MONITORING AND CONTROL SYSTEM FOR BUILDING APPLICATION USING MODBUS REMOTE TERMINAL UNIT PROTOCOL WITH AVR AT MEGA FAMILY MICROCONTROLLER UNIT

Arief Wisnu Wardhana\textsuperscript{1}, Eka Firmansyah ST. M.Eng, Ph.D\textsuperscript{2} and Addin Suwastono ST. M.Eng\textsuperscript{3}

\textsuperscript{1}Postgraduate Master Student at Department of Electrical Engineering, Engineering Faculty, Gadjah Mada University
\textsuperscript{2,3}Lecturer at Department of Electrical Engineering, Engineering Faculty, Gadjah Mada University
Jalan Grafika 2 Yogyakarta 55281 Indonesia

ABSTRACT

This paper presents the design of a monitoring and control system that will be installed in buildings and used as a building management system for monitoring and controlling mechanical and electrical devices embedded in the building. The system implements the master slave RS485 multidrop configuration. The system hardware consists of sensor, controller, and actuator. Arduino board with AT Mega series microcontroller unit (MCU) is used as controller. MCU’s ADC will be used as sensor. MODBUS remote terminal unit is used as protocol and implemented inside the master and slave program inside the MCU. At the end of this paper, the result of transmission with various baud rate setting, various cable length, multiple message frames and are presented.

KEYWORDS

Building management system, MODBUS RTU, Multidrop, RS-485, master slave, Microcontroller Unit, Analog to Digital Converter

1. INTRODUCTION

Monitoring and control network has been gaining many applications in our daily life. The network could be a system applied in industrial area to control and monitor various devices in the area. Within automotive area, the system is used to control and monitor various sensor and actuator on modern vehicles. For building application, it is used as a building management system. This research is about monitoring and control system for building application.

Building management system is an electronic-information based control system which is installed in buildings, which will control and monitor mechanical and electronic devices on the buildings.

Some kind of communication protocol is needed for this network. The protocol connects sensors, controller, and actuators. The already known MODBUS communication protocol is used here.
Many publications discussing about monitoring and control for building applications have been done [6][11][12] and some of them utilise MODBUS protocol [5][8][10]. The hardware part of this Building management system consists of sensors, controller, and actuators. Communication protocol will build up the software part.

The research emphasises on the utilisation of Microcontroller Unit (MCU) as a hardware controller device. The objective is to build a monitoring and control network for building applications using MODBUS Remote Terminal Unit protocol. The final result is a controller with an embedded software inside which is ready for monitoring and control function.

2. THE SYSTEM
2.1 SYSTEM OVERVIEW

A master slave configuration with RS485 physical medium is used for this network, as depicted in the figure 1 below [7].

![Figure 1. RS485 multidrop configuration for building management system application](image)

It consists of one sensor, one master controller, and two slave controllers with their corresponding actuator. They are all connected with a twisted pair RS - 485 cable. The Master controller is connected to sensor and gives command to the slave controllers. Slaves are connected to actuators and will control the actuators according to master’s command. Master controller is installed at a particular place in a building while the slave controller are placed in some other location in the building.
2.2 SYSTEM IN DETAIL

2.2.1 Sensor

Sensor acts as a device which respond to physical stimuli. It could be in the form of temperature sensor, pressure sensor, etc. It will convert analog signal into digital signal. Sensor is connected to the build-in Analog to Digital Converter (ADC) fiture of the MCU.

2.2.2 Controller

Controller is the main part of this system. There are three controller, one is a master controller, and the other two are the slave controllers. Master controller is used to receive the signal from sensor and then give command to the slave. Slave controller receives the command from master and using it to control the actuator.

2.2.3 Actuator

Actuator is connected to slave controller. Figure 2 below is a typical circuit used for controlling actuator from MCU. As shown, I/O node is connected to the PORTn to be controlled.

2.3 HOW THE SYSTEM WORK

Sensor measures variable value from the surrounding building and transmits the output signal to controller. The controller then uses the input signal which is transmitted to actuator directly or via controller. The actuator receive the signal from controller and acts accordingly.

2.4 COMMUNICATION

2.4.1 Sensor – Controller Communication

This is the monitoring part of the system. The communication between sensor and controller take in the form of reading the sensor output signal. Build – in Analog to Digital Converter is used to read the output voltage from sensor.
2.4.2 Master controller – slave controller communication

This is the control part of the system. The communication between them uses the MODBUS Remote Terminal Unit communication protocol, which can be briefly explained as follows. To communicate with a slave controller, the master sends a message containing:
- Slave Address
- Function Code (the command for the slave what to do)
- Data field
- Error Check

Once the slave finish to do the command, it sends a respond back to master. The respond contains:
- Slave address (so master knows who is responding)
- The requested Function code
- Data
- Error Check

2.5 SOFTWARE

The main part of the software is the serial communication, the role of which is accomplished by USART fiture of the Microcontroller Unit. The following are some routines that build master – slave software:

Transmission routine and its interrupt

```c
// USART Transmit Complete Interrupt
interrupt [USART_TXC] void usart_tx_isr(void) {
    if (tx_counter)
    {
        --tx_counter;

        // The program lines which show an option between sending the address frame or data frame starts from here .......... if (adc_data > (3 * 1023) / 5) {
            if (tx_rd_index == 0) UCSR0B |= (1 << TXB80);
            else if (tx_rd_index > 0)
                UCSR0B |= (0 << TXB80);

            UDR0 = tx_buffer_relay_on [tx_rd_index];
        }
        else if (adc_data < (2 * 1023) / 5) {
            if (tx_rd_index == 0) UCSR0B |= (1 << TXB80);
            else if (tx_rd_index > 0)
                UCSR0B |= (0 << TXB80);

            UDR0 = tx_buffer_relay_off [tx_rd_index];
        }
        // .... until here
```
Routine for receiver and its interrupt

```c
// Transmission function
#ifndef _DEBUG_TERMINAL_IO
#define _ALTERNATE_PUTCHAR_RELAY_ON
#pragma used=
void putchar relay_on(unsigned char c)
{
    while (tx_counter == TX_BUFFER_SIZE);
    asm ("cli")
    if (tx_counter && (UCSR0A & UREADY != 0))
    {
        tx_buffer relay_on[tx_wr_index] = c;
        if (++tx_wr_index == TX_BUFFER_SIZE)
            tx_wr_index = 0;
        ++tx_counter;
    }
    else
        UDR0 = c;
    asm ("sei")
    #pragma used=
#endif

// USART Receive Complete Interrupt
[USART_RXC] void usart_rx_isr(void)
{
    char status;
    unsigned char data;
    status = UCSR0A;
    data = UDR0;
    UDR0 = rx_buffer[rx_wr_index];

    // Program lines which shows whether or not the message is for a particular slave,
    // starts from here ...........
    if (1 << RXBS0)
    {
        if (rx_buffer[0] == 0x41)
        {
            UCSR0A &= ~(1 << MPCM0);
            if (rx_buffer[1] == 0x50)
            {
                if (rx_buffer[2] == 0x51) PORTD = 0xF0;
                else if (rx_buffer[2] == 0x52) PORTD = 0x0F;
            }
        }
    }
```
3 RESULT AND DISCUSSION

3.1 SENSOR – CONTROLLER

When the conversion from analog to digital is completed, the result can be found as follows [1][2]:

\[
\text{ADC} = \frac{V_{\text{in}} \cdot 1024}{V_{\text{REF}}} \tag{1}
\]

where \(V_{\text{in}}\) is the voltage on the selected input and \(V_{\text{REF}}\) is the reference voltage. The value 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB. Experiment was done for two input values, giving the following result:
<table>
<thead>
<tr>
<th>ADC Input value $V_{in}$ (Volt)</th>
<th>ADC output value (ADCW register) in ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00</td>
</tr>
<tr>
<td>5</td>
<td>0xff</td>
</tr>
</tbody>
</table>

### 3.2 CONTROLLER – ACTUATOR

The communication between controller and actuator is based on the following. If the sensor output signal is 0 Volt, then, by referring to the set value in the software (program) at master controller (master MCU), master controller will consider the output value from sensor lower than the value already set. Master controller then sends a message to actuator (via slave controller) to act accordingly.

On the other way, if the sensor output signal is 5 Volt, master controller will consider the output higher than the set value. Master controller then sends a message to actuator via slave controller for the output to act accordingly.

Message frame format from master controller to slave controller can be shown in the figure 3 below:

<table>
<thead>
<tr>
<th>Slave address</th>
<th>Function Code</th>
<th>PORT address</th>
<th>Number of bit in PORT</th>
<th>Cyclic Redundancy Check</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To send the message, a 9-bit frame format is used. Firstly, master controller will send slave address slot (byte). Then, in turn, function kode, PORT address, number of bit in the PORT, and CRC-16 are sent.

The following table shows the result (slave controllers are given an address 0x41 (slave ‘A’) and 0x42 (slave ‘B’)):

<table>
<thead>
<tr>
<th>Slave address sent</th>
<th>ADC Input $V_{in}$ (Volt)</th>
<th>PORT address</th>
<th>Number of bit at the PORT</th>
<th>Actuator Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x41</td>
<td>0</td>
<td>PORTD</td>
<td>upper nibble</td>
<td>upper &amp; lower nibble OFF</td>
</tr>
<tr>
<td>0x41</td>
<td>0</td>
<td>PORTD</td>
<td>lower nibble</td>
<td>upper &amp; lower nibble OFF</td>
</tr>
<tr>
<td>0x41</td>
<td>5</td>
<td>PORTD</td>
<td>upper nibble</td>
<td>upper nibble ON</td>
</tr>
<tr>
<td>0x41</td>
<td>5</td>
<td>PORTD</td>
<td>lower nibble</td>
<td>lower nibble ON</td>
</tr>
<tr>
<td>0x42</td>
<td>0</td>
<td>PORTF (ATMega)</td>
<td>upper nibble</td>
<td>upper &amp; lower nibble OFF</td>
</tr>
</tbody>
</table>
The first byte is a field for slave address which is slave 0x41 or 0x42. The second byte is a field for function code. The function code is a command. Which function code to be sent is depended on the digital input from Analog to Digital Converter. The third byte is a field for PORT address (which PORT on the addressed slave) to be controlled. The fourth byte is number of bit at the controlled PORT (which bit at that PORT which going to be controlled).

<table>
<thead>
<tr>
<th>Message</th>
<th>CRCLo byte</th>
<th>CRCHi Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x41, 'P', 'a', 'n', 'a', 's', 'i', 0x43, 'i', 0x52, 'i'</td>
<td>0x86</td>
<td>0xD6</td>
</tr>
<tr>
<td>0x41, 'D', 'i', 'n', 'g', 0x43, 'i', 0x51, 'i'</td>
<td>0x86</td>
<td>0x26</td>
</tr>
<tr>
<td>0x42, 'P', 'a', 'n', 'a', 's', 'i', 0x43, 'i', 0x52, 'i'</td>
<td>0x82</td>
<td>0xC3</td>
</tr>
<tr>
<td>0x42, 'D', 'i', 'n', 'g', 0x43, 'i', 0x51, 'i'</td>
<td>0x82</td>
<td>0xD2</td>
</tr>
<tr>
<td>0x42, 'P', 'a', 'n', 'a', 's', 'i', 0x46, 'g', 0x52, 'i'</td>
<td>0x82</td>
<td>0x22</td>
</tr>
<tr>
<td>0x42, 'D', 'i', 'n', 'g', 0x46, 'g', 0x51, 'i'</td>
<td>0x82</td>
<td>0x1E</td>
</tr>
<tr>
<td>0x42, 'P', 'a', 'n', 'a', 's', 'i', 0x46, 'g', 0x51, 'i'</td>
<td>0x82</td>
<td>0xEE</td>
</tr>
</tbody>
</table>

The last byte of the message frame is for error checking CRC-16. Examples of CRCLo byte and CRCHi byte (in HEX) for some messages are shown at table 3 above.

The two bytes CRCLo and CRCHi are then added to the message being sent, appended after the last data byte of the message.

### 3.3 Transmission Check

To guarantee the robustness of the system, transmission check was done. There are three transmission check. The first one is by using various baud rate value. With oscilation frequency of 16 MHz, the following baud rate values were tried : 2400, 4800, 9600, 19200, 38400, 57600 and 115200 bps. The result can be shown in table 4 :
Table 4. Result for various baud rate setting

<table>
<thead>
<tr>
<th>Baud rate (bps)</th>
<th>Result</th>
<th>Baud rate error (in %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2400</td>
<td>Functioning</td>
<td>-0.1</td>
</tr>
<tr>
<td>4800</td>
<td>Functioning</td>
<td>0.2</td>
</tr>
<tr>
<td>9600</td>
<td>Functioning</td>
<td>0.2</td>
</tr>
<tr>
<td>19200</td>
<td>Functioning</td>
<td>0.2</td>
</tr>
<tr>
<td>38400</td>
<td>Functioning</td>
<td>0.2</td>
</tr>
<tr>
<td>57600</td>
<td>Functioning</td>
<td>2.1</td>
</tr>
<tr>
<td>115200</td>
<td>Data succesfully sent, but error</td>
<td>-3.5</td>
</tr>
</tbody>
</table>

It can be shown that the system was sucessfully run for baud rate values of 2400, 4800, 9600, 19200, 38400, and 57600 bps.

Baud rate error was calculated by using the following equation \[1 \] \[7 \]:

\[
\text{Error (\%)} = \left( \frac{\text{Baud Rate}_{\text{Closest Match}} - 1}{\text{Baud Rate}} \right) \times 100, \quad (2)
\]

The next transmission checking is by using various cable length (various distance). With a fix baud rate value of 9600 bps, we have the following result:

Table 5  Result for various cable length

<table>
<thead>
<tr>
<th>Distance / cable length (in meter)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Functioning</td>
</tr>
<tr>
<td>20</td>
<td>Functioning</td>
</tr>
</tbody>
</table>

The last transmission check is sending with more than one message. This was done at the program, both master MCU and slave MCU.

This test was done to check the Multiprocessor Communication Mode fiture. Also for testing with different messages, with inter frame time between message. Master firstly sends a number of message to slave number 1, and then continued with the next group of message to slave number 2. The result were shown in table 6 below.
Table 6. Transmission with more than one message frame

<table>
<thead>
<tr>
<th>Number of messages (jumlah frame)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Four messages for slave number 1</td>
<td>Successfully sent, functioning</td>
</tr>
<tr>
<td>Four message for slave number 2</td>
<td>Successfully sent, functioning</td>
</tr>
<tr>
<td>Eight messages, the first four for slave number 1 and the next four for slave number 2</td>
<td>Successfully sent, functioning. Slave number 1 receive the first four message and ignoring the next four messages. Slave number 2 ignores the first four message and receive the next four messages</td>
</tr>
</tbody>
</table>

4. CONCLUSION

Some conclusion can be derived from the above experiment:

1. A monitor and control network for building application has been designed in this research.
2. A master-slave multidrop RS – 485 configuration can be used as monitoring and control system for building management (automation) system application.
3. MODBUS RTU can be used as a communication protocol for this system. It works well when using baud rate value of 9600 and 19200 bps.
4. Microcontroller Unit (MCU) as part of a development board can be used as controller.
5. Generally speaking, the system can be run well with baud rate setting of 2400, 4800, 9600, 19200, 38400, dan 57600 bit per second.

REFERENCES

[2] Atmel ATmega 640/V-1280/V-1281/V-2560/V-2561/V 8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash DATASHEET


Author

Arief Wisnu Wardhana was born in Solo, Indonesia in 1972. He received his engineering degree in Electronic and Information from The University of Huddersfield, Huddersfield, The United Kingdom in 1997. Since 2005, he is with Department of Electrical Engineering, Faculty of Engineering, Jenderal Soedirman University Purwokerto, Indonesia. His current interest is in embedded system and embedded programming. Starting from September 2013, he has been doing his master degree in Department of Electrical Engineering, Engineering Faculty, Gadjah Mada Universiaty, Yogyakarta, Indonesia.