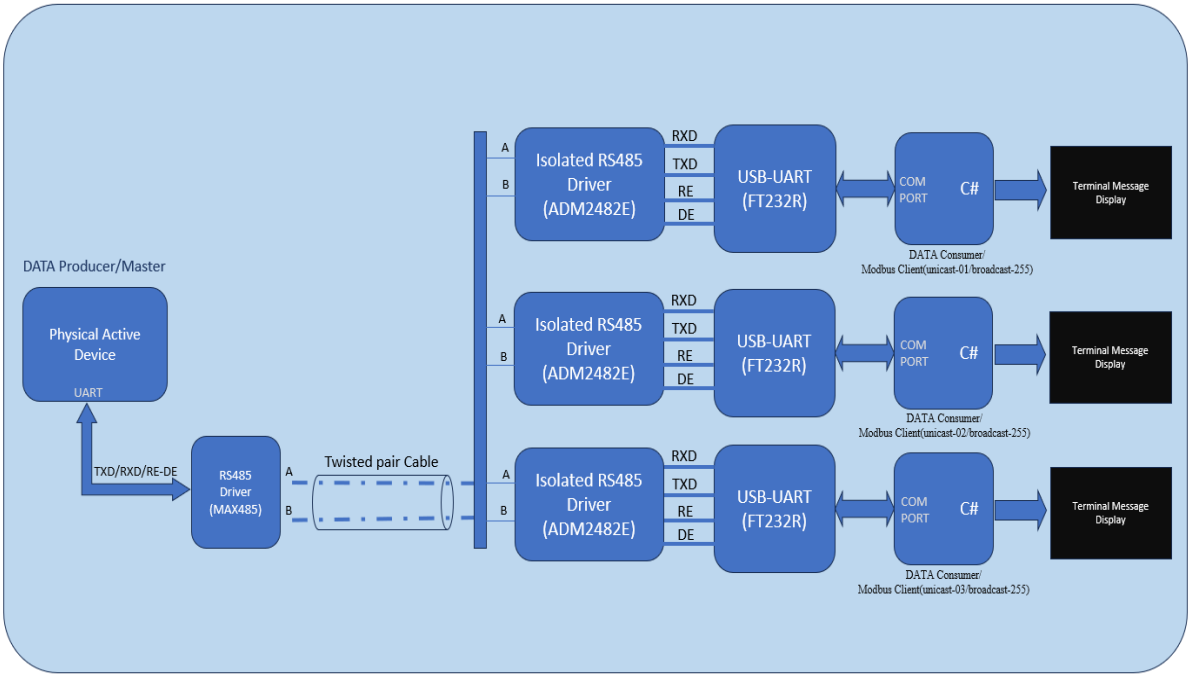

Fig. 1: Block diagram of the research work [Source: Author's]

Figure 1 depicts the block diagram of our research work. We will see what exactly is happening inside the block diagram.

(1) **Controller**: On the left side, there is the Modbus data producer. The primary source of the data is the physical device. It is a microcontroller device. Mainly it should be one UART port to send and receive the data. The device must display an understandable message to make debugging easy. Generally, the RS485 driver is not available inside the microcontroller. So we need to connect externally with the serial port. With the drive, RXD, TXD, and RE-DE signals are coupled between the controller and the driver.

(2) **RS485 driver**: the RS485 driver converts the signal from UART to differential pair voltage. It has several advantages. It carries long distances, and it is noise immune also. In the industrial environment, there are lots of noise. Through the noise, we need to flow the signal without interruption. So it is the best way to pass the signal using different voltage pair. Other signal-carrying variants are also available, like ethernet, fiber optic, etc., but the cheapest way over long-distance channels is RS485. In the RS 485 bus, we can connect around 32 devices. In some scenarios, we need to adjust the impedance matching, that is, signal pair balancing using two resistors (like 120E) for better performance.

(3) **Data Consumer/client**: The Modbus client receives the data and responds to the master. The master initiates all communication. Master sends the command and response back from the specific client which is addressed for. For broadcast messages, no clients are answered. When the master wants to send the display message, it must send the data with the target address. The broadcast address can be any address. The client firmware must be hardcoded for different clients to assign the device Id.

(4) **Isolation**: Here, The A and B signals are fed to the isolated RS485 chip. It isolates from high voltage spikes, so our connected system, like a laptop, does not affect any unwanted spikes. After that, we use RS485 to UART converter chip. The popular and reliable chip is FT232RL. This chip acts on two things. One converts RS485 to UART and the next convert UART to USB. This chip manages all USB enumeration and communication states. When we connect to the system, the operating system (OS) exchange a couple of USB token. The token is nothing but data with a specific standard format. The OS creates a COM port like COM5 or COM3.

(5) **Application(C#)**: Now, applications like C# open the port first using the assigned COM port number. Once it is successfully opened, it can send and receive the data. The application is based on the graphical user interface (GUI). We have taken a list box to view the message and a couple of buttons to control the port, like connect/disconnect Button. When the COM object

receives a message, it is received by an interrupt vector and sets the received flag. The main application runs a timer with a one-millisecond interval. It continuously checks whether the data is present or not. Once data is obtained, it starts to parse. If the client iD matches, extract the data and push the data to the Listbox.

```
2  ////////////////////////////////////////////////////////////////
3  void msg(unsigned char *bfr)
4  {
5      unsigned char char_count;
6      unsigned char temp[100];
7
8      char_count=0;
9      do{
10        temp[char_count]=*bfr;
11        char_count++;
12        bfr++;
13     }while(*bfr!=0);
14
15     Modbus_Multiple_Register_Write(200,0, &temp[0],char_count, bfr);
16 }
17 ////////////////////////////////////////////////////////////////
```

Fig. 2: Embedded Side Code: message function [Source: Author's]

(6) **Firmware**: Figure 2 depicts the sample code for Modbus multiple registers write, which function code is 16 in decimal and 0x10 in hex. It is a simplistic approach. It can be a more complex structure. It uses to send our debug message to the remote client; in our code, where we need to view some message, we call the "**msg**" function and pass the message as an argument. Figure 3 shows the example of a Modbus write multiple register function. According to Modbus standards, there are some data formats to maintain the standard Modbus protocol. After receiving the message, this function creates the Modbus packet. We make a temporary register array called "temp" and form the packet inside. The first byte is the device id to whom the data will receive. If it is for all, we need to set the broadcast address. Else, set the unicast or the specific client id. The next one is function code 16 in decimal or 0x10 in hex. The next two registers are the first address from which data will start writing.

(7) **Address**: We deal with 16-bit addresses, so we must split into the high and the low. The next parts are 16-bit data. Inherently Modbus is a 16-bit register format. It is popular. So the following two registers split into high and low bytes for the quantity of the int16 register. The next position is the $6^{th,}$ the number of bytes we send. Next, we insert the data we want to transmit: the payload. It will be displayed on the other end. After the data fillup, calculate the checksum, split it into two bytes, and insert it at the last position of the array. Once packet formation is complete, it sends through the UART engine.

```
//_____________________________________________________________________
void Modbus_Multiple_Register_Write(unsigned char device_id,unsigned int addr, unsigned char *data,unsigned char count)
{
    long TimeOut;
    unsigned int crc;
    unsigned char i;
    unsigned char temp[0xff];
    unsigned int reg_qty;
    unsigned char pkt_len;

    temp[0]=device_id;   // slave address
    temp[1]=0x10;        // function code

    temp[3]=(unsigned char)(0x00ff & addr);       // addr Lo
    addr=addr>>8;
    temp[2]=addr;        // Addr Hi

    reg_qty=count/2;
    temp[5]=(unsigned char)(0x00ff & reg_qty);       // quantity of register(int16) lo
    reg_qty=reg_qty>>8;
    temp[4]=(unsigned char)reg_qty;                  // quantity of register(int16) Hi

    temp[6]=count;                                   // byte count
    for(pkt_len=7;pkt_len<(7+count);pkt_len++)
    {
        i=*data;
        temp[pkt_len]=i;
        data++;
    }
    crc=Get_CRC(&temp[0],pkt_len);
    i=(unsigned char)(0x00ff & crc);        // CRC Lo
    temp[pkt_len]=i;
    crc=crc>>8;
    pkt_len++;
    temp[pkt_len]=(unsigned char)crc;        // CRC Hi

    pkt_len++;
    temp[pkt_len]=0x00;

    SCIB_Purge();
    scib_send(&temp[0],pkt_len);
}
//_____________________________________________________________________
```

Fig. 3: Embedded Side Code: Modbus multiple register write function [Source: Author's]

5. EXPERIMENT :

Now we can do some experiments to see what is happening. We follow the below steps:

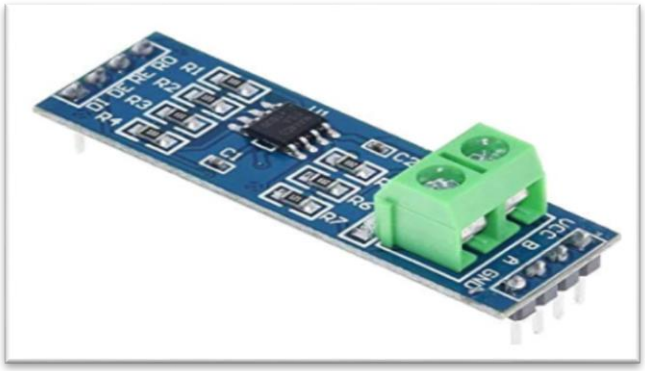

Fig. 4: TTL To RS485 Module [Source: https://www.amazon.in]

(1) We need two types of modules. One is TTL to RS485 module depicted in Figure 4. It will enable our microcontroller device into RS485. The module is available online:- https://www.amazon.in/Converter-Adapter-Raspberry-Integrated-Circuits/dp/B08243L4VX/ref=sr_1_3?crid=2S39UXX86AK3A&keywords=rs485+module&qid=1688284413&sprefix=rs485+modul%2Caps%2C261&sr=8-3

Fig. 5: USB to RS232 / RS485 / TTL Industrial Isolated Converter [Source: https://robu.in/]

(2) The other module is USB to the RSR85, depicted in Figure 5 it is available online https://robu.in/product/waveshare-usb-to-rs232-rs485-ttl-industrial-isolated-converter/
(3) Take two modules. Take two wires. Connect one module's A with another module's A—one Module's B with another module's B.
(4) Connect the USB to the working system using a USB cable.
(5) Take the TTL-RS485 module. The module's RX pin will connect with the controller TX, and the module's TX will connect with the controller RX pin. Make RE and DE pin short and connect with one controller-free GPIO. In the program, we must change the state when we receive or transmit the signal. Check the chip datasheet https://www.analog.com/media/en/technical-documentation/data-sheets/MAX1487-MAX491.pdf. Now our hardware is ready to test. Let us do some software work.
(6) Inside the existing firmware, we need to add the function "write multiple holding registers" like Figure 3 and call the function like Figure 2. the embedded code is available in the project download folder. The provided code is for the C2000 controller. It may change for other controllers, but the concept is the same.
(7) For Modbus protocol details, https://www.modbustools.com/modbus.html#lrc and https://camatsystem.com/wp-content/uploads/2015/12/Modbus-manual-TD80.pdf
(8) Build and upload the firmware. Our firmware is ready now. Let us make ready the PC application.
(9) Download and install visual studio from the Microsoft website.
(10) Download the project from the repository. https://github.com/sudipchakraborty/Industrial-Automation-Debug-Message-Display-Over-Modbus-Using-C-sharp-And-CoppeliaSim-.git
(11) Under the C# folder, open the "**Modbus_Debug_Terminal.sln**" solution file. It will open the project. Build and run the project.
(12) Connect converter module. The OS will assign a port like "COM1" or others. These port numbers need to add apps PORT text box.
(13) Now press connect Button. If the connection is okay, the button color turns green.
(14) Now run the firmware. We will see the message is displaying. If it does not happen as described, be patient and debug one by one according to the process flow.

Now we see various control on the GUI depicted in Figure 6.

(1) **Message**: under this group box, the incoming message is displayed.
(2) **Receive count**: A textbox on the top right displays the packet receipt count. When new receive count will added. It will show the total number of valid received data counts.
(3) **PORT**: The PORT textbox is the assigned COM port of the operating system after connecting the USB-RS485 module. It will assign automatically. For the Windows system, On the start menu, click "Device Manager." Under the port, the port number is available. It needs to be input into the application port textbox.
(4) **Baud Rate**: It is the speed of communication. By default, the port speed is 9600. According to the application, we can change the baud rate. The industrial standard baud rate is 9600, and 11.5 KBPS is also acceptable for communication,
(5) **Connect/Disconnect Button**: the next Button is "**connect**," which is used to communicate with the device. Pressing the Button opens a COM port and is ready to communicate with that device.

The next Button is the "**disconnect**" Button. This Button is used to disconnect with the system; actually, it will disconnect that COM port.

(6) **Device ID**: This is the field of the Modbus device ID. By default, we set 200. It can be changed anytime. Set this ID before sending any command to the target devices.

(7) **Top checkbox**: This checkbox displays the incoming message at the top. The info display can be at the top and maybe at the bottom. If we check on top, it will always be the incoming message on the top, and uncheck, keep the incoming message at the bottom.

(8) **"do not repeat the same message"**: sometimes, we need to see the same message repeatedly. For that, Uncheck it. If checked, the repetitive message is displayed in a single line with the number of repetitions.

(9) **Start/Stop reading the Button**: This Button is used to start the timer when we want to read. When we start that application, the reading start is on by default. If we want to stop reading, press "**Stop reading**" Button. The stop reading Button is used to stop that timer. One timer is always running to read the data at every millisecond. It will check whether there is any incoming message and if the incoming messages are available. It will display in that message box.

(10) **Search textbox**: It is used to search content from the listed data. Sometimes we need some data which is already received. Type any text containing a number or value inside the textbox and enter. It will stop the display and start searching from the entire row. If found, the row will be highlighted.

(11) **Do not process CRC**: sometimes, processing the CRC of an incoming packet takes time to display the message. We can skip CRC processing by unchecking it. But it has an advantage—unwanted or partially unhealthy data is not shown, leading to confusion.

Fig. 6: Application Interface [Source: Author's]

6. RECOMMENDATIONS :

- The complete project work is written in a simplistic approach to understanding better.
- Experience researchers can also add various helpful functionality.
- The Researcher can implement an efficient search method.

7. CONCLUSION :

Through this research, we learn how to create a debug message display using Modbus. The are various debus message display procedures are available. Among them, it is less complex and has several advantages. The researchers can implement this into their projects to speed up their experiments using this display procedure.

REFERENCES :

[1] Bingol, O., Tasdelen, K., Keskin, Z., & Kocaturk, Y. E. (2014). Web-based intelligent home automation: PLC-controlled implementation. *Acta Polytechnica Hungarica, 11*(3), 51-63. Google Scholar↗

[2] Yunzhou, Z., & Chengdong, W. (2011, May). Design of a training and experimental platform based on multi-protocol communication. In 2011 *International Conference on E-Business and E-Government (ICEE) (pp. 1-4). IEEE.* Google Scholar↗

[3] Bajer, M. (2014). Dataflow in modern industrial automation systems. Theory and practice. *Int. J. Appl. Control Electr. Electron. Eng*, *2*(4). 01-11. Google Scholar↗

[4] Pfrang, S., Meier, D., Friedrich, M., & Beyerer, J. (2018). Advancing Protocol Fuzzing for Industrial Automation and Control Systems. In ICISSP (pp. 570-580). Google Scholar↗

[5] Wang, H., Lu, J., Li, W., & Jiang, Z. (2017, June). Development of a three-dimensional virtual PLC experiment model based on Unity3D. In 2017 First International Conference on Electronics Instrumentation & Information Systems (EIIS) (pp. 1-4). IEEE. Google Scholar↗

[6] Familiar, B., Barnes, J., Familiar, B., & Barnes, J. (2017). Sensors, Devices, and Gateways. Business in Real-Time Using Azure IoT and Cortana Intelligence Suite: Driving Your Digital Transformation, 127-168. Google Scholar↗

[7] Mathas, C. M., Vassilakis, C., Kolokotronis, N., Zarakovitis, C. C., & Kourtis, M. A. (2021). On the design of IoT security: Analysis of software vulnerabilities for smart grids. *Energies*, *14*(10), 2818. Google Scholar↗

[8] Del Carmen Curras-Francos, M., Diz-Bugarín, J., García-Vila, J. R., & Orte-Caballero, A. (2014). Cooperative development of an Arduino-compatible building automation system for practical electronics teaching. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, *9*(3), 91-97. Google Scholar↗

[9] Müller, M., Götting, M., Peetz, T., Vahs, M., & Wings, E. (2019, July). An Intelligent Assistance System for Controlling Wind-Assisted Ship Propulsion Systems. In 2019 IEEE 17th International Conference on Industrial Informatics (INDIN) (Vol. 1, pp. 795-802). IEEE. Google Scholar↗

[10] Aguilar, R. P., & Moreno, A. P. (2021). Design of a general-purpose automation software based on Raspberry Pi. *International Journal of Embedded Systems*, *14*(6), 563-577. Google Scholar↗

****