

Remote Water Metering in Venecia, Nicaragua

Jeffrey Wettstein

ECE 499 Capstone Design

Advisor: Professor Hedrick

March 17th 2015



Executive Summary:

The senior Capstone Design Project is the culmination of over three years of undergraduate work in electrical engineering, and other related fields. The project allows students to demonstrate their knowledge on a tangible design. Starting from a blank slate, the Capstone process asks students to carefully move through every stage of design in order to create the most effect output.

Water shortage is a major problem for the entire world including the United States, but particularly in developing countries like Nicaragua. The community of Venecia, Nicaragua has a unique water problem. Community members leave taps running in fear that if they are shut off, they will not turn back on. This behavior leads to the overconsumption of a very valuable and scarce resource. One way to reduce water consumption is to make the user aware of just how much water they are sing. The system discussed in this paper is a prototyped design developed in conjunction with community leaders of Venecia. The final prototype provides real-time remote metering of water to demonstrate the benefits of conservation. Data is collected and processed at the meter before it is transmitted wirelessly to a display site. At the display site, users can access real-time flow information as well as totalized values. This system will be installed in Venecia later this spring.

Table of Contents

<u>SECTION</u>	<u>PAGE</u>
EXECUTIVE SUMMARY	2
LIST OF TABLES AND FIGURES.....	4
INTRODUCTION	5
BACKGROUND	7
DESIGN REQUIREMENTS	12
DESIGN ALTERNATIVES	17
PRELIMINARY PROPOSED DESIGN	20
REVISED DESIGN	27
FINAL DESIGN AND IMPLEMENTATION	32
PERFORMANCE RESULTS	42
PRODUCTION SCHEDULE	46
COST ANALYSIS	48
USER’S MANUAL	49
DISCUSSION AND CONCLUSIONS	51
REFERENCES	53
APPENDIX.....	55

List of Tables, Figures, and Equations

Type	Page	Description	Type	Page	Description
Figure 1	11	Photo of Venecia, Nicaragua	Figure 12	28	RS-232 Voltage Levels
Table 1	12	Preliminary Design Requirements	Figure 13	28	RS-232 Shield
Figure 2	15	Amplitude Shift Keying	Figure 14	30	Packet Protocol
Figure 3	15	Frequency Shift Keying	Figure 15	32	Meter-Side System Diagram
Figure 4	17	EKM Water Meter Error Plot	Figure 16	33	Meter Test Setup
Figure 5	20	EKM 2" Pulse Output Water Meter	Figure 17	34	Packet Modem and Handheld Radio
Figure 6	21	Hall Effect Sensor Diagram	Figure 18	34	Sample Arduino Output
Figure 7	21	Hall Effect Output Plot	Figure 19	35	Arduino Output Waveform
Equation 1	21	Flow Rate Calculation	Figure 20	35	RS-232 Shielded Arduino
Figure 8	22	Arduino Uno Microcontroller	Figure 21	36	Display-Side System Diagram
Figure 9	23	433 MHz RF Transmitter	Figure 22	37	Display Code (Baud Set)
Figure 10	26	Preliminary System Block Diagram	Figure 23	37	Display Screenshot
Table 2	27	Revised Design Requirements	Figure 24	38	Display Code (Parsing Scheme)
Figure 11	28	USB Voltage Levels	Figure 25	40	Final Display Screenshot
			Figure 26	41	Display Code (Quitting)
			Figure 27	43	Periodic Arduino Output
			Figure 28	46	PERT Chart
			Table 3	48	Cost Analysis

Introduction:

Engineers Without Borders (EWB) is a group whose goal is to develop relationships around the world and help create physical and social improvements to their environment. Often times, projects that EWB take on deal with supplying water to groups who have limited access. The Vermont Professional Chapter has taken on a project in Venecia, Nicaragua. The community currently has no existing method to monitor the consumption of water. This lack of information causes households to continuously run their water, in fear that it won't turn back on. The purpose of this project was to build a device that convinces households that there is water, and as long as it is used appropriately, it will not run out. By simply presenting the metering data, I hypothesize that this will be enough to change water use behaviors and substantially lower overall consumption. Although water metering and data collection is common, this project encompasses more than just that. It aims to trigger changes in water use, leading to a more sustainable and flourishing community.

To accomplish the goals outlined above, I built a system that obtains water-metering data, and presents it in a fashion that can be understood by all. Obtaining the metering data requires a piece of hardware, which allows water to pass through it and simultaneously "counts" the quantity of water. Devices like this exist in the form of pulse output water meters. The meter is installed in line with the existing pipe, and uses a magnetic sensor to generate a pulse each time a known quantity of water flows through. The main difference between this type of meter and a generic water meter is the ability to read the metering data from a distance. With metering information available it will need to be conditioned and processed in a way that it can be sent wirelessly to a pre-determined site. Sending stand-alone pulses wirelessly is challenging. It can be difficult

to differentiate them from other wireless noise that inherently exists. This is the reason for the data conditioning stage. Radio frequency, or RF, employs radio waves to accomplish the task of transmission. The frequency of these radio waves will be chosen to meet requirements like transmission distance, and avoidance of interference. After the data has reached it's final destination, some processing needs to occur. The processing unit will extract the conditioned information from the radio signal and transform it back into usable data.

This report will begin by giving a brief background of water metering and it's applications throughout time. These applications encompass aspects of economics, environmental sustainability, and social trends. Next, more details regarding the design requirements will be introduced. These design requirements give the details necessary to select useful, and implementable components. This section will also provide insights into the major functions of the system. Through a literature search, this report will give a justification for the approach of building the system, component by component, as well as the overarching goal of reducing water usage. Finally, the report will outline the how the system was constructed, and some of the details that went into accomplishing the design requirements.

Background:*History of Water Infrastructure*

The idea of supplying a community with water distant from the epicenter of population has been tinkered with for thousands of years. In early civilization, it was advantageous to live at higher elevation. It was easier to protect inhabitants with an uphill position. This was not always necessary, which allowed communities to form close to natural bodies of water like springs, rivers, and aquifers. At higher elevations new methods had to be developed to get water to citizens. The Romans were first in developing large-scale water distribution systems. Networks of supply tunnels fed central water towers, where it could be rationed to the community (Bromehead 184). In late 19th century America, the need for an adequate supply of drinking water was growing. Population was rising, commercial establishments were forming, and new public health standards were introduced. Cities like Boston experienced political tension when deciding whether supplying water would be a private, or public matter (Broomehead 193). With the growth of water infrastructure in the United States and around the world, suppliers began charging based off water consumption. In order to accomplish this, devices were needed to keep accurate records for each home, spawning the current era of water metering (Melosi 243).

Origins of Water Metering

A water meter is not unlike a gas or electricity meter. It is a device that provides the supplier with information to determine how much to charge customers. In recent decades, there has been a change in mentality regarding water supply. The business of domestic water supply has evolved from a public service, to a revenue-making endeavor.

In order for the companies supplying the water to make money, it has become more common to monitor households' water use. For some, this was beneficial. Before metering, high-value properties paid higher monthly water rates than lower value properties. If a high-value property were not using a lot of water, a meter would enforce this and lower their monthly cost of water. On the contrary, households that used a lot of water now were required to pay for the high use (OFWAT, Water Meters).

Although the main implementers of water meters tend to be private water suppliers, there has also been an increase in metering for other purposes. In Brazil, metering data provides customers with reports containing water usage patterns, and projected savings due to changes in consumption (Lima). In developing countries like Brazil, the data compiled by metering can be helpful in raising awareness of overconsumption. Two classifications of water metering can now be established. Metering for profit, and metering for the advancement of developing regions of the world. The aim of this project is to employ the second of the two classifications and provide Venecia with useful information regarding water use.

Economic Benefits and Drawbacks of Metering

Countless studies have been conducted to determine the economic usefulness of water metering. A study conducted by Aquacraft Inc, and the National Research Center examined water use in metered apartment buildings vs. non-metered buildings. The data showed that metered buildings used 15% less water per year (Apartment Meters). This study supports my hypothesis that providing water usage data leads to a decrease in overall consumption. A research project performed by Dutch researchers Harrison Mutikanga and Saroj Sharma showed that water metering in Kampala, Uganda reduced

water usage by 18%. Kampala is a developing city in Africa that previously had no means to measure water consumption.

Regulations have not been fully established to enforce how third parties would charge for water consumption. In multi-family homes, landlords are not given specific regulations to follow if they wish to sub-meter each tenant. The implications of this are large. This means that landlords could potentially profit off their tenants water usage by overcharging (Apartment Meters). To inhibit this situation from occurring, state level policies must be established that would not allow landlords to profit of tenant's water use. Fortunately this project does not involve any aspect of charging for water use. The primary goal is reduction in water consumption and highlighting to potential environmental benefits that water metering can bring to the table.

Environmental Impacts of Metering

Global water consumption has tripled since 1950. In 2010, the yearly consumption rate was estimated to be over 4,000 cubic kilometers. This mammoth number can be attributed to the fact that populations have risen in parts of the world past a sustainable limit (Postel 2332). While water scarcity is mainly a problem for developing countries, the United States has spent billions of dollars trying to tackle the challenge of preserving this necessary resource (Lundeen 4).

The primary consumer of water is agriculture, accounting for more than two thirds of all consumption. The economy in Nicaragua is focused on the agricultural sector. 15% of the countries exports are coffee. Among beverages, coffee requires the most water to produce. A study claims that over 1,000 liters of water goes into producing

one liter of coffee after growing, processing, and preparation is taken into account.

Venecia is a coffee producing community, meaning that a lot of water is used to sustain their current lifestyle. If water prices were to increase or a shortage was to occur, it would be difficult for the community to sustain the quality of life they have now. By metering the water consumed for drinking, bathing, and cooking, the overall consumption can be cut down; which would leave more water for agriculture (Water Use: Thirsty Work).

Engineers Without Borders

Engineers Without Borders is an organization that began in 2000. The goal of the organization is to collaborate with community partners to design and build sustainable engineering projects in the United States, and internationally. The Vermont chapter of EWB took on a project in Venecia, Nicaragua to help deal with issues surrounding water. More specifically, these issues include water shortages and water quality. Today, the community has the necessary infrastructure set up to address these issues. Although the problems surrounding quality have been solved, water shortage still affects the community. In collaboration with EWB-VT, I hope to assist in solving the water shortage problem that Venecia is facing. A photo of Venecia is shown below in Figure 1.



Figure 1

Design Requirements (ECE 498):

The table below outlines the design goals and how each will be accomplished. More details regarding each goal can be found below the table with justifications for each choice of component.

Design Requirement/Goal	Means of Accomplishing
Obtain metering data in a form that can be processed	Implement a meter that can withstand maximum flow rates of up to 8L per second. Meter must have a readable output such as a pulse
Calculate flow rate from pulse output of water meter and create a data package.	Implement Arduino code that uses pulse frequency to determine flow rate. The Arduino will output data at an interval chosen by the user. Data will be packaged with error-checking information
Transmit data 400 meters to display site	Use RF wireless transmission at 433MHz. The transmitter will use ASK to encode the data into the radio wave.
Receive data	RF receiver paired with transmitter will detect signals at a frequency that has been chosen for transmission.
Decode data and prepare for display	A microcontroller will be implemented to obtain the data out of the RF signal. The data will then be converted from binary form back to decimal, and stored/displayed for use in water consumption analysis.

Table 1

The method that water meters use to measure flow rate can vary from meter to meter. There are three types of velocity-based water meters. One type of velocity-based meter uses turbines. Turbine meters are typically used for applications where the volume of flow will be quite high. Fire hydrant water meter use the turbine method. The

advantage to this system is that it can handle very high flow rates, while the disadvantage is that it is typically only used for pipes that measure about 12 inches across. The second type of velocity based water meter is called a compound water meter. A compound meter is used when the velocity of water can vary from low to high. The device has two different measuring components. One is for high flow rates, while the other is used for low flow rates. To switch between the two measuring components, a valve is used to direct the water to one of two channels. The last type of velocity-based meter uses either electromagnetic or ultrasonic principles. The electromagnetic meter is based on fluid induction principles and the ultrasonic meters uses sound waves to obtain a water volume reading. The benefit of this type of meter is that it can also be used for large volume measurements. The disadvantage is that they are typically used for fluid other than water (Lima).

While the methods of measuring flow listed above do work well, the type of meter I implemented was a volume based turbine meter. When the water flows through, an impeller spins. The impeller has a magnet on it that passes by another magnet located on a stationary part of the meter. When the two magnets pass by each other, a pulse is generated. This effect is called the Hall effect. A pulse is generated each time a known amount of water passes through, allowing the user to calculate the total flow. The benefit of single and multiple jet volume meters is that they typically do not clog. This is important, as a low failure rate is necessary.

The turbine meter I used also has a pulse output. The pulses correspond to a known amount of water, and can be counted to calculate a total flow rate over time. To count the pulses or calculate a flow rate, a microcontroller was implemented. The

microcontroller's duty was to calculate pulse frequency, then use the known information about the meter to determine flow rate. Attaching other information to data is a common practice is wireless information transfer. Protocols are developed in order to effectively accomplish the task of error-free data transfer.

During my visit to Venecia in December, I got a chance to do wireless testing. This was an important step in determining how much power would be needed to transmit over 400 meters without errors. The testing involved handheld radios that had a large frequency range, and multiple power output settings. I stood at the metering side by the collection tanks and asked someone to carry the other radio to the display site. Using three different power settings and many different frequencies in the hundreds of MHz range, it was determined that 500mW was sufficient to transmit voice the required distance. All frequencies tested (from 120MHz up to 440MHz) successfully accomplished the task of transmission. Knowing this information, I was able to use a similar power output and frequency in the range tested in Venecia. Due to Nicaragua's developing state, the equivalent of our FCC has not yet allocated all usable frequencies. The radios implemented in the final design will be programmable, which will allow for adjustments in the transmitting frequency if necessary.

After an appropriate frequency was selected, the mode of transmission was considered. Amplitude shift keying (ASK) and frequency shift keying (FSK) are two common types of wireless transmission. An ASK transmitter uses the amplitude of the wave to encode the data. Figure 3 below shows the methodology used in ASK. ASK is more susceptible of noise corruption because noise is generally additive. Since the data is encoded in the amplitude of the waveform, noise has an easier time corrupting the signal.

FSK uses frequency to code data. Figure 4 below shows how FSK works when transmitting a binary signal. FSK is typically more reliable because noise is less likely to corrupt the frequency of the signal, rather than the amplitude. This device will be located in a relatively remote region of Nicaragua, meaning there will be less interference from other signals to corrupt the transmission used for this project.

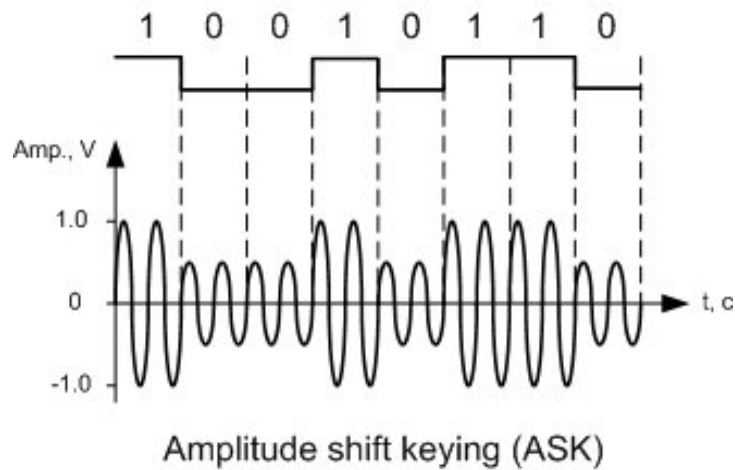


Figure 2

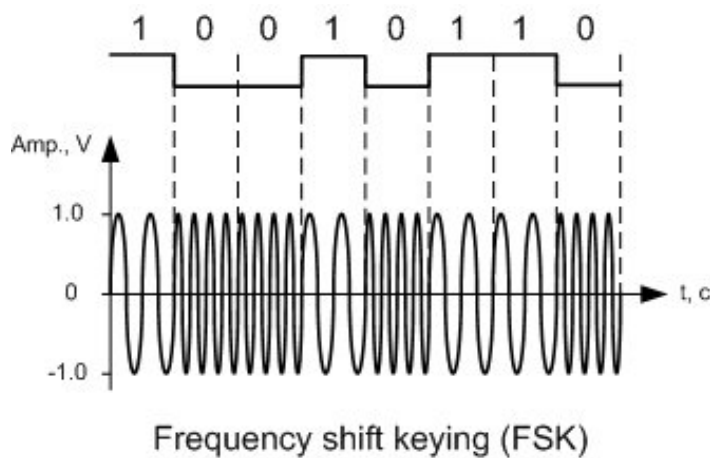


Figure 3

After the data has been transmitted, it will need to be received and displayed. Receiving the data will be as simple as having a radio operating on the same frequency. To reverse whatever transmission protocol was used, processing will occur after the radio. The display needed to accomplish a few tasks. It needed to create a structure for the data, and put it on a screen in a way that was clear to the user.

Design Alternatives:

EKM 2" Pulse Output Meter

The typical flows that the turbine meter will encounter in Nicaragua range greatly. The meter will be placed after the primary collection tank, inline with the pipe that carries water into the community. The collection tank is about 40m^3 and can empty in 45 minutes when the valve is fully open. This yields a maximum flow rate of approximately 8 liters per second. Figure 4 below was taken from the data sheet of the meter that will be used. The meter's overload flow is 30m^3 per hour or 8.33L per second. This means that at maximum flow, the percent error will still be within reason. Ideally, the meter would measure with a percent error within the American Water Works Association accepted range, which is 2-3.5%.

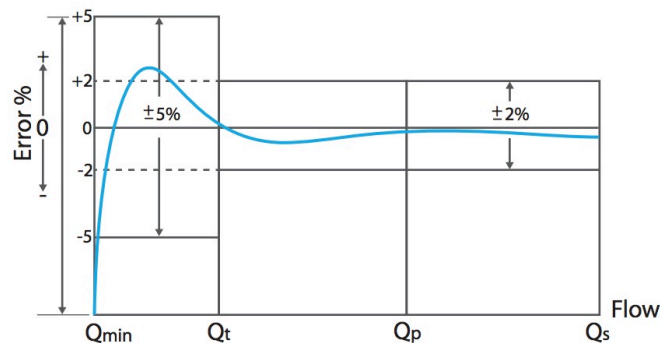


Figure 4

The justification behind the choice to use the pulse output meter manufactured by EKM involved several criteria to be considered. The first being that it was the most

affordable meter that fit my design constraints. These constraints included things like typical flows, reliability, accuracy, and size. The GPI TM Series meter was an alternative I was considered. Unfortunately, this meter was made from PVC and uses potentially dangerous solvents to attach to the pipe it sits inline with. Also, the pulse output that it generates relies on an external power source instead of a Hall effect sensor. This means that the batteries would need to be replaced up to every 6 months. The meter by EKM is constructed out of stainless steel and does not need a power supply to operate.

Arduino Uno Microcontroller

One of the main reasons why the Arduino was selected was my previous experience working on that platform. I am familiar with the coding language used, allowing me to create the script without much background research. This time saved can be allocated elsewhere, in order to create a better system. The task of receiving pulses and calculating flow rate is also very simple. Although the Arduino Uno is not a very robust microcontroller, it was able to perform this task sufficiently to provide the information at an acceptable rate.

433MHz RF Transmitter

When first considering the method of wireless transmission to use, many options seemed viable. Wi-Fi was one of those options. Unfortunately, the village that the system will be located in does not have any Wi-Fi connections I could tap in to. Also transmitting at Wi-Fi frequencies typically requires more power, since the signals are at a significantly higher frequency (RF Propagation Basics). The next option I considered was radio frequency, or RF. There are particular bands of the RF spectrum set aside for

applications like this one. The decision to use a 433MHz transmitter comes from the fact that this frequency is set aside for scientific projects. As long as the transmission power is below a certain limit, it can be used without having to obtain a license. Radio frequency devices also seemed to be less expensive than Wi-Fi components, which is another factor that led me to select a 433MHz RF transmitter.

Preliminary Proposed Design (ECE 498):

A photo of the EKM meter is shown below in Figure 5. The meter is 2" in diameter, which suites the pipe that it will be installed inline with. In Figure 4, the typical flow rates are shown for the selected meter. The maximum flow rate as preciously mentioned is approximately 8 liters per second. This rate corresponds to just below the overload flow of the EKM meter. At the overload flow, the meter accuracy begins to diminish greatly. The meter will not see flows greater than this after installation, meaning the accuracy will remain sufficient and adhere to AWWA standards. The EKM meter selected also generates a pulse output. It does this with a Hall effect sensor. Figure 6 below shows the mechanics behind a Hall effect sensor. The rotating disc shown corresponds to the magnet on the turbine, while the second magnet is located on the inner wall of the meter. When the magnet located on the turbine passes the sensor, a pulse is generated. Sample output from a Hall effect sensor can be seen in Figure 7. The EKM meter generates a pulse for every 0.01m^3 . Knowing the pulse output to flow relationship allows for a flow rate to be calculated using time. Equation 1 below shows how the pulses and collection are calculated to find flow rates.



Figure 5

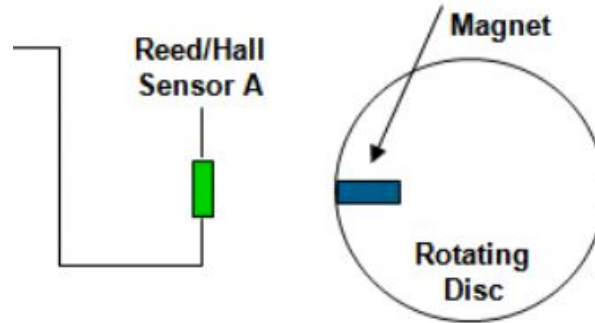


Figure 6

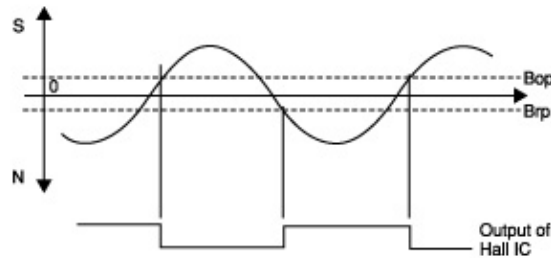


Figure 7

$$\text{Flow Rate} \left(\frac{m^3}{\text{second}} \right) = \frac{\text{Number of Pulses}(m^3)}{\text{Time of collection}(\text{sec})}$$

Equation 1

The calculation to determine flow rate will occur at the metering stage of the system. To do this calculation, a microcontroller will be implemented. The microcontroller selected for stage is the Arduino Uno. Figure 8 below is an image of this microcontroller. The Arduino platform offers many libraries of built-in functions. I plan on taking advantage of a pulse counting function. With the ability to keep track of the

number of pulses seen by the Arduino, the next important step will be keeping track of time. It will be important to ensure that each time interval will be the same so the data is as accurate as possible. Once the flow rate has been calculated, it will be stored in a variable that is updated to reflect the total amount of water that has gone through the meter. Approximately every 30 minutes, the data will be outputted where it will be processed and conditioned before being sent to the display.

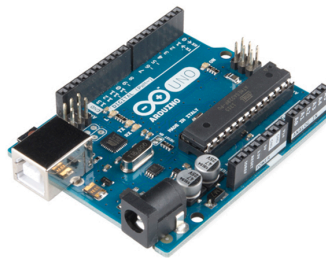


Figure 8

Preparing the data for transmission means developing a unique transmission protocol. The protocol will be developed to suite this system specifically. The data itself will only be one part of the entire data package that will be transmitted. The data package will include information that tells the receiver what type of information it is, error checking information, and instructions on how to handle the data. The protocol that I will implement has four bytes. The first byte is an identifier. Its purpose is to identify itself to the receiver so it can be handled in the correct way. The different kind of identifier packages could be data, error, or a request for maintenance. After the identifier byte, the instruction byte will follow. Based off what type of information it is, this byte will tell the receiver what to do with it. For example, if the identifier byte tells the receiver that the

information contained is data, the instructions will say to decode the data and display the flow rate. The third byte will be the data itself and the final byte will be for error checking. Error checking can be done with parity bits, or more complex methods. For this project, parity or a method similar to parity should suffice. This protocol will be uploaded to the microcontroller located near the meter and prepared for transmission.

To transmit the data, I have selected a 433MHz RF transmitter. The maximum distance of this transmitter is 2000m, which satisfies the requirement of 400m given in the design requirements section. Figure 9 below is a photo of the transmitter, along with the paired receiver. The transmitter uses approximately 20mW at 5VDC and has a working voltage range of 3V-9V. The transmitter will send the total flow for each collection period from the Arduino and use ASK to transmit the signal. Due to the remoteness of the project site, I do not anticipate significant interference from other devices using this frequency band. The sensitivity of the receiver is -105dBm which means there can be approximately 80dB of loss between the transmitter and receiver and the signal can still be received uncorrupted. The receiver module is powered with 3-5VDC and has an antenna that is 18cm long. It will be important to position the receiver to give it the best chance to receive the signal.



Figure 9

Once the data is received, I will interface it with another computer. The purpose of this computer is to take the transmitted waveform and reform the data that has been encoded in the amplitude of the waveform. The data will mostly likely be coded into the signal as a binary representation of the flow rate. For example, if the flow rate was 20m^3 per hour, that would correspond to 10100 in binary. Using this coding means that the only processing that needs to be done on the receiver side is to take the binary representation and convert it back to a decimal representation. Another Arduino board could be implemented to do this. There are built-in binary to decimal converters that I could utilize to perform this task. Once the data is back in decimal form, it will need to be displayed. The display I will utilize for testing will most likely be a LED display or 8-bit display. When the system is fully tested and implemented, it will mostly likely be connected to a computer so more involved analysis can be performed. The scope of this project encompasses testing and not final implementation; therefore using a small display for proof of concept will suffice.

To effectively test the system, multiple stages of testing will need to be used. First, it will be important to do water meter accuracy and reliability testing. To do this, some sort of bench top unit will need to be built. The unit will need to be small enough where it can be moved around if needed, but still be able to thoroughly test. The tests that will occur on the meter will involve different flow rates, including very low rates. This is where most meters are the least accurate, so it will be important to test this case thoroughly. To control the flow rate, a known amount of water will be located at one end of a pipe, and depending on the size of the hole that is opened; this will correspond to

different flow rates. For example, a large opening the size of the entire pipe will result in a higher flow rate, while a smaller opening will test the lower flow rate. It will be important to calibrate this test set up in order to ensure the results that were being obtained are reliable enough to base the study off. To calibrate the test set up, a known amount of water can be placed in the container. A timer will let us know how long it takes for all of the water to flow through, and then using the two known values, a flow rate can be calculated. The tests of flow rate should be realistic, and simulate the flow rates that the meter will see once it is implemented.

After successfully testing the sensor, testing the wireless transmission system will be next. In order to test this, data will need to be sent to the transmitter. If the first stage of testing is complete, this could be real data referencing flow rate. If the first part of testing is not complete, pulses could be generated. The receiver will initially be located very close to the transmitter to ensure that the data is not corrupted at all in transmission. After it is confirmed that the data is making it to the receiver, more rigorous tests can be used. For example, distance tests can be completed to find the maximum, unobstructed distance that the transmitter is capable of handling. After the distance tests are complete, obstruction tests would be done next. It would be important to test the penetration of different common materials, including trees, walls, and other common building materials. Finally, it will be important to test the battery life of the transmitter.

An overall system diagram can be seen below in Figure 10. The top section of this figure outlines the design goals of the project. The bottom section gives the components that will accomplish the goals above. The water from the main collection tank will flow through the EKM meter, where the pulses will be processed and transformed into data to

be transmitted by the Arduino Uno. Next, the 433MHz RF link will transmit the data using ASK. The data will be received and processed by another microcontroller to prepare it for display and analysis.

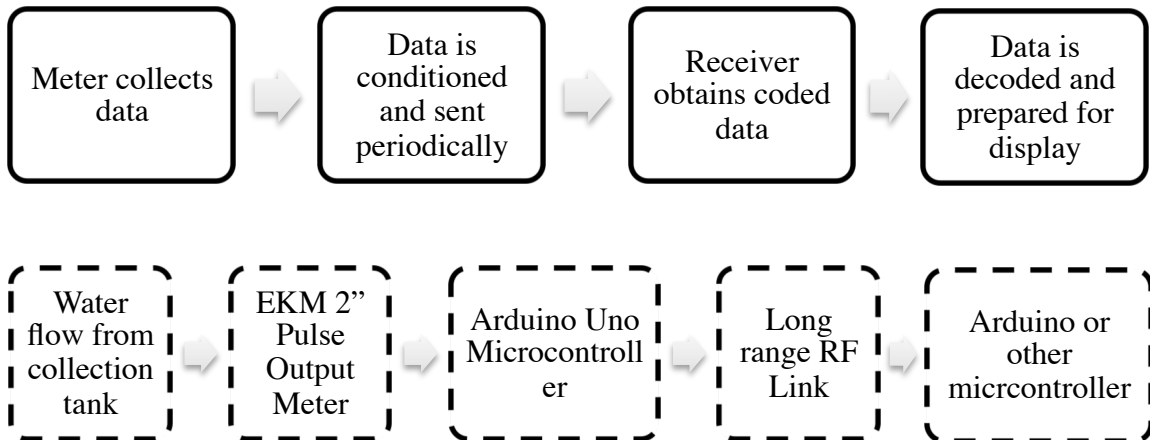


Figure 10

Revised Design (ECE 499):

When building the prototyped system, opportunities and issues were encountered along the way resulting in some changes in the original proposed design. The changes are highlighted in the revised design requirement table below as well as the justifications for doing so.

Design Requirement/Goal	Means of Accomplishing
Obtain metering data in a form that can be processed	Implement a meter that can withstand maximum flow rates of up to 8L per second. Meter must have a readable output such as a pulse
Calculate flow rate from pulse output of water meter and create a data package.	Implement Arduino code that uses pulse frequency to determine flow rate. The Arduino Mega will output data at an interval chosen by the user. Interface using RS-232 shield. Data will be packaged by Kantronics KPC-3 packet modem with error checking and other identifying information
Transmit real time flow data 400 meters to display site	Handheld radios configured to use FSK modulation and a frequency of 145.05MHz
Receive packet and extract the desired data	Paired receiver and modem obtain and unpack the data using the additional information passed along by modem.
Display real time current flow information as well as totalized flow data	Use algorithm written in C to take in data one character at a time through the RS-232 port and display in NCURSES window

Table 2

RS-232 Shield for Arduino Mega 2560

In order for the Arduino to talk to the Kantronic modem, they need to be speaking the same language. In hardware communications, voltage levels define language. The Arduino uses USB voltage levels, where 5V corresponds to a binary 1 and 0V corresponds to a binary 0. In serial communications, it is common for a binary 1 to be represented by +10V or +12V and binary 0 to be -10V or -12V. A figure of serial communications levels is shown below in Figure 12. In order for the devices to communicate, a level converter needs to be implemented. The converter used in this design can be seen below in Figure 13.

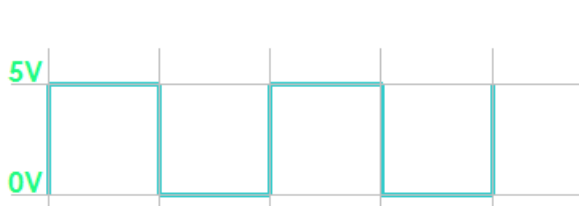


Figure 11

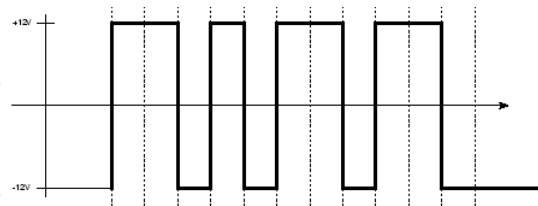


Figure 12

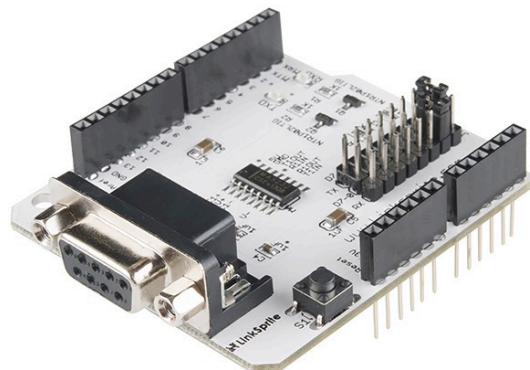


Figure 13

The level converter also provides a convenient RS-232 port. With the level converting shield on the Arduino, there are now two output ports. It is important to configure the device to set the primary output port as the RS-232 port since the original USB output will not be used. Universal asynchronous receiver/transmitters or UARTs are responsible for sending data to ports. Due to the limitations of the Arduino board, a primary UART needs to be established. The default setup uses the USB UART for transmitting data. To configure the device to set the RS-232 UART as the master, an assignment in the Arduino script can be made to change the primary UART on the device. In order to use the shield, the Arduino Mega 2560 was substituted in for the Arduino Uno.

Kantronics KPC-3 Packet Modems

For the system to be effective, data needed to be transmitted without errors. The Kantronics packet modems provided a commercial-grade protocol that is able to handle real-time rates. The modems were also provided, which allowed for financial resources to be allocated elsewhere. To build a protocol that would result in error rates as low as the Kantronics modems one would need to dedicate a lot of time, which was simply not available to me. This modification to the system lowered costs, increased functionality, and allowed for more time to be spent on more critical stages of the design.

After the packet modem receives the data, it follows a protocol called x.25 to package it with error checking information and other tags that allow for delivery confirmation to occur between the paired modems. Figure 14 below shows what a typical

packet contains when it leaves the Kantronics modem. A stream of data is often sent in a short burst of multiple packets.

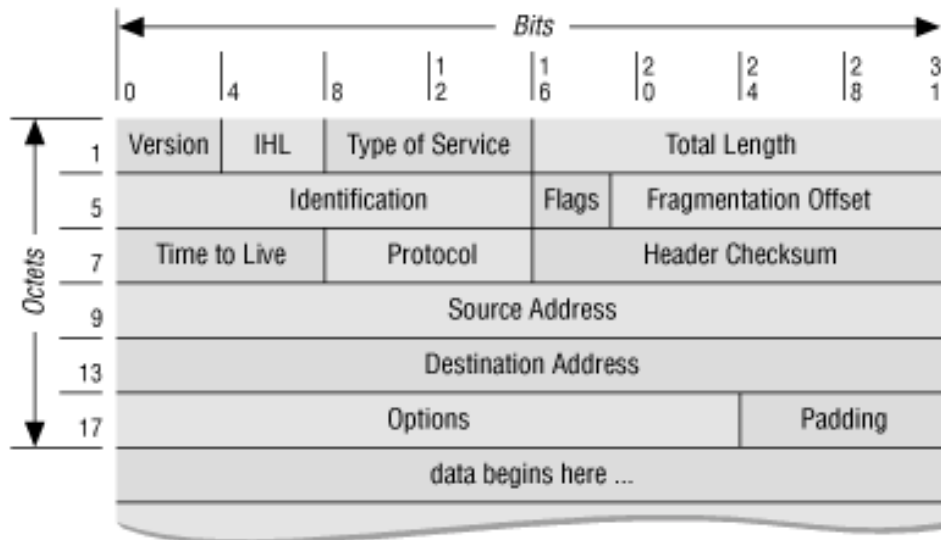


Figure 14

As shown above, a packet contains much more than just the data. To oversimplify, the packet contains six key pieces of information. The first is a time stamp of when the data is transmitted. This is useful if timing is critical in a system, or the system is using a network of sensors. For this design no synchronization needs to occur, so it is not as useful. The next piece is information to check for errors once received. The method of error checking used in this particular configuration is a checksum scheme. Some of the more popular checksum schemes include parity and redundancy (Linux Information Page). Two addresses are included in the packet: one for the source, and one for the destination. This information is useful when operating in a region that may have a large volume of traffic on the chosen frequency. I don't anticipate interference from any other packet modem, so this information will simply be along for the ride. The fifth

chunk of information passes the configurable parameter settings like baud rate. After all of that information has been packaged, the data gets attached. As previously mentioned, the data is often broken up and sent in many different packets.

Handheld FSK Modulating Radios

My original design called for two inexpensive fixed power and frequency transceivers. Having adjustable, reliable, and easy to use radios was very important in the testing process. The handheld radios provided all of that. I was able to adjust parameters like transmit power, frequency, and transmission method (FSK or ASK). Since these radios were only used for the prototype, I was given them to use for testing with no cost.

Data Display

Displaying both the real-time flow information and totalized flow information can be done infinite ways. During my site visit, I received input from community members about how they would like to see the data displayed. The consensus was that a simple window that updates automatically would be sufficient. Instead of opting to buy commercial software to do this, I decided to write the program myself. By writing the program I was able to save on cost, as well as have total access to display parameters in case something needed to be altered in the future. Commercial software is often closed source and expensive.

Final Design and Implementation:

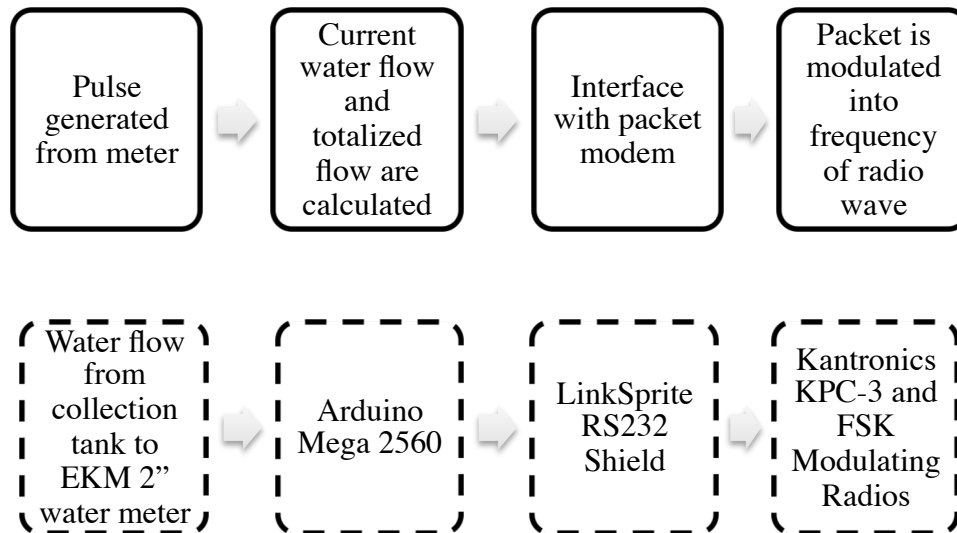


Figure 15

To effectively present the design used in the final implementation, the system has been split up into two primary sections: meter-side and display-side. The functional block diagram for the meter-side subsystem is shown above in Figure 15. The solid boxes represent the tasks that are completed at each stage, while the dotted boxes give the hardware or software that will perform each task. The meter used for implementation is the same as described in the preliminary proposed design section. When the meter arrived, a test setup was constructed in order to generate pulses for system testing. This test setup is shown below in Figure 16.



Figure 16

The meter was connected directly to an Arduino Mega 2560 board. The flow rate code shown in the Appendix was uploaded to the board. To do this calculation, the code initializes the relationship between one pulse and total flow as 0.1 pulses for every liter of flow/second. The code tracks the pulses on the falling edge of a state change from the sensor using the “interrupt” feature. The interrupt feature will stop the program, add to the **pulseCount** function, then return to it’s normal routine whenever a pulse is detected. It establishes the pin corresponding to the Hall effect output as data pin 2. After the initializing is complete, the code enters its main loop. Each iteration of the main loop runs for 1000 milliseconds to ensure timing accuracy. To calculate the flow rate for each second, the number of pulses are counted then divided by the 0.1 pulses for every liter of flow/second factor. Therefore, if 2 pulses are counted in one second, the flow rate is 20 liters/second. The flow rate for each second is saved in a variable that is called **currentFlow** and that same value is added to the variable **totalFlow**. The pulse counter is then reset to zero, so the loop can start again.

The packet modem and one of the handheld radios is shown below in Figure 17. To interface the Arduino with the Kantronics packet modem, the RS-232 shield was soldered to the Arduino board. Sample Arduino output is shown below in Figure 18, along with a waveform representation in Figure 19 demonstrating that the level converting function of the RS-232 shield was operational. A photo of the shielded Arduino board is shown below in Figure 20.



Figure 17

Flow rate: 9.9L/sec Output Liquid Quantity: 38
L

Figure 18

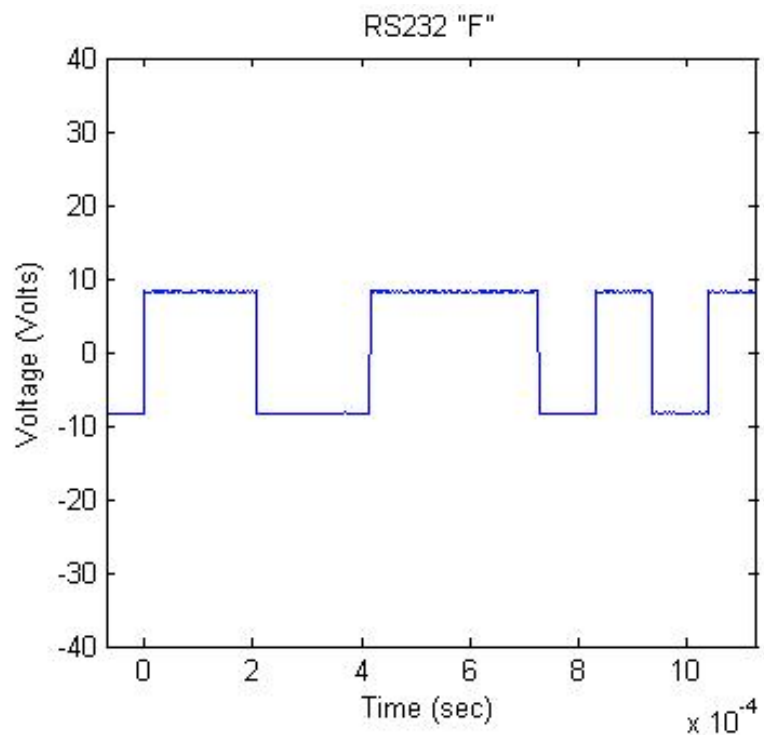


Figure 19

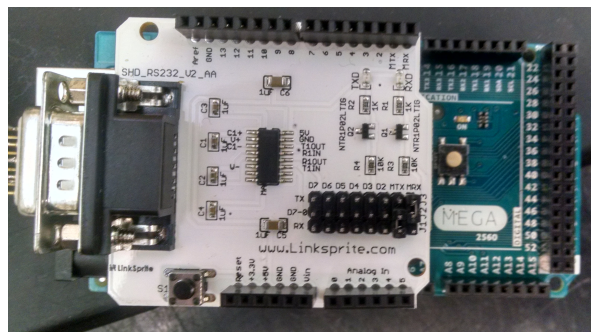


Figure 20

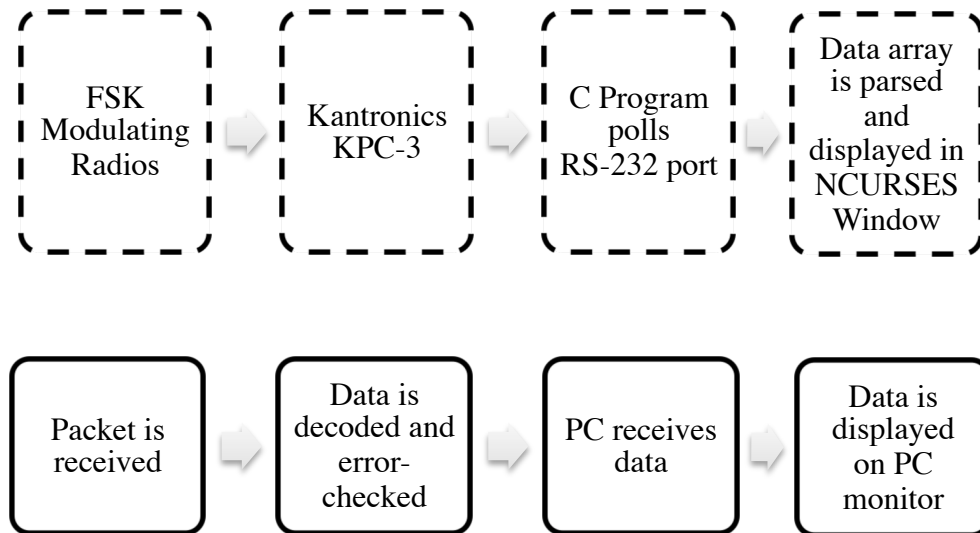


Figure 21

On the display side of the system, a paired radio and modem were used to receive the signal and decode the packet to extract the desired information. With the data in its raw form, a PC pulls it in. A script running on the PC displays the data. A block diagram of the display-side subsystem is shown above in Figure 21.

The program written to take the data from the modem and display it can be found in the Appendix. The algorithm begins by initializing a program called `NCURSES`. Instead of using a small LED display as described in the preliminary proposed, `NCURSES` allows for a PC monitor to be utilized and a window to be constructed for displaying information. The port is defined to tell the computer where to pull information from, then opened to initialize the flow of information. The baud rate, or rate at which information is flowing, is set to match the rate of the Kantronics modem. If the baud rates

do not match, it is possible that errors will occur in the information exchange process.

Figure 22 below is a snippet of code that includes the command to set the baud rate.

```
/* set baud rate */
cfsetispeed(&new_flags, B9600);

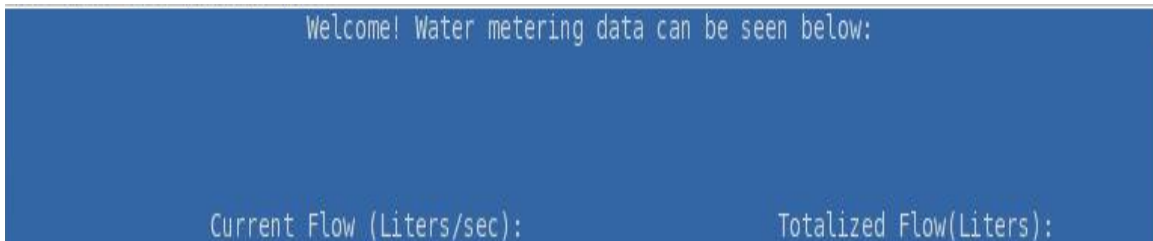
move(0,25);
printw("Welcome! Water metering data can be seen below:\n");
refresh();

move(5,17);
printw("Current Flow (Liters/sec):\n");
refresh();

move(5,64);
printw("Totalized Flow(Liters):\n");
refresh();
```

Figure 22

Also shown in Figure 22 above are three commands to print information to the window. The **move** command tells the program where to put the text, and the **printw** command prints it to the window. The result of these commands can be seen below in Figure 23.



```
Welcome! Water metering data can be seen below:

Current Flow (Liters/sec):                Totalized Flow(Liters):
```

Figure 23

Now that the window is configured to display the data, the algorithm enters the main program loop. This loop will run constantly until the user decides to exit. The pseudo code for the main program loop is as follows. An empty data array called **in_msg** is created to house the data before it is parsed for display. The algorithm will only parse **in_msg** when it is the length of a full data message. To determine what constitutes a full message, the length of the sample output shown in Figure 18 must be considered. When the total flow is less than 10L, the length of the message is 47. If the total flow is greater than 10L but less than 100L, the length is 48. If the total flow is greater than 100L, the message is 49 characters long. The code to handle each of these situations is shown below in Figure 24.

```

// parse message
//refresh();
if (strlen(in_msg) == 47){
    strncpy(current_flow, &in_msg[11], 3);
    current_flow[3] = 0x00;
    move(6,28);
    addstr(current_flow);
    // refresh();

    move(6,73);
    strncpy(total_flow, &in_msg[44], 2);
    total_flow[2] = 0x00;
    addstr(total_flow);
    refresh();
}
if (strlen(in_msg) == 48){
    strncpy(current_flow, &in_msg[11], 3);
    current_flow[3] = 0x00;
    move(6,28);
    addstr(current_flow);
    //refresh();
    move(6,73);
    strncpy(total_flow, &in_msg[44], 3);
    total_flow[3] = 0x00;
    addstr(total_flow);
    refresh();
}
if (strlen(in_msg) == 49){
    strncpy(current_flow, &in_msg[11], 3);
    current_flow[3] = 0x00;
    move(6,28);
    addstr(current_flow);
    refresh();

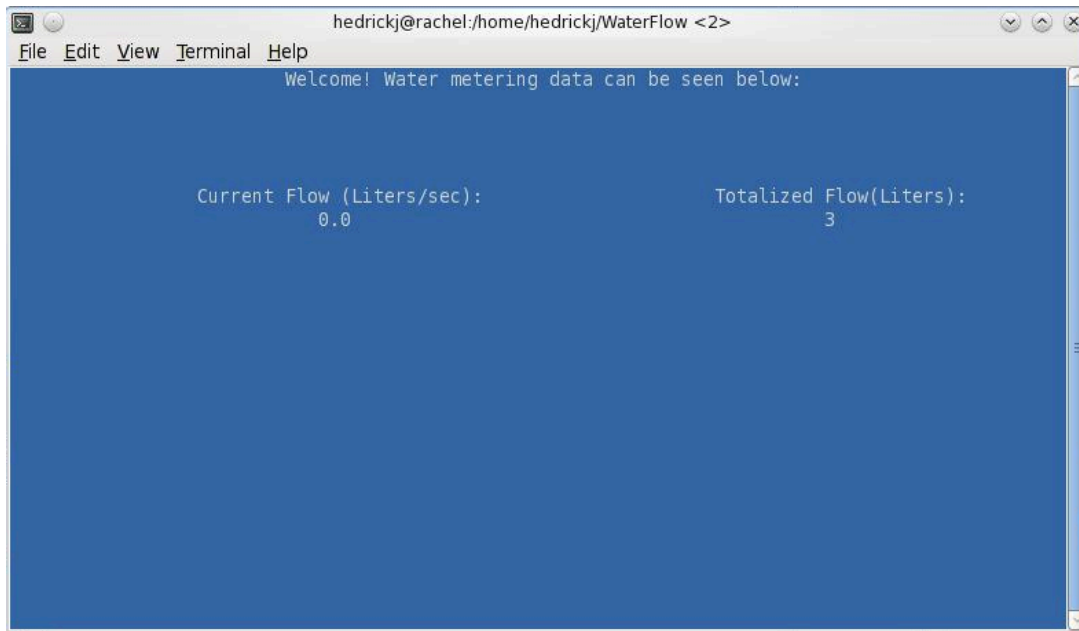
    move(6,73);
    strncpy(total_flow, &in_msg[44], 4);
    total_flow[4] = 0x00;
    addstr(total_flow);
    refresh();
}

```

Figure 24

With conditions to handle realistic values developed, the parsing can begin. To parse the information, two new empty arrays were defined. The first, **current_flow**, displays the real-time current flow information. The second, **total_flow**, displays the real-time totalized flow data. Since each of these values is located at different positions in the **in_msg** array, each must be handled separately. The current flow information is located in positions 11 through 14 in **in_msg**, **strncpy** allows for one section of an array to be copied; therefore indices 11 through 14 are copied to **current_flow**. After the information is copied over, a null character is added to the end of **current_flow** to tell the computer that the string complete. The parsing of **current_flow** does not change in any of the conditions shown above.

As discussed earlier, parsing the totalized flow information does change in each condition. If the value is less than 10L, only two characters need to be copied to **total_flow** to represent the number and the space before it. In Figure 24 above, it is easy to see how the section parsed in the **strncpy** command for **total_flow** changes for each length condition. Null characters were added to the end of **total_flow** to signal the end of the string. After the strings were copied, the **move** function positioned the cursor to the desired location on the window. Instead of using the **printw** function used to print a non-varying string to the window, **addstr** was used. After any information was added to the display window, it needed to be refreshed using the **refresh** command. The result of this parsing scheme can be seen below in Figure 25. The screenshot looks similar to Figure 23, but the key data has been extracted from the data sentence shown in Figure 18.



```
hedrickj@rachel:/home/hedrickj/WaterFlow <2>
File Edit View Terminal Help
Welcome! Water metering data can be seen below:

Current Flow (Liters/sec):      Totalized Flow(Liters):
      0.0                          3
```

Figure 25

To exit the main program loop and stop displaying data, the user can enter a “Q” on the keyboard. Clicking the X in the corner does close the window; but the program may continue running and can cause some errors. The section of the algorithm that properly exits the program is shown below in Figure 26. After each iteration of the main loop, the program checks to see if any characters have been entered from the keyboard. If there was a character entered, and it was a “Q”, the program closes the connection to the RS-232 port and exits the window.


```
// check keyboard
    /* get a character from the keyboard */
    ch = getch();
    if (ch != ERR){
        printf("%c ", ch);
    }
    if (ch == 'Q') {

        /* tcsetattr(port, TCSANOW, &old_flags);*/
        close(port);
        endwin();
        return(0);
    }
```

Figure 26

Performance Results:

Flow Rate Calculation

It was estimated that flow rate could be calculated accurately with the Arduino Uno board. The proposed algorithm used a built-in pulse counting function to track the meter output and calculate a totalized flow rate to be transmitted approximately every 30 minutes. In the final design, a change in hardware required the use of an Arduino Mega 2560 board. Along with substitution, there was a change in the methodology behind calculating flow rate. Instead of using a pulse counting function, the final prototype utilized the interrupt feature described above. This feature allowed for the program to keep track of both current flow and totalized flow. With both of these values now available, the system became more robust. These two pieces of information were also sent to the transceiver at a rate of 2-3 times per second. This was crucial in order to attain real-time data display. Figure 27 below shows the Arduino output every 300ms. One unanticipated result was the reliability of the connection between the meter and Arduino board. The thin gauge wires were difficult to deal with, and sometimes resulted in inaccurate data. To deal with this issue, the ends of the wire were tinned with solder. A more permanent solution, like wire nuts, is something that will be explored in the future work.

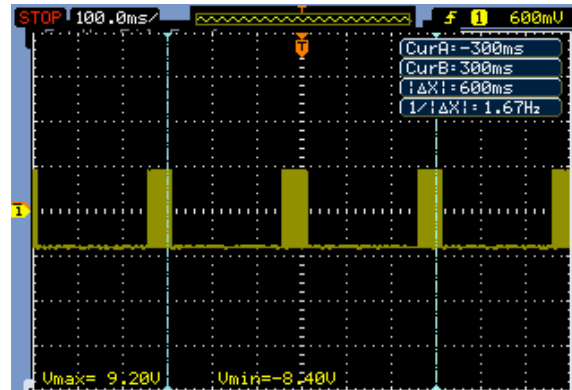


Figure 27

Wireless Transmission Protocol

The proposed design discussed a wireless protocol and estimated that it could provide error-free communications. The anticipated error-checking method highlighted the use of parity bits to confirm that the data had been transmitted without errors. The proposed design also talked about the limitations of this type of protocol, and how real-time data transfer would not be practical to implement because of processor limitations. In the final design, the wireless protocol was provided by the Kantronics modems. These modems used the same error-checking scheme originally proposed to ensure reliable communications. The biggest difference between the estimated performance and actual performance regarding data packaging and protocol was the ability to do real-time information transfer. The Kantronics modems provided the hardware and software needed to transmit the two outputted variables from the Arduino immediately after receiving them. The use of the packet modems also allowed for the overall system cost to decrease, since they were gifted for use in this system.

Transmitter/Receiver Performance

The original design called for two low cost transceiver units. The units were at a fixed frequency and power, which would limit my ability to test different methods of wireless transmission. Handheld radios were used in the final prototype for more flexibility in testing the system. The estimated results were that the data would be transmitted using ASK because of its ease of use and anticipated lack of other radio interference in the region. With the handheld radios I was able to test both ASK and FSK, as well as a wide range of frequencies in the hundreds of MHz range. From testing, it was concluded that FSK would be a better choice because of its reliability and resilience. This addition to the system made it more reliable, and allowed for more thorough testing.

Before my visit to the site, I did not have a clear picture of the transmission medium. I proposed that the 2km range of the 433MHz transmitters would have been sufficient. In actuality, more power was required. The final prototyped design used the 500mW setting on the handheld radios. As part of the future work, programmable radios will be implemented. When installing the system, I will be able to adjust power levels in order to attain maximum efficiency.

Information Display

The proposed design called for a barebones LCD display for testing. The estimated results regarding the display was that whenever data was received (approx. 30min intervals), the display would show an updated totalized volume of water. The actual results were significantly better than this estimate. Because of the packet modems more time was spent developing a display, which could not handle the real-time

information that was being received. Both the current flow measurement and updated totalized flow values could be displayed. These values were also updated each time a new measurement was received (approx. 300ms intervals).

Production Schedule

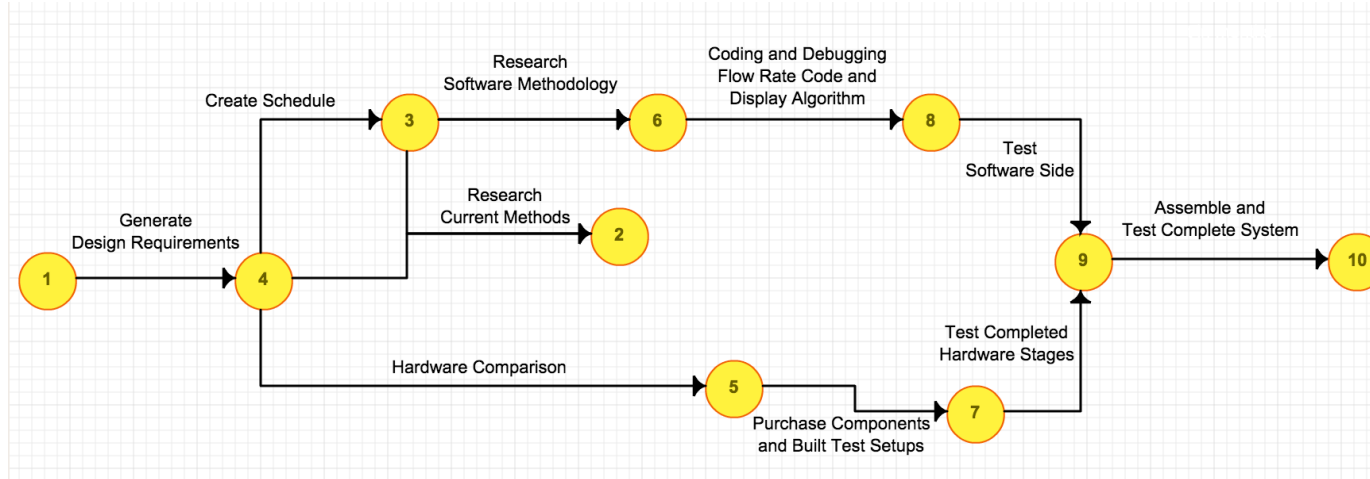


Figure 28

Figure 28 above is a graphical illustration of the steps taken to complete this prototyped system. In ECE 497, most of the time was spent learning how to generate effective design requirements. The generation of design requirements coincided with creating a timeline of how things would be accomplished in ECE 498 and 499. While this was occurring, it was important to research similar projects to understand the methodology used in their solutions. Many times the applications would be slightly different than the one desired, but this was useful in determining how to best accomplish the design requirements. After the design requirements were generated, component research and comparison began. Based on the knowledge gained from researching current methods, this process was more precise. During this research phase, software strategies were examined.

After the hardware and software strategies had been selected, the work on the system began. On the software side, this means writing pseudo code and generating

algorithm outlines. For hardware, components were tested individually using test setups. Once the individual testing was complete, the pieces were connected together and software was uploaded to the appropriate devices. More stages of testing occurred before the entire system was tested. Finally, the entire system was tested and debugged. Because of the unanticipated changes in some hardware components, the original schedule developed in ECE 497 was not always followed. To improve this, it would have been valuable to include time in the original schedule for unexpected changes to the overall system design.

Cost Analysis

Expenditure Description	Cost
EKM 2" Pulse Output Meter	\$520.00
Kantronics KPC-3 Packet Modem*	2 x \$98.00
Arduino Mega 2560	\$13.73
LinkSprite RS-232 Shield V2	\$11.90
2 inch PVC Piping	\$8.22
PVC Cement	\$5.41
Teflon Tape	\$1.48
PVC Fittings	\$2.19
PVC Valves	\$5.41
Total (With Kantronics Modem)	\$764.34
Total (Without Kantronics Modem)	\$568.34

Table 3

The initial proposed budget was \$406.24. The final cost was \$568.34, 28.5% higher than proposed. To satisfy my design requirements, a different meter was required. Without the change in meter included in the analysis, the project cost was actually over 30% under budget.

The marketability of this system is quite good. A commercially available system would require the purchase of expensive, closed source software. This software may offer more features, but the user is typically limited to predetermined settings from the manufacturer.

User's Manual

To operate the prototyped remote metering system, the user must follow a few simple steps before data is displayed. The first step is to power all of the devices. The devices that require the user to turn them on are the handheld radios, shielded Arduino board, packet modems, and the PC on the display side. Next, confirm that the frequency on both handheld radios is set to 145.05MHz. The dial on top of the handheld radio allows the user to navigate frequencies. (If this particular frequency is not ideal for the application, the user may change to any desired as long as it does not violate and communications laws established by the FCC or comparable entity). On the display side, the meter output should be connected to the Arduino. To do this, connect the **red** wire to **pin 2** and the **white** wire to **ground**.

The next step is to initialize a connection between the two packet modem links. To accomplish this, the user should first open up a terminal window and begin a program called Kermit. A Kermit session is started by navigating to the "home" directory of the computer and typing "Kermit" into the command line. Kermit is an open source piece of software used for serial communication devices. Once Kermit is open, the user can then connect with the device connected to the PC. The command **c** initializes the connection. If the computer has not been used to connect with the packet modem beforehand, it may be necessary to set the baud rate. Typing **baud rate 9600** into the command line after connecting with the device does this. To begin talking with the packet modem located on the meter side, the user will type **c** followed by the callsign of the second modem. This will initialize and call and response protocol between the two modems. Once the devices are connected, the **CONN** light on each modem will be solid green. To begin speaking to

one another, each modem will need to be placed into **converse** mode. Typing **convers** in the command line of Kermit accesses this mode. The meter side modem will automatically be in **converse** mode, so no further configuration is required. Before moving forward, ensure that data is being transmitted from the meter side. If the Arduino is wired correctly, the user should see flow rate information appearing in the Kermit window.

To run the script, the user should first close out of the Kermit program. This requires two commands to be typed by the user: CTRL + \, followed by “quit”. This closes the connection that Kermit has established with the modem. This step is very important because the data collection program requires the full attention of the modem. In the terminal, navigate to the directory that contains the program called **getdata**. Run the program by typing, “run getdata.c”. This command should open a new window similar to the one shown in Figure 25. As soon as the modem on the display side receives any data, the information will appear on the screen.

Troubleshooting:

- If a connection between modems cannot be established:
 - First ensure both modems and radios are powered **on**
 - Check for **matching frequencies** between radios
 - It may be necessary to **squell** any noise. To adjust this, use the secondary knob on top of the radio
- If no data is being displayed:
 - Ensure the Arduino is powered properly
 - Check that the **CONN** light is green on **both** modems
 - Confirm that Kermit has been closed **properly** before running **getdata**
- If data is erratic:
 - Check the connection between the meter output and Arduino
 - It may be necessary to tin the ends of the wires using some common household solder to create a better connection

Discussion, Future Work, and Conclusions

Successfully monitoring water remotely in a remote region like Venecia required a unique, custom solution. The pulse output of the water meter was piped to a shielded Arduino microcontroller that calculated current flow rates, as well as a totalized value using a program designed specifically for this application. Wireless data transmission was accomplished by borrowing hardware that was made popular by amateur radio operators. Another program was written exclusively for this system to display the data once it had been received. Tailoring every piece of the system for Venecia may have resulted in more work, but will provide the community with exactly what they asked for.

The prototype will undergo more changes in the spring to ensure that it is reliable and effective for the people of Venecia. Meter side components will have to be weatherproofed as well as powered. Enclosures exist that provide electronics with the necessary protection against all types of inclement weather. Providing power to the components on the meter side will most likely utilize the ample amounts of sunlight at the metering site. More robust radios will be used instead of the handheld radios described in this report. The Ritron radios are multi-channel as well as fully programmable. The ability to add sensors to this system is very important, as discussions to expand the metering network are underway. Finally, I would like to add some more features to the display. Trending, and historical data access are two that seem especially useful.

By participating in the capstone design project, I was able to expand on the basic skills I learned in the classroom while at Union. These skills include specifying design

requirements, selecting components, and constructing an entire system. I was pushed to make decisions on my own, which is not a comfortable experience initially. Capstone gave students experience in time management, budgeting, and actual implementation of our work.

Although this project was ultimately an engineering project, I feel that I gained most through my interactions with people in Nicaragua. During my site visit, the community members welcomed me to Venecia with open arms. I was given the unique opportunity to interact with the end-users, while simultaneously performing testing that would help in the final design of the system. Before the visit, I anticipated that the community members would have a list of specifications and design requirements of their own. After walking around the community and receiving a tour of their water system, it was clear that no specific design requirements had been discussed. They had one goal in mind, and it was to give their families the necessary means to continue living. I returned to Union with heightened enthusiasm, as well as some anxiety, knowing that most of the critical design specifications would be left for me. Although learning skills like C programming and serial communications may come in handy some day, it was these anxiety-inducing decisions that gave me the most.

This project would not have been possible without Engineers Without Borders, the entire ECE department and the support from both of my parents. My advisor, Professor James Hedrick, was more than just an excellent engineering resource during this yearlong endeavor. His experience in the social realm of engineering and wealth of compassion played a larger role in the completion of the project than he will ever admit.

Work Cited:

- A. Chambouleyron, "An Incentive Mechanism for Decentralized Water Metering Decisions," *Water Resources Management*, vol.17, pp. 89-111, 2003.
- Hall Effect Output. Digital image. *AKM*. N.p., n.d. Web. 16 Nov. 2014. <http://www.akm.com/image/gr-outline-12_1.gif>.
- Amplitude Shift Keying(ASK)*. Digital image. *TMAntlantic*. N.p., n.d. Web. 15 Nov. 2014.
- "Apartment Meters Cut Water Use 15%: Study." *Contractor Magazine* 51.10 (2004): 52. *Business Source Premier*, Web. 15 Nov. 2014.
- Bromehead, C.E. "The Early History of Water-Supply." *The Geographic Journal* 99.4 (1942): 183-93. *JSTOR*. Web. 15 Nov. 2014.
- C. Lima, "Smart Metering and Systems to Support a Conscious use of Water and Electricity," *ENERGY*, vol 45, pp. 528-540, 2012.
- "Checksum Definition." Checksum Is a Simple Method of Detecting Errors in Data. Linux Information Page, 4 Nov. 2005. Web. 13 Mar. 2015.
- "EkoTek Low Power Radio Mesh Networking," *EkoTekWeb*, <<http://www.lotsab.se/wp-content/uploads/2011/07/Low-Power-Radio-Mesh-Networking.pdf>>.
- Frequency Shift Keying (FSK)*. Digital image. *TMAntlantic*. N.p., n.d. Web. 15 Nov. 2014.
- G. Richards, M. Johnson, and S.Barfuss, "Metering Secondary Water in Residential Irrigation Systems," *American Water Works*, vol. 100, pp. 112-21, 2008.
- H. Mutikanga, S. Sharma, and K. Vairavamoorthy, "Investigating Water Meter Performance in Developing Countries: A Case Study of Kampala, Uganda," *Water S. A.*, vol. 37, pp. 567-74, May 2011.
- Hall Effect. Digital image. *EET*. N.p., n.d. Web. 16 Nov. 2014. <http://m.eet.com/media/1160906/flow_metering_tutorial_2_fig6.jpg>.
- Kulkarni, P.; Gormus, S.; Zhong Fan; Motz, B., "A mesh-radio-based solution for smart metering networks," *Communications Magazine, IEEE* , vol.50, no.7, pp.86,95, July 2012.
- Lundeen, Tim. "Reducing Water Scarcity." *Feedstuffs* [Chicago] 8 Sept. 2014, 86th ed., sec. 36: 1+. Web. 15 Nov. 2014.
- Melosi, Martin V. "Pure and Plentiful: The Development of Modern Waterworks in the United States, 1801–2000." *Water Policy* 2.4-5 (2000): 243-65. *ScienceDirect*. Web. 15 Nov. 2014.
- "Metering systems." *WaterWorld* vol. 44, Web. 8 May 2014.
- Postel, Sandra. "Water Scarcity." *Environ. Sci. Technol* 26.12 (1992): 2332-333. *ACS Publications*. Web. 15 Nov. 2014. <<http://pubs.acs.org/doi/pdf/10.1021/es00036a600>>.

"RF Propagation Basics," *Sputnik*, Web. 14 May 2014,
<https://www.sputnik.com/resources/support/deployment/rf_propagation_basics.pdf>.

Sadi, Yalcin; Ergen, Sinem Coleri; Park, Pangun, "Minimum Energy Data Transmission for Wireless Networked Control Systems," *Wireless Communications, IEEE Transactions*, vol.13, no.4, pp.2163,2175, April 2014.

Water Meters: Your Questions Answered: Information for Household Customers. Birmingham: Ofwat, 2010. OFWAT, 2010. Web. 15 Nov. 2014.

"Water Use: Thirsty Work." *The Economist*. N.p., 25 Feb. 2009. Web. 15 Nov. 2014.

Appendix:

Arduino Flow Rate Calculation Code:

```
1. byte statusLed    = 13;
2.
3. byte sensorInterrupt = 0; // 0 = digital pin 2
4. byte sensorPin     = 2;
5.
6. // The hall-
   effect flow sensor outputs approximately 0.1 pulses per second per liter/second
7. float calibrationFactor = 0.1;
8.
9. volatile byte pulseCount;
10.
11. float flowRate;
12. unsigned int flowLiters;
13. unsigned long totalLiters;
14.
15. unsigned long oldTime;
16. void setup()
17. {
18.
19. // Initialize a serial connection for reporting values to the host Serial.begin(38400);
20. // Set up the status LED line as an output
21. pinMode(statusLed, OUTPUT);
22. digitalWrite(statusLed, HIGH); // We have an active-low LED attached
23. pinMode(sensorPin, INPUT);
24. digitalWrite(sensorPin, HIGH);
25. pulseCount      = 0;
26. flowRate        = 0.0;
27. totalLiters     = 0;
28. oldTime         = 0;
29.
30. // The Hall-effect sensor is connected to pin 2 which uses interrupt 0.
31. // Configured to trigger on a FALLING state change (transition from HIGH state to LOW state)
32. attachInterrupt(sensorInterrupt, pulseCounter, FALLING);
33. }
34.
35.
36. /**
37. Main program loop
38. */
39. void loop()
40. {
41.
42. if((millis() - oldTime) > 1000) // Only process counters once per second
43. {
44. // Disable the interrupt while calculating flow rate and sending the value to
45. // the host
46. detachInterrupt(sensorInterrupt);
47.
```

```

48. // Because this loop may not complete in exactly 1 second intervals we calculate
49. // the number of milliseconds that have passed since the last execution and use
50. // that to scale the output. Also apply the calibrationFactor to scale the output
51. // based on the number of pulses per second per units of measure (liters/second
52. // in this case) coming from the sensor.
52. flowRate = ((1000.0 / (millis() - oldTime)) * pulseCount) / calibrationFactor;
53.
54.
55. oldTime = millis();
56.
57.
58.
59.
60. // Add the liters passed in this second to the cumulative total    totalliters
61. += flowRate;
61. unsigned int frac;
62. // Print the flow rate for this second in liters / second    Serial.print("Flow
63. // rate: ");
63. Serial.print(int(flowRate)); // Print the integer part of the variable
64. Serial.print(".");           // Print the decimal point
65. // Determine the fractional part. The 10 multiplier gives us 1 decimal place.
66.     frac = (flowRate - int(flowRate)) * 10;
66.
67. Serial.print(frac, DEC) ;    // Print the fractional part of the variable
68. Serial.print("L/min");      // Print the number of liters flowed in this second
69.
70.
71. // Print the cumulative total of liters flowed since starting    Serial.print("
72. // Output Liquid Quantity: "); // Output separator
72. Serial.print(totalliters);
73. Serial.println("L");
74.
75.
76. // Reset the pulse counter so we can start incrementing again    pulseCount = 0
77. // Enable the interrupt again now that we've finished sending output    attachI
78. // nterrupt(sensorInterrupt, pulseCounter, FALLING);
78. }
79. }
80. /*
81. Interrupt Service Routine */ void pulseCounter() { // Increment the pulse co
82. unter    pulseCount++; }

```


Data Collection and Display Code:

```

1.  /*
2.  getdata.c
3.  J. Wettstein
4.  02/20/2015
5.  get flow data from RS232 port and display with NCURSES
6.  */
7.  #include <ncurses.h>
8.  #include <termios.h>
9.  #include <unistd.h>
10. #include <signal.h>
11. #include <stdlib.h>
12. #include <stdio.h>
13. #include <fcntl.h>
14. #include <string.h>
15.
16.
17. #define MESSAGE_LENGTH 256
18. #define PORT_NAME "/dev/ttyS0"
19. #define TRUE 1
20. #define FALSE 0
21. #define LF 0x0A
22.
23. int main()
24. {
25.     int port;
26.     int portchar;
27.     char char_cnt;
28.     char in_msg[MESSAGE_LENGTH + 5];
29.     char current_flow[20];
30.     char total_flow[20];
31.     char num_str[25];
32.     char in_ch;
33.     int i, ch, lf_flg;
34.     struct termios old_flags, new_flags;
35.     initscr();
36.     start_color();
37.     init_pair(1, COLOR_WHITE, COLOR_BLUE);
38.     wbkgd(stdscr, COLOR_PAIR(1));
39.
40.
41.
42.
43.
44.     /* open the RS-1232 port */
45.     if ((port = open(PORT_NAME, O_RDWR | O_NDELAY | O_NOCTTY | O_NONBLOCK )) == -
1)
46.     {
47.        printw("Error opening RS232 port\n");
48.         refresh();
49.         exit(-1);
50.     }
51.
52.     /* set up raw/non-canonical mode */
53.     tcgetattr(port, &old_flags);
54.     new_flags = old_flags;
55.     new_flags.c_lflag &= ~(ECHO | ICANON | ISIG);
56.     new_flags.c_iflag &= ~(BRKINT | ICRNL | IXOFF | IXANY);

```

```

57.  new_flags.c_oflag &= ~OPOST;
58.  new_flags.c_cc[VTIME] = 0;
59.  new_flags.c_cc[VMIN] = 1;
60.  new_flags.c_cflag |= CS8;
61.  if (tcsetattr(port, TCSANOW, &new_flags) < 0) {
62.      printf("Error setting raw mode!! \n");
63.      exit(-1);
64.      endwin();
65.  }
66.
67.  /* set baud rate */
68.  cfsetispeed(&new_flags, B9600);
69.
70.
71.  move(0,25);
72.  printf( "Welcome! Water metering data can be seen below:\n");
73.  refresh();
74.
75.  move(5,17);
76.  printf("Current Flow (Liters/sec):\n");
77.  refresh();
78.
79.  move(5,64);
80.  printf("Totalized Flow(Liters):\n");
81.  refresh();
82.
83.
84.  /* set input for non blocking */
85.  nodelay(stdscr, TRUE);
86.
87.  char_cnt = 0;
88.  lf_flg = FALSE;
89.  /* receive the data message and print it to the screen */
90.  for (;;) {
91.      i = read(port, &portchar, 1);
92.      if (i > 0) { // START CHARACTER FOUND AT rs-232 PORT
93.
94.          if ( (((char)portchar > 0x1f) && ((char)portchar < 0x7f)) || ((char)port
char == LF ) ){
95.              // if character read is not a LF store in in_msg
96.              if ((char)portchar != LF){
97.                  /* character read from the RS232 port is not a Line Feed
98.                   so store in the message array */
99.                  in_msg[char_cnt] = (char)portchar;
100.                  char_cnt++;
101.                  /*addch((char)portchar);*/
102.                  //refresh();
103.                  // check to see if there was no second LF
104.                  if (lf_flg == TRUE){
105.                      lf_flg = FALSE;
106.                  }
107.              }else { // start handle LF
108.                  // line feed found replace with NULL and zero char count
109.                  /* handle line feed. If the first one set flg and check to see
110.                   if the next character is also a line feed */
111.                  if (lf_flg == FALSE){ // start first LF
112.                      in_msg[char_cnt] = 0x00;
113.                      lf_flg = TRUE;
114.                      char_cnt = 0;
115.

```

```

116.         } else { // handle second LF
117.             //ignore second LF
118.         }
119.     } // end of handle LF
120.
121.
122.     // parse message
123.     //refresh();
124.     if (strlen(in_msg) == 47){
125.
126.         strncpy(current_flow, &in_msg[11], 3);
127.         current_flow[3] = 0x00;
128.         move(6,28);
129.         addstr(current_flow);
130.         // refresh();
131.
132.         move(6,73);
133.         strncpy(total_flow, &in_msg[44], 2);
134.         total_flow[2] = 0x00;
135.         addstr(total_flow);
136.         refresh();
137.     }
138.     if (strlen(in_msg) == 48){
139.
140.         strncpy(current_flow, &in_msg[11], 3);
141.         current_flow[3] = 0x00;
142.         move(6,28);
143.         addstr(current_flow);
144.         //refresh();
145.         move(6,73);
146.         strncpy(total_flow, &in_msg[44], 3);
147.         total_flow[3] = 0x00;
148.         addstr(total_flow);
149.         refresh();
150.     }
151.     if (strlen(in_msg) == 49){
152.
153.         strncpy(current_flow, &in_msg[11], 3);
154.         current_flow[3] = 0x00;
155.         move(6,28);
156.         addstr(current_flow);
157.         refresh();
158.
159.         move(6,73);
160.         strncpy(total_flow, &in_msg[44], 4);
161.         total_flow[4] = 0x00;
162.         addstr(total_flow);
163.         refresh();
164.     }
165.
166.
167.
168.     /* addch((char)portchar);*/
169. }else{
170.     // addch('.');
171. }
172.     //refresh();
173.
174. } // END OF CHARACTER FOUND FROM rs-232
175.
176. // check keyboard

```

```
177.         /* get a character from the keyboard */
178.         ch = getch();
179.         if (ch != ERR){
180.             printf("%c ", ch);
181.         }
182.         if (ch == 'Q') {
183.
184.             /* tcsetattr(port, TCSANOW, &old_flags);*/
185.             close(port);
186.             endwin();
187.             return(0);
188.         }
189.
190.
191.
192.     } // end forever
193.
194.     /* reset original port settings and close port */
195.
196. } // end main
```