Environment Temperature Control Using Modbus and RS485 Communication Standards

Kosta Papasideris, Chris Landry, Brad Sutter and Archie Wilson
Engineering Technology and Industrial Distribution Department
Texas A&M University
Faculty Advisor: Dr. Joseph Morgan, D.E., P.E.
January 10, 2009

***Abstract* – This paper reports an undergraduate engineering team's effort to develop a system which emulates a curing process chamber found in many industry settings. Based on industry utilization of bus protocols, it has been determined that the de facto standard Modbus protocol under the industry recommended standard interface RS485 is an appropriate communication method for this closed-loop control system. This paper will outline and discuss the implementation of these standards within this system and will also explain the differences between an established and de-facto standard and why this is important when deciding which protocols should be implemented for a particular project.**

INTRODUCTION

The Engineering Technology and Industrial Distribution Department at Texas A&M University is pursuing an initiative to integrate curriculum from Electronics Engineering Technology, Mechanical and Manufacturing Engineering Technology and Nuclear Engineering to create a Power Engineering Technology program. In addition to developing new courses for this path of study, existing courses are being redeveloped. In the Electronics Engineering Technology Program, the Instrumentation and Control Systems course is being restructured to focus on a systems-level development process. In the power industry, master and multi-slave control over a standardized architecture is a prime example of changes in the program. A team enrolled in the Instrumentation and Control Systems course has undertaken a project to develop a system which will be used to facilitate the laboratory requirement for this course. This project will also provide a reference design for future students learning similar communication systems.

SYSTEM OVERVIEW

The temperature control system, depicted in Figure 1, emulates a curing process chamber found in many industry settings. The system reads the temperature of the chamber and adjusts it to track a set of user-defined temperatures. It consists of a single master and two slaves that together monitor and control the temperature of the chamber. One slave communicates with the master to control environment temperature using a heat source and fan, and the other slave consists of a temperature sensor and the signal conditioning circuit necessary for data collection from the chamber. Using both de facto and established standards, Modbus and RS485, the master communicates the necessary information to each slave to complete the temperature control system.

The team was responsible for creating the signal conditioning slave referred to as Slave 1. Slave 1 includes a thermistor, signal conditioning circuitry and an embedded microcontroller which communicates with the master computer to provide the temperature of the chamber. Slave 2 is a laptop with a data acquisition card, or DAQ, running a LabVIEW Modbus Virtual Instrument (VI). Modifications to the LabVIEW VI incorporate the control for heating and cooling sources which are the chamber temperature control devices. Finally, a Master LabVIEW VI is hosted on a desktop computer to accept curing profiles entered by the user. The Master VI is

modified to include an array of user-defined time and temperature settings. Operating together, the Master and two slaves, along with the temperature chamber, create the closed-loop control system.

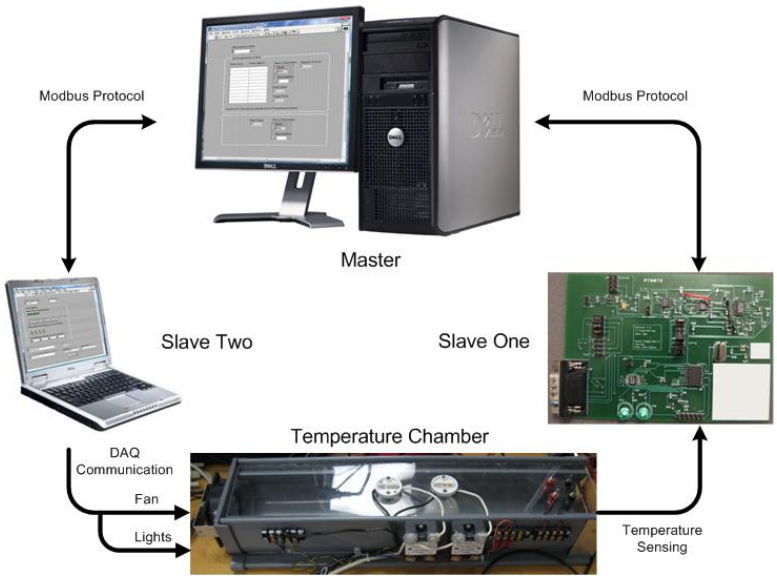

FIGURE 1:
CONCEPTUAL SYSTEM DIAGRAM

TEMPERATURE SENSING

Upon request from the Master, Slave 1 must provide a current and accurate temperature reading for the chamber. Slave 1 can be divided into analog and digital sections. Slave 1 includes temperature-sensing, conversion, conditioning circuitry, analog-to-digital conversion and Modbus message framing. Figure 2 shows a functional block diagram of how this is achieved.

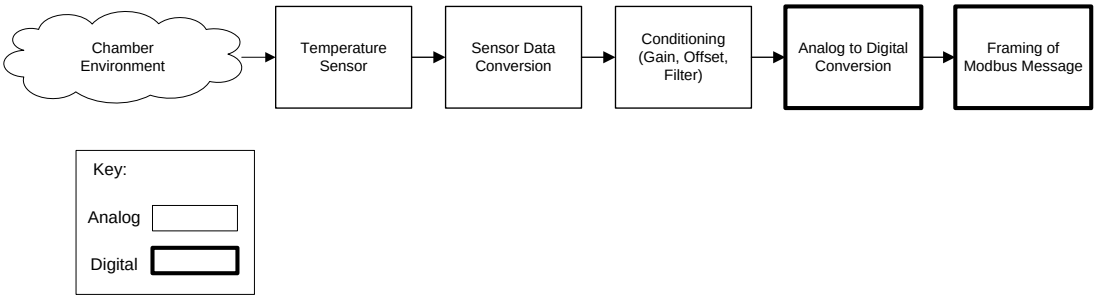

FIGURE 2:
SLAVE 1 FUNCTIONAL BLOCK DIAGRAM

Analog Section

The following constraints were used to develop the conditioning circuit:

1) System power: One nine-volt battery
2) 0 – 3.3 volts operational range for microcontroller (Microchip PIC24 series) and RS485 transceiver chip (Linear Technology LTC2855)

3) 0.3 – 3 volts at respective minimum and maximum temperature (0° to 100°) levels to provide a 20 percent buffer for the A/D
4) A/D Conversion
5) Master-to-Slave communications software development

Since Slave 1 is powered by a nine-volt battery, a voltage regulator providing 3.3 volts is required to provide stable power to the ICs on the board and serve as the voltage reference for the conversion process which in this case is a Wheatstone bridge. Next, the Wheatstone bridge produces a differential voltage based on the thermistor and resistor network configuration. The thermistor allows the Wheatstone bridge differential output voltage to change based on the temperature of the environment. These two voltages are received by an Instrumentation Amplifier allowing a first-stage gain. Complementing this gain is an Offset Amplifier that offsets the voltage received by the Instrumentation Amplifier meeting the specific constraints. The conversion, which is performed by the microcontroller, takes the analog voltage and converts it to a digital ten-bit value which is then converted to a temperature based on a software implemented algorithm. Finally, a message using Modbus protocol is created, encapsulating the temperature measurement.

Temperature Sensor

The majority of Slave 1's responsibility is to use a thermistor to sense the temperature of the curing chamber. Since the change in temperature is inversely proportional to the change in resistance of the thermistor (as temperature rises, resistance decreases), data was collected to characterize the sensor. Data collected from this experiment was provided from 0°C to 100°C as well as the corresponding resistance values in kilo-ohms. Figure 3 depicts the data and relationship of temperature versus resistance.

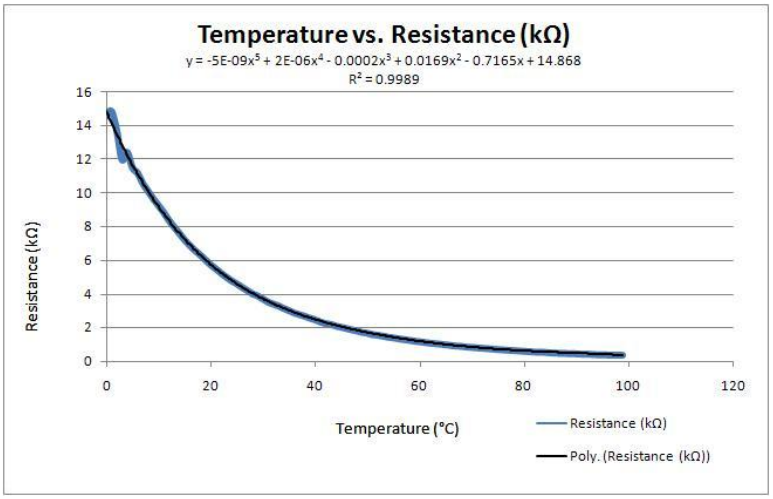

FIGURE 3:
RESISTANCE VS TEMPERATURE CHARACTERIZATION RESULTS

The resistance value of the thermistor is converted to a voltage and eventually conditioned to fit a given range. This process is referred to as signal conditioning. Upon configuration of the entire analog section of the circuit, we needed to characterize the change in the final voltage output as it related to the change of temperature.

As shown in Figure 4, the characterization allows for the formulation of a sixth-order polynomial equation which was used in software to give the proper temperature value of the chamber by using the final conditioned voltage.

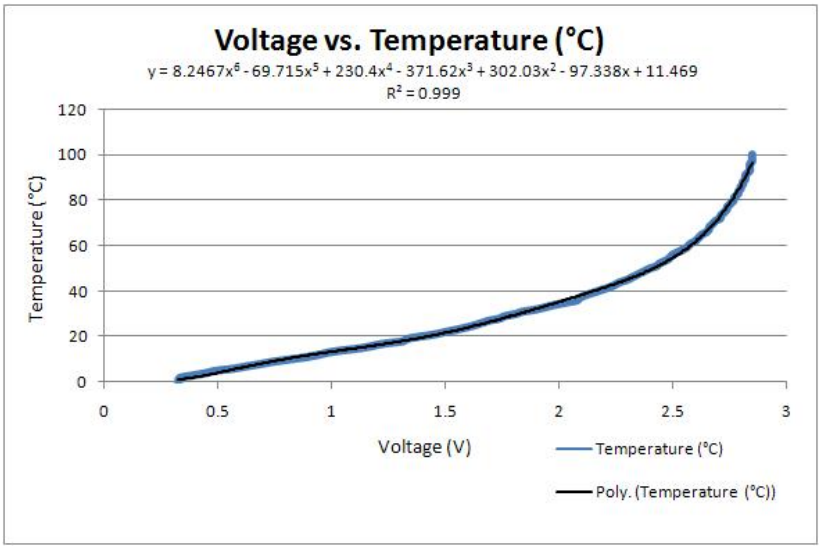

FIGURE 4:
TEMPERATURE VS VOLTAGE CHARACTERIZATION RESULTS

Digital Section

With the analog signal conditioning circuit design completed, the PIC24 was now ready to receive the voltage related to the temperature sensed in the environment.

Analog-to-Digital Conversion

Once an analog voltage was provided to the microcontroller's A/D, the ten-bit conversion process within the PIC24 was utilized to convert the analog voltage to a digital value. The voltage from the conditioning channel was converted by the PIC24's A/D using the voltage-to-temperature polynomial equation, representing the temperature reading.

SYSTEM COMMUNICATION USING MODBUS AND RS485 STANDARDS

The first decision when implementing this system was to decide how communications would be achieved between the master and slaves using specific standards. In this case, Modbus protocol over RS485 would be most applicable because of their wide uses in industry to perform multipoint communication, as well as their open source availability, requiring no royalties for use. Specific to technical standardization, there are differing forms of recognition. Two of these forms include de facto and de jure standards. It is important to understand the distinction between the two when starting to develop a project where standards are utilized.

De facto vs. De jure Standards

Although Modbus is a widely used protocol in industry, it is not recognized as a formally adopted standard in the way RS485 is. Modbus is a serial communications protocol that was originally created in 1979 by Modicon and has since become a de facto standard through its use in many commercial electronic devices.

A de facto standard is, by definition, a custom or product that has achieved dominance through public acceptance and use. A good example of a de facto standard is the Windows operating system or the Mozilla Firefox web browser. On the other hand, a de jure standard (i.e. RS485) has been reviewed by organizations such as the International Organization for Standardization (ISO) and legally accepted.[1] The goals of these organizations are to establish at least four levels of standardization when reviewing: compatibility, interchangeability, commonality, and reference. What all four of these levels try to accomplish is to create interoperability between products and universal agreement on its use in the industry. For example, ISO standards are first developed in response to a need, usually expressed by an industry sector. After being proposed by that sector, the ISO must formally recognize and agree to proceed to the first phase of standardization, defining the technical scope of the standard. This is usually carried out by experts in the area in which the standard is to be applied. Once agreement is reached on the scope of the standard, the second phase begins in which multiple countries begin negotiating the specifications of the standard. Once a consensus is reached, a two-thirds approval by the ISO members involved in the standardization process as well as seventy-five percent of all members who vote is needed. Only after all these criteria are met does a standard become recognized as an ISO international standard. RS485 is one of those de jure standards, and is recognized by the Electronic Industries Alliance or EIA, a trade organization for electronics manufacturers in the United States. They are accredited by the American National Standards Institute (ANSI) to develop standards for consumer electronics and telecommunications. They have a very similar procedure as the ISO for the development of standards in that a consensus must be reached on the scope and specifications of the standard as well as approval by the majority of EIA and ANSI members involved in the process. In addition, the standard is reviewed every five years to guarantee it still maintains the four levels of standardization.[2] Table 1 shows some of the advantages and disadvantages each type of standard offers.

TABLE 1:
DE FACTO VS. DE JURE COMPARISONS

	Advantages	**Disadvantages**
De Facto	1. Widely used and accepted 2. Often proprietary 3. Lots of different options	1. Controlled by a single vendor 2. Not recognized by standard committees 3. Offers less interoperability
De Jure	1. High quality 2. Widely recognized 3. Vendor neutral	1. Strict reviewing process 2. Not market driven

Modbus Protocol[3]

Modbus is a communication protocol which supports master to multi-slave communications between electronic devices. In a standard Modbus network, there is one master and up to 247 slaves -- each with a unique slave address from 1 to 247. In the current configuration, there is one master and two slaves. Data is stored through the use of coils and registers. Coils store simple binary values while registers store numerical values. Both coil

and register values are stored in tables having specific addresses relating to the stored values. This relationship allows for the proper message to be framed under the Modbus protocol.

Modbus RTU vs. Modbus ASCII[4]

Figure 6, the Master Temperature Controller VI Front Panel, shows a selector called "Slave X Parameters." This selector contains a "Mode" dropdown menu requesting Slave mode options: ASCII or RTU. While the choice is arbitrary, there are differences between the two modes.

Modbus RTU sends bytes consecutively with no space in between them. Any delay between bytes will cause Modbus RTU to interpret it as the start of a new message. This keeps Modbus RTU from working properly with modems. Modbus ASCII marks the start of each message with a colon character ":" ($3A_{16}$). The end of each message is terminated with the carriage return and line feed characters ($0D_{16}$, $0A_{16}$). This allows the space between bytes to be variable making it suitable for transmission through some modems. Modbus RTU requires that each byte is sent as a string of eight binary characters framed with a start bit, and a stop bit, making each transmission ten bits in length. In ASCII, the number of data bits is reduced from eight to seven. A parity bit is added before the stop bit which keeps the transmission size at ten bits. In Modbus ASCII, each data byte is split into the two bytes representing the two ASCII characters in the Hexadecimal value. For example, Table 2 displays RTU vs. ASCII Modbus mode data representation.

TABLE 2:
MODBUS RTU VS. ASCII EXAMPLE

Modbus Mode	Data	ASCII Data	Hex Data	Binary Data
Modbus RTU	®	®	AE	1010 1110
Modbus ASCII	®	A, E	41, 45	0100 0001, 0100 0101

The range of data bytes in Modbus RTU can be any characters from 00_{16} to FF_{16}, while Modbus ASCII's range of data bytes represents only the sixteen hexadecimal characters. Therefore, every data byte in Modbus ASCII must be one of those sixteen.

Modbus Message Formatting

The following are examples of Master Request/Slave Response for writing and reading to Coils and Registers using Modbus protocol. The key below gives a description of the parts found in the messages.

Key:

SLV: Slave Address (1 byte)
FUN: Function Code (1 byte)
DATA ADR: Data Address of First Coil/Register (2 bytes)
#COILS: Number of Coils to Write/Request to/from the Slave (1 byte)
#REG: Number of Registers to Write/Request to/from the Slave (1 byte)
#BYTES: Number of Bytes to Follow (1 byte)
COILS: Coil Values to Write/Read (1 bit)
REG: Register Values to Write/Read (2 bytes)
CRC: Cyclic Redundancy Check (2 bytes)

Writing Coils (Function code – 0F for writing multiple coils)

Master Request:
Sent in the form: (SLV FUN DATA ADR #COILS #BYTES COILS CRC)
Slave Response:
Sent in the form: (SLV FUN DATA ADR #COILS CRC)

Reading Coils (Function code – 01 for reading coils)

Master Request:
Sent in the form: (SLV FUN DATA ADR #COILS CRC)
Slave Response:
Sent in the form: (SLV FUN #BYTES COILS CRC)

Writing Registers (Function code – 16 for writing multiple registers)

Master Request:
Sent in the form: (SLV FUN DATA ADR #REG #BYTES REG CRC)
Slave Response:
Sent in the form: (SLV FUN DATA ADR #REG CRC)

Reading Registers (Function code – 04 for reading registers)

Master Request:
Sent in the form: (SLV FUN DATA ADR #REG CRC)
Slave Response:
Sent in the form: (SLV FUN #BYTES REG CRC)

Modbus Error Checking

There are two types of errors that could possibly occur during communication using Modbus.

1. The request is received by the slave with a CRC error. When this occurs, the slave ignores the request and sends no response.
2. The request is received by the slave without any error, but cannot proceed for another unknown reason. When this occurs, the slave replies with an exception response.

The exception response that occurs in the second case consists of a single byte that replaces the byte(s) after the function code but before the CRC. These exception codes range from 01 to 08 that correspond to errors. For example, an illegal function code being sent to a slave device.

RS485 Interface

RS485 provides the physical interface between the master and slaves. Referring to Figure 5, this interface provides a multipoint serial channel that allows communication between multiple devices. Since it is a four-

wire, 120Ω terminated interface, it uses two sets of receive (Rx) and transmit (Tx) lines. Tx+/Tx- on the slaves are connected to Rx+/Rx- on the master, and Rx+/Rx- on the slave are connected to Tx+/Tx- on the master. The effect of this wiring scheme is that the master is able to communicate with all slaves, and all slaves are able to communicate with the master. However, no slave is able to talk to any other slave directly. Information required by one slave from another must go through the master first to be relayed or processed for the next slave.[5]

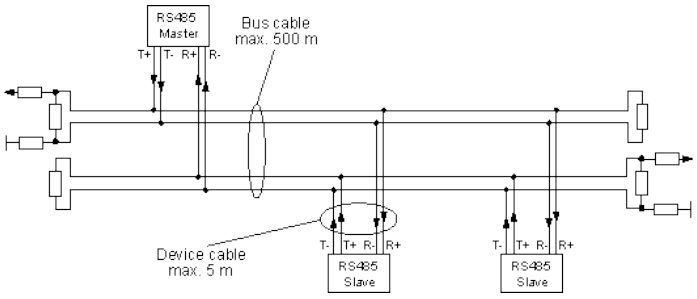

FIGURE 5:
RS485 4-WIRE INTERFACE SETUP[6]

Utilization of Modbus and RS485 in Slave 1

In utilizing both communication standards, specific messages are structured between the Master and Slave 1. The first example is what is requested by the Master to Slave 1 and stays consistent throughout communication. This is the Master asking for the temperature from Slave 1. The second example is how Slave 1 responds to the Master's request and contains the actual temperature value of the chamber received from the A/D of the PIC24. Both of these messages end with cyclic-redundancy checking, or CRC, that are calculated based on a look-up table in the software.

Master Request: Read Registers from Slave 1

Example: 01 04 0000 0001 31CA

01 = Address of Slave 1
04 = Read Multiple Registers (Function Code)
0000 = Data Address of the First Register
0001 = Number of Registers Requested
CRC = 31CA

Slave 1 Response: Send Temperature back to Master

Example: 01 04 02 0032 E4D2

01 = Address of Slave 1
04 = Read Multiple Registers (Function Code)
02 = Number of Data Bytes to Follow
0032 = Contents of Register (Temperature Value)
CRC = E4D2

Interpreting the Master Request was the first objective in creating the software code for Slave 1. The beginning of the sequence starts with Slave 1 listening for a request from the Master. Upon receiving an appropriate request, Slave 1 scans the message looking for the Function Code 04 as discussed in the Master Request above. Once that message is confirmed by the PIC microcontroller, it acquires a voltage at its A/D port and converts it to the chamber's temperature, based on a look-up table. A Modbus message is framed with the corresponding CRC bits that are needed for the specific message. The message or response that is sent back to the Master is similar to the example for Slave 1's Response. This message is sending the temperature of 50°C, which is 0x32 in hexadecimal, framed with a CRC of E4D2 fulfilling the Master's request. The PIC24's UART transmission hardware is used for serial communication. With the addition of an RS485 transceiver chip which converts the

UART signals into CMOS logic-level signals, transmission on the RS485 interface is possible. The software and interface hardware together allow Slave 1 to effectively communicate with the Master.

Utilization of Modbus and RS485 in Slave 2

Slave 2 serves as the controller for the heating and cooling elements. In modifying National Instrument's Modbus Slave VI, an additional set of controls have been added allowing the slave to receive information from the Master VI and also use the data for control. Based on the type of Modbus message that is received, the Slave extracts the coil or register values from the message(s) and relays them to the DAQ for use in controlling the chamber elements. The Slave 2 VI has been modified to receive two coils, referencing the two light bulbs providing heat by turning them on or off. Registers, referencing the duty cycle of the fan, have been modified to control fan speed to cool the system.

Utilization of Modbus and RS485 in Master

The Master serves as the bridge between sensing (Slave 1) and controlling (Slave 2) the chamber temperature. The Master first requests the chamber temperature value from Slave 1 and then, based on Slave 1's response, compares that value to the pre-defined user profile. This profile allows control of the temperature in the chamber by altering it through user-defined time and temperature settings. Figure 6 represents the front panel of the Master VI. The Modbus protocol implemented in the previous Slave 1 section requests and reads the temperature of the chamber from Slave 1. The Master, unlike Slave 1, is not responsible for message framing. This framing is done internally by LabVIEW. However, a control sequence that reads the register values from Slave 1 containing the chamber temperature was needed to be created. Based on that value, the Maser writes the coil and register values to Slave 2 needed to control the chamber temperature.

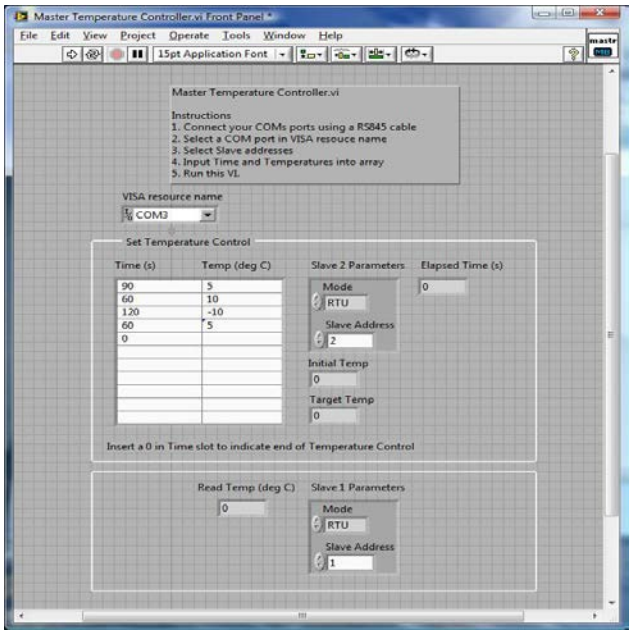

FIGURE 6:
MASTER TEMPERATURE CONTROLLER VI FRONT PANEL

When running the Master VI, the initial temperature of the chamber is read from Slave 1 and stored in the Initial Temp indicator. This is simply the starting temperature at which the profile begins its pseudo-curing process.

The Time table in the left column represents time values at which the temperature in the Target Temp indicator is sustained. The Target Temp is based on the initial temperature plus the value found in the Temp table in the right column. For the example in Figure 6, the Initial Temp box will display the temperature read from Slave 1 and will immediately know to raise that temperature by 5 degrees Celsius and maintain it for 90 seconds. All of this information is relayed to Slave 2 to control the light bulbs and fan speed maintaining that temperature. After a value of 90 seconds appears in the Elapsed Time indicator, that stage of the profile is complete. The Target Temp indicator will then add 10 degrees to its current value and hold it for 60 seconds, and the process repeats for the remaining stages of the profile. At all points during this process, the user may view the current temperature, as it is maintained, in the Read Temp indicator. Putting a 0 in the Time table indicates the final stage, and once it is reached, the VI will know to end, and the pseudo-curing process is complete.

CONCLUSION

The sequence of receiving the chamber temperature, creating the control, and applying the controlling instructions works flawlessly in highlighted execution or step mode in LabVIEW. However, it was found that when running in continuous mode there are timing issues that prevent the operation from running smoothly. Based on the troubleshooting performed both by this development team as well as the professor, the problem was narrowed down to the switching of communication from one slave to the other. Upon receiving the temperature from Slave 1, the Master's instructions began a sequence that, based on whether or not the temperature needs to be increased or decreased, will send the corresponding coil and register values to Slave 2. Slave 2, however, never received these values.

As mentioned earlier, this project is now being fully implemented as the lab requirement for the Instrumentation and Control Systems course. Each semester, the new class of students will have a chance to work on the system develop by this project. They will not only be making improvements to this system, but they will also be taking the initiative to expand on it. Additions so far include several more sensing slaves added to the system allowing for a more precise detection of temperature over a wider area.

The Modbus and RS485 standards are very useful in many applications for multipoint serial communication. The Engineering Technology and Industrial Distribution Department at Texas A&M University has taken the initiative to help undergraduates utilize such standards in many of their class projects. Through a small restructuring process, the Instrumentation and Control Systems course has undergone many changes in the area of applied theory, focusing on the design and development of a system from the theory taught in the classroom. This approach to experiential learning using industry-grade toolsets and employing relevant standards adds significant value to the students' education. Utilizing standardized protocols such as Modbus and RS485 is but one of the many ways the department prepares its students for entry-level positions within industry.

REFERENCES

[1] http://en.wikipedia.org/wiki/De_facto_standard
[2] http://www.iso.org/iso/home.htm
[3] http://www.simplymodbus.ca/
[4] http://www.simplymodbus.ca/ASCII.htm
[5] http://www.lammertbies.nl/comm/info/RS-485.html
[6] http://www.wut.de/kpics/e-6wwww-13-grus-000.gif