Interfacing of Yokogawa Controllers with Matlab using Modbus Protocol for Process Control Applications

Dr. S. Meenatchisundaram

***Abstract*—**Modbus is an industrial protocol standard that has been in use for many years. Its simplicity and easiness of writing program codes made it as one of the most used industry protocol to transfer data from controllers to personal computers or servers. This project aims to develop an interface using Matlab and Yokogawa single loop feedback controllers with Modbus protocol. Matlab has distinct advantages in terms of powerful, simple and vast number of commands but having poor real-time interfacing features. This paper describes the design of lab experiments for basic process control plants such as temperature, flow, level and pressure, interfacing features of Yokogawa single loop controllers with Matlab using Modbus and Modbus TCP protocols.

***Keywords*—**Matlab Interfacing, Modbus Protocol, Modbus TCP, Yokogawa Controllers.

I. Introduction

REAL-TIME interfacing requires user friendly interface to understand the features of physical variables and the plant dynamics. Process control applications need moderate sampling time, standalone PID control interfacing hardware with good features of data logging and alarm facilities. Proper selection of proportional band, Integral and Derivative time plays a major role in the control of process parameters with PID control algorithm. On the back end, an appropriate communication protocol should be used for data transfer and storage. Yokogawa standalone single loop controllers are having the above said features with Modbus RTU / Modbus TCP communication protocols, which is used in this project.

Modbus is an open protocol, meaning that it's free for manufacturers to build into their equipment. It has become a standard communications protocol in industry, and is now the most commonly available means of connecting industrial electronic devices. It is used widely by many manufacturers throughout many industries. Modbus is typically used to transmit signals from instrumentation and control devices back to a main controller or data gathering system, for example a system that measures temperature and humidity and communicates the results to a computer. Modbus is often used to connect a supervisory computer with a remote terminal unit (RTU) in supervisory control and data acquisition (SCADA) systems. Versions of the Modbus protocol exist for serial lines (Modbus RTU and Modbus ASCII) and for Ethernet (Modbus TCP) [1].

In this paper, design of a set of lab experiments to understand the process plant operations such as temperature, flow, level and pressure is presented. The interfacing is created using the GUIDE toolbox, and the Matlab code is developed to equip Modbus RTU / TCP protocols. It is important to understand the format of Modbus protocol and is explained in the next section.

Dr. S. Meenatchisundaram, Associate Professor, Department of Instrumentation and Control Engineering, Manipal Institute of Technology, Manipal, Karnataka, India. Phone: +91-9448547155; e-mail: meenasundar@gmail.com.

II. Modbus Protocol

A. Introduction

Modbus is a stateless client-server protocol, based on *transactions*, which consist of a *request* (issued by the client) and a *response* (issued by the server). In the field where this protocol is usually applied, there exists a Master-Slave concept that is one of the possible schemas governing the lower level communication behavior on a network using a shared signal cable [2]. A transaction and it's context is visualized in Figure1.

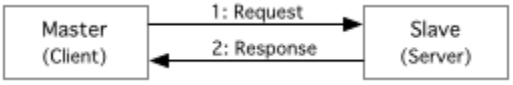

Fig 1 Master-Slave visualization

B. Protocol Data Unit (PDU)

The stateless communication is based on a simple package, which is called Protocol Data Unit (PDU). The protocol specification defines three types of PDU's:

Request PDU, consisting of: a code specifying a function (Function Code, 1 byte) and function specific data (Function Data, varying number of bytes)

Response PDU, consisting of: the function code corresponding to the request (Function Code, 1 byte) and response specific data (Response Data, varying number of bytes)

Exception Response PDU, consisting of: the function code corresponding to the request + 0x80 (128), (Error Code, 1 byte) and a code specifying the exception (Exception Code, 1 byte) [2].

C. Application Data Unit (ADU):

To enable the actual communication, the implementation

extends the PDU with additional fields, into a package with a *header* and an *error checksum*. The resulting package is defined by the protocol specification as Application Data Unit (ADU).

The header is composed of an address field (1 byte) and the tail is an error checksum over the whole package, including the address field. For transmission the Modbus message is placed into a frame that has a known beginning and ending point, allowing detection of the start and the end of a message and thus partial messages. There exist two transmission modes for serial communication and one transmission mode for Ethernet, which differ in encoding, framing and checksum as explained below:

a). ASCII:

Frames are encoded into two ASCII characters per byte, representing the hexadecimal notation of the byte (i.e. characters 0–9, A–F). The error checksum is represented by a longitudinal redundancy check (LRC; 1 byte) and messages start with a colon (':', 0x3A), and end with a carriage return – line feed ("CRLF", 0x0D0A). Pauses of 1 second between characters can occur.

b) RTU:

Frames are transmitted binary to achieve a higher density. The error checksum is represented by a cyclic redundancy check (16 bit CRC; 2 byte) and messages start and end with a silent interval of at least 3.5 character times. This is most easily implemented as a multiple of character times at the baud rate that is being used on the network. The maximum pause that may occur between two bytes is 1.5 character times.

C) Modbus/TCP ADU:

The IP specific header (called MBAP in the specification) is 7 bytes long and composed of the following fields:

i) the invocation identification (2 bytes) used for transaction pairing; formerly called transaction identifier

ii) the protocol identifier (2 bytes), is 0 for Modbus by default; reserved for future extensions

iii) the length (2 bytes), a byte count of all following bytes

iv) the unit identifier (1 byte) used to identify a remote unit located on a non-TCP/IP network

III. Implementation

The physical setup used in this project is shown in Figure 2. The process setup consists of supply water tank fitted with pump for water circulation. The level transmitter used for level sensing is fitted on transparent process tank. The process parameter (level) is controlled by Yokogawa single loop PID indicating controller which manipulates pneumatic control valve through I/P converter. A pneumatic control valve adjusts the water flow in to the tank. These units along with necessary piping are fitted on support housing designed for tabletop mounting. The controller is connected to computer through USB port for monitoring the process in SCADA mode. Similar setups are used for temperature, flow and pressure control trainers. The screenshots of the developed SCADA interface using Matlab GUIDE is shown below. In Figure 3, the interface using RS485 interface with Yokogawa 321E controllers is shown and in Figure 4, the interface using Ethernet interface with Yokogawa 35A controller is shown.

Fig 2. Level Control Trainer with Yokogawa UT321E Controller

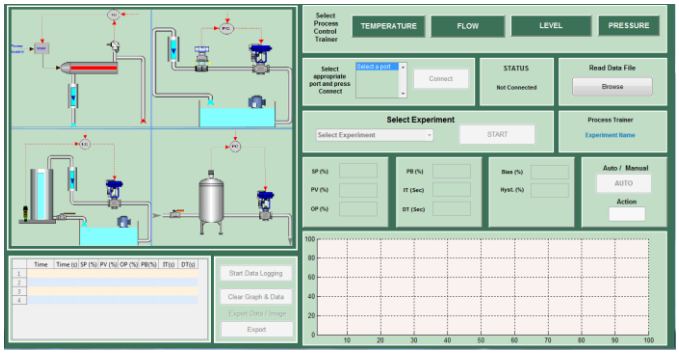

Fig 3. Interface for UT321E Controllers

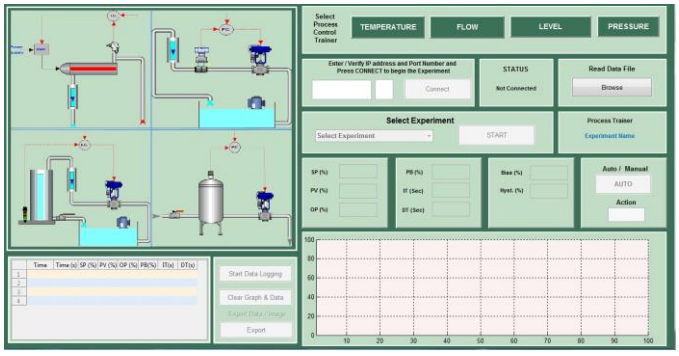

Fig 4. Interface for UT35A Controllers

IV. Modbus Code And Design Of Lab Experiments

A. Sample code

Matlab follows file open, read / write and file close procedure to establish interface with other devices. Matlab GUIDE toolbox provides necessary push button / toggle buttons and other graphical user interface facilities along with the coding in .m file. A set of sample code is given below to

establish communication between Matlab GUI and Yokogawa controller UT321E with Modbus protocol [3,4,5]:

```
%% To read Process Value, Set Point, and
Controller Output
adr = 2; % Controller Address
Fcun = uint16(3); % Function - Read (3) /
Write (6)
Reg = 2; % Register Address
Reg1 = uint16(fix(Reg/256)); % Register
No. Byte1
Reg2 = uint16(mod(Reg,256)); % Register
No. Byte2
No_Reg1 = uint16(0); % Number of Registers
to be read - Byte1
No_Reg2 = uint16(3); % Number of Registers
to be read - Byte2
h = [adr;Fcun;Reg1;Reg2;No_Reg1;No_Reg2];
g=append_crc(h);
pause(0.1);
fwrite(serConn,g);
e=fread(serConn);
```

The response of the controller is stored in a variable 'e' in the above code and is processed further to display and log the process data. Similarly, a sample code is given below to establish communication with UT35A and Modbus TCP protocol.

```
%% To read Process Value, Set Point, and
Controller Output
FunCod = uint16(3); % Function code
UnitIDFunCod = bitor(FunCod,UnitID);
%Add = uint16(2113); % 16b Address of the
resister
Add = uint16(2002);
Val = uint16(3); % 16b Data
message = [transID; ProtID; Lenghf;
UnitIDFunCod; Add; Val];
%disp(message);
pause(0.1);
fwrite(tcpip_pipe, message,'int16');
while ~tcpip_pipe.BytesAvailable,end
tcpip_pipe.BytesAvailable;
res=fread(tcpip_pipe,tcpip_pipe.BytesAvail
able);
```

Here, the COM port and TCP port objects are created as *serConn* and *tcpip_pipe* respectively. Modbus RTU requires appending a CRC for error checking and the code for CRC is written separately as a function file and called whenever needed.

B. Lab Experiments

To understand the process plant behavior and to understand the importance of selection of Proportional Band (PB), Integral Time (IT) and Derivative Time (DT), a set of 10 experiments are framed and the codes are developed in Matlab GUI. A snapshot of the experiment is shown in figure 5 and is listed below.

1. Open Loop
2. On – Off Experiment
3. P Mode
4. PI Mode
5. PD Mode
6. PID Mode
7. Process Reaction Method
8. Closed loop
9. Auto Tuning
10. Stability Analysis

The above experiments are designed to understand the process behavior in open loop as well as in closed loop. In the open loop controller output is directly controlled by the user. In the On – Off mode the controller will perform On – Off control action and a value of hysteresis needs to be provided using the option given in the interfacing screen.

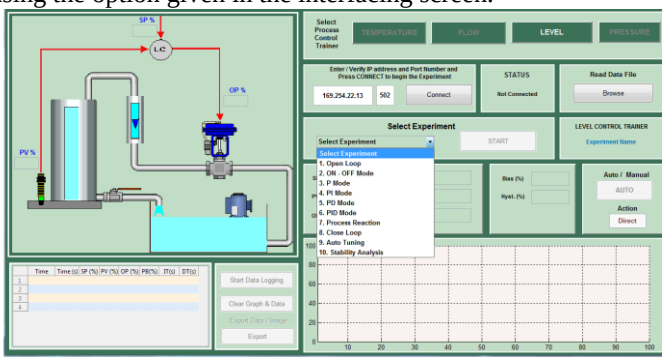

Fig 5. Level control trainer with list of experiments popup menu

In feedback mode, the impact of selection of P, I, D values will be inferred by performing the P, PI, PD and PID mode experiments. The software provides necessary option to feed Proportional Band (PB), Integral Time and Derivative Time values and the response of the plant is displayed in real-time graph. Real-time data can be stored as a jpeg file, or as a excel file. A set of real-time jpeg images exported using the above said interface is given below to demonstrate the power and flexibility of the developed software.

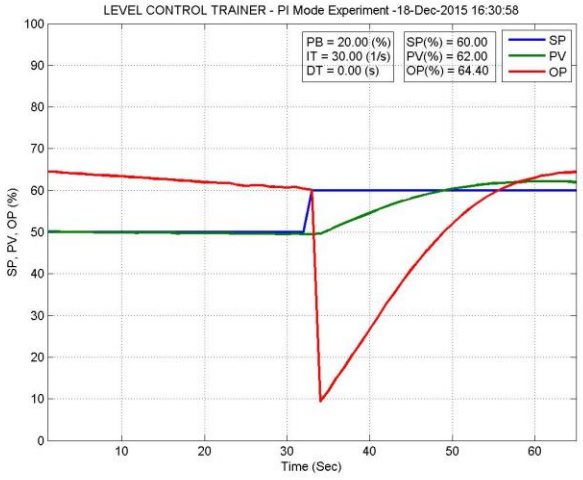

Fig 6. PI mode experiment on Level control trainer

Fig 7. PD mode experiment on Level control trainer

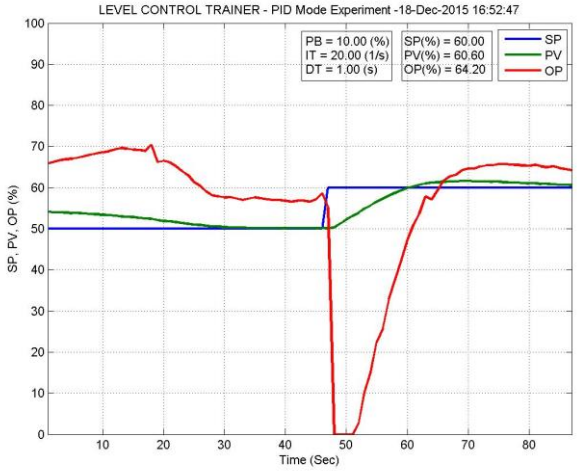

Fig 8. PID mode experiment on Level control trainer

Apart from the basic experiments, the interfacing also caters the experiment for tuning of PID controllers and auto tuning procedure. In the auto tuning mode, the controller itself calculates the optimum P, I and D values using Zigler-Nichols closed loop tuning method. As the last experiment, the stability analysis of the plant can also be performed. In this mode, the controller provides a sinusoidal output with required amplitude and period.

V. Features of the Proposed SCADA Interface

1) The user interface is compiled and created as an application file so that it can be used as platform independent software.
2) The interface has no time / tag limitations, whereas the conventional SCADA software will run in demo mode or will have tag / time limitations.
3) Automatic detection of COM port is done, so that user can easily connect the hardware with the software.
4) The interface uses USB / Ethernet ports and, PC link, Modbus RTU or Modbus TCP communication protocols.
5) The interface supports 4 process control trainers such as Temperature, Flow, Level and Pressure in a single GUI screen.
6) It supports data exporting options in the form of excel 2007, 2010, jpeg, txt and matlab formats.
7) It supports 32 bit as well as 64 bit, Windows XP, Windows 7 and Windows 8 OS.

VI. Conclusion

A SCADA software is developed in Matlab environment to communicate with Yokogawa UT321E, UT32A or UT35A standalone single loop PID controllers. The interface uses Modbus protocols and PC communication protocols as standards to read / write data from the controller through personal computers. A set of lab experiments are developed to understand the plant operations and plant parameters. The interface has good exporting options and the alarm features can be included at a later stage.

Acknowledgment

I would like to acknowledge the help provided by M/s. Apex Innovations Private Limited, Sangli, India, for their technical support and necessary hardware support for the project work. I also would like to thank Manipal University for the lab facilities provided in the department of Instrumentation and Control Engineering, Manipal Institute of Technology, Manipal, Karnataka, India.

References

[1] http://www.simplymodbus.ca/FAQ.htm
[2] http://jamod.sourceforge.net/kbase/protocol.html
[3] http://in.mathworks.com/discovery/matlab-gui.html?s_tid=gn_loc_drop
[4] User Manual, Yokogawa Green Series Communication Functions, doc. No. IM 05G01B02-01E.
[5] Instruction Manual, Level Control Trainer, Product Code 313A, Apex Innovations Pvt. Limited, Sangli, India.

Dr. S. Meenatchisundaram has obtained his Bachelor's Degree in Instrumentation and Control Engineering, Master's Degree in Digital Electronics and Advanced Communication in Madurai Kamaraj University and Manipal University, India respectively. He also obtained his Ph.D for his work entitled 'Design, Simulation and Optimization of Micro Pressure Sensors' at Manipal University. He is having an experience of over 18 years in both Industry and Teaching. He has published about 50 research papers in journals and conferences. His field of interests includes MEMS, Sensors and Instrumentation Systems. Currently he is working as an Associate Professor in the Department of Instrumentation and Control Engineering, Manipal Institute of Technology, Manipal, India.

.