Modbus to DNP3 Protocol Converter for use in Water and Wastewater Control Systems

Josh Daly, Daniel Doherty, Mac Lightbourn, Joseph Wilkinson

Department of Electrical Engineering and ComputerScience, University of Central Florida, Orlando, Florida, 32816-2450

***ABSTRACT* — In water and wastewater control systems, secure, reliable, and efficient data transfer are key, while the most common SCADA protocol for storing and transmitting data – the Modbus protocol – does not sufficiently meet these standards in all applications. The newer DNP3 protocol offers an improvement in each of these key areas, and so it is becoming more common in the water and wastewater industries. The device outlined in the following paper will address the problem of a network of mixed protocols. It will act as a gateway in a network of pump controllers at nodes where a device using the Modbus protocol needs to communicate with another device using the DNP3 protocol.**

***Index Terms* — DNP3, Ethernet networks, industrial control, master-slave, Modbus, protocols, SCADA, wastewater.**

I. Introduction

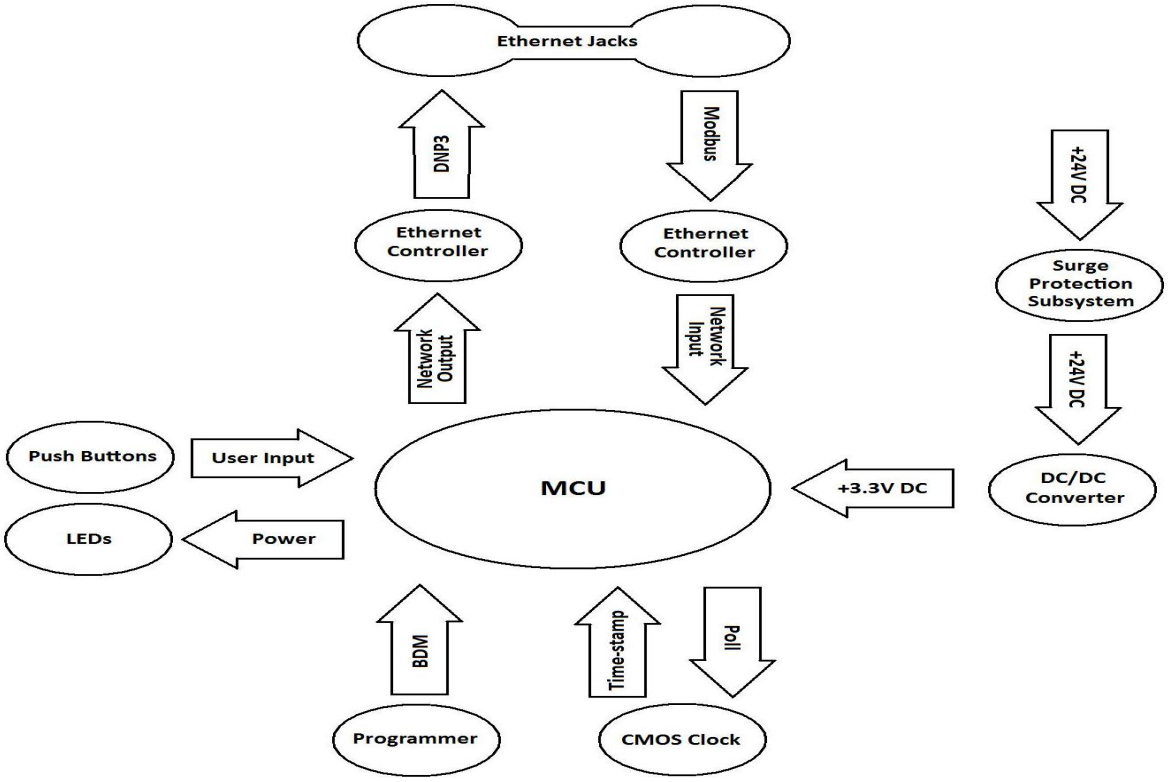
Motor Protection Electronics commissioned our group to create a small device that converts a stream of network communication data from the Modbus protocol to the DNP3 protocol using two Ethernet ports. MPE is a small company out of Apopka, FL, that designs and manufactures controllers for lift stations, which govern the water level and flow rate of wastewater moving from one station to the next. Groups of controllers communicate with each other as part of the smart grid, and there is surprisingly no universal standard for how information is transmitted and stored in this network. Right now, every product that MPE manufactures works only on the widely used Modbus protocol, but in the coming years they hope to introduce some products that use the newer DNP3 protocol. Modbus is by far the most common networking protocol used in industrial controls, and comparing it to DNP3, it's easier to implement because it is a closed standard with well-established specs. Because of that, devices that use Modbus are also cheaper. In the water and wastewater industry, though, time-stamped recordings of events are more crucial than in other industries because of the constantly-changing nature of the fluid being measured. This is where DNP3 outshines the Modbus standard. It provides much faster data transmission than Modbus and records each register change to produce a sequence of events with accuracy to the millisecond. DNP3 also provides enhanced security – a key consideration for devices on the smart grid – and offers room for growth because it is an open standard.

In a network composed of both Modbus and DNP3 devices, a protocol converter is necessary for facilitating communication between devices of different protocols. Existing converters tend to include compatibility with many different protocols, driving up their price and making the units bulky. The project sponsor currently mounts all of their lift station equipment on a DIN rail, so the converter will have to be small enough to fit on one of those and should only convert from Modbus to DNP3. Our project will fill this specific need.

Because of the complexity of the DNP3 protocol, the project has been split into two phases, with our group tackling only the first phase of the project. This includes the device build and the coding of the user interface firmware. Our work will allow for a second group to tackle the difficult problem of coding the protocol conversion firmware with a single-minded focus.

II. Design Overview

While giving us great leeway for a large portion of the design, the sponsor gave us some requirements that the protocol converter must meet. It will be subjected to harsh temperatures, so it must operate between -10 and 80 degrees Celsius. Since it will be used in Florida – the lightning capital of the world – surge protection is a major consideration. The device should withstand surges of up to 220V. The Modbus master and the DNP3 slave will both use 100 base-T rated Ethernet connections and the user must have some way of viewing and setting network parameters associated with Ethernet. The key specification for the project is that the MCU should store 20,000 events in memory, where an event is defined as a register change. It should do this with very little data loss while time stamping each event and notifying the master device of each event. The MCU must be connected to a CMOS clock that can power the system when its main



power source is lost, and it must send an alarm to the master device when this happens.

Fig. 1 Overall block diagram

Existing protocol converters are highly proprietary devices for which virtually no scholarly documentation is available. For this reason, the group had to go to the drawing board, and figure 1, above, is the result. It shows the elements that we deemed vital and represents our system from a birds-eye view.

All equipment near the lift station shares a 24V DC power supply, so no AC to DC conversions were necessary. This power will be filtered through our surge protection subsystem and stepped down to 3.3V, the voltage at which the microcontroller will operate. To step down the voltage, a DC to DC converter was chosen over a cheaper and easier to implement passive voltage regulator because the current drawn by the regulator would likely be too great.

Modbus input comes in from one Ethernet port, and a dedicated Ethernet controller handles the incoming data before passing it along to the MCU. The MCU sends a signal to the CMOS clock, which drives the frequency of the MCU to its own, much higher frequency to allow for accurate time-stamping of data. The MCU will periodically poll the Ethernet controller that deals with the Modbus input, and it must transmit to and receive data from an external clock upon each poll operation. After being recorded, the output data is sent to another Ethernet controller that processes the data for transmission to the master device.

Clocks are also needed on each of the two Ethernet controllers. On the suggestion of our sponsor, the design team chose to couple CMOS clocks with the controllers as well. This allows for a circuit that is less susceptible to electromagnetic interference versus using a crystal oscillator.

The input and output of the converter box will both use the networking technology of Ethernet. For the hardware designers, this necessitated the inclusion of two Ethernet jacks into the circuit. Ethernet jacks come in several varieties, some of which include electromagnetic isolation modules, so the design team made sure to choose one such jack to bypass the need for incorporating an isolation module separately. System input and output are both interfaced with the MCU using Ethernet controller chips, which will govern all network communications.

On the left side of figure 1, one can see the elements that make up the user interface. The guiding principle in the design of the user interface was that of simplicity. As

such, the only hardware requirements for it are a handful of push-buttons, a couple of stand-alone LED lights, and a 3-digit 7-segment display. Not shown are the various drivers and latches that help control each segment or LED light through a multiplexed design.

A programming module connects to the MCU via the background debug module, a set of four pins on the microcontroller. We have extended these pins out to pads on the edge of the PCB so the board can always be reprogrammed.

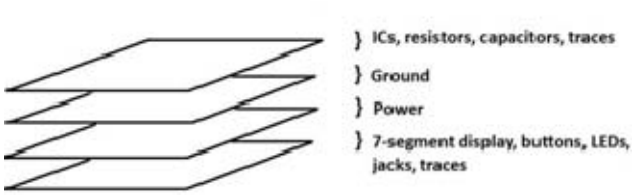


Fig. 2 Board Layers

In the end, our system was composed of nearly 200 discrete parts. To allow for many elements in a compact space, we implemented a four-layer PCB design. All of the surface mounted components, including capacitors, resistors, and ICs, as well as the majority of the traces connecting them, were soldered to the top layer. Dedicated ground and power planes were sandwiched in the middle so that any component needing power or ground can connect with a drilled via placed close to the component. The bottom layer is used for through-hole-mounted components including the Ethernet jacks, LED lights, seven-segment displays, and push buttons. Some traces that did not fit neatly on top were also placed on the bottom.

III. Microcontroller

The specifications given about the project gave serious constraints on the choice of a microcontroller. Storing 20,000 events calls for a chip with a large amount of memory, and because the phase II design team needs to be prepared for a lengthy debugging process, that memory should at least in part be flash or EEPROM so that the chip can be reprogrammed many times. Another specification is that the device must deal heavily in Ethernet networking, a technology not typically built into microcontroller chips – it must take in an Ethernet input, perform the protocol conversion, and give an Ethernet output. It should include a memory stack to help streamline the coding process and it must be able to handle a reasonable current for driving all of the other hardware components in the system. For all of these reasons, the choice of a microcontroller was limited to those on the more sophisticated end of the spectrum.

The group's immediate thought was that one of the few chips with built-in Ethernet support would be the best fit for this project. After a careful examination of offerings from the PIC, Stellaris ARM Cortex M3, and Freescale 68HC12 families, one such chip – the MC9S12NE64 by Freescale Semiconductor – was chosen initially. It is part of Freescale's 68HC12 family of microcontrollers which is used in other devices produced by the project sponsor. Indeed, their flagship pump controller implements this very chip. Therefore an additional advantage of this choice would be the low learning curve for the sponsor when the finished product is handed over to them. Also, an application note published by the manufacturer gave the schematic for Ethernet implementation by interfacing RJ-45 Ethernet jacks directly to the MCU, effectively completing a large portion of the design.

It was originally thought that the 64 KB of flash memory of the MC9S12NE64 would be more than enough to handle the task called for in the spec, making it the perfect choice as the heart of the protocol converter. The project sponsor, however, had doubts. In order to be absolutely sure that enough memory was available, the design team was forced to choose another chip. It erred on the side of caution and settled on the MC9S12XDP512, another member in the same family as the previous choice of MCU. This provides 512 KB of flash memory – probably overkill, but undoubtedly enough to avoid the disaster of running out of memory when the hardware design is finished and the programming is well underway.

The MC9S12XDP512 is a robust and powerful chip running on RISC architecture. With it, Freescale introduced the XGATE module, a built-in peripheral co-processor that is available in three of the four modes of operation for this chip. The module is active any time the internal clock (not to be confused with the external CMOS clock) is active. Using a separate RISC core, it pre-processes instructions and data to soften the interrupt load seen by the CPU, increasing instruction and data throughput. This is quite the significant aide for the protocol converter, which will continually process input data and deliver output data to the next device in the pump control system.

The MC9S12XDP512 MCU comes in three varieties: 80-pin, 112-pin, and 144-pin. The design calls for roughly 70 pins to be used, but the 80-pin option does not include enough ports with which to work, and so we decided to use the 112-pin version. It includes three serial peripheral interface (SPI) ports, which are useful for controlling peripherals such as Ethernet controllers in a

master-and-slave arrangement. Many of its pins are multi-purpose, so a given pin might be used for part of an SPI port as well as a standard address port as well as a pin for another, more specialized signal, depending on the intended application and mode of operation.

All variations of the MC9S12XDP512 include a phase lock loop circuit to regulate power consumption automatically. This is a tremendous benefit for an application such as the protocol converter in which the MCU is delivering power to multiple components. The chip features a low-voltage reset that asserts itself whenever the driving voltage is detected as low and a power-on reset to adjust all output pins to standard operating voltage upon startup. This feature makes it easy to synchronize the resets of all components of the system and keep tabs on the status of all MCU pins when the MCU is powered on.

IV. Ethernet Incorporation

In the course of researching RJ-45 Ethernet jacks, the design team learned that some RJ-45 jacks produced by Tyco come with integrated magnetics modules. Such modules provide shielding from electromagnetic interference events, a feature that must be included one way or another in the final product, so incorporating these jacks led to a simpler design and a less expensive product.

For the Ethernet controllers, the final design uses the ENC28J60, a chip that the sponsor is already familiar with. The specs given for the Ethernet clocks are that they must be 20 MHz and they must operate at a strict minimum voltage of 2 V. For this, the ASEMPC-20.000MHZ-LR-T fits the bill, as it is able to operate anywhere from +3.3 V (which is what the MCU will supply if no voltage divider is used) to +2.25 V. To keep the brand of CMOS clocks consistent throughout the design, this offering from Abracon will be used.

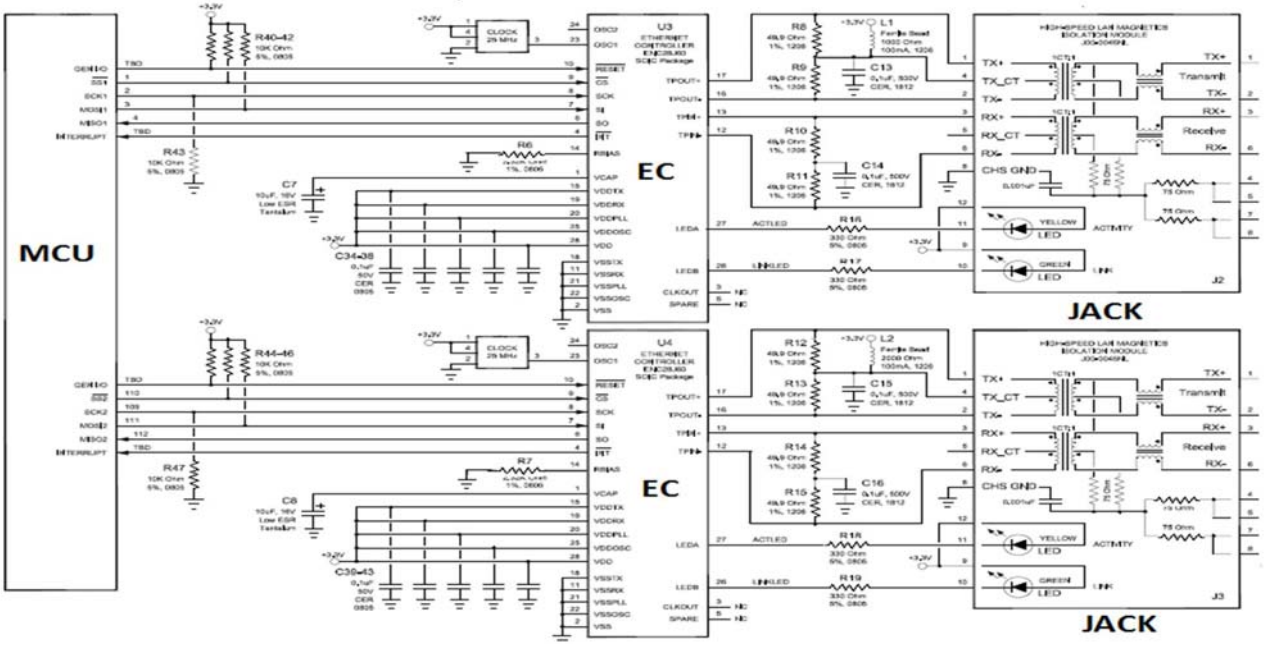


Fig. 3 Ethernet Subsystem Schematic

Each RJ-45 jack with built-in magnetic isolation modules will connect with an Ethernet controller close by on the board, and those will interface with the MCU through SPI ports. SPI lines are the industry standard for high-frequency communication between ICs that are in a master-and-slave arrangement and the protocol converter may transmit at very high speeds – up to 100 MHz.

Figure 3, below, shows our schematic for the Ethernet subsystem. Between the jacks and the controllers are the transmit and receive buses as well as lines tied to LEDs on the jacks that indicate when data is being transferred or received. On the MCU side a reset line and some of the SPI signals use a pull-up resistor for noise cancellation and there's a large bank of capacitors tied to key voltage pins on the Ethernet controller for more control over the noise.

Noise is an extremely common problem for SPI signals. It happens because the master-out-slave-in (MOSI) and the master-in-slave-out (MISO) lines transmit data at very high frequencies, and recall from physics,

$$\mathbf{X_L} = 2\pi\mathbf{F_L}. \quad (1)$$

Inductive reactance is a function of frequency as well as inductance. Even short traces have a parasitic with the high frequency of transmission lines leads to a field effect that interferes with the surrounding signals – noise.

To cut back on noise even more, we shielded the MOSI and MISO pins from nearby signals as well as

from each other by surrounding them with a copper island. These islands carry the ground signal.

V. Push Buttons and LEDs

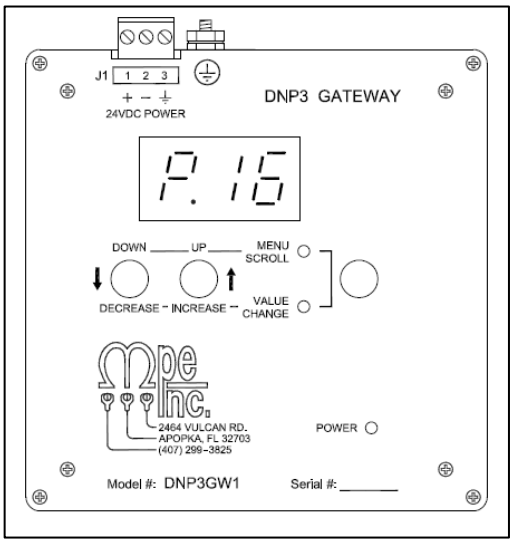


Fig. 4 Faceplate

The user interface is based on that of MPE's existing product line. To aid the user in viewing and changing system parameters, LED lights are used to indicate either 'Menu Scroll' mode or 'Value Change' mode and one of the push buttons is used to toggle between these two modes. Two more push buttons are labeled 'UP' and 'DOWN' – these allow the user to navigate the list of parameters in 'Menu Scroll' mode and raise or lower the value of a parameter in 'Value Change' mode. All of this is reflected on the 3-digit display.

Our design uses four pairs of latches and passive drivers in a multiplexed design that allows for 26 LED lights and segments to be controlled by 9 general I/O pins on the MCU. This was also done to mirror MPE's existing products so that the protocol converter can be built with parts on hand when it goes into production. Each latch has an enable line that connects to a general I/O port at the MCU, which will be programmed to enable only one of them at a time. Clear, data, and three separate address lines share nodes with all of the latches, and then connect to the MCU via five more general I/O pins. Push buttons are driven high when depressed, and pull-down resistors are used to debounce the signal.

VI. Power Subsystems

Luckily for the design team, no AC to DC power conversion needed to be done and no UL standards needed to be considered. The system is supplied with 24 V of DC power which is used by all devices in the lift station, and devices operating at such a low voltage bypass many of the tests called for by the UL standards. Unluckily for the design team, the risk of electromagnetic interference and voltage spikes is heightened in the industrial setting in which the protocol converter will be housed. We took extra measure to guard against these.

Many anti-interference devices were researched, including the metal oxide varistor (MOV), transient voltage suppression diode (TVS), Zener diode, and gas discharge tube. We chose to build a circuit that uses an MOV and a TVS in a shunt configuration to harness the benefits of each – the TVS reacts very rapidly to transient voltages and the MOV withstands many surges. Figure 5 shows the design.

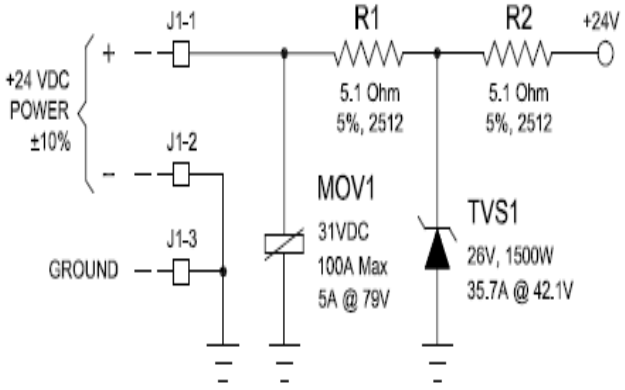


Fig. 5 Surge Protection Subsystem

The DC/DC converter is a common circuit found in many applications, consisting of a switching regulator and Schottky diode. It steps the voltage down from 24V to 3.3V, at which the microcontroller and the other ICs in the system operate.

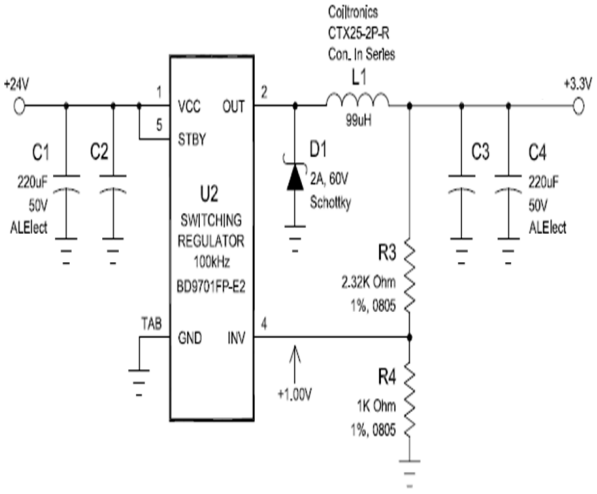


Fig. 6 DC/DC Converter

VII. Firmware Design Summary

The original scope of the project was to create a device that was designed around two SCADA protocols, Modbus and DNP3. Modbus was considered an industry standard for the last 30 years. It allows for a master controller to control many slave devices. For the purpose of this project a master control center controls many waste water lift stations. Modbus uses a modified 5-layer data model. The 5-layer model is very similar to what is used in the current internet. It takes a standard TCP/IP data stream and modifies it to carry the device id, function code and relevant data. Figure 7 shows what a typical Modbus TCP data stream looks like.

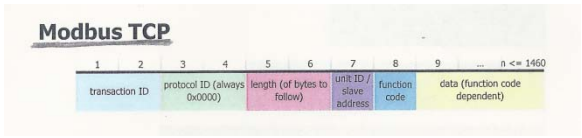


Fig. 7 Modbus Data Stream

DNP3 is a newer SCADA standard that has found a home mainly in the electric utilities. Only recently has it begun to move into waste water utilities. Benefits include the increased security provided by specialized security objects in the data stream as well as being Wi-Fi capable. DNP3 used the same modified 5-layer model. Along with the increased security measures, it has the capability to change the Modbus master-slave relationship to a master-master relationship. This allows messages to be sent and received by the device and the master control center. The functionality of DNP3 far exceeds that of Modbus, Modbus has 10 function codes that might be applicable to this device where DNP3 has closer to 20 function codes that are applicable.

The DNP3 gateway device has a detailed firmware design. The model that was chosen was the waterfall model figure 8. We chose this because it was a lifecycle model that wouldn't slow down the development of the product by constant testing every step taken and yet it still provided a nice outline and flow overall. The design team could follow the waterfall life cycle step by step up to the maintenance which is where it would hand off the product to the sponsor, MPE to then be given to the next senior design group.

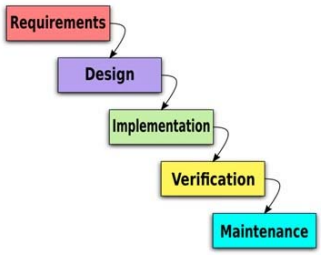


Fig. 8 Waterfall Lifecycle Model

The fact that this model had sequential progression through its steps allows one to easily work through the design process. While generally this method is considered more for hardware, the firmware team found that it would fit the project's purposes quite well solely on the time factor. The team would not have the more than 4 months to code all the firmware for this project. The thought that anyone would have a working model and it would need an improvement in four months is just unrealistic. After that time frame, MP electronics would have the product and the design team would not even be involved with it anymore. The other perk of using this model is it was very easy to integrate into the life-cycle of the project overall, which had a lot of hardware design too. The team wasn't even able to start the process until the hardware had been designed anyways so it ended up connecting the firmware life cycle between the design and the implementation portion of the hardware design and then connected the two at the verification step.

For the "requirements" stage the team sat down as a group once the hardware was designed and wrote out what features needed to be implemented to make the controller function at basic levels. In addition to the features, the limitations of both the hardware itself had to be worked out. This included how many resources were available and the limitations of the development environment.

At the start of the spring semester it was determined that the scope of this device was more than what one senior design group could handle. The scope of the firmware was modified to account for this change.

A. New Requirements

The DNP3 gateway device will have complete functionality over the 3 buttons (scroll up, scroll down and menu select). The buttons will control the values displayed on the 3 7-segment displays. The parameters will be stored in RAM for access while the device in on but will be stored in EEPROM in the event of power loss to the device.

With the new requirements outlined we needed to redesign our high level firmware flow. We started with an overall picture of what the firmware needed to do. When the device powers on it first loads the initialize LED group then it initializes the parameters for the device. After parameters are loaded the device waits for the button input. This is illustrated in figure 9.

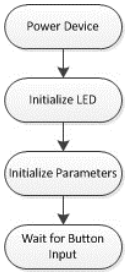


Fig. 9 High level firmware design (re-designed 1/2013)

B. Initialize LED Design

The initialize LED routine is broken into 5 parts. First the Initialize Latch Objects is called. This function sets all of the latches used in the seven segment display circuit. It sets them into reset or low for our purpose. This allows the latch to initialize clear of any data. It then initializes the seven segment display object function. This function goes to set all of the A-G and DP (decimal point) led points to low to they start in the off position. Initialize LED objects resets the display to clear any lingering data that was found on load up. This is a final check and clearing of data before information is fed to the LEDs. Set LED state is called to turn the power LED on to show the device is active. Initialize display is a function that cycles each of the 3 7-segements through a count 0-9. This is a verification routine on power-up to make sure all the LED's are working. Figure 10 illustrates the LED design flow.

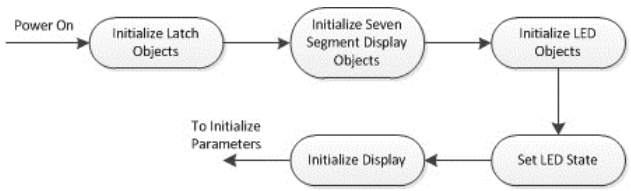


Fig. 10 LED 1 Design

C. Initialize Parameters Design

The initialize parameters routine is 2 parts but is a foundation of the entire project design. When the device enters this routine it checks a flag that is stored at the start of the Parameters data structure in EEPROM. On the first power up this value will not be set. With this value unset the device runs the parameters first load function which fills this flag and the parameters in RAM and EEPROM. On subsequent power-ups the flag value will be set and it will load the parameters from EEPROM allowing the device to meet one of the crucial requirements of storing parameter changes. Figure 11 highlights the direction of the data in this routine.

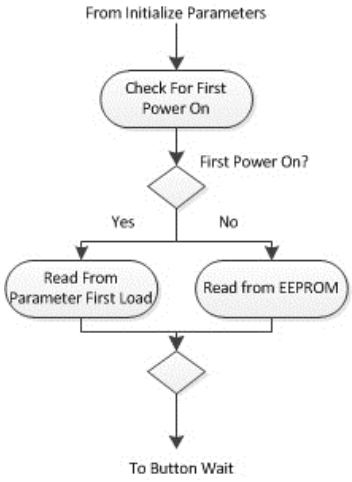


Fig. 11 Initialize Parameters Diagram

D. Final Design Requirements

Once everything has been initialized and its waiting for button input the system operates as an event-driven system. On button press (scroll up/down) the LED will display the next or previous parameter in the list while the Menu/Scroll LED light will be illuminated. When the Menu button is pressed the LED will display the stored value for the parameter. This value is only modifiable after holding down the scroll up or down button for 4 seconds. This triggers program mode within the program. The user is able to adjust the parameter value at this point. Once they press the menu button again the LED display

reverts back to the parameter list and displays the current parameter. It also calls the EEPROM write function and stores the changed value in the current parameter structure.

VIII. Conclusion

The Modbus to DNP3 Protocol Converter, phase I proved to be a formidable challenge for all involved and gave each team member valuable experience in their respective fields, whether electrical or computer engineering. Working closely with MPE, we gained helpful insight on how a real-world engineering project is taken from the drawing board to the physical product.

The result is a device into which a protocol converter that meets all of the sponsor's specifications can be programmed. At the heart of the device is a robust and sophisticated microcontroller that has more than enough memory to store a time-stamped sequence of events. It is interfaced successfully with an Ethernet input and output, and the end user can interact with the system easily with a user interface that's in line with other products by MPE. By all accounts, the next group should be in a position to hit the ground running and see the protocol converter through to completion.

Acknowledgements

The group would like to extend a special thank you to all of the engineers at Motor Protection Electronics for working so closely with us and giving us instant feedback on our progress. John Evans came up with the concept of the protocol converter from its earliest stages and managed its design through phase I, generously providing us with all the necessary resources. Chris Parker gave invaluable insight on which parts to choose, how to lay out a PCB design, soldering technique, and practical engineering advice. Steve Pickles helped to create a timeline and forced us to nail down concrete goals. He also shared with us his wealth of experience in firmware coding.

We would like to also thank the review committee, currently consisting of Dr. Michael Haralambous and Dr. Zakhia Abichar, for taking time out of their schedules to come and see what we've been working on for the last two semesters. We thank to Dr. Arthur Weeks for his in-depth education on a variety of topics, without which the group would have been lost. We thank Dr. Samuel Richie, as well, for his guidance through the last couple of semesters.

Biographies

Joshua Daly is a 30 year old Computer Engineering student. Josh is finishing his second bachelor's degree at UCF. He has worked as a developer for almost 4 years at a local healthcare company. He is moving to Pittsburgh, PA after graduation after accepting a position with PNC bank as a Senior Software Engineer in their asset and wealth management division.

Daniel Doherty is a 24 year old Computer Engineering student. He is finishing his first bachelor's degree at UCF and currently interning at C3Pathways working on designing training simulations for emergency responders. He specializes in software programming and is planning on staying with C3Pathways after graduation. He is moving to San Diego, CA after graduation for a year and working remotely.

Joseph Wilkinson is a 30 year old Electrical Engineering student graduating Magna Cum Laude in the Spring Semester of 2013. His interest is in modeling feedback control systems

Mac Lightbourn is a 23 year old Electrical Engineering student with an interest in power systems. He plans on returning to his home country of The Bahamas to pursue work as an electrical engineer.