A Flexible Platform for Industrial Applications Based on RS485 Networks

Phan Duy Hung[1], Vu Van Chin[2], Nguyen Thanh Chinh[3], and Ta Duc Tung[4]
[1] FPT University, Hanoi, Vietnam
[2] FPT Software, Ho Chi Minh, Vietnam
[3] Nippon Steel Metal Products Vietnam co., ltd
[4] The University of Tokyo, Japan
Email: Hungpd2@fe.edu.vn; Chinvv0110@gmail.com; Chinh.nt@nsmv.com.vn; Tung@akg.t.u-tokyo.ac.jp

Abstract —Despite the popularity of wireless networks, wired serial ones continue to provide the most robust and reliable communication, especially in harsh environments. These networks provide effective communication in industrial and building automation applications, in which noise, electrostatic discharge and voltage faults are very critical to the life of the system. A common example of the above-mentioned networks is the RS485, in which communication between nodes happen in a Master-Slave model. This paper proposes a flexible platform based on RS485 networks, which support multidrop communication links. This platform with a simple bus wiring and a long cable length is ideal for joining remote devices and can be used in a wide range of applications such as smart homes, electronic queue management system, interconnection between security control panels and devices such as access-control card readers. The proposed platform allows the management of network nodes through adding or deleting nodes, editing information, etc. In addition, the firmware of all network nodes can be directly programmed using a graphical programming language on an Android device. This solution can bring about quick and flexible system deployments, especially in cases where the system's requirements change over time or depend on the deployment environment, during which the staff or end-users can modify the software easily.

Index Terms—flexible platform, RS485 network, industrial applications.

I. Introduction

With the explosion of the Internet of Things, the number of devices (Things) has increased rapidly. However, the modes of communication between them can vary and bring about complications. Communication methods between devices can be classified into wired communication (RS232, 1-wire, RS422, RS485, Ethernet…) and wireless communication (Wi-Fi, Bluetooth, LoRaWAN, Nb-IoT, cellular networks…). Each one of those has its own characteristics and is applicable to a number of problems. The choice of a communication method depends on a number of factors such as cost, signal transmission environment, signal transmission between two or more points, network configuration, speed, etc.

Manuscript received July 30, 2019; revised February 11, 2020.
Corresponding author email: hungpd2@fe.edu.vn.
doi:10.12720/jcm.15.3.245-255

In industrial applications, dozens of serial data interfaces are used today. Most have been developed for specific applications. A few have become universal, such as I2C, CAN, LIN, SPI, Flex, MOST, and I2S. Then there are modern standards such as Ethernet and USB and other application-specific serial interfaces like FireWire, HDMI, and Thunderbolt.

Meanwhile, two of the oldest interfaces, RS-232 and RS-485, are not obsolete and are still used in many applications. RS232 communication is most commonly used to program or to communicate between computers or electronic devices with circuit boards. And RS485 is the most popular to connect multiple control points [1], [2].

The RS485 standard specifies differential signaling on two lines rather than single-ended with a voltage referenced to ground of the RS232 interface. The differential format produces effective common-mode noise cancellation. The maximum cable length is commonly defined as 1200 meters. The typical maximum data rate at 1200 meters is 100 kilobits per second. The RS485 interface can be used in simplex mode with a single-pair cable or full-duplex with a two-pair cable. A common configuration is a bus network with multiple drops or connections. The standard specifies a maximum of 32 drivers (transmitters) and 32 receivers (Fig. 1). All receivers are fully connected, while line drivers are disconnected from the line when not transmitting and the bus line is terminated in a load matching resistance [3].

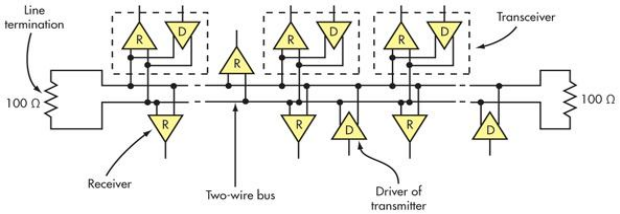

Fig. 1. Common configuration of RS485 network [3].

There have been many studies related to the RS485 network. With regards to hardware, the authors in [4] give the design of a silicon-based Transient Voltage Suppressor to meet the IEC system-level ESD specification for a RS485 transceiver. In [5], the work describes the design of a RS485-to-1553B Bus Bridge. A radiation-hardened ESD protection for RS485 drivers is

given in [6]. Meanwhile, some other studies focus on the application aspect. For example, the authors in [7] present an Intelligent Parking Management System Based on RS485 and RFID. A communication network of stereo garage devices based on RS485 is described in [8]. In [9], the authors discuss about a computerized Fuse Auto Changeover System with RS485 Bus Reporting & Multiple IOT Cloud Connectivity Avenues. While the authors in [10] design A hazmat transportation monitoring system based on Global Positioning System/Beidou Navigation Satellite System and the RS485 bus.

The above studies or solutions prove that the RS485 network can provide effective communication in industrial environments and in construction, automation applications, in which stability and accuracy are the most important requirements. However, there exists a big gap in which there is no complete platform study for RS485 networks. It is understood that because RS485 networks are often deployed in complex environments, requirements can change over time or depend on the deploy location. And in some situations, the deployment staff or the end-users would probably want to change the requirements themselves for time or convenience's sake.

The main contribution of this paper is the development of a complete platform based on the RS485 network that includes nodes (master and slaves) and an Android application. Through the Android application, all network nodes can be managed, by adding, modifying or deleting. In particular, all nodes can be programmed using a graphical language. The platform then converts the program into text and transmits it to the server node, where it will be compiled and sent to the corresponding node, replacing the program currently running. The whole process is done completely automatically, and the user only needs to understand the requirements to program it using said simple graphical programming language on Android phones (Fig. 2).

The remainder of the paper is organized as follows: section 2 describes the platform design, section 3 analyzes the example system and its results. Brief conclusions are discussed in Section 4.

Fig. 2. Method of programming all devices in the network.

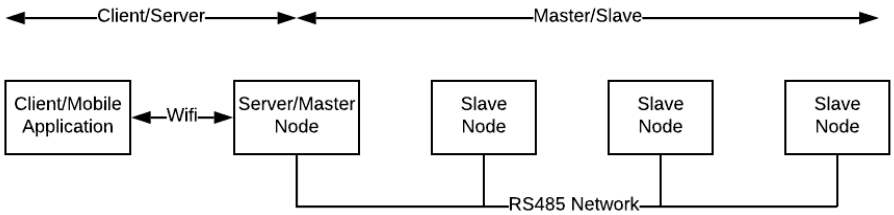

Fig. 3. Overview of the proposed platform.

II. Platform Design

A. Overall Architecture

The platform contains three main components: A Server/Master node, Slave nodes and the Mobile Application (Fig. 3).

The Server/Master component performs both the roles of a Server and a Master. It acts as a Server when working with the application on mobile devices. At the same time, when working with other nodes on the network, it acts as the Master in the Master-Slave communication model. Slave components are networked together with each Slave acts as an interface between the platform and the field device that needs to communicate over the network. A Slave can also act as a new node connected to sensors and actuators.

Our mobile application is written for the most popular mobile operating system, Android. The user can edit the programs of the Master and Slave nodes through the application using a graphical language. The application then generates a text program and sends it to the Master node. The compilation and uploading of executable files are done by the Master node. The application also shows the status of the nodes in the network.

In this platform, the communication between the mobile application and the Master node is done via Wi-Fi for convenience.

B. Requirements

The system has 38 functions in total and is distributed to the components in the system as follows:

The Server/Master performs 32 functions, the Slave nodes 2 and the application 4 (Fig. 4). In addition, there are some non-functional requirements as well: stability, accurate operation, user-friendly interface, cheap hardware and industrial support. A Raspberry Pi is selected to run the Master/Server Node. The Server application is therefore written in Python. The Slave nodes utilize Arduino Pro mini, an open-source electronics platform with a common program structure. The chosen platform for the mobile application is Android, as it is the most popular operating system at the present time.

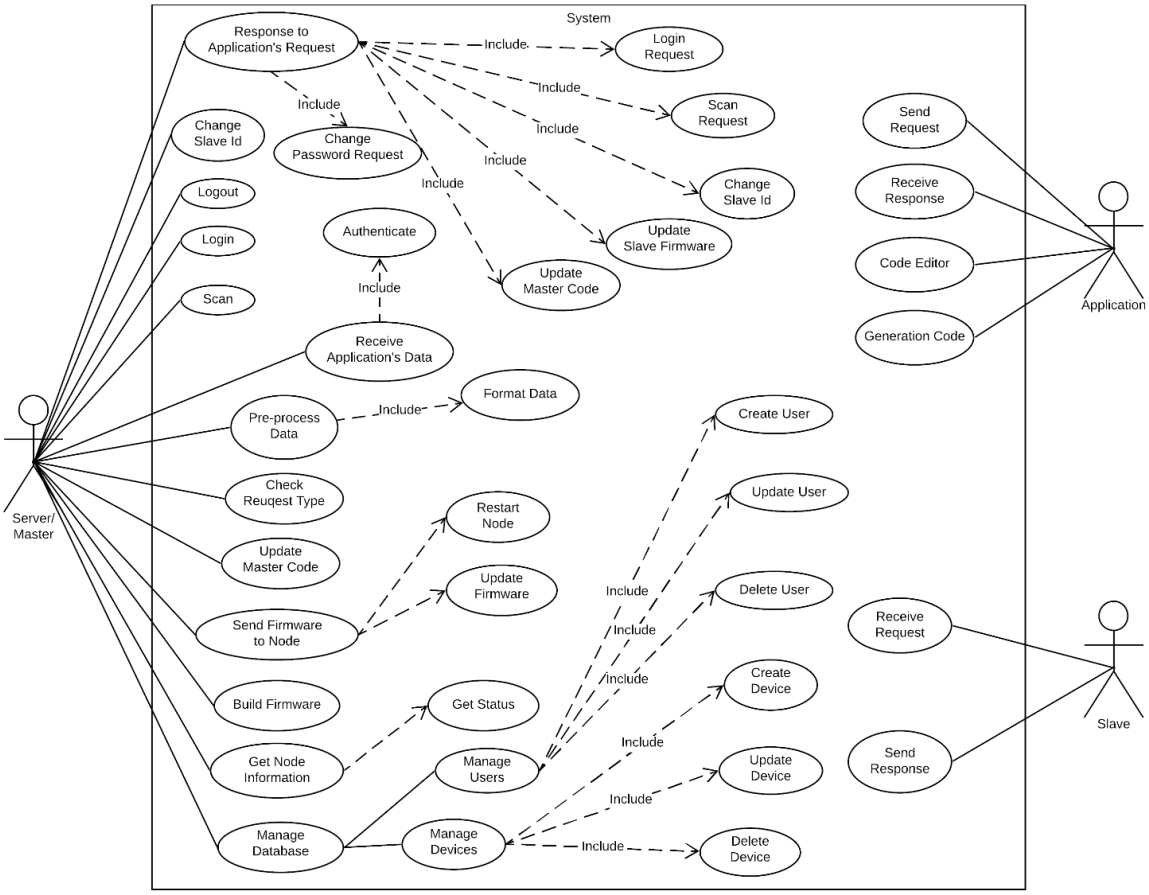

Fig. 4. Use case diagram.

C. Mobile Application

The application is designed as per Fig. 5 with the following functions: Send request, Receive response, Code Editor and Code Generation. The application contains four layers: the Activity layer, the Objects layer, the Network layer and the Utils layer.

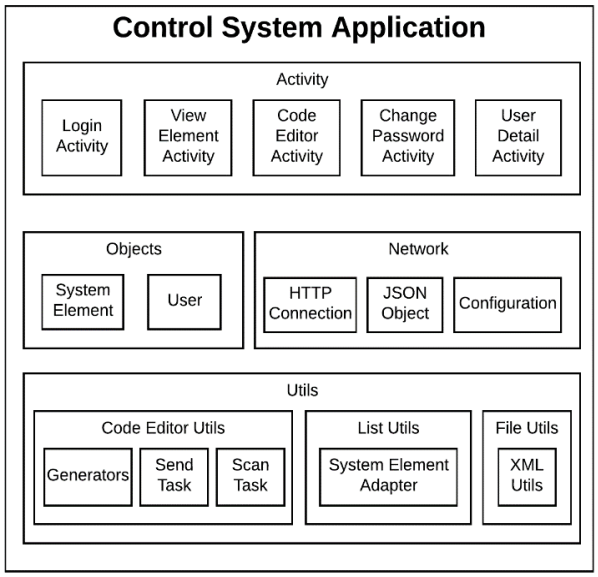

Fig. 5. Block diagram of mobile application.

The Activity layer contains the graphic user interface which the user interacts with. A Login Activity is first displayed. After a successful login, the View Element Activity will be displayed. The View Element Activity contains a list of Slave nodes in the system, which the user can click to view and edit its status.

The most important part of the Activity layer is the Code Editor Activity. When the user chooses a Slave node to update its function, the application will show a code editor screen in a graphical language. The Blockly library [11] is used for this purpose, which presents interlocking, graphical blocks correspond to computing concepts like variables, logical expressions, loops, and more. There are two types of Code Editor Activity: Master and Slave, to create and update the respective device's functions. In addition, the Activity layer has two activities: Change Password Activity and User Detail Activity. The Objects layer contains the object-oriented components: system element and user. Each system element represents a Slave node's status: name, id, data. The user object contains information about a user: name, password, user_data.

The Network layer handles connections to the Server via Wi-Fi, including: HTTP connections, JSON data packaging and parsing and network configurations.

The Utils class is divided into sub-functions that support the application: the Code Editor Utils generates

text code from graphic blocks, the Send Task dispatches the text code to the Server and the Scan Task checks the status of network nodes. The List Utils is an adapter for the System Element and the File Utils performs xml file handling.

Some of the main interfaces of the application are shown in Fig. 6. The Code Editor screen (Fig. 7) includes drag and drop graphic blocks, and the corresponding Code under the text format described.

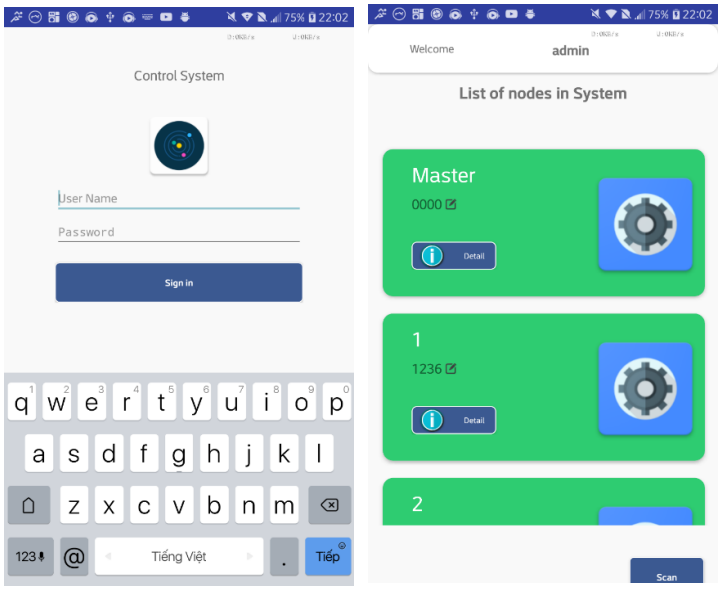

Fig. 6. Main interface of the mobile application.

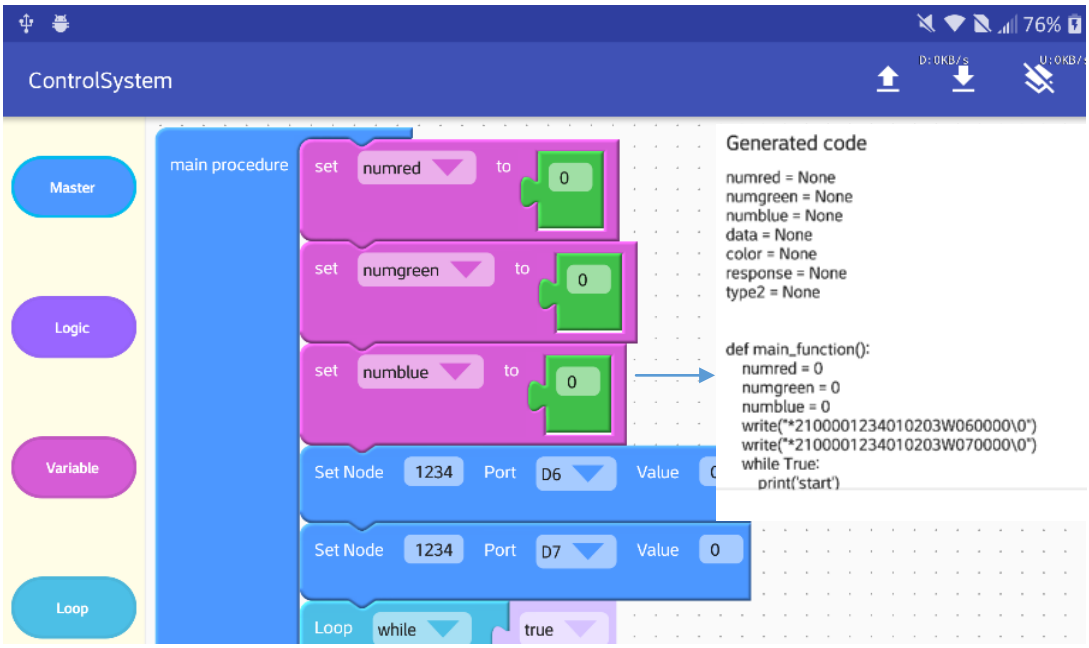

Fig. 7. Code editor screen.

D. Software in the Server/Master Node

The flowchart of the software on the Server/Master Node is described in Fig. 8 and 9. A web server based on Flask, a micro web application framework written in Python, is created for a Raspberry Pi [12]. Server-side functions include creating a database, parsing HTTP Requests, checking Type Requests. When receiving a request from a Client, this node will play the Master role in the RS485 network. These commands include "scan node" to check the status of Slave nodes, modify the default identity (ID) of a new node joining the network, or send new firmware to a Slave node.

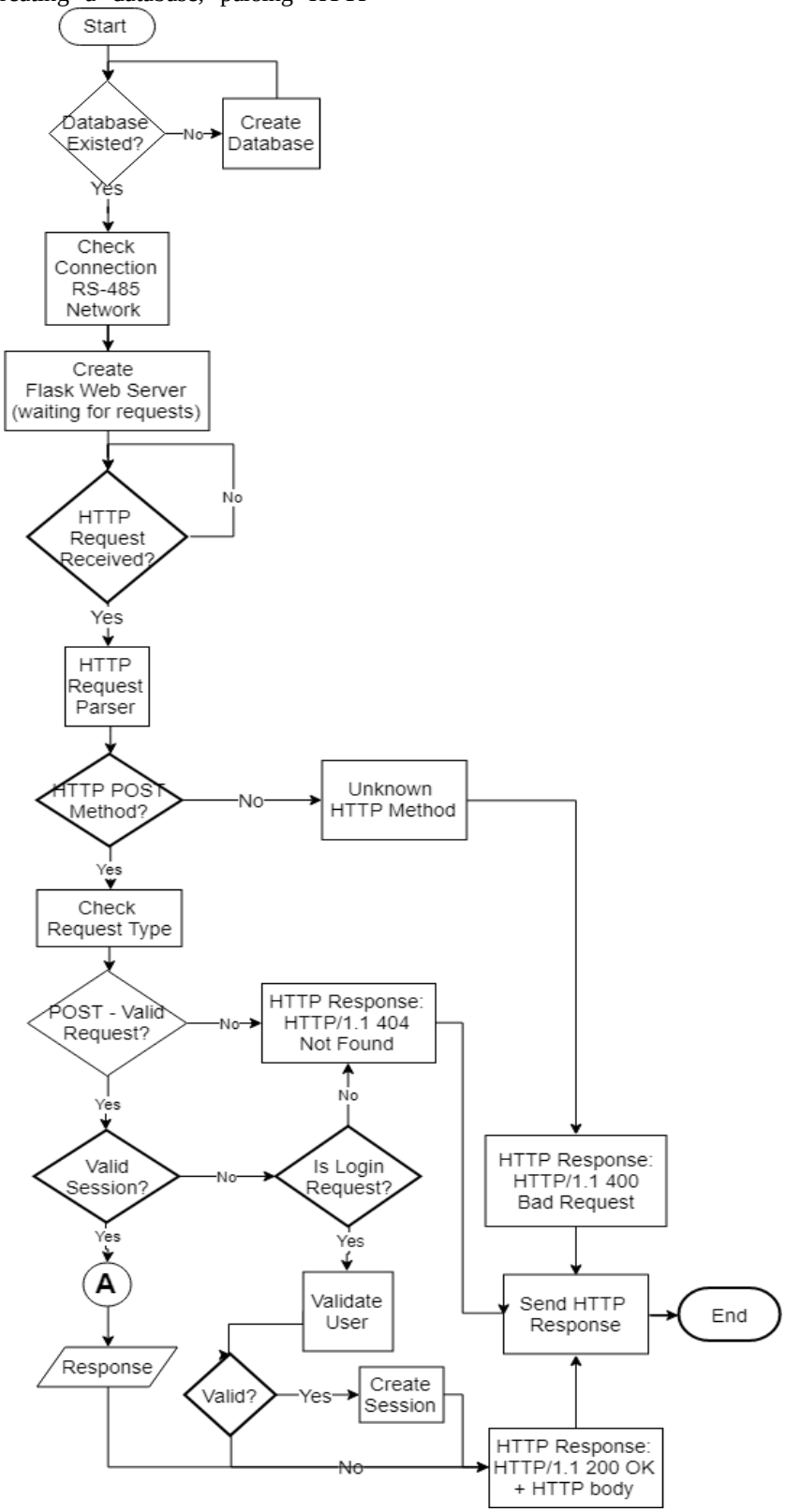

Fig. 8. Server flow chart.

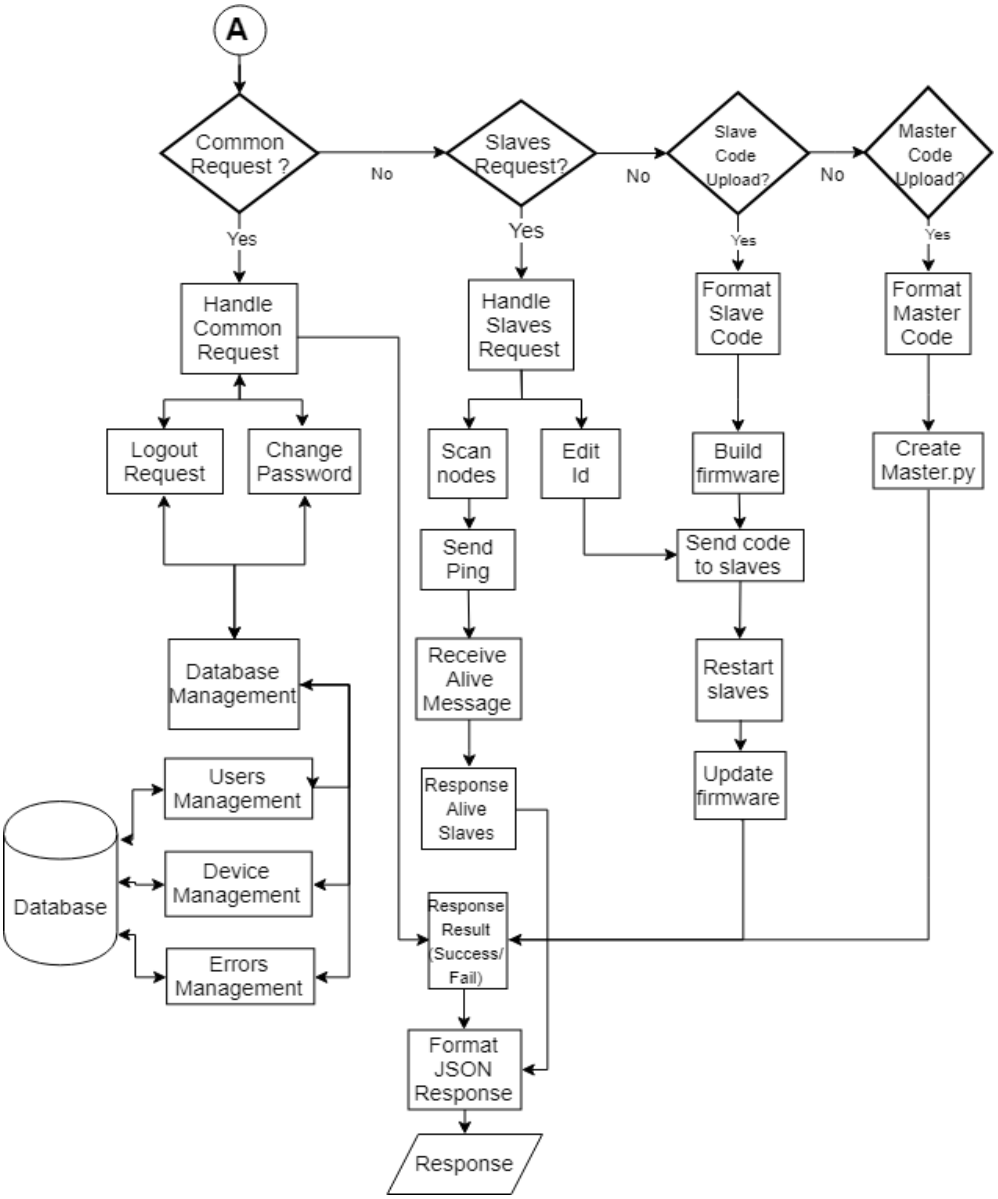

Fig. 9. Master flow chart.

E. Firmware of the Slave Nodes

The flowchart of a Slave node's operations is shown in Fig. 10. The entire source code of a Slave node is prepackaged in the form of a template, including the fixed part and the changeable part, i.e. the part that the user can program on the mobile application. The pseudo code of a Slave node is described in Fig. 11.

The firmware of a Slave after being received from the application will be compiled at the Server/Master node (the Raspberry Pi). The Slave nodes are pre-loaded with a bootloader which allows remote firmware upgrade. The firmware update operation is described in Fig. 12. First, the Slave node is restarted. When it is powered on, the bootloader will wait for 1 second to get a new firmware. Then the new firmware is written to the microcontroller's program memory. Finally, execution jumps to the beginning address of the new program.

F. Firmware of the Master Node

Similar to the slave node, the Master node's flowchart is shown in Fig. 13 and all the source code of the Master node is prepackaged in the form of a template as in Fig. 14.

G. Command Format

The command format for the network is described in Fig. 15, where:

- First byte: marks the start an incoming command. An asterisk (*) character is used in this project.
- Next 2 bytes: the number of bytes after these 2 bytes.
- Next 4 bytes: the store sender's id.
- Next 4 bytes: the store receiver's id.
- Next 4 bytes: the store command id.

- Next 2 bytes: the store data type (00 for requests, 01 for responses).
- Next 7 bytes: command data, including:
  - ✓ First byte: store command.
    - U: restart to upload new firmware.
    - R: read port.
    - W: write port.
    - D: send data.
    - L: sleep command (default to 8 seconds to wait for another slave firmware upload).
  - ✓ 2 next bytes: store port name.
  - ✓ 4 last bytes: store port value.

For example, if the Master with ID of 0000 would like to restart a slave with ID of 1234, we can use this command: *1500001234000100U.

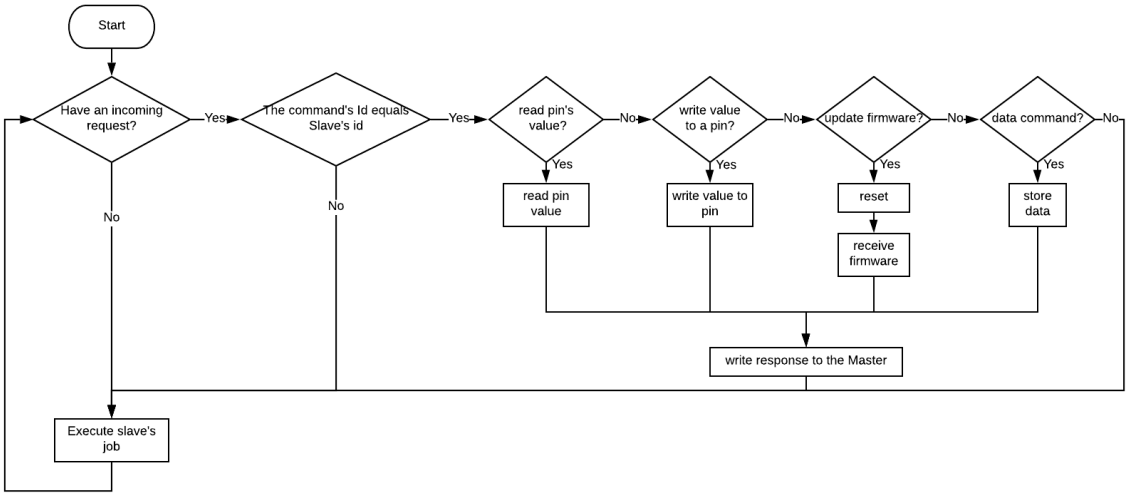

Fig. 10. Firmware flowchart for slave.

```
1. declare rs485 pins
2. declare incoming request analyzer variables
3. declare list response message (for ping, request to the Master message..etc)
4. setup:
5.     get id from specific memory zone (the memoryafter flash memory)
6.         setup incoming request analyzer variables
7.         setup pin mode for RS485 network
8.
9. loop:
10.    receive incoming request
11.    if have incoming request:
12.        parse incoming request and response corresponding data:
13.        U = upload firmware
14.        R = read port value
15.        W = write port value
16.        D = send data
17.        L = request sleep for uploading firmware for another Slave
18.        reset incoming request analyzer variables
19. Do user generated job
```

Fig. 11. Pseudo code of slave template.

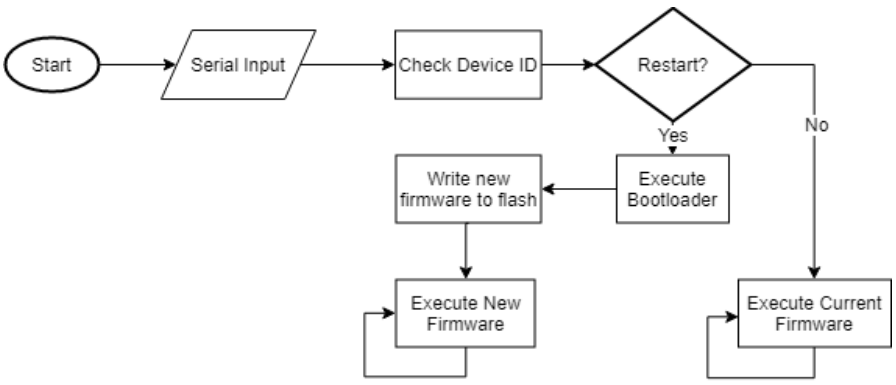

Fig. 12. Firmware upgrade flowchart.

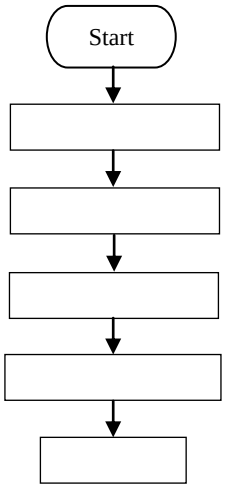

Fig. 13. Master flow chart.

```
1.   import neccessary library :serial , time, array, sys, os
2.   declare global variables
3.   define global funtions:
4.     getNormalChar: elimiate bad byte
5.     isPing: check if the incoming message is ping message
6.     isSlaveRequest: check if the incoming message is slave request
7.     constructResponseSlave: build response message for the Slave request
8.     write: write data to specific Slave with RS485 network protocol
9.     read: read data to specific Slave with RS485 network protocol
10.    clear_serial_buffer: clear buffer in the RS485 network
```

Fig. 14. Pseudo code of master template.

Definition	Start	Total Bytes	Sender Id	Receiver Id	Command Id	Data type	Content		
							Command	Port name	Value
Number of bytes	1	2	4	4	4	2	7		

Fig. 15. Command format.

III. Example System

This project demands a product classification system based on color, including receiving products, checking product status, and product classification. The functions of each network node are described in Fig. 16.

The system hardware includes: 1 Raspberry Pi 3 B+ that functions as the Server/Master, 3 Arduino pro mini as 3 Slaves, plus 3 Max485 modules, 1 LCD, 2 Servos, 2 Proximity sensors, 1 Color sensor and wiring. The wiring diagram of the system and the prototype are shown in Fig. 17. The "User Code" has been modified from the original Template and is programmed using the graphical programming language on the mobile application.

To demonstrate the flexibility of the platform, consider a simple situation. At Slave 1, the LCD is currently displaying the total number of products of the same color as the newly classified product. At the time of deployment, the user wants to display all the detected quantities of different product colors when a button on Slave 1 is pressed. Because the common blocks for communicating with digital I/O pins, sensors, LCDs, etc. have been packaged graphically, the user just needs to drag and drop a few graphic blocks to complete their requirements. Some command blocks are modified as shown in Fig. 18. Next, the code is sent to that Slave. Everything is handled automatically by the platform.

IV. Conclusion and Perspectives

This study presents a flexible platform where the users can easily program system functional requirements using a graphical programming language. The platform is dedicated to the RS485 network, which is a popular and robust communication solution. An Android application that allows the management of all network nodes is also developed. The users only need to understand the requirements. Then they can make use of the graphical programming language to change the functions of a node. The entire compilation, communication and firmware upgrade process for network nodes is done automatically and accurately. The hardware used is popular, cheap, and suitable for different deployment environments.

A prototype system is built as a proof of concept for the proposed platform. For future development, voice control and artificial intelligence can also be integrated into the system [13].

This paper can also serve as a reference for research in embedded software [14], smart embedded systems [15], [16], etc.

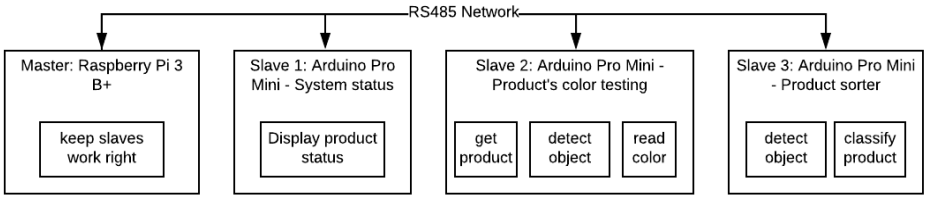

Fig. 16. Product classification system.

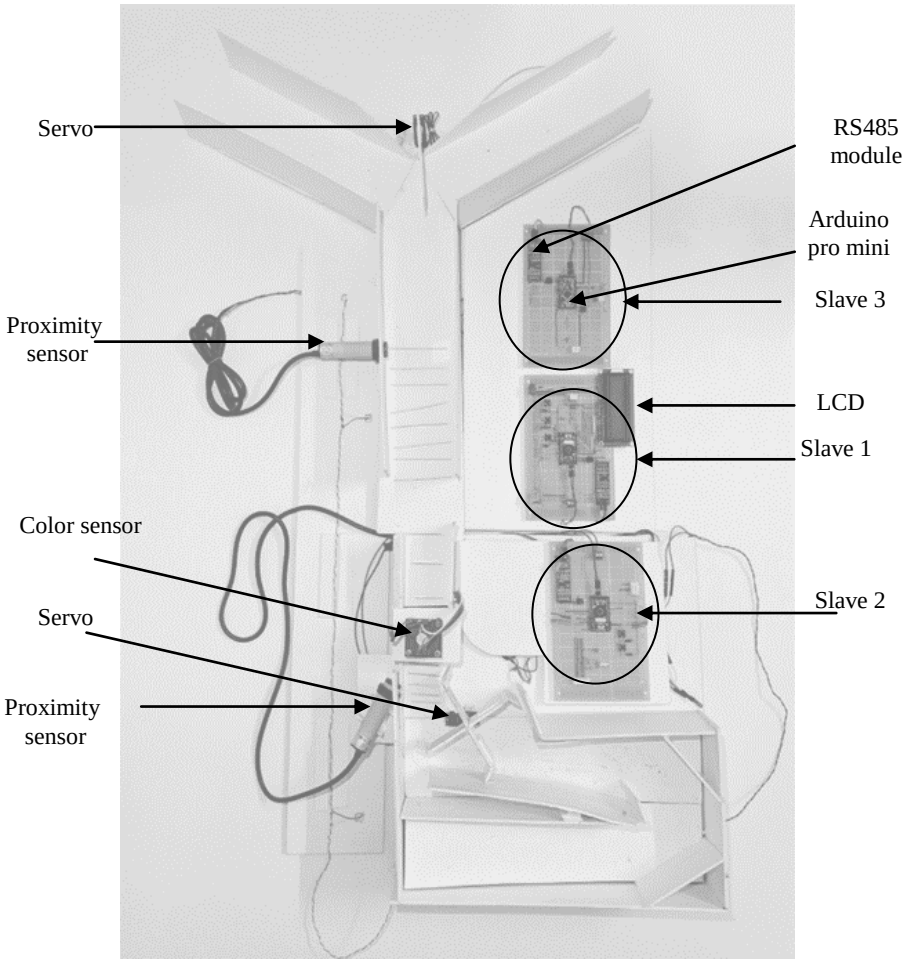

Fig. 17. System prototype.

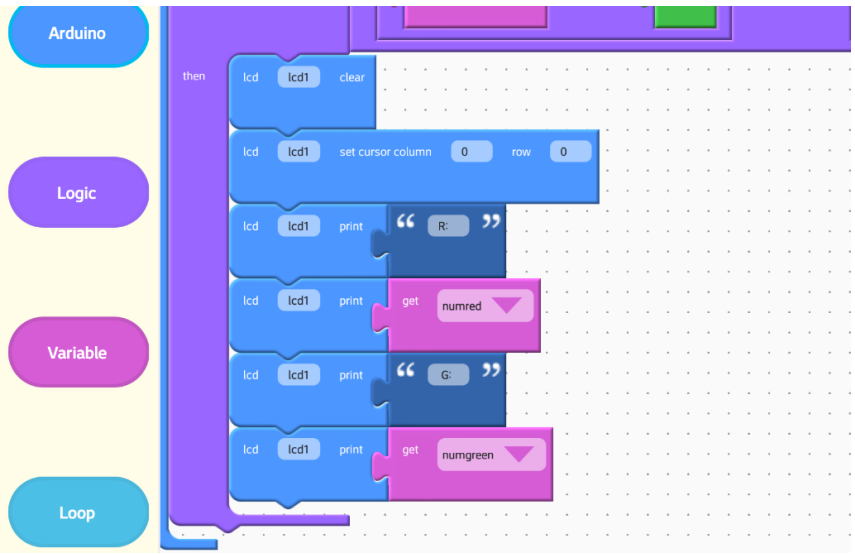

Fig. 18. Graphics blocks have been changed for Slave 1.

Conflict of Interest

The authors declare that they have no conflicts of interest in the subject matter or materials discussed in this manuscript.

Author Contributions

Phan Duy Hung gives the idea and solutions for the problem; all authors participate in programming and making prototype; Phan Duy Hung writes the draft paper; all authors edit and review the final version.

Acknowledgment

This work is partly supported by the research project No. ĐHFPT/2019/01 granted by FPT University. We would like to thank our colleagues in the Information Technology Specialization Department, FPT University, Hanoi for their critical and relevant comments on the manuscript; Colleagues in the English Department who have helped to polish English text.

References

[1] TUTORIAL RS-485: Still the Most Robust Communication, Maxim integrated. [Online]. Available: https://pdfserv.maximintegrated.com/en/an/RS-485-Still-The-Most-Robust-Communication.pdf

[2] H. Marais, RS-485/RS-422 Circuit Implementation Guide. AN-960 Application Note. ANALOG DEVICES, 2008.

[3] [Online]. Available: https://www.electronicdesign.com/what-s-difference-between/what-s-difference-between-rs-232-and-rs-485-serial-interfaces

[4] X. Jin, H. Yuan, Q. Jiang, and Y. Wang, "Design of silicon-based transient voltage suppressor to meet IEC system-level ESD specification for RS485 transceiver," in *Proc. 12th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, Guilin, 2014, pp. 1-3.

[5] L. Zhou and J. An, "Design of a RS485-to-1553B bus bridge," in *Proc. International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, Beijing, 2013, pp. 255-258.

[6] L. Fan, G. Xingguo, X. Xun, H. Xiangzong, and L. Zhiwei, "Design of radiation-hardened ESD protection for RS485 drivers," in *Proc. IEEE International Nanoelectronics Conference*, Chengdu, 2016, pp. 1-2.

[7] H. Zhou and Z. Li, "An intelligent parking management system based on RS485 and RFID," in *Proc. International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, Chengdu, 2016, pp. 355-359.

[8] J. Shen, Y. Ye, Z. Zheng, and W. Hong, "Communication network design of stereo garage devices based on RS485," in *Proc. 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)*, Dengleng, 2011, pp. 3831-3834.

[9] H. Bhojwani, G. K. Sain, and G. P. Sharma, "Computerized fuse auto changeover system with RS485 bus reporting & multiple IOT cloud connectivity avenues," in *Proc. 3rd Technology Innovation Management and Engineering Science International Conference (TIMES-iCON)*, Bangkok, Thailand, 2018, pp. 1-5.

[10] Y. Xie, M. Yu, J. Fu, D. Chen, and C. Yang, "A hazmat transportation monitoring system based on Global Positioning System / Beidou Navigation Satellite System and RS485 bus," in *Proc. 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, Datong, 2016, pp. 1059-1063.

[11] Join GitHub today. [Online]. Available: https://github.com/google/blockly-android

[12] Flask. [Online]. Available: http://flask.pocoo.org/

[13] P. D. Hung, T. M. Giang, L. H. Nam, and P. M. Duong, "Vietnamese speech command recognition using recurrent neural networks," *Journal of Advanced Computer Science and Applications*, vol. 10, no. 7, pp. 194-200, 2019.

[14] P. D. Hung and D. Q. Linh, "Implementing an android application for automatic vietnamese business card recognition," *Pattern Recognition and Image Analysis*, vol. 29, no. 1, pp. 203-213, 2019.

[15] P. D. Hung, L. H. Nam, and H. V. Thang, "Flexible development for embedded system software," in *Proc. 4th International Conference on Research in Intelligent and Computing in Engineering*, Hanoi, Vietnam.

[16] L. T. Kien, P. D. Hung, and K. H. My, "Evaluating Blockchain-IoT frameworks," in *Proc. 4th International Conference on Research in Intelligent and Computing in Engineering*, Hanoi, Vietnam.

Phan Duy Hung received his Ph.D. degree from INP Grenoble France, in 2008. Since 2009, he has worked as a Lecturer, Head of Department, Director of Master Program in Software Engineering at FPT University in Hanoi, Vietnam. His current research interests include Digital Signal and Image processing, Internet of Things, Big Data, Artificial Intelligence, Measurement and Control Systems and Industrial Networking.

Vu Van Chin received his B. Sc from FPT University, Vietnam. Since 2018, he is as a software expert at FPT Software in Hanoi, Vietnam. His current research interests include Internet of Things, Artificial Intelligence.

Nguyen Thanh Chinh received his B. Sc from FPT University, Vietnam. He works as a software expert in the fields of Software engineering and Industrial Networking at Nippon Steel Metal Products Vietnam co., ltd. His current research interests include Internet of Things, Artificial Intelligence.

Ta Duc Tung received his PhD degree in 2019 from the University of Tokyo, Japan. Now he works as Postdoc at the University of Tokyo, Japan. His main research is in design and implementation of Digital Fabrication, Robotics and Human Computer Interaction.