Modbus-A: Automated Slave ID Allocation Enabling Architecture for Modbus Devices on RS485/232

Bharath Sudev*[#], Iain Kinghorn*, Dongbing Gu[#], Doug Gower*

* Fläkt Woods UK
Colchester, England CO4 5ZD
Email: {bharath.sudev, iain.kinghorn, doug.gower}@flaktgroup.com

[#] School of Computer Science and Electronic Engineering
University of Essex, Colchester, England CO4 3SQ
Email: {bs16733, dgu}@essex.ac.uk

***Abstract*—If Modbus devices are to be connected on to the same communication infrastructure, each device must be powered up individually and manually given a unique ID. This can be a time-consuming and laborious process if the system has a plurality of devices. This paper presents the architecture, implementation and testing information of Modbus-A, an architecture that allows the master device on a communication infrastructure to autonomously set the IDs of live devices, which are already connected to the bus. Furthermore, the concept is validated using a software simulator as well as using a hardware prototype.**

***Keywords*— Modbus; Modbus-A; autonomous Modbus ID allocation.**

I. Introduction

Modbus [1] is a single master multiple slave protocol. As per the specification, the master should be connected to the slaves in a daisy chain topology, as shown in Figure 1. The slave devices will have unique slave IDs, using which the master will address them during communication.

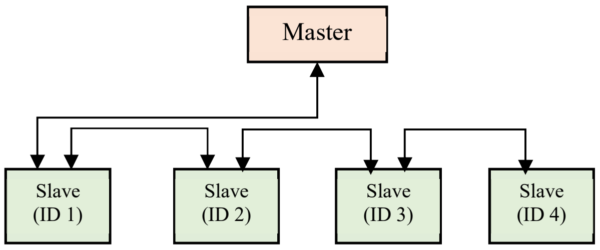
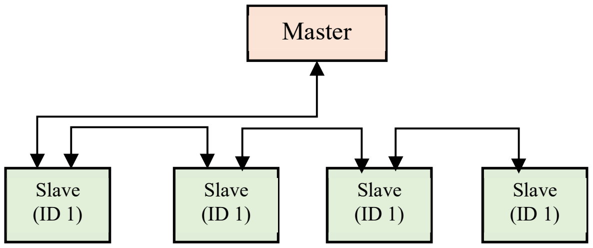


Figure 1. Normal multi slave operation

To get information from a slave or to set a parameter on the slave, the master sends a message with the slave ID through the daisy chain. As an example, if the master sends a message with the ID=3, the slaves with ID1 and ID2 will ignore the message and forward it down the chain. When the slave with its ID set as 3 receives the message, it will process the instruction and send a response message through the chain back to the master.
However by default, Modbus devices out of the box typically will have an ID of 1. So, one would not be able to connect devices out of the box into a daisy chain topology as there will be multiple devices with the same slave ID of 1, as shown in Figure 2.



Under such a situation, the devices will have to be disconnected, powered up individually and given unique slave IDs individually before connecting them into the daisy chain. This can be a time consuming and difficult process when there are a wide number of devices. It is even harder in situations where the devices are already installed and needs reconfiguration.

This paper will present Modbus_A architecture along with the implementation and test result of our patent pending (GB2017008577) concept that will enable automatic reconfiguration of slave IDs of devices which are already in a daisy chain.

This paper is organised as follows. Section II presents related work on Modbus communication followed by the architecture and functionality of Modbus-A in Section III. Section IV presents information on software simulation conducted and its results. Section V then presents the hardware implementation followed by conclusion as Section VI.

II. Related work

Modbus is a protocol widely used in industrial devices like inverters, sensors and control systems. As previously mentioned, every message from the Modbus master will have a slave ID in it to designate the intended recipient of the message.

Followed by the slave ID, the master will send a function code specifying the requested operation followed by some data (if required) and error check code. The frame format of a Modbus message is shown in Figure 3.

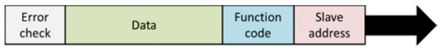


Figure 3. Modbus frame format

Though every slave in the system will receive each message, only the slave with the same address (ID) as in the message will accept the message while the other slaves ignore it. On reception of the message, the slave will do the requested operation and respond with a message back to the master containing its slave ID as a confirmation. If the master fails to receive a confirmation message, the master will retransmit the message as the pervious message is assumed to be lost [2]. So, it is critical that the slaves have unique Modbus IDs lest multiple slaves will attempt to reply to the master's message simultaneously thereby failing the protocol. Since the devices typically will have a Modbus ID of 1 out of the box, there will be duplication of IDs on the bus if there are multiple devices.

In such a situation, each must be powered up separately and given a unique Modbus ID before connecting them to a common communication medium. The difficulty of setting the ID will increase with the number of devices and the location of its deployment.

Naismith et. al. [3] present a concept where the master device polls through all the possible address in the network on start-up. This enables detection of devices on the network if all of them have unique Modbus IDs. However, this is a time-consuming process and would not work if there are devices with same Modbus IDs.

Liang et. al. [4] present a concept that will enable reconfiguration of slave IDs when there are ID conflicts. The system employed slaves with unique identification numbers in them. In case of devices with conflicting Modbus IDs, the master requests the conflicting slaves to send their unique identification number to the master within a certain pre-set interval like in Ethernet [5]. The slaves then will attempt to send their ID back after a random waiting period so that the IDs are received at the master reliably. In case the IDs are not received by the master, the same process will be re-initiated several times until the identification numbers of all the slaves are received. Once the identification numbers of the slaves are received successfully by the master, the master will then send configuration messages to set Modbus ID of each slave referring to it by its identification number. Since the technique relies on each slave having its own unique identification number, there could be difficulty in adopting this as a universal standard as use of different types of devices or devices from different manufactures can result in duplication of identification numbers on the network. Furthermore, in a system with many slaves, the master will have to send multiple retransmission messages to the slaves; and the slaves will have to attempt to respond with its identification number multiple times per message from the master for a successful transmission. So, the configuration time will increase significantly with the number of devices on the communication infrastructure.

A similar concept is presented in [6] where the master initially will send a broadcast message to retrieve Modbus IDs of all the devices on the network. This technique also involves the use of a unique identification number in the slave to enable communication to devices with same Modbus slave IDs.

In case of ID conflicts, the master will request the slaves with conflict to send their Modbus IDs combined with their unique identifier thereby formulating a total ID. To prevent conflicts of reply messages, each slave will delay the message back to the master for a time period proportional to its total ID. Due to the use of unique identifiers on the slaves, this technique could also suffer from the limitations as with [4] as discussed previously. A similar approach is seen in [7] where the slaves are identified using unique identifiers and communication is possible either using Modbus or TCP (Transmission Control Protocol).

On the contrary, Lloyd [8] presents a concept that enables slave devices to send unsolicited messages to the master requesting identification/configuration. Successful transmission of the message is ensured by using principles used in Ethernet like Address Resolution Protocol [9] and Dynamic Host Configuration Protocol [10]. On receipt of the message, the master is able to allocate ID to the device. This technique is thus able to resolve ID conflicts by using complex logic on the slaves which enables unsolicited messages transmission using certain protocols. This paper presents the Modbus-A protocol that enables resolution of ID conflicts using a computationally light method with minimal overheads.

III. MODBUS-A ARCHITECTURE AND FUNCTIONALITY

Modbus-A slaves have two ports for communication. The slaves have two states of operation, a default state where the two Modbus ports are connected internally using a Modbus bridge as in Figure 4(a) and configuration mode where the Modbus bridge is disconnected, as shown in Figure 4(b).

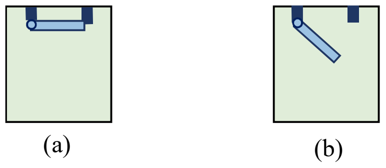


Figure 4. Modbus-A slave modes
(a) Default mode (b) Config mode

As a result, Modbus-A slaves will function as any other traditional Modbus slave under normal operation, as shown in Figure 5.

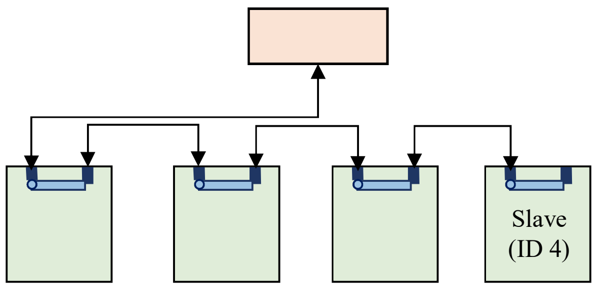


Figure 5. Normal multi slave operation

Consider the condition where the slaves are new out of the box and hence have ID=1, as shown in Figure 6.

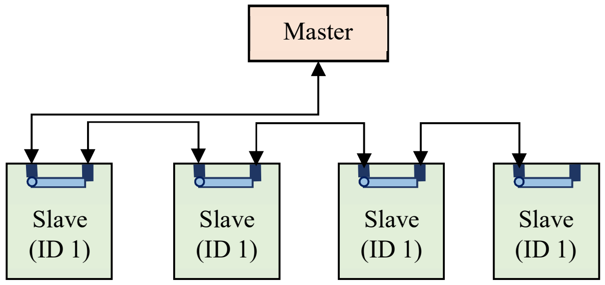


Figure 6. Configuration out of the box

To make the system work, the slaves must be set with unique IDs. With Modbus-A, the master can be made to do this automatically in two stages as explained below.

A. Reset stage

The master sends a reset message to the daisy chain. Regardless of the slave ID, every device that receives the message will accept it and transmit it to the next slave. Once the message is transmitted to the next device, each slave will switch to configuration mode thereby disintegrating the daisy chain by opening the Modbus bridge, as shown in Figure 7. The slaves will also clear their own slave ID. After sending the reset message the master switches to re-config stage.

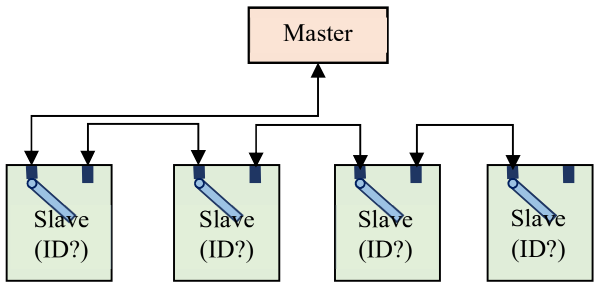


Figure 7. Auto-reconfiguration sequence step 1

B. Re-config stage

In re-config stage, the master sends messages with incrementing slave ID so that the slaves which are in configuration mode can set its ID and go into default mode. In the example in Figure 7, the master first sends a configuration message with ID=1.

Since the slaves are in configuration mode and hence have disconnected Modbus bridges, only the first slave will receive the message. On receiving the message, the slave will set its ID to 1 and then change its mode to default, as shown in Figure 8. On setting the slave ID, the slave will send an acknowledgement message to the master thereby confirming the configuration.

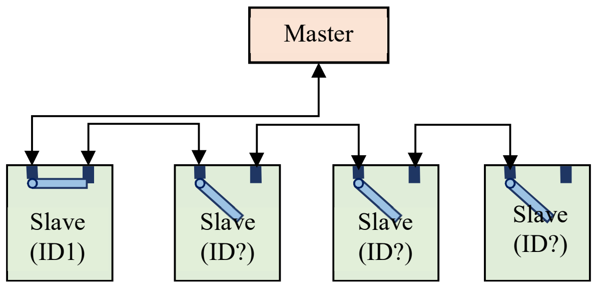
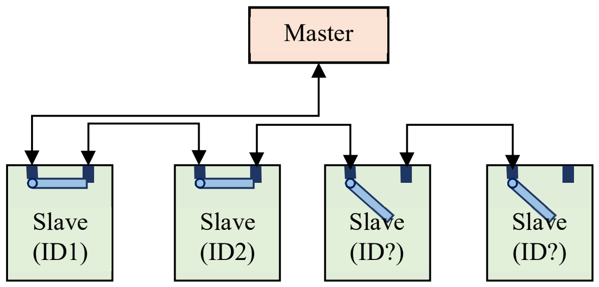


Figure 8. Auto-reconfiguration sequence step 2

As the master is in the re-config mode, it will keep on sending configuration messages with unique slave IDs using an incrementing counter. So, the second message from the master will have an ID of 2. Since the first slave in the example already has a slave ID (of 1) and is in the default mode, it will ignore the message and transmit it to the second slave.

Since the second slave is in configuration mode, it will set its ID in accordance with the ID in the message and switch to normal mode without transmitting the message to the next slave. This is shown in Figure 9. The slave will also send an acknowledgement message back to the master.



Similarly, the third message from the master will configure the third slave and the fourth message the fourth slave, as shown in Figure 10 and Figure 11.

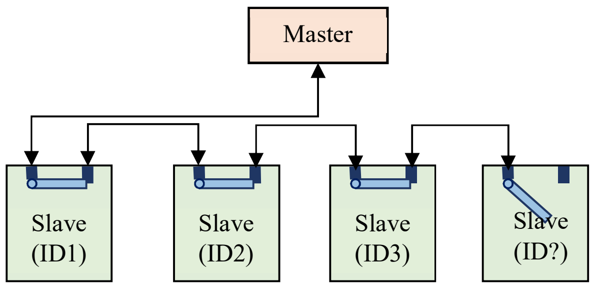


Figure 10. Auto-reconfiguration sequence step 4

So, ultimately, all the devices in the chain will be configured and set to default mode, as shown in Figure 11.

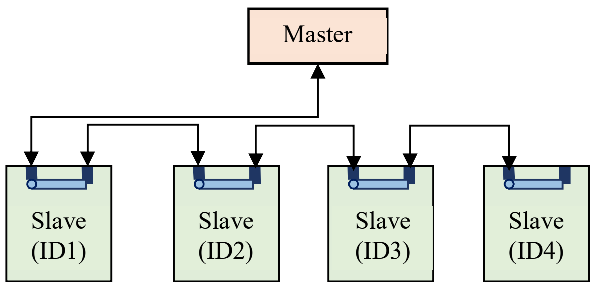


Figure 11. Auto-reconfiguration sequence step 5

Even after the last slave in the chain is configured (as in Figure 11), the master will still be sending configuration messages as before.

However, since all the slaves are in default more and hence no more re-configuration happens, the master will stop to get re-configuration acknowledgement messages back. This will enable the master to deduce that all the slaves in the chain are configured.

IV. SOFTWARE SIMULATION AND RESULTS

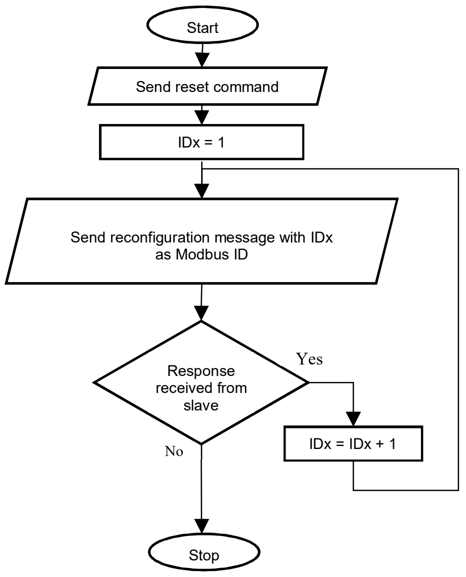
A. Simulator architecture

To ensure expected system functionality before development of a hardware prototype, we developed a Modbus slave simulator with added Modbus-A functionality. The Cycle-approximate TLM (Transaction-level modelling) [11] based simulator is capable of simulation of a parametrizable number of slave devices. In the simulation, slave IDs 98 and 99 were reserved for the reconfiguration procedure such that a message (from the master) with the ID 98 will be treated as the reset message that will instruct all the slaves to disconnect their Modbus bridges and to switch into reset mode.

Following that, the master will send messages with ID 99 accompanied with a new slave IDs thereby configuring the recipient slaves.

The same function can be implemented differently with broadcast messages. In Modbus, messages with ID=0 is treated as broadcast messages. Further work will involve use of broadcast messages coupled with function codes depicting reset and config messages.

The autonomous algorithm that enables the master to configure the slave addresses of devices on the network is shown in Figure 12.


Start
Send reset command
IDx = 1
Send reconfiguration message with IDx as Modbus ID
Response received from slave
Yes
IDx = IDx + 1
No
Stop

Similarly, the algorithm shown in Figure 13 allows the slaves to get configured autonomously by the master.

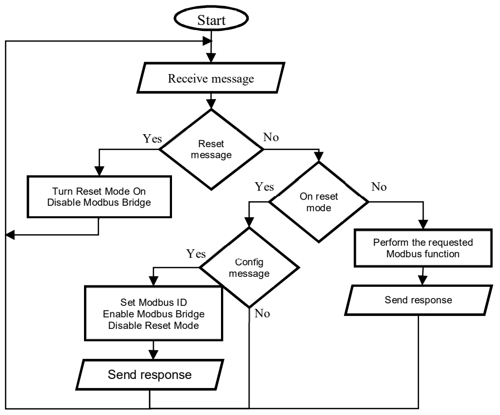


Figure 13. Configuration algorithm on slaves

This enables the slaves to function like any other traditional Modbus slave device under normal working conditions in an infinite loop.

B. Simulation results

Here we present the simulation result of the system with 42 slave devices; from the data captured in the simulator's log. Figure 14 shows the address of the slaves at the start of the simulation. It can be seen that all of the devices have an address of 1; as devices would have out of the box.

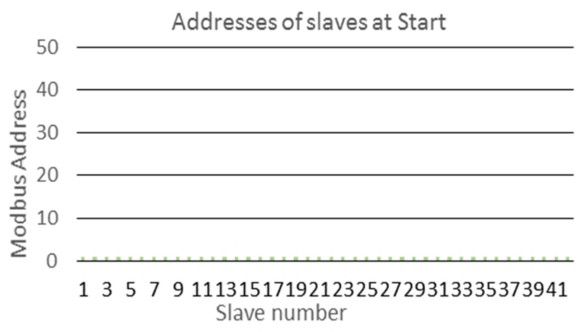


Figure 14. Address of slaves at the start of the simulation

As mentioned in Section III, the Modbus-A master will initally send a reset command which will reset the slave IDs on all the slaves and disconnect its Modbus bridges.
Following that, it will send reset messages to each slave with new slave IDs enabling each slave to set its own ID as per the allocation by the master.

The Modbus addresses of the slaves after the 9th iteration is shown in Figure 15. It can be seen that the first 8 slave devices were give unique IDs from 1 to 8 by the master. It can also be noted that the slaves 9 to 42 do not have IDs as they are in reset mode.

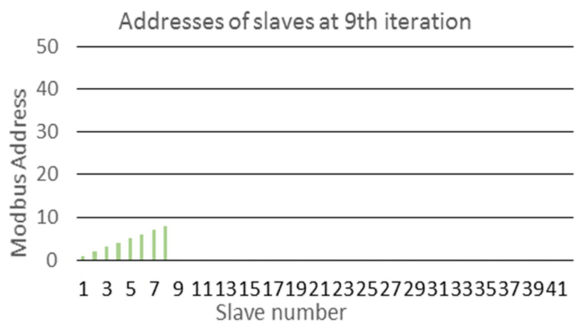


Figure 15. Addresses of slaves after the 9th iteration of the reset logic

Similarly, Figure 16 and Figure 17 show the addresses of the slaves after the 18th and 30th iteration, respectively.

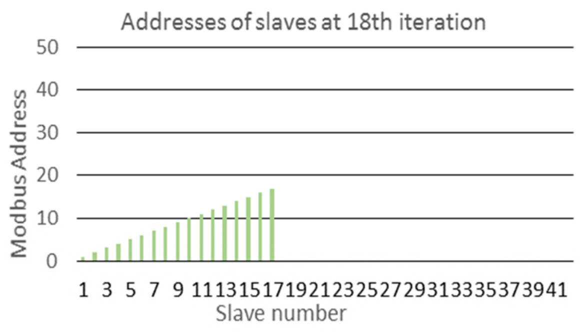


Figure 16. Addresses of slaves after the 18th iteration of the reset logic

It can be seen that 17 of the slaves were reset with unique IDs by the 18th iteration (Figure 16) and 29 of the slaves were reset with unique IDs by the 30th itteration (Figure 17).

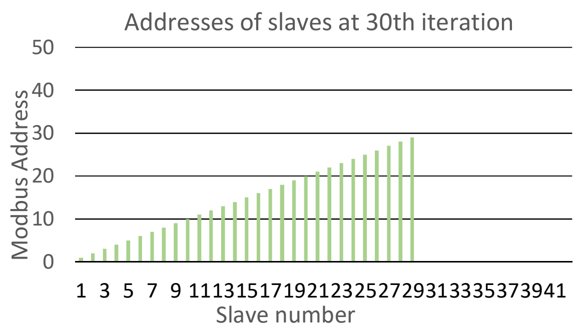


Figure 17. Addresses of slaves after the 30th iteration of the reset logic

Figure 18 shows the address of the slaves after the 44th itteration after the whole reset algorithm was performed. As visible, the master device was able to configure each slave with unique IDs autonomously.

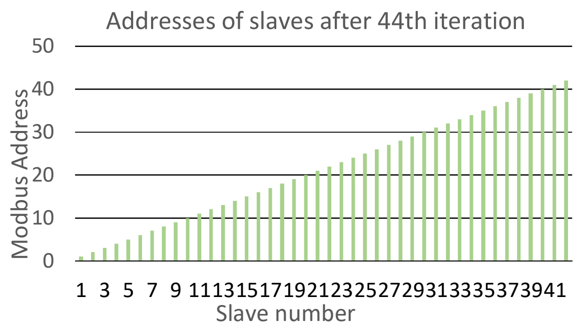


Figure 18. Addresses of slaves after the 44th iteration of the reset logic

V. HARDWARE IMPLEMENTATION

A RS485 based hardware implementation of a Modbus-A slave is shown in Figure 19.

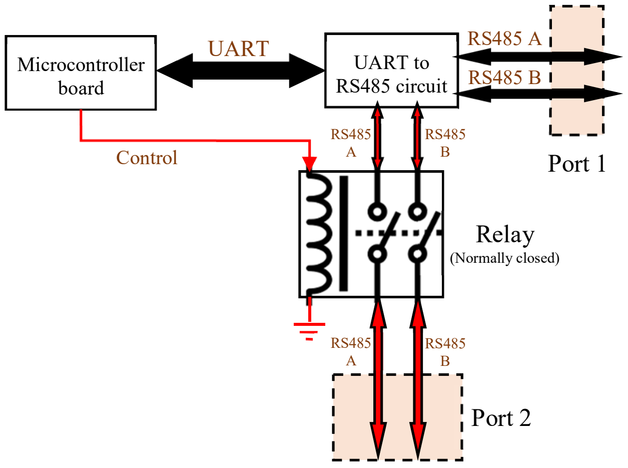


Figure 19. Modbus-A slave implementation

Each Modbus-A slave circuit consisted of a microcontroller board along with a UART (Universal Asynchronous Receiver-Transmitter) to RS485 circuit and a relay. Port 1 is directly connected to the RS485 A and RS485 B connections from the UART to RS485 circuit and the connections to Port 2 are made through the normally closed relay. A control line from the microcontroller board was added to turn the relay on and off as needed.

The test system utilised four such Modbus-A slaves and a PC (with a USB to RS485 adapter) as the master device. The components used in the design are listed in Table I.

TABLE I. COMPONENTS USED

ITEM	MODEL
Microcontroller board	Arduino Mega2560 with 8-bit ATmega2560 microcontroller
Relay	Good Sky GS-SH-205D 5V GS-D Series 1A DPDT Relay
RS485 chip	Maxim MAX3072E

The hardware tests confirmed the functionality of the device as previously established during software simulation.

VI. Conclusion

The paper presented Modbus-A, an architecture that will enable a Modbus master device to autonomously configure Modbus slaves already connected on to the common communication medium.

This was made possible using a configuration algorithm on the slaves aided by an additional hardware component. This enabled the Modbus-A master to autonomously reset Modbus IDs of every slave connected using a configuration algorithm. This eliminates the need for individually powering up each Modbus slave and setting its Modbus ID before connecting them into a common communication medium.

The paper explored the implementation aspects of the architecture along with details on tests conducted using a software simulator. The paper also detailed a possible hardware implementation of the system and evaluated its functionality.

Acknowledgement

This research is partly funded by Innovate UK as part of the Knowledge Transfer Partnership [12] initiative of the government of the United Kingdom.

References

[1] Modbus Organization, Inc, “Modbus,” 1979. [Online]. Available: http://www.modbus.org/specs.php. [Accessed 19 May 2017].

[2] Modbus Organization, “Modbus application protocol specification v1.1b,” 28 December 2006. [Online]. Available: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf. [Accessed 2 June 2017].

[3] R. H. Naismith and D. Areces, “Automatic configuration of network automation devices”. USA Patent US 2005/0256939 A1, 17 November 2005.

[4] W. C. Liang, Y. H. Liu, and K. H. Chu, “Method for setting addresses of slave devices in communication network”. USA Patent US 9015267B2, 21 April 2015.

[5] IEEE Standard 802.3, “Part3: Carrier sense multiple access with collision detection,” 2000 Edition.

[6] Y. P. Grain et al., “Automatic Address Identification Method By Utilizing MODBUS Communication Protocol On RS-485”. China Patent CN201410330307.8, 10 February 2016.

[7] C. J. Ching , A. L. Ting, and L. C. Chin, “Design the DNS-Like Smart Switch for Heterogeneous Network Base on SDN Architecture,” in *International Computer Symposium (ICS)*, Chiayi, Taiwan, 2016.

[8] C. A. Lloyd, “Automated configuration of device communication settings”. USA Patent US 8190697B2, 29 May 2012.

[9] T. Alharbi and M. Portmann, “SProxy ARP - efficient ARP handling in SDN,” in *26th International Telecommunication Networks and Applications Conference (ITNAC)*, Dunedin, New Zealand, 2016.

[10] C. Lin, T. Su, and Z. Wang, “Summary of high-availability DHCP service solutions,” in *4th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*, Shenzhen, China, 2012.

[11] J. R. Harbin and L. S. Indrusiak, “Fast transaction-level dynamic power consumption modelling in priority preemptive wormhole switching networks on chip,” in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII)*, Samos, Greece, 2013.

[12] Innovate UK, “Knowledge Transfer Partnerships,” Innovate UK, 2017. [Online]. Available: http://ktp.innovateuk.org/. [Accessed 13 June 2017].