THE WATERBOY

PROGRAMMABLE IRRIGATION SYSTEM

Miguel Castillo - Computer Science
Graphical User Interface

Ten Dunaj - Electrical Engineering
Overview and Power Supply

Andrew Hopp - Mechanical Engineering
Flow Meter and Packaging

Bill Keech - Electrical Engineering
Water Valve and Valve Control

Todd Kulesza - Computer Science
Microcontroller and Wireless Units

Norm Parker - Mechanical Engineering
Design Principles and Business and Marketing

Jack Turner - Electrical Engineering
Water Valve Testing and Position Sensing

The WaterBoy is a functional example of a programmable sprinkler. It is able to irrigate specific, user-defined patterns by measuring position and water flow and varying water pressure. A wireless link allows full sprinkler control from distances up to 50 yards away.

Table of Contents

Overview

The Waterboy introduces a new lawn watering system that will cut the cost of underground installation and avoid over watering of a lawn. This is done by placing a single watering head that can reach up to a 31 foot radius in the middle of the watering area. This means only one line has to be installed from a water and power source. With only one head this will prevent over spray from other sprinkler heads. The sprinkler head will rotate 360° and vary the spraying distance according to the position of the sprinkler head. This allows the water to cover only the designated area. This system will be controlled by a programmable user interface on a PC with the input being the distances at the designated position. The position of the head will be found by a positioning sensor that is connected through a gearing system from the rotating head to the shaft of the position sensor. Depending on the position of the head the senor will transfer a voltage reading to the processor. The processor then matches the distance at that position to the voltage necessary to open the actuator that controls the valve. The valve controls the water flow through the system which determines how far the water will spray. Once the water passes through the valve it then flows through a flow sensor. The flow sensor then relays back the amount of water flow through the system to the main system where water flow can be observed. Therefore the system controls the amount of water used and the area that the water is dispersed. This allows for maximum conservation of water and lawn watering made smart. Please see Appendix J for a complete system-level block diagram.

Design Principles

Theory

In order to properly describe the sprinkler system, it is important for us to understand how the water flow is affected by numerous variables within the system, and what variables we need to consider and which ones we can neglect. There are several reasons why this is important, the most important being; when a customer asks, "will this work with my system," we have the models (and understanding) to give qualified answers.

The first step to modeling the sprinkler was to take all of the physical measurements of the sprinkler itself, and then on to measuring the distance the water shoots under various pressures.

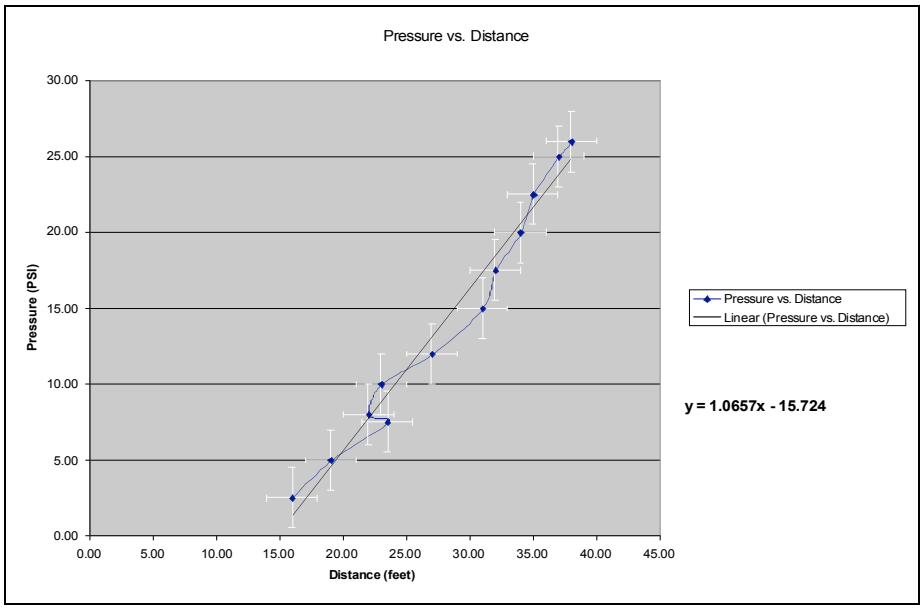
Nozzle Data

Brand: Nelson
Model: 1202
Desc: Gear Driven Sprinkler
Nozzle Diameter: 3.175mm
Nozzle Area: 7.917×10^{-6} m^3
Nozzle Angle: 30°

Since a flowmeter was not available during the initial testing of the sprinkler, we set up a gauge pressure system, which would allow to us to determine the amount of water passing through the system at different times. First, using the set-up shown at right, we tested the sprinkler with various pressures by logging the gauge pressure, and then measuring the physical distance the water was shooting at that pressure.

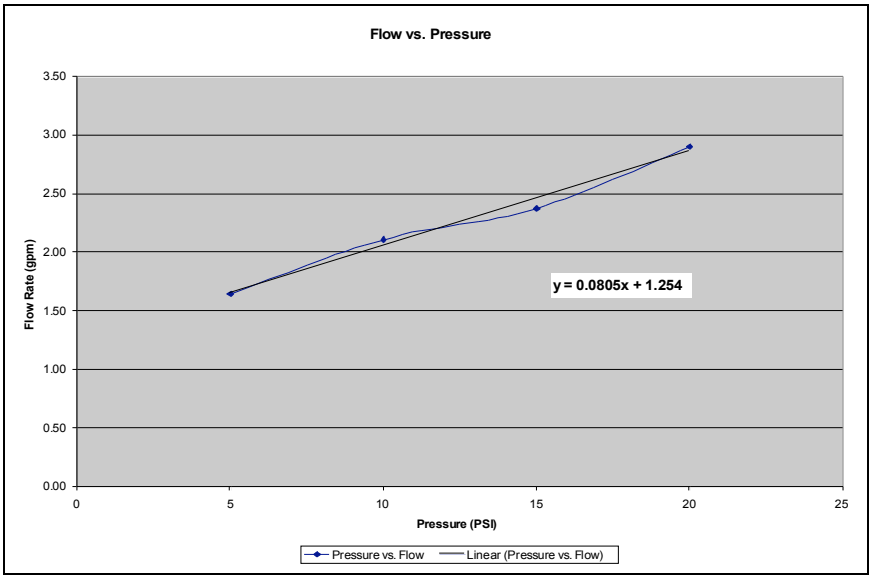
The second part of this test involved measuring the flow rate. Using the same setup as in the first part, though this time, rather than measuring the distance the water was shooting, we collected water in a bucket over a given period of time. This process would allow us to relate gauge pressure, which we had, to flow rates, which we did not have. This process is described below:

First, we manually obtain data plots for pressure vs. distance as seen in graph 1.



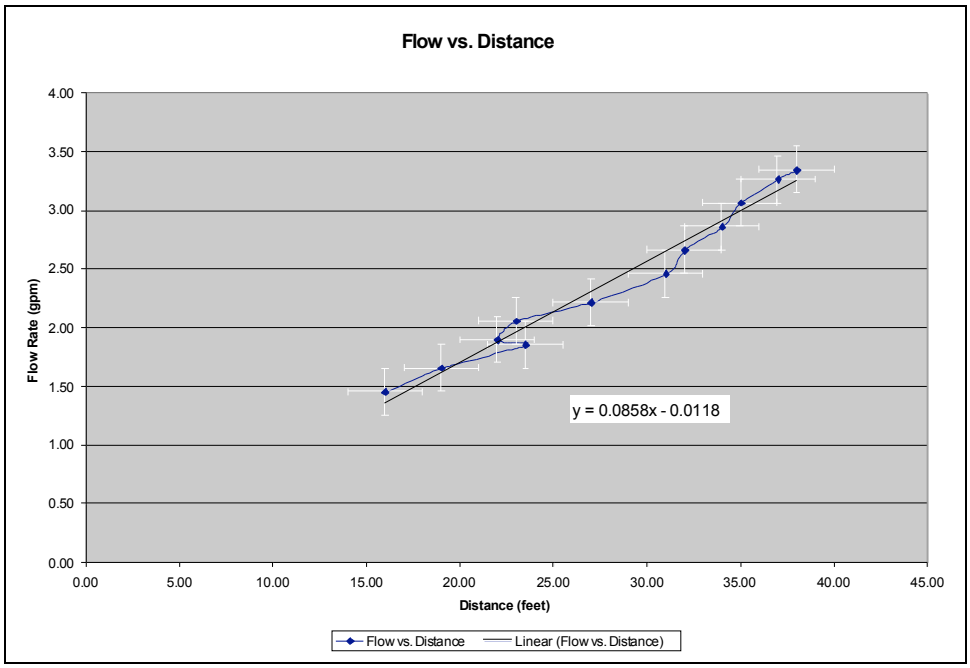
Graph 1: Pressure vs. Distance

Using an allowable margin of error, we were able to describe the points using a linear equation. The second step was to relate several pressure points to flow rates. It is important to note that in order to use this technique with accurate results; the same hose must be used under the same conditions so that the pressure and flow rate relationship remains the same. Graph 2 shows the plots of flow rate vs. pressure:



Graph 2: Flow vs. Pressure

Again, describing this relationship with a linear equation, we then used this data to indirectly describe the equation needed for the system, flow rate vs. distance:



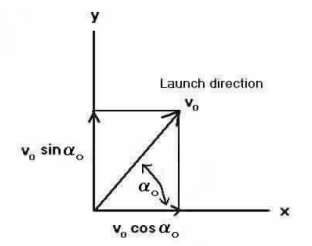
Graph 3: Flow vs. Distance

Modeling

Early in the testing process, a problem was encountered with the spray of the nozzle. In an attempt to simplify the data, we had plugged some of the auxiliary holes in the nozzle. The problem this created was a very uneven spray pattern, sending most of the water to the furthest distance of the stream. To correct this, the auxiliary holes were used; however, this left a very complicated nozzle pattern, which would be difficult to measure directly. It was important that we found this area for when we model distances that may involve wind resistance, we would need to know the nozzle velocity, which is dependent on nozzle area. In order to solve for the area, we would use the assumption that for very short distances, we could neglect wind resistance (we will not make this assumption for further distances as will be shown later).

We begin this process by writing the equations of motion:

In the x direction:



$$\frac{d^2x}{dt^2} = 0$$ Where x(0) = 0 and x'(0) = Vx

Integrating twice, we have:

$$\iint \frac{d^2x}{dt^2} = vt$$ or $x(t) = V_x t$ **(1)**

Since we do not have v or t, we will find t from the y equation of motion:

$$\frac{d^2y}{dt^2} = -g$$ where y(0) = 0 and y'(0) = V_y

Integrating twice yields:

$$\iint \frac{d^2y}{dt^2} = -\iint g\,dt$$

$$\int_0^t \frac{dy}{dt} = \int_0^t (gt + V_y)\,dt$$

$$y(t) = V_{oy}t - \tfrac{1}{2}gt^2 \quad \textbf{(2)}$$

Now, we do not have V_{oy}, V_{ox} or t, but we can write both velocities in terms of V_o:

$$V_{0X} = V_0 \cos(\alpha)$$

$$V_{0Y} = V_0 \sin(\alpha)$$

And, from equation (1), we can write t in terms of V_o and x:

$$t = \frac{x}{V_o \cos(\alpha)}$$

Substituting these values back into equation (2), we have:

$$y(t) = V_o \sin(\alpha)\frac{x}{V_0 \cos(\alpha)} - \frac{1}{2}g\left(\frac{x}{V_0 \cos(\alpha)}\right)^2$$

Setting y=0 and rearranging to solve for V_0:

$$V_0 = \sqrt{\frac{gx}{2\tan(\alpha)\cos^2(\alpha)}}$$

And with a measured nozzle angle of 30 degrees, we plug in these values to solve for Vo at this particular distance

$$V_0 = \sqrt{\frac{(9.81)(2.13)}{2\tan(30)\cos^2(30)}} = 4.9\,{}^{m}\!/_{s}$$

Now to solve for V_0 for some small value (recall, we are just trying to find the nozzle area at this point), we use a couple of test data points we are confident with. Fortunately, at this point of the project the team had received a flowmeter equipped with an output signal in Hz (described in detail in the *flowmeter* section). With this, we were able to obtain more accurate flow rates that had previously been described. Using the same testing method to relate the actual flow rate to gallons per minute, the following relationship was found: $gpm = .0161Hz + .0201$

In this case we had 2 data points that we were able to perfectly produce twice at 2.13m:

Hz	x	GPM	m3/s
92	2.13	1.5013	9.47x10^-5
92	2.13	1.5013	9.47x10^-5

Using the principle of conservation of mass, we now have enough data to find the nozzle area.

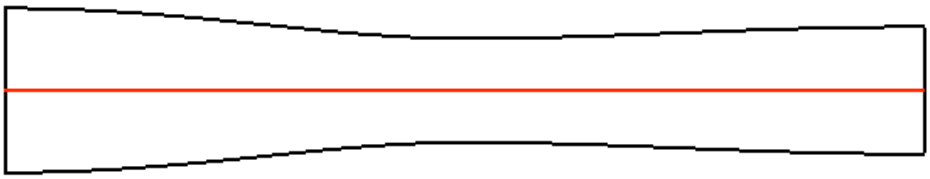


Figure 1: Sprinkler Nozzle

Where $\dot{V} = VA$ Since we know the volumetric flow rate and exit velocity, we just solve for A: A = $\frac{(9.47\times10^{-5}\ {}^{m^3}\!/_{s})}{4.9\,{}^{m}\!/_{s}} = 1.93\times10^{-5}\,m^2$, or about 19mm^2, which seems reasonable.

With the nozzle dimension taken care of, we can move on to modeling the actual stream of water leaving the sprinkler nozzle. In this case, we will not assume that wind resistance is negligible simply by observing how sensitive the stream of water is to light crosswinds.

In the y direction, since the nozzle angle is relatively low and how wind resistance will somewhat balance equation in the up and down direction, we can use the same equation of motion that we had used previously.

$$y(t) = V_{oy}t - \tfrac{1}{2}gt^2$$

This is not the case in the x direction, we write:

$$\frac{d^2x}{dt^2} = -b\frac{dx}{dt}$$

Where b is some coefficient of wind resistance that we will have to calculate using our test data, this may or may not be constant. To solve for this second order system, we will employ the use of Laplace Transforms.

$$L\left\{\frac{d^2x}{dt^2}\right\} = L\left\{-b\frac{dx}{dt}\right\}$$

Which translates to:

$s^2X(s) - sx(0) - x'(0) = -b(sX(s) - x(0))$ Where x(0) = 0 and x'(0) = V_{0x}

Substituting and solving for X(s):

$$X(s) = \frac{V_x}{s(s+b)}$$

In order to solve, we have to break this polynomial down using the method of partial fractions:

$$\frac{V_x}{s(s+b)} = \frac{A}{s} + \frac{B}{s+b}$$

A(s+b)+Bs = Vx

Let s = -b

B = $-V_x/b$

Let s = 0

A = V_x/b

$$\Rightarrow \frac{V_x}{s(s+b)} = \frac{V_x}{b}\left[\frac{1}{s} - \frac{1}{s+b}\right]$$

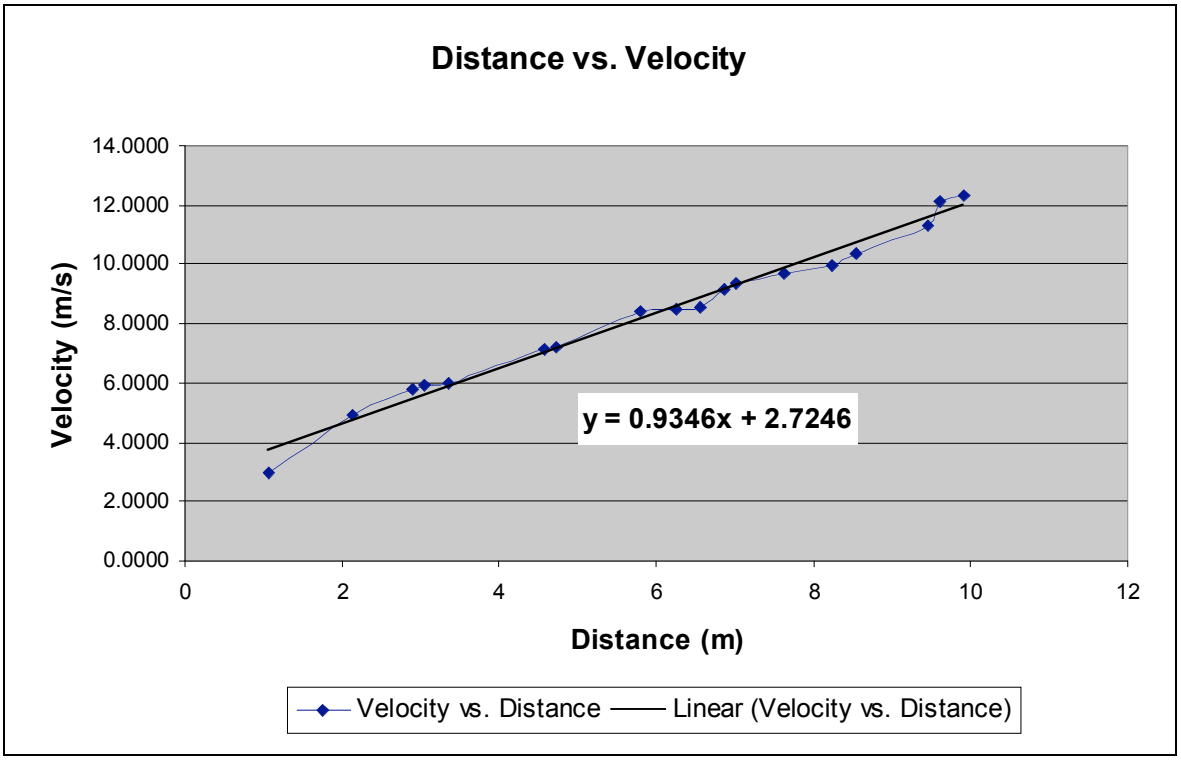
Taking the inverse Laplace Transform:

$$L^{-1}\left\{\frac{V_x}{s(s+b)}\right\} = \frac{V_x}{b} L^{-1}\left\{\left[\frac{1}{s} - \frac{1}{s+b}\right]\right\}$$

And now we have the equation of motion in the x direction with wind resistance:

$$x(t) = \frac{V_x}{b}(1 - e^{-bt})$$

In order to find b, we will need some test data. Below is a graph of velocity vs. distance. The distance was physically measured and velocity was calculated by dividing the mass flow rate by the nozzle area we derived earlier.



Graph 4: Velocity vs. Distance

In finding b, we must first determine if b is constant for this set of data. In order to accomplish this, we find b values for several sets of data (Table 2) and compare using our previously described equations of motion.

$$y(t) = V_y t - \tfrac{1}{2} g t^2 \qquad x(t) = \frac{V_x}{b}(1 - e^{-bt})$$

x meters	V
3.048	5.907578
5.7912	8.433793
7.0104	9.328494
8.5344	10.38108
9.906	12.32837

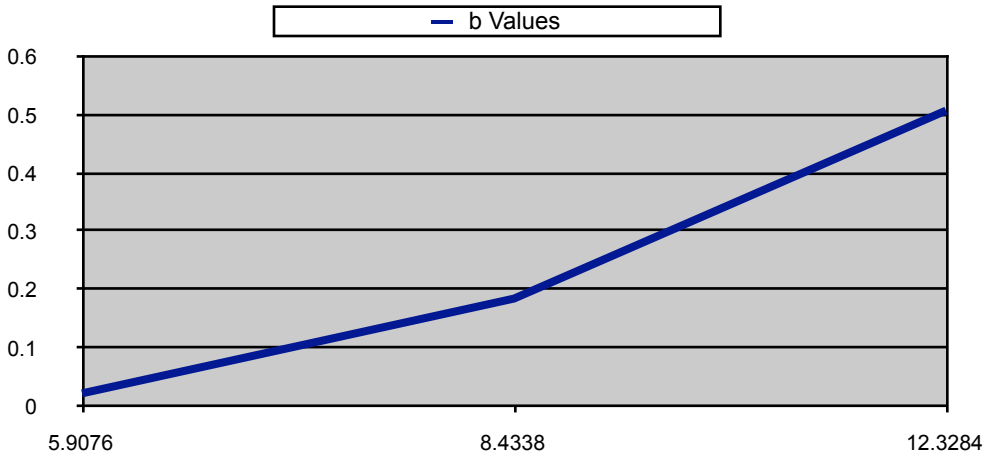
Table 2: Distance / Velocity

By performing manual calculations to solve for b, first solving for time in the y equation and then plugging in that value to solve for b in the x equation, we quickly find that b is not constant, rather increases with velocity. This is not surprising, for with non-rigid bodies, in this case water droplets, the droplets deform more with increasing velocity, thus increasing the drag. Table 3 shows the values of b as they relate to velocity.

time	velocity	b
0.6000	5.9076	0.0236
0.858	8.4338	0.1864
0.95	9.3285	0.1938
1.057	10.3811	0.2071
1.256	12.3284	0.5093

Table 3: Calculated b values

For the sake of convenience and practicality, we would not want to have to look up b values for every value of velocity, we would want this described within the formula. We plot b against the velocity values and determine if some relationship can be made. Graph 5 shows the results.



Graph 5: b vs. Velocity

Here we find a nice linear relationship between b and velocity. Note in the linear best fit equation, y is the b value and x is the velocity, and we rewrite the relationship as:

$$b = .0763V - .4382$$

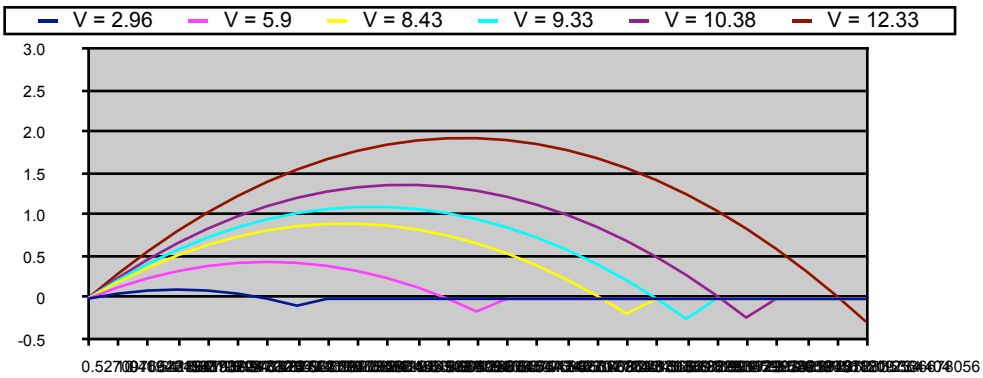
Now we rewrite our equation of motion in the x direction and substituting b with our b-V relationship:

$$x(t) = \frac{V_x}{b}(1 - e^{-bt}) \longrightarrow x(t) = \frac{V_x}{.0763V_0 - .4382}(1 - e^{-bt})$$

And for convenience, we will write everything in terms of V0.

$$x(t) = \frac{V_0 \cos(\alpha)}{.0763V_0 - .4382}(1 - e^{-bt}) \qquad y(t) = V_0 \sin(\alpha)t - \tfrac{1}{2} gt^2$$

Finally, with our b values calculated, we can plot our theoretical streams of water using several different velocity points shown in graph 6.



Graph 6: Calculated Streams of water

The graphs shown derived from our formulas appear to be a perfect model for our system. The distance is correct, of course, because we solved for the b value based on the desired distance. Furthermore, the shape of the stream is exactly what was witnessed during testing, a nearly straight stream coming from the nozzle, then a slight fade or stall near the end of the stream. We are very confident with these results.

With our model in place, it would be a trivial task to modify our system for the customer to fit any type of sprinkler head and whatever type of flowmeter, or even pressure sensor that may be used at the time.

Graphical User Interface

Overview

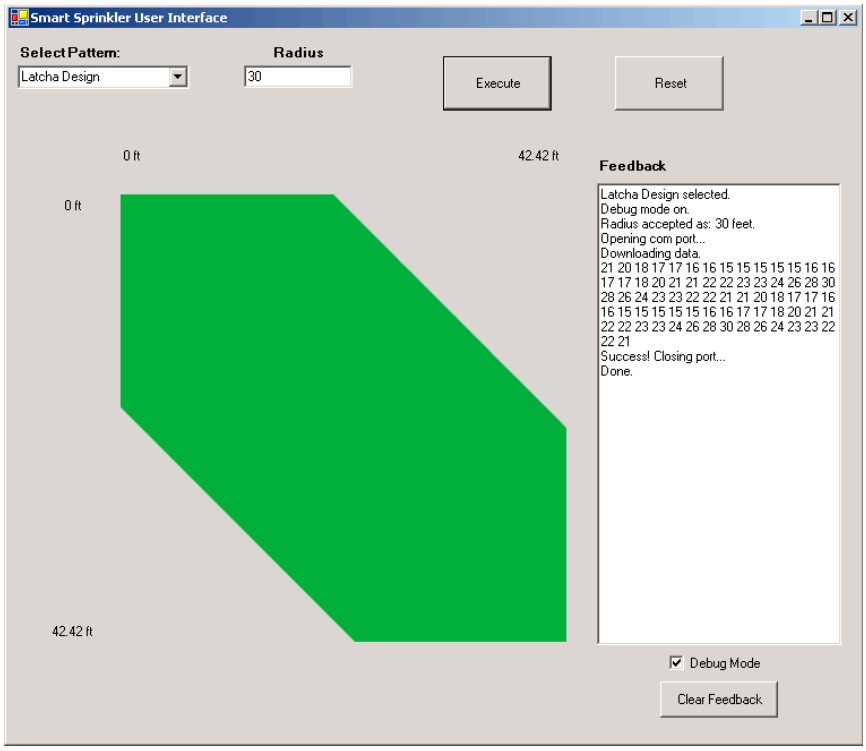
The user interface for the system will allow the user (or technician) to configure the pattern that the sprinkler will spray. The user selects a predefined pattern (such as a circle or square) or a custom shape, and inputs the dimensions of the shape. The user is then given the option to cut out areas of the shape in order to avoid those areas on the lawn.

The program allows for cutting out square areas on the lawn. When the dimensions of the square cutout are specified, the UI calculates the new distances from the center point to the cutout border and stores those values in an array along with the rest of the lawn distances. The data is then formed into 72 8-bit unsigned integer packets (corresponding to 72 distance calculations in the circle of spray) and sent through the serial port to the HCS08 micro controller.

Methods

The GUI is written in Visual Basic .NET. The main components of it are the form itself, the equations for calculations, and the Rs232 communication object. [1]

The form consists of a drop down menu for selection of the shape, a radius input text box, a picture of the shape selected, and a feedback text box. For our demonstration, the user selects a pre-defined shape and inputs the radius of the circle that the shape is cut out from. Once the shape and radius are defined, the user clicks the execute button and the program begins running.
(Refer to Appendix C for all code references)



The program first calls the calculation function corresponding to the shape selected, and passes in the radius. The program loops through the 72 positions on the circle (every 5 degrees) and calculates the distance to spray based off the equations listed in Appendices F, G, and H. Once the calculations are complete, it then calls the serial output function serialO().

The serial output function loops through the distances array and sends out each number as an unsigned 8 bit integer. The trick to this part is that the Rs232 object communicates only in ASCII values. So in order to get the proper distance value output, it is necessary to convert the number to its ASCII equivalent. For example, to send out the number 65 in binary, the function must actually send out the letter "A." This way, the HCSO8 board does not need to know anything about the conversions taking place, but simply takes it as raw data. To do the conversion, the Chr() function is called, which is built into .NET, on the binary value.

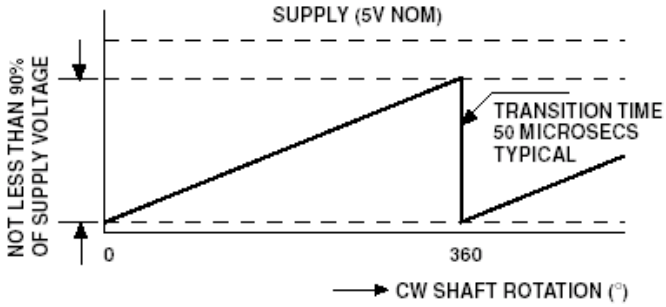
Obstacles Encountered

Due to the complexity of the equations of the shapes and cutouts and time restrictions, the prototype GUI only can program a few shapes. Ideally, the GUI will be able to calculate distances of the shapes and cutouts based on a pixel to feet ratio.

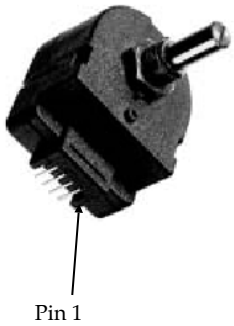
Another obstacle encountered was the serial communication portion of the program. In Visual Basic .NET, serial communication was antiquated, and was no longer built in to VB. However, after a long amount of time searching and testing various serial objects built by other independent sources, the Rs232() object from the MSDN library was found suitable. There were problems with it in the beginning, and as a result, it was originally thought buggy and not working. The reason for this was the baud rate selection. If a non-standard baud rate is selected, it gives a runtime exception with no reason, and points to a line that has nothing to do with the baud rate. The error was found when a sample piece of code was used for testing and the realization that 36600 is not a standard baud rate (which is what we thought the HCS08 worked on). Once we figured out that the baud rate is actually 38400 and tried the code out with that, we were able to get the data from the program through the serial port properly.

Position Sensing

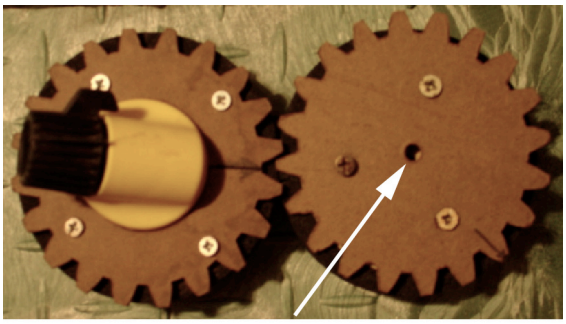
Knowing the position of the sprinkler head is an important aspect of Waterboy 's control system. This is accomplished by using a Vishay Spectrol Full 360° Smart Position Sensor (Model #601-1045). This device is essentially a continuous rotary potentiometer in a self-contained package, shown below on the right. The sensor provides an analogue electrical output over a full 360° without the need of external electronics, and because of its low power consumption and non-volatile output (it remembers its position in the event of power loss), it is a cost effective alternative to absolute encoders. Although the output provides a fully continuous voltage signal ~0-5VDC, where the exact output is dependant on the input, ranging from 4.5 to 5.5VDC, the position sensing control subsystem of Waterboy will convert the analog output to an 8-bit digital signal. Due to limitations of the microcontroller's A/D input port (the microcontroller can accept a maximum of ~3 VDC) the output voltage is limited with a voltage divider circuit. This is discussed further in the power supply section of this report. The shaft of the position sensor runs through the center of a disc, where a one-to-one gearing system, shown below on the left, allows the position sensor to turn directly with every incremental turn of the sprinkler head. The figure on the bottom right shows position sensor from the underside of the lid. There are four pins on the position sensor where pins 1&2 are for power supply (5 and 0V respectively), pin 3 is an output voltage, and pin 4 determines the direction of the output, or rather the ramp polarity. The graph below shows the increasing linear relationship between output voltage and the shaft's rotational position in degrees for the default option of leaving pin 4 disconnected.



DEFAULT OUTPUT [Terminal #4 Open-Circuit]



Pin 1



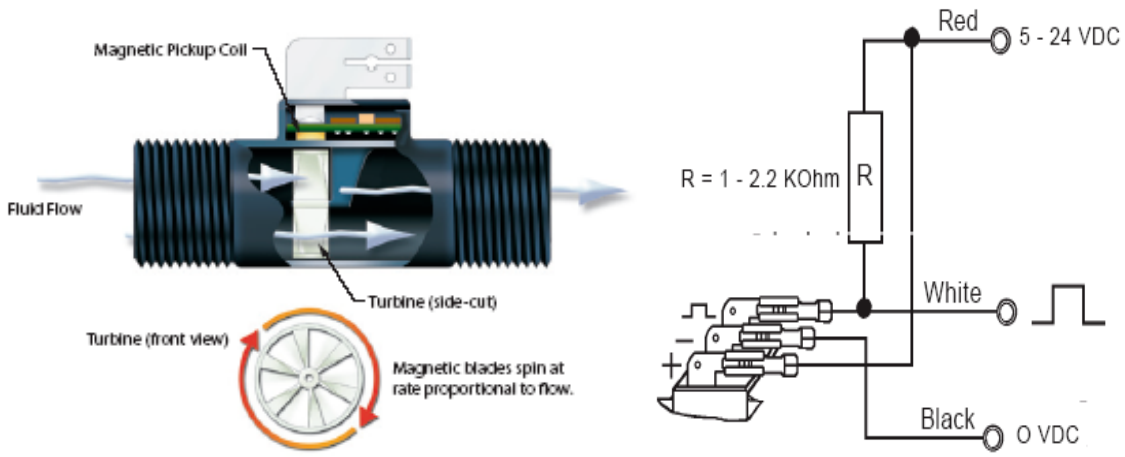
Top position of sensor shaft

Flow Meter

In order to sense in real time the distance that our system sprays water, we have to know the area of the sprinkler nozzle and the velocity at the sprinkler nozzle. The easiest way to find the velocity would be to measure the flow through the system, and then apply the conservation of mass principle to obtain the unknown velocity. Flow measurement was accomplished using the most common instrument on the market, a turbine flow meter.

The caption to the right displays the turbine meter used in our system. It was obtained through Gems Sensors Company, who graciously made a donation to our project. [2] This meter accepts a 5-24V signal and outputs a square wave with amplitude equal in magnitude to the voltage input. The frequency of this wave increases linearly with flow and thus the flow can be obtained by dividing the frequency by the inverse of the slope of this linear relationship.

The meter functions via the hall- effect. As shown below, when flow is established through the meter, a small Nylon composite turbine rotates at a speed proportional to the fluid velocity, and generates an output pulse when its blades pass over a magnetic pickup. The pulse takes the form of a square wave signal with a frequency equal to the blade passage frequency of the turbine. This frequency varies linearly with flow rate. A proper signal can be obtained when electrically configured as shown at the above right. The meter is grounded using the middle pin, while a DC signal of anywhere between 5-24 volts can be supplied to the front pin. This range provides flexibility for any application. For our system, it is convenient to power the meter using a 9V battery.



The purpose of measuring the real time flow of the system is to obtain the velocity at the nozzle of the sprinkler head. Once this is known, the spray distance can be calculated using the basic equations of projectile motion. Real time flow data is used to obtain the exit velocity by drawing a control volume around a selected portion of the system diagram and applying the continuity equation to this control volume (Please see Appendix E for derivation).

The table below summarizes flow specifications for our meter. Experiments were conducted to verify these spec points. Results can be found in Appendix E

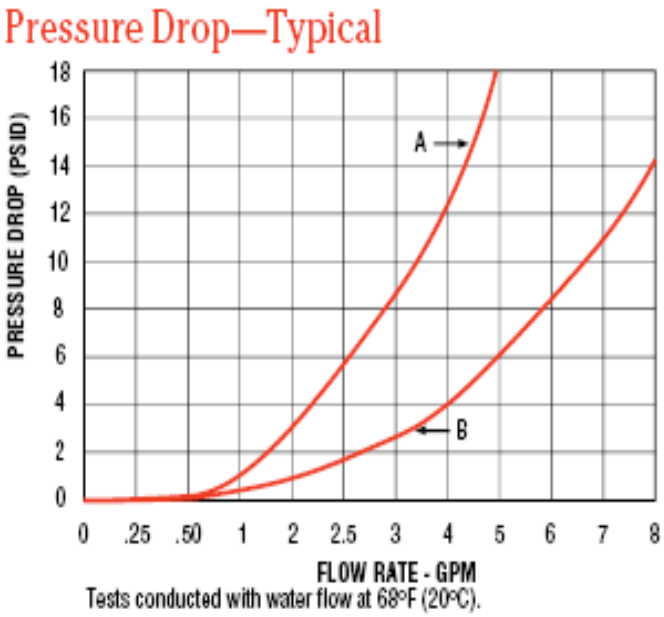
Part Number	Normal Flow Range	Extended Flow Range	Pulses	Frequency Range
#	GPM	GPM	Per Gallon	Hz
173935	0.53 - 7.9	0.13 - 7.9	3800	33 - 500

Table 4

Note that normal flow is specified to a minimum of about 0.53 GPM or 33Hz. Experimental data was taken at frequencies as low as 10.75 Hz or 0.16 GPM. Data here is still very linear, promising accuracy even in the extended flow range as shutoff is approached. Also note that the meter is specified for a maximum flow of about 7.9 GPM or 500Hz. This will work out fine for our system since we don't expect any household outdoor faucets to produce a flow greater than 6 GPM.

The durability of this meter makes it a good fit for this application. The body and turbine of the meter are made from Nylon plastic, making it resistant to thermal cycling and high stress. The specification sheet located in Appendix E indicates an operating temperature range of -4F to 212F, and a burst pressure of 2500 PSI. Since such extremes in temperature and pressure are not expected for our system, the meter will have a long operating life.

The restrictive characteristics of this meter also tend to favor our system. The figure below, (also available on the spec sheet in Appendix E), shows restriction curves for two of the Gems Sensor products, ours being characterized by curve B. This curve shows that even for a high flow rate such as 4 gallons per minute, the system will only see a 4 PSI drop in pressure due to the meter. Considering that typical house pressures are on the order of 40PSIG, such a drop would not severely impact spraying distance for our system.



One final threat to this meter is particulate matter entering through the user hose line. The meter is equipped with a 5 micron filter, but would be susceptible to large particulates such as leaf debris. This could tangle around the turbine rendering it useless. To prevent such a disaster a wire filter washer was installed in the female hose connection at the inlet to the system. This filter is protective enough to ensure that particulates passing through it will not be detrimental to the meter and system as a whole.

Water Valve and Valve Controller

Premise

The premise of the controllable sprinkler system is based upon the application of controlling a water valve using a microcontroller. Using the input from the position sensor connected to the sprinkler head, the microcontroller will direct the valve to the proper position to achieve the desired watering distance. The distances required are determined by the user's requirement. The mechanical sprinkler used makes a circle while watering. By controlling the water valve position, other designs are possible. For each design, equations describing distance are required. Depending on the design, only a select number of valve positions are required. A square, regardless of the radius of circle chosen, only requires 10 select valve positions covering 72 segments of 5 degrees each. However, the circle's radius must be within the limits of the sprinkler system. The Latcha design requires 19 valve positions over 72 segments of 5 degrees each. A triangle design requires 13 valve positions over 72 segments. The appendices F, G, and H covers the square, Latcha design and triangle, respectively. The valve positions related to distance sprayed will be discussed later and are discovered while testing the system.

There are several things to consider when deciding what kind of valve to use and how to control that valve. Some of the considerations required are as follows:

1. What is the best method to operate the valve?

2. What type of interface is required to operate the valve?

3. How fast does the valve need to open, close or move from one position to another?

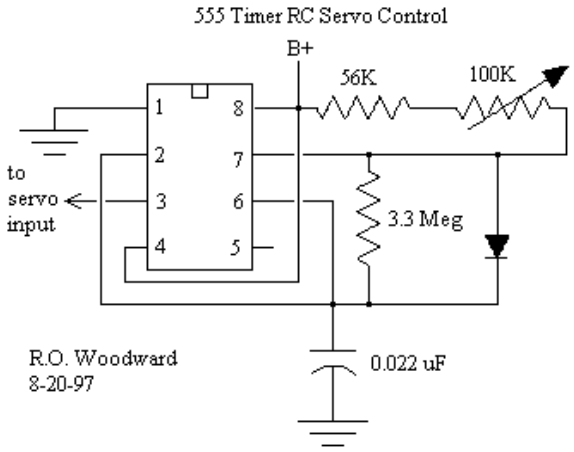
4. What style of valve to use (ball, butterfly, etc.)?

5. What size of valve is required?

Hardware

There are many possible ways to operate a valve. Several ways were considered and include a servo motor, a stepper motor, or an electric actuator. Each method has positive and negative aspects. Both servo and stepper motors are fairly inexpensive, whereas electric actuators can be expensive. All offer good positional holding torque and is determined by the properties of each motor. A stepper motor can be operated in full and partial step modes; however the rated torque is reduced when the stepper motor is held at partial step position. A servo motor is operated using pulse width modulation, or PWM. Stepper motors are controlled using binary inputs in a select order that determines the step size and direction. An electric actuator can be operated in a number of different ways, which will be covered later. A servo has a built in feedback system as to position based upon the input signal. The position of the stepper motor cannot be determined without the use of a secondary sensor.

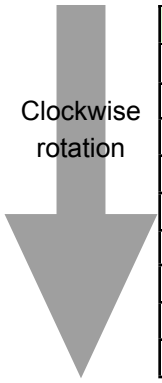
A servo motor will utilize the microcontroller's ability to generate PWM signals. There are a variety of servo valves available of different sizes, torque ranges, and degree of arm motion. The typical RC servo motor is small and able to rotate either 45 or 60 degrees each way from the center position, resulting in a total of 90 or 120 degree range of motion. RC servos have very good response times for arm control. RC servo motors normally operate with an input duty cycle between 1 and 2 milliseconds, with 1.5 milliseconds being the center position. The first controllable test valve made incorporated a S75 sub-micro servo motor and a typical plastic in-line garden hose 90 degree turn ball valve. A picture of the test valve set-up is located to the right. The S75 servo motor has the ability to rotate a total of 120 degrees with 17.9 ounces/inch of rated torque. The 17.9 ounces/inch of torque converts to approximately 200 grams/cm. The time required to move the arm 60 degrees is approximately 120 milliseconds. The 120 degree of arm motion is more than required to operate a 90 degree ball valve. For the purposes of this experiment, the servo will only be operated 90 degrees. The valve will be half open when the servo input is 1.5 milliseconds. The valve is completely open or closed when the PWM input is approximately 1.9 milliseconds and 1.1 milliseconds, respectively. The torque required to operate the valve is unknown. To test the servo, a 555 timer circuit was erected based upon the diagram below to left [1].



By adjusting the 100 kilo-Ohm potentiometer, the positive output pulse will range from 0.9 to 2.1 milliseconds. There is approximately a 40 milliseconds off period between positive pulses. The off time can be increased by increasing the value of the 3.3 Mega-Ohm resistor. The servo motor had just enough torque to operate the valve without water flowing through it. It was determined that the S75 servo isn't large enough to operate the valve correctly. If the project were to use a servo, it would have to have a higher torque rating.

Stepper motors are very common and can be found in computer disk drives, printers, as well as many other devices. A big drawback of a stepper motor is that an additional sensor is required to determine the position of the stepper motor. This project incorporates a FT-110 Turboflow flow meter to indirectly determine the valve position. There are different types of stepper motors and a couple of types are bipolar and unipolar motors. There are variations within each type depending on the number of wires going into the motor. Another way to separate the types of stepper motors is by the number of steps per revolution. An example is a stepper motor that is described as a 7.5 degree per step motor has 48 steps per revolution. The amount of holding torque is highest at each 7.5 degree increment. A stepper motors degree per step can be further divided into ¼, ½ and other partial steps; however the holding torque is no longer the rated holding torque. How much less depends on the motor. These steps can be divided further by using micro-stepping technology. An example of using micro-stepping involves a stepper motor with 1.5 degree per step, which translates into 240 full steps per revolution. Micro-stepping this motor will generate 240 additional steps between each full step. A motor with 240 full steps per revolution now has 57,600 steps per revolution. This allows a stepper motor to have very precise positioning regardless of its full step degree value.

The method of direct input from the microcontroller was the first idea investigated. A bipolar stepper motor has a higher torque/size ratio than that of a unipolar stepper motor. Unipolar stepper motors typically requires less complex driver circuitry than a bipolar stepper motor. A stepper motor is operated by inputting a series of binary digits in a specific order. They both use the same input logic, but unipolar stepper motors require only a on/off signal represented by a 1 or 0, respectively. A bipolar stepper motor's inputs refer to the polarity of the input voltage. The order and frequency that the digits are sent determines the direction and speed of the motor. To reverse the motors direction, the order of digits is reversed. This section will discuss three possible ways to operate a stepper motor and they are as follows full-step method, half-step method and high-torque method. The high torque method requires the current input to be double that of the full and half step methods. The holding torque of a half-step method is about half that of a standard full-step method. The input sequences for the three methods discussed are listed in the Table 5. The input sequence order is inverted from bottom to top to achieve a counterclockwise rotation. The project considered using one of the described methods, but it was not determined which one would work best.



Standard full step					Half-step					High torque full-step				
Index	1a	1b	2a	2b	Index	1a	1b	2a	2b	Index	1a	1b	2a	2b
1	1	0	0	0	1	1	0	0	0	1	1	0	0	1
2	0	1	0	0	2	1	1	0	0	2	1	1	0	0
3	0	0	1	0	3	0	1	0	0	3	0	1	1	0
4	0	0	0	1	4	0	1	1	0	4	0	0	1	1
5	1	0	0	0	5	0	0	1	0	5	1	0	0	1
6	0	1	0	0	6	0	0	1	1	6	1	1	0	0
7	0	0	1	0	7	0	0	0	1	7	0	1	1	0
8	0	0	0	1	8	1	0	0	1	8	0	0	1	1

Table 5

A properly selected stepper motor would be able to move the valve quickly in either direction, but may involve quite a bit of coding and hardware assembly to do so. There are other options available to control stepper motors and two of them are stepper motor control chips, and stepper motor microcontrollers. A CMOS stepper motor control chip was located and viewed as a possibility. It would replace most of the coding required to operate the stepper motor. The wires from the stepper motor are connected to the predetermined pins on the stepper control chip. Using this chip to operate the stepper motor only requires three high or low inputs from the microcontroller. The first input tells it to step or not to step with a 1 or 0, respectively. An input of 1 tells the chip to drive the motor clockwise and a 0 tells the chip to move the motor counterclockwise. The last input determines whether the motor will be moved a full step or a half step.

Peter Norberg at Peter Norberg Consulting, Inc. was contacted for information regarding the operation and control of stepper motors. His company specializes in building stepper motor microcontrollers. For purposes of clarity, stepper motor microcontrollers will be referred to as a stepper controller. Based upon statements about the project, he recommended using a stepper motor with a minimum torque rating of 2000 grams/cm. He assisted by suggesting a Jameco stepper motor that had a torque rating of 3600 grams/cm. Peter suggested using the BiStepA06 stepper motor controller which is able to operate up to two stepper motors simultaneously or one stepper motor at double current. The stepper motor list price is approximate $30 and the Peter's stepper controller cost $99. The stepper controller allows for easier interfacing with both the main microcontroller and the stepper motor. The stepper controller is operated using TTL serial input from the main microcontroller. An input varying from 4 to 20 milliamps operates the stepper motor over a 90 degree span. This input style offers direct feedback as to valve position. The valve is completely closed or open when the input is 4 or 20 milliamps, respectively. The main microcontroller used doesn't have TTL serial capabilities, but is does have a RS232 serial port. Peter's company offers a RS232 serial to TTL converter for $12. This appeared to be the valve control system best suited to our needs. All of Peter's controllers are well documented and provide very good install and operation instructions. It offers a tutorial for first time use. This controller design is a good prototype set-up, but the BiStepA06 microcontroller is not made for long-term use in a sprinkler system application. There are a number of companies that produce stepper motor controllers; Peter's website had the best documentation and instruction files. The use of a stepper controller and stepper motor would be a very good option for this project. This is discussed in more detail in the future considerations section.

A phone call was placed to Chris Bele at the Hayward Flow Control division of Hayward Industrial Products, Inc. in Clemmons, NC at approximately the same time Norberg was contacted. Chris was contacted to get more information regarding the water valves and electric actuators the manufacture. He was updated of the idea and direction of the sprinkler project. He suggested several methods to control distance of spray besides using a varying water valve, including but not limited to changing the trajectory of water from the sprinkler head and adjusting the nozzle head to change the water dispersion from the head. He was informed that the other options had been discussed and in some cases investigated, but controlling the flow of water turned out to be the best option for this project. He suggested using their Profile2 proportional control ball valve. The actual valve component is a true union ball valve as represented in the picture located to the right, but what makes this valve special is the design of the ball used inside the valve. The Profile2 proportional control ball valve is pictured to the right. The ball's design incorporates two opening rates that can be changed by reversing the orientation of the ball within the valve unit. The slow opening rate is created by installing the ball so that the water flows through the small part of the ball first. The smallest valve available utilizing the Profile2 ball design is 1". The Profile2 ball valve has a linear flow curve when set-up for slow opening.

A chart showing comparing the theoretical flow curves of the Profile2 ball valve versus a typical ball valve is can be viewed in Figure 2. The percentage of flow through a typical ball valve is 80 percent before the valve is half open.

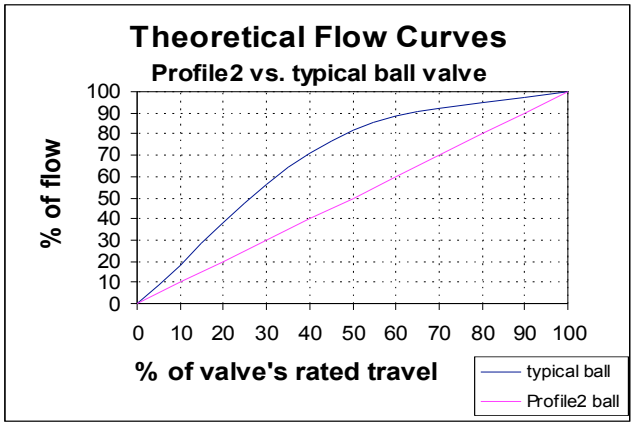
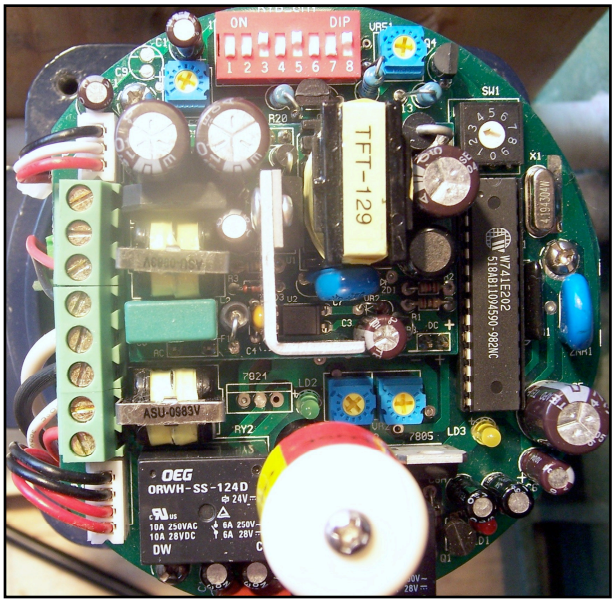


Figure 2

The benefit of using the Profile2 style ball valve is that the percentage of flow rate is determined linearly by the valve position. There was only one downfall using the 1" True Union Profile2 ball valve for our project. The smallest True Union valve size that utilizes the Profile2 ball has a 1 inch opening. We are using 5/8 inch garden hose to feed the system from the house and have restrictions in the system after the valve which include the flow meter and sprinkler head. A saturation level was expected at some point while opening the valve where the flow ceases to increase as the valve continues to open. During testing, the saturation level occurred when the valve was about half open.



To control the True Union Profile2 ball valve, Chris suggested using an EJM electric actuator powered by 115 volts with a 4 to 20 milliamp positioner. The Profile2 proportional ball valve can handle an input pressure up to 150 psi and the EJM's torque rating is 3500 lb/in which are both sufficient for this application. The EJM actuator on its own would not work within our system, due to it having a 25 percent duty cycle and takes 8 seconds to travel 90 degrees. A 25 percent duty cycle dictates that for every 5 seconds it is operated, it must be idle for at least 15 seconds. By using the 4 to 20 milliamp positioner in conjunction with the EJM actuator, the duty cycle becomes nearly 100 percent and the valve's travel time for 90 degrees was reduced to approximately 5 seconds. Chris said the cost associated with a set-up containing the EJM actuator, 4-to-20 millisecond positioner and the True Union Profile2 ball valve would be about $1200. However, he said that Hayward Industries occasionally grants a valve system for a project being done by students at a university and was able to grant us a valve system to this project. A picture of the Hayward unit received, which includes the actuator, positioner and valve are pictured below to the left. Pictured to the left is a top view of the positioner board's design.

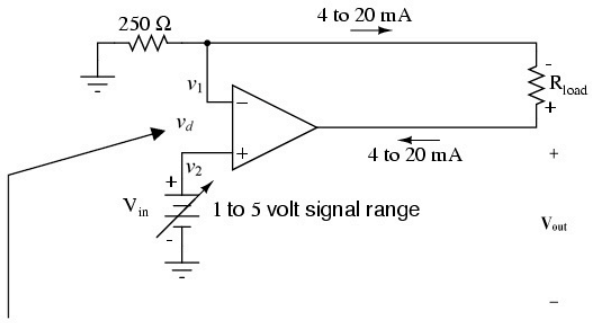
The positioner operation is pre-set at the factory for 4-to-20 millisecond input control. There are other options based upon the positions of the eight dip switches available, which can be viewed in the top portion of the picture above to the right. Dip switches 1 and 2 control the type of input signal that the valve operates on. There are three possible input types available to control the position of the valve and they are 4-to-20 milliamps, 1-to-5 volts, and 2-to-10 volts. If dip switches 1 is on and 2 is off, the input signal required is 4-to-20 milliamps. The lower value of amperage and voltage refers to a completely closed valve position, while the higher values refer to a completely open valve position. To use a 1-to-5 volt input signal, both dip switches 1 and 2 are off. A 2-to-10 volt input signal is required when dip switches 1 and 2 are off and on, respectively. The best option for this project was the 1-to-5 volt input signal. A PWM signal from the microcontroller can drive the valve over this range. The interface between the valve and microcontroller is discussed in the section covering the microcontrollers used for this application.

This particular positioner system allows for output signals to be received in two ways. The output signal can be set for a 4-to-20 milliamp signal or a 2-to-10 volt signal, by setting dip switches 3, 4 and 5 to the proper orientation. The output signals were not utilized in this application. Another useful setting allows the input signal to be reversed, so that an increasing input signal opens the valve. Dip switch 6 was placed in the on position for this project, so that increasing signal opens the valve. The valve can be opened, closed, or locked into place upon loss of signal from the microcontroller. This can all be accomplished by setting dip switches 7 and 8. The valve was set to close on loss of signal, so pins 7 and 8 were set to off and on, respectively.

The valve and positioner is a continuous system. If the valve is closed and an input signal of 5 volts is applied, the valve will smoothly open completely in about 5 seconds. However, when inputting small voltage differences the valve acts as though it has discrete movement. It was discovered during testing that the valve would move when the change in voltage applied approached 0.1 volts. Assuming that these 0.1 volt increments are considered deadbands, there is an adjustable deadband setting on the positioner board. If the SW1 is adjusted clockwise, the deadband will increase and decreases as SW1 is turned counterclockwise. The SW1 setting was not moved, but could be adjusted to allow for more valve positions over the 90 degrees.

Testing

Before it was realized that the Hayward valve could accept other types of inputs besides 5-20mA, the others being 1-5V or 2-10V, (the input option is dependant on a particular DIP switch setting, which is described above) a transconductance amplifier circuit, found from http://www.allaboutcircuits.com/vol_3/chpt_8/7.html, shown below for quick reference, was built to provide a voltage-to-current signal conversion, where transconductance is the ratio of current change divided by voltage change (ΔI/ΔV) and the transconductance ratio of this circuit, fixed by the 250Ω resistor, provides a linear current-out/voltage-in relationship of 4-20mA out / 1-5V in. For testing purposes the 1-5 volt input came from an ELENCO DigitalAnalog Trainer Board Model XK-150. This circuit is just a noninverting amplifier where the voltage gain is $\frac{V_{out}}{V_{in}} = 1 + \frac{R_{load}}{250}$, but in this case we are not interested in the voltage gain, rather the output current, so the value of Rload is irrelevant, so long as the op-amp has a high enough power supply voltage to output the necessary voltage to pass 20mA through Rload. The polarity of Rload is indicated such that a positive voltage would display when hooked up to a voltmeter.



To simplify analysis, since $v_d = v_1 - v_2 \approx 0$, we can treat v_d as a virtual short circuit, which yields the following relationship $v_1 = v_2 = V_{in}$

Microcontroller and Wireless Units

Overview

We selected a pair of Freescale 13192 SARD boards to handle the logic processing and wireless communication. These particular boards are reference designs provided by Freescale. Each includes an MC13192 wireless transceiver and an HCS08 microprocessor; these components are currently available in mass quantities for under $5 each,

allowing for inexpensive mass production of the final system. The 8-bit MCU includes all of the capabilities needed for the sprinkler while maintaining very low power consumption and keeping an ultra-compact form factor. The MC13192 unit allows for fast, reliable transmission of small data packets, and seemed a perfect fit for our application. All of the software design was completed in Metrowerks Codewarrior 3.1 using Embedded C or HSC08 Assembly; the resulting ROMs were flashed to the SARD boards using the USB HSC08 Multilink Programmer. Freescale's SMAC application template was used as a general framework for ZigBee wireless communications. Appendix I includes a product overview of the Freescale 19192 Developer's Starter Kit.

Operation

Our prototype makes use of two SARD boards. One is attached to the user's PC and is used to wirelessly transmit pattern information to the sprinkler. The second is integrated into the sprinkler housing; this board acts as the system controller, reading data from the position and water flow sensors, computing necessary adjustments to create the desired pattern, and appropriately controlling the water valve.

Pattern Transmission and Storage

The user-selected pattern is sent from the computer to the PC Link board via the serial interface. An interrupt handler catches this serial data and stores it in a global array; the pattern is a collection of seventy-two integers specifying the desired distance at each position, padded with a zero before the pattern and a one following the data. Zero and one where chosen as the sprinkler can not reliably water such short distances, thus the numbers will never occur in a pattern. When the interrupt handler catches a zero, it sets the array counter to zero and increments through the array as each new value is detected. When a one is found, the array is considered to be complete and wirelessly transmitted using the stock ZigBee protocol stack provided by Freescale. The complete source code can be found in the *main()* method of Appendix A.

The Valve Controller listens for the ZigBee packet and, once detected, calls *flash_pattern_store()* (Appendix B) to move the pattern into flash memory. When writing to the HCS08's flash memory, all code must be executed from the stack, as the onboard memory will be unavailable during the procedure. The file *flash_utility.asm* includes full assembly source code, largely provided by Freescale, for erasing or writing flash sectors. The files *flash_utility.h* and *flash_utility.c* (Appendix B) were written to provide convenient C wrappers for flashing or reading an entire pattern; the sequential order of the *pattern[]* array is exploited and allows the use of simple loops to handle both reading and writing the nonvolatile memory.

Position Sensor Input

The position sensor outputs an analog signal between zero and five volts. The SARD board includes an analog-to-digital converter, but the input range is only zero to three volts. Thus, it is necessary to halve the output of the position sensor before it is fed into pin 0 of ADC1 on the SARD board. Control register ATD1SC is set to 0x00, selecting the first of eight ATD channels. The software then loops until the Conversion Complete flag of ATD1SC is set, indicating that a value is ready. This value, stored in ATD1RH, is a decimal number between zero and 255. Our sprinkler's circular range is divided into seventy-two segments of 5º each; each value from the position sensor is mapped to a corresponding segment via the simple formula $position = value / 2.75$. The *compute_position()* method (Appendix B) performs this calculation using integer-only arithmetic and returns the current position of the sprinkler head.

Flow Meter Input

The HCS08 includes a pair of dual-channel timer modules, TPM1 and TPM2. The first channel of TPM2 is used in input capture mode to read the output of the inline flow meter. The module's status and control register, TPM2SC, is set to 0x0f, enabling use of the onboard clock with a divider of 128 to slow it down to an appropriate range for the flow meter. TPM2C0SC is set to 0x44 to turn on input capture and enable an interrupt for the channel. An interrupt handler (*input_capture()* in Appendix B) is then defined to respond to the input event. The HSC08 features as 16-bit timer, but as the chip is only 8-bits, care must be taken to read both the high and low bytes of the timer registers, else the timer will hang. Once the two bytes have been read, they are combined into a 16-bit integer and compared with the value from the previous input capture event; the difference is the period of the frequency sent by the flow meter. The period is stored in the global variable *input_period* so that it can be referenced outside of the interrupt handler.

Water Valve Control

An output current is required to control the operation of the motorized water valve. The first channel of TPM1 is dedicated to controlling the valve; it operates in edge-pulse mode. The timer for the PWM's period is set to 0xff and the divider varies between 0x00 and 0xff, resulting in the output of zero to three volts. For each iteration of the main program loop, the sprinkler's position and flow rate are examined. The position is used to lookup the desired distance in the array *pattern[]*. This data is fed into the *compute_flow()* method (Appendix B), which uses the equation *flow = (distance + 5.707) / 0.154* to determine the necessary flow rate for the requested distance. The result is compared with the current flow rate as detected by the *input_capture()* interrupt handler. If there is a significant difference between the actual and desired flow rates, the output signal of the PWM is slightly increased or decreased. The program then pauses for approximately 500 milliseconds while the valve adjusts itself to the new voltage output, then repeats. This algorithm allows for a self-correcting system based on ever-changing water flow; whether the sprinkler is used on city water, well water, or next door to a laundry-mat, no calibration is necessary as distance always corresponds to the same flow rate.

Board Modifications

The SARD boards, as shipped by Freescale, lack interfaces for the two TPM units. One board needed to be modified to link the first channel of each TPM to the pin interface JP105. This was accomplished by tracing the appropriate pins out of the MCU and soldering wrapping wire to them, then running the wire underneath the board to the bottom side of the JP105 connector. Two unused pins, six and eight, were selected and soldered to the respective wire traces. The five-volt output of TP104 was similarly connected to JP105 pin ten; TP104 is nothing more than a dot of solder, but by connecting it to a pin, we were able to use it to power the position sensor and flow meter.

Power Supply

This system is made up of several different components which need different voltages and currents to operate properly. A list of components along with the currents and voltages is shown in Figure 3.

Component	Voltage (V)	Current (mA)
Position Sensor	2.5	Maximum 20
Flow Sensor	5 to 24	8
Processor	9	
Valve actuator	110	

Figure 3

One option is to power the position sensor, flow sensor and processor with a 9 volt battery. This works very well and was implemented during the testing process. During this process the position sensor was actually powered from the 5 volt output off of the processor. This will have to be stepped down with a voltage divider as shown in figure 4.

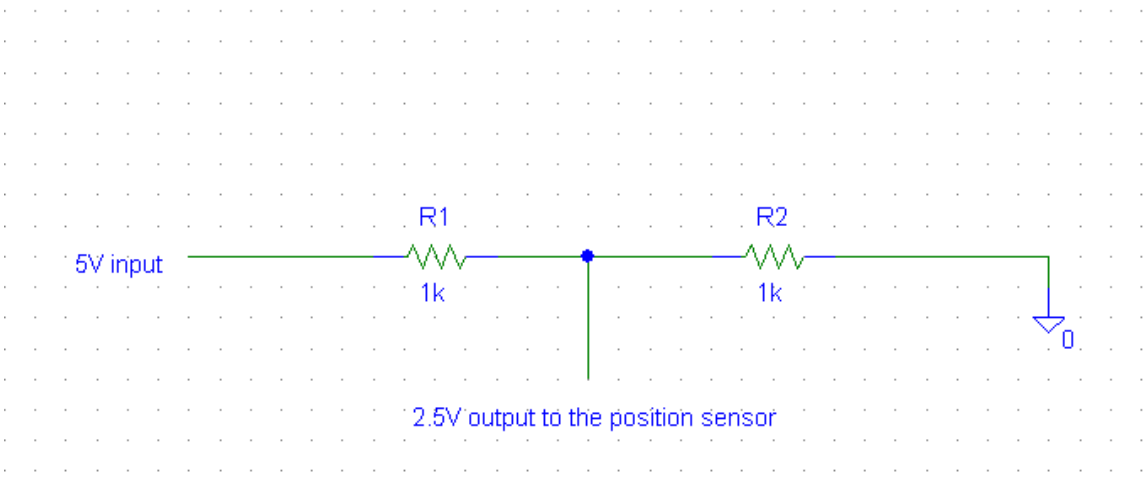


Figure 4: Voltage Divider

Ideally the system will be powered by a 120V line which will run from the house or power source. The system will need to be converted from AC to DC and then stepped down to proper operating conditions. To help with this process a power supply from a Canon BJ-200 b/w bubble jet printer will be used as the AC to DC converter. Once the conversion takes place there a three power leads off the power supply which will be used to finish powering the system. Two of the leads from the power supply have voltage a 27.6V. The third lead has a voltage of 5V. One of the lead with a voltage of 27.6 can be used to power the flow meter and the position sensor. The power coming from this lead is 29.8 Watts. The power required operating the position sensor and the flow sensor is approximately .2 Watts. This means there is a lot of wasted power from this power supply. Therefore it is in the best interest of the system to get a smaller power supply for the final system.

The flow sensor has an operating voltage between 5 and 24 volts. The reason for such a variance in the operating voltage is because it depends on the desired out put amplitude on the frequency. The information needed from the flow sensor does not need specific amplitude there for any voltage with in these parameters will work. An operating voltage 12 volts was chosen for the sensor. The operating current need is only 8mA therefore the current also has to be reduced along with the voltage. The proper currents for this these two components are obtained by the circuit shown in Figure 5:

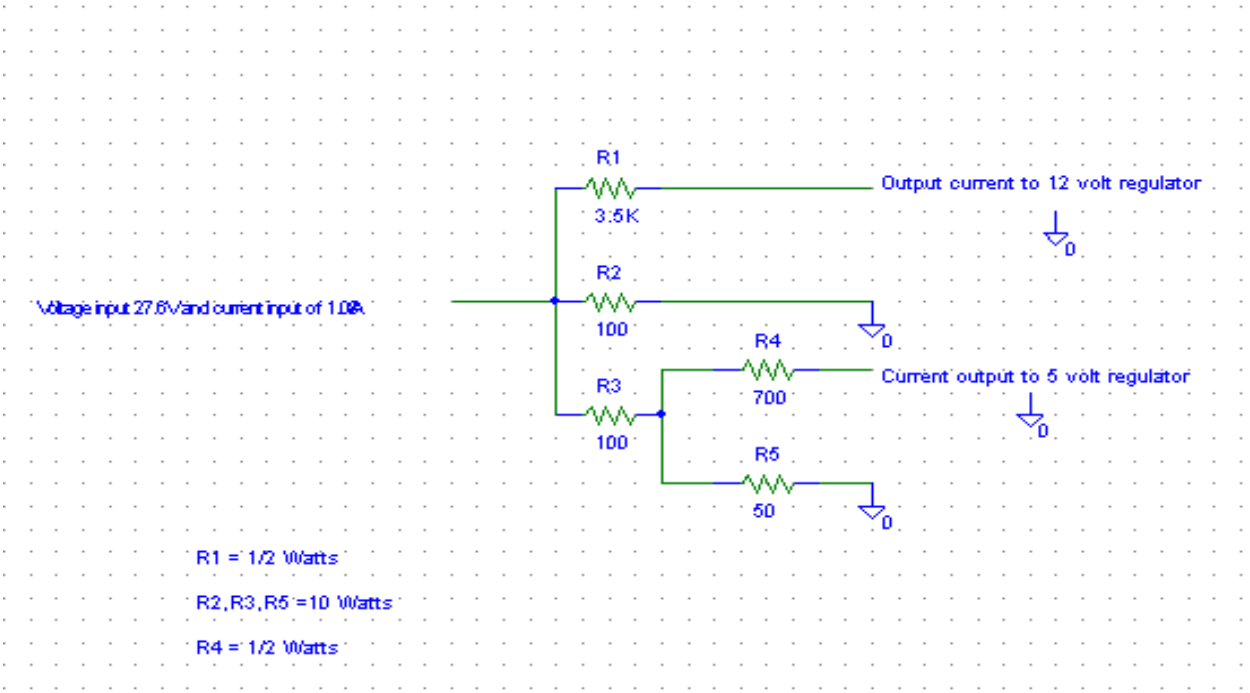


Figure 5: Current and power divider circuit

Once the currents are set for each component the voltages have to be correct. For the flow sensor a 12 volt voltage regulator will be used to control the input in the flow sensor. This means the circuit for the flow sensor will have a voltage of 12 volts and current of 19 mA.

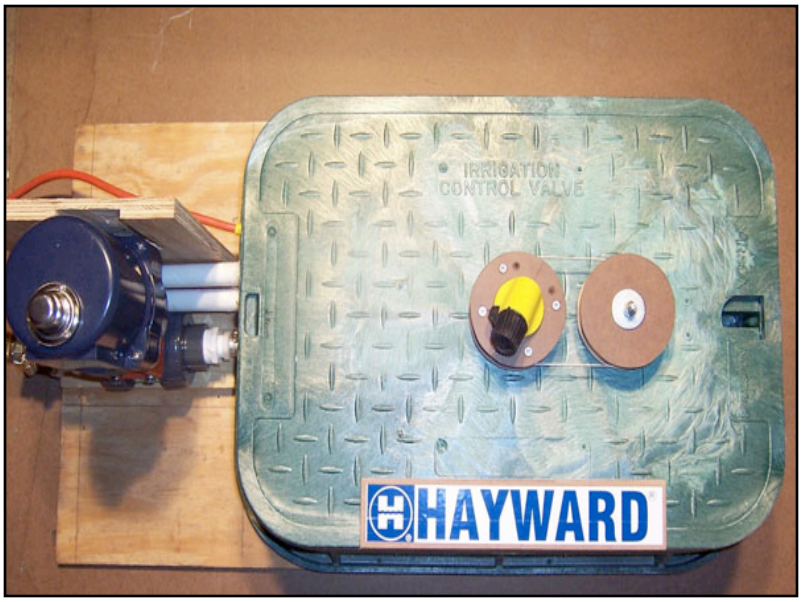
For the position sensor, the lead with 8 mA flowing through it will go through a 5 volt voltage regulator to reduce the voltage from 27.6 voltages to 5 volts. Once the 5 volts is coming out of the voltage regulator it can be passed through the voltage divider shown in figure 2 to get the desired 2.5 volts.

The processor board is designed to be operated by a 9volt battery. For the convenience this project a 9 volt battery will continue to be used for the processor board. It is understood if this product were to go into production the board would be powered off the power supply instead of the battery. This would be done to insure the consumer not having to change the battery every 2 to 3 months.

The final component is the controlling of the actuator. This is accomplished using the three volt out put on the processor board. The board communicates directly to the actuator and by varying the voltage controls how much to open the valve. This is only controlling the actuator. The powering of the actuator is a direct 110V line. This is the only component that has no altering of the input current or voltage.

Packaging

The photos below illustrate the overall packaging scheme for the prototype WaterBoy system. All major components, except for the valve, are contained in the valve box. Future improvements would be aimed toward finding a smaller valve that more appropriately fits the system capacity. This would allow for relocation of the valve to the inside of the box, making for a far more aesthetically pleasing final product. This prototype is contained by a valve box which is sealed to a plywood floor using screws and silicon. The lid of the box is lined with weather stripping to prevent water entry from the top. To ensure an airtight seal for the lid, it was proposed that clamps be installed on the side walls.



The underside of the lid of the valve box was used to store the major electrical components of the system including the SARD board. This area contains convenient compartments in which these types of components can be mounted in a protective and discreet manner. A hole was drilled in the middle of the lid to accompany the sprinkler head. Since the lower body portion of the sprinkler does not move, it could be secured to the lid and sealed. This leaves the upper portion of the head as the only component, aside from the valve, exposed to the elements. The pulley system, which was fabricated to enable proper function of the position sensor, is mounted in a similar fashion as the sprinkler head. Though leaks are prevented from entering through these two points, the system is made from particle board, a material susceptible to water damage. Future improvements would specify that this pulley system be composed of a weather resistant plastic material.

Business and Marketing

Business Talk

To begin prudent business research, we must know what type of market we are trying to approach along with the size and scope.

According to the US Census Bureau, consumers spent over $11 billion on landscaping services and maintenance in 2002 [4]. Additionally, "Installing and maintaining an irrigation system is one of the primary jobs for any landscape contractor or grounds maintenance business," according to the online magazine ProGardenBiz [5]. Of course, nearly all businesses start locally and expand, therefore it is important for a sound business plan to have strong roots in the local economy, or more specifically, southeast Michigan. According to the Michigan Nursery and Landscape Association (MNLA), a statewide trade association, Michigan is the 5th largest state in terms of dollars spent on landscaping services, generating $1.2 billion in annual sales, $655 million of that specifically in contractor and designers (irrigation falls into this group) [6].

It becomes quite clear there is no shortage of people looking to install home irrigation systems. The next question we must address is are people interested in the water conserving aspect of the system we are marketing? The Michigan Green Industry highlights some key aspects some key points of the Water Legacy Act that was adopted by the American Water Works Association on January 21, 2005 [7,8]:

Best Management Practices for Non-Agricultural Irrigation

- Include in the design water conserving technology, whenever economically feasible.
- Sprinkler head location should take into account possible obstructions.
- The use of pressure compensating materials in the system will maintain water distribution uniformity throughout the zone.

Clearly there is a strong demand, in fact legislation, promoting water conservation in Michigan. If our system can compete with the current systems monetarily, the added benefit of water conservation alone provides a strong argument for the feasibility of the product. If we also take into consideration lowered maintenance costs of the system, which we have shown to be a significant portion of landscaping business income, it becomes evident that a superior product could lead the untapped market of variable flow sprinkler controls.

Patent Information

The next step in our business research would be to determine patent-ability of the product. A search was conducted using the free database supplied by the US Patent and Trademark Office http://www.uspto.gov/. It is important to note that with a patent application, a required exhaustive search is performed by the USPTO for a substantial fee. With millions of patents in the database, we are by no means claiming here that these results are conclusive. The results are as follows:

Accurain – This is a variable distance control nozzle that adjusts the water distance based on the angular displacement of the water at the nozzle. The nozzle outputs a small stream of water (1 GPM max.) and "traces" out the pattern using a back and forth motion while continually adjusting the spray angle [9]. Patents 6,688,535 / 6,402,048

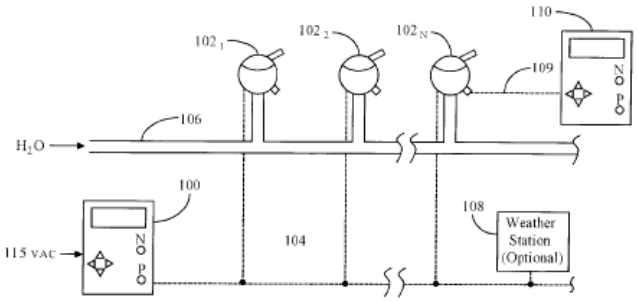


Figure 6: Accurain System

While the system appears to perform the stated claims, there are several drawbacks to this system: 1. At 1 GPM, to put one inch of water on the lawn per week (a generally accepted number) for a typical ¼ acre residential lawn, that would equate to 116 hours the sprinkler would have to be running…there are only 168 hours in the week. In fairness, they claim you can run two at a time, so that is only 58 hours per week. Another strong drawback to the system is that in order to reach the claimed 60 ft. diameter, the head must be mounted on a 4' pole, obstructing the yard in both practicality and aesthetically.

Again, as we mentioned, this is certainly no claim that this is the only patent addressing electronically controlled sprinkler nozzles, rather, in our novice search, this is the only one we found.

Marketing Aspects

While our claims our many, one of the greatest benefits of purchasing the WaterBoy system over a conventional system is what you can't see, the underground system. With the advantage of being able to spray any water pattern, compared to a typical residential system with 15' heads, you could reduce the number of heads by as much as 85%, taking an 18 head, 5 zone system down to 3 heads and three zones. Not only do you get the benefit of fewer heads, the underground plumbing, wiring and valve bodies saved will not only save money now, but also year after year as maintenance costs are dramatically reduced.

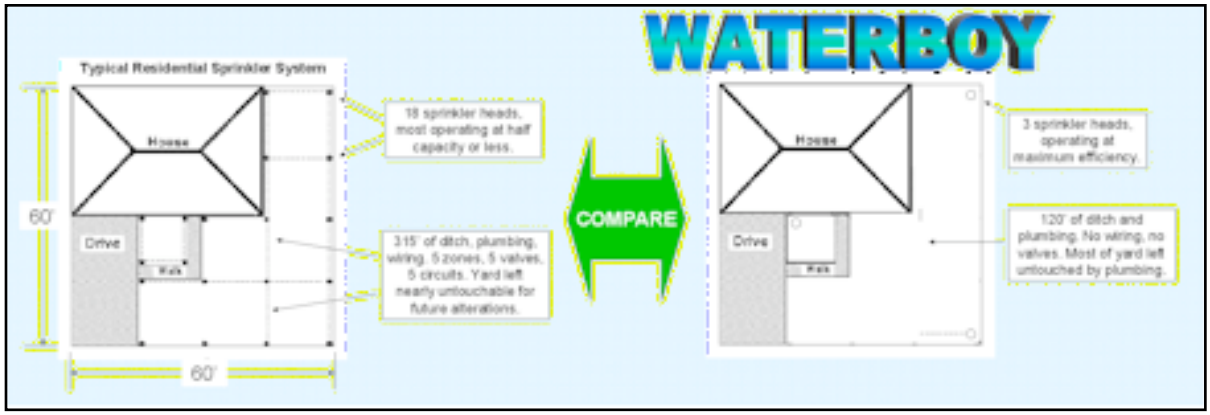


Figure 7: Residential vs. WaterBoy

Let us compare what is currently being used and the typical material sheet for an in ground system. The following chart was provided by DYI irrigation; a company that markets their system as being a cheaper, do-it-yourself kit opposed to professionally installed systems [10].

	DIY Irrigation	*Versus*	*Home Centers*
1 Each	6 Zone Controller	2 Each	6 Zone Controller
22 Each	Nelson 6000 Gear Driven Heads	26 Each	R-50 Ball Driven Heads
20 Each	6" Rain Bird Spray Bodies	24 Each	6" Rain Bird Spray Bodies
20 Each	Rain Bird Spray Nozzles	24 Each	Rain Bird Spray Nozzles
7 Each	1" Nelson Globe Valves "Auto"	13 Each	¾" Rain Bird Auto-valves
14 Each	Waterproof Wire Connectors	26 Each	Waterproof Wire Connectors
1 Each	Rain Shut-off	1 Each	Rain Shut-off
1 Each	PVC Cutter	1 Each	PVC Cutter
1 Each	PVC Cement	1 Each	PVC Cement
1 Each	PVC Primer	1 Each	PVC Primer
5 Each	18X1X500 Wire	7 Each	18X1X500 Wire
7 Each	Valve Boxes	13 Each	6" Valve Boxes
50 Each	Plotting Flags	50 Each	Plotting Flags
16 Each	SX1/2FPT Elbow 1"	9 Each	SX1/2FPT Elbow ¾"
26 Each	SXSX1/2FPT Tee 1"	27 Each	SXSX1/2FPT Tee ¾"
27 Each	SXS Elbow 1"	24 Each	SXS Elbow ¾"
14 Each	Male Adapter 1"	27 Each	Male Adapter ¾"
22 Each	Barbed Elbow ¾"	14 Each	SX3/4fpt Elbow ¾"
62 Each	Barbed Elbow ½"	12 Each	Sxsx3/4fpt Tee ¾"
1100 Feet	PVC Pipe 1" PR200	1520 Feet	PVC Piping ¾" Sch. 40
100 Feet	Swing Pipe (Funny Pipe)	152 Each	Couplings ¾"
8 Each	Rubber Gloves	50 Each	Cut-Off Risers
1 Each	Custom Cad Drawing	1 Each	Auto Cad Drawing
1 Each	How-To-DVD	Add $200.00 to upgrade Rain Bird R-50 Heads to Rain Bird T-Bird Heads and 12-zone model ESP Controller.	
500 Feet	Drip-Line		
DIY Irrigation Price: $1241.4		*Home Center Price: 1475.05*	

Table 6

"In the comparison system, we have estimated that a contractor using DIY Irrigation's design and comparable products would be in the price range of between $3600.00 and $4200.00" [10].

The material sheet for the system outlined above is a typical 12-zone irrigation system. Though our sprinkler heads are considerably more expensive than a basic pop-up, estimated at $250 per head, the rest of the material sheet drops by 2/3. The WaterBoy system would likely be installed by trained professionals, which charge by the man-hours involved. With only 2/3 of the plumbing and a fraction of the wiring, we would estimate a drop in labor on the order of 30%. These combined saving bring our system in right around $2000 installed, which is right in line with what is currently being charged.

If the labor and maintenance savings weren't enough, you can expect to drop your water usage by an estimated 30% with the WaterBoy system. How? By eliminating over coverage. In-ground system use what is referred to as head-to-head coverage in order to completely cover the yard. The problem with is, while you cover the whole yard, roughly 30% received double coverage.

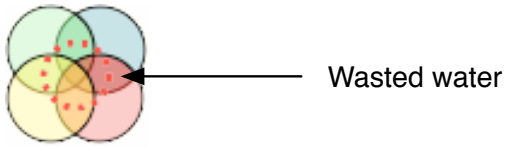


Figure 8: Water Coverage

Lets talk about water savings. If you live in or around southeast Michigan, you are likely paying at least $3.50 per 100 ft3 of water, or 748 gallons. If you water your lawn the recommended 1" per week, in a typical subdivision lot, you are going to put nearly 7,000 gallons of water on your lawn. Let us cut that in half since we usually receive a decent amount of rain, we will say 3,000 gallons per week. That comes out to a monthly "grass watering" bill of $56 per month. Using WaterBoy to cut that bill by 30% brings your monthly bill to $39 per month, a savings of $17 per month. During a full yard season of 6 months, that's over $100 per season.

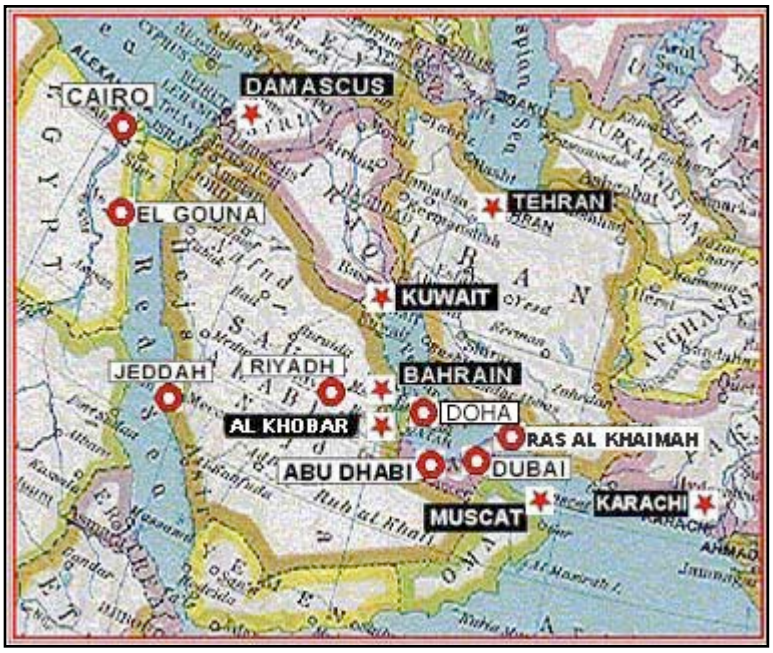
Our system would work in conjunction with a timer and rain sensors that are already on the market. Our goal is to make the WaterBoy system as user friendly, adaptable and universal as any system on the market. Replacement parts would be a readily available as any system.

Global Marketing

It used to be that watering grass was largely an American phenomenon. However, with the spotlight on us, that is quickly changing. Using the Toro website, one of the leading manufacturers of lawn irrigation equipment, it was more difficult to find a country that did not have a significant demand for turf irrigation supplies. Short of Africa, Toro products have a dominating presence throughout the world including Asia, The Middle East and Europe [10].

Dirab Golf Course, near Riyadh Saudi Arabia.

Hydroturf International, a leading Middle Eastern landscape irrigation company and distributor of Toro products, that serves both commercial and residential areas, supplied the picture above. Below is their service area where staffed distribution centers are circled [11].



With the labor savings, maintenance savings, and of course, water savings and a virtually unlimited global market, WaterBoy could have a very bright future.

Appendix A

Source code for main_pc.c

```
/*****************************************************************************
main_pc.c

Purpose: Accept data from a PC serial port and relay it to the
sprinkler valve controller.

Author: Todd Kulesza

Application Note: Based on SMAC Application Template
*****************************************************************************/

#include <hidef.h> /* for EnableInterrupts macro */
#include "device_header.h"
#include "pub_def.h"
#include "simple_mac.h"
#include "mc13192_hw_config.h"
#include "main_pc.h"

/* Global Variables */
UINT8 gu8RTxMode; /* needed for s-mac, application can read this variable */
INT8  gi8AppStatus = 0;
extern UINT8 gu8SCIDataFlag; /* These two are from SCI.c */
extern UINT8 gu8SCIData[2];

void main(void) {

       tRxPacket sRxPacket;
       tTxPacket sTxPacket;
       UINT8 au8RxDataBuffer[20];
       UINT8 au8TxDataBuffer[96]; /* Our packet should fit in 74 bytes */
       UINT8 pressed = 0;
       UINT8 current = 1;

       /*Initialize the packet.*/
       sTxPacket.u8DataLength = 0;
       sTxPacket.pu8Data = &au8TxDataBuffer[0];
       sRxPacket.u8DataLength = 0;
       sRxPacket.pu8Data = &au8RxDataBuffer[0];
       sRxPacket.u8MaxDataLength = 100;
       sRxPacket.u8Status = 0;
       gu8SCIDataFlag = 0;

#ifdef BOOTLOADER_ENABLED
       boot_init(); //Initialize the bootloader.
       boot_call(); //Check for user request to run bootloader.
                                  //App will not return, if Bootloader is requested.
#endif BOOTLOADER_ENABLED

       MCUInit ();
       MC13192Init ();
       MLMESetMC13192ClockRate(0);  /* Set initial Clk speed to 16MHz */
       UseExternalClock();    /* switch clock sources to 8MHz CPU bus*/
       SCIInit ();

       /* Init LED's */
       LED1 = LED_OFF; /* Default is off */
       LED2 = LED_OFF;
       LED3 = LED_OFF;
```

```
	LED4 = LED_OFF;

	LED1DIR = 1; /*Output*/
	LED2DIR = 1;
	LED3DIR = 1;
	LED4DIR = 1;

	/*****************************************************************
	To adjust output power call the MLME_MC13192_PA_output_adjust() with:

	MAX_POWER     (+3 to +5dBm)
	NOMINAL_POWER (0 dBm)
	MIN_POWER     ~(-16dBm)

	or somewhere custom ? (0-15, 11 (NOMINAL_POWER) being Default power)

	*****************************************************************/
	MLMEMC13192PAOutputAdjust(MAX_POWER);    /*Set MAX power setting*/
	//MLME_MC13192_PA_output_adjust(MIN_POWER);    /*Set MIN power setting*/
	//MLME_MC13192_PA_output_adjust(NOMINAL_POWER);    /*Set Nominal power setting
*/


	/* include your start up code here */
	EnableInterrupts; /* Turn on system interrupts */
	MLMESetChannelRequest(0);

	gi8AppStatus = INITIAL_STATE;

	/* FIXME: Debugging code, used to easy push-button operation */
	PB0PU = 1;     /* Pushbutton directions and pull-ups */
	PB0DIR = 0;
	PB1PU = 1;     /* Pushbutton directions and pull-ups */
	PB1DIR = 0;
	PB2PU = 1;     /* Pushbutton directions and pull-ups */
	PB2DIR = 0;
	PB3PU = 1;     /* Pushbutton directions and pull-ups */
	PB3DIR = 0;

	/* Stay in RX mode. */
	for (;;)
	{
		switch (gi8AppStatus)
		{
			case INITIAL_STATE:
				gi8AppStatus = RECEIVER_ALWAYS_ON;
				break;
			case RESET_STATE:
				MC13192Init();
				gi8AppStatus = RECEIVER_ALWAYS_ON;
				break;
			case RECEIVER_ALWAYS_ON:

				break;
			default:
				break;
		}

		/* Push button 2 is an easy way to start the sytsem */
		if (PUSH_BUTTON2 == PRESSED && !pressed)
		{
			pressed = 1;
			LED1 = LED_ON;
```

```
                        /* Send start packet */
                        sTxPacket.u8DataLength = 2;
                        sTxPacket.pu8Data[0] = 0;
                        sTxPacket.pu8Data[1] = 0;
                        MCPSDataRequest (&sTxPacket);
                }
                /* Push button 3 turns everything off */
                else if (PUSH_BUTTON3 == PRESSED && !pressed)
                {
                        pressed = 1;
                        LED1 = LED_ON;
                        /* Send stop packet */
                        sTxPacket.u8DataLength = 2;
                        sTxPacket.pu8Data[0] = 1;
                        sTxPacket.pu8Data[1] = 1;
                        MCPSDataRequest (&sTxPacket);
                }
                else if (PUSH_BUTTON1 != PRESSED &&
                        PUSH_BUTTON2 != PRESSED &&
                        PUSH_BUTTON3 != PRESSED)
                {
                        LED1 = LED_OFF;
                        pressed = 0;
                }

                /* Check for data on the serial port */
                if (gu8SCIDataFlag)
                {
                        UINT8 data = gu8SCIData[0];
                        /* If this is the start of a pattern, reset our array position */
                        if (!data)
                        {
                                sTxPacket.u8DataLength = 0;
                        }
                        /* Store the distance for the current position in the send buffer
*/
                        else
                        {
                                sTxPacket.pu8Data[sTxPacket.u8DataLength++] = data;
                        }
                        /* If data is '1' our pattern is complete, so send the packet */
                        if (data == 1)
                        {
                                LED4 = LED_ON;
                                MCPSDataRequest (&sTxPacket);
                                LED2 = LED_OFF;
                                LED3 = LED_OFF;
                                LED4 = LED_OFF;
                        }

                        gu8SCIDataFlag = 0;


                }
                delay (0x0FFF);
                delay (0x0FFF);
                delay (0x0FFF);
        }
}

/* This is a Freescale stub function, we don't actually need
 * to do anything here. */
void MCPSDataIndication(tRxPacket *sRxPacket)
{
```

```
        if (sRxPacket->u8Status == TIMEOUT)
        {
                /* Put timeout condition code here */

        }

        if (sRxPacket->u8Status == SUCCESS)
        {
                /*Packet received.  Handle it.*/

        }
}

void MLMEMC13192ResetIndication()
{
        /* Notifies you that the MC13192 has been reset.
         * Application must handle this here.
         */


        gi8AppStatus = RESET_STATE;

}

void delay (INT16 count)
{
        INT16 i;
        for (i=0; i < count; i++);
}
```

Appendix B

Source code for main_valve.c

```
/*****************************************************************************
main_valve.c

Purpose: Process data from a position sensor and flow meter,
adjusting a motorized water valve to create the desired
water pressure for shooting water a specified distance.

Author: Todd Kulesza <todd@dropline.net>

Based on SMAC Application Template

*****************************************************************************/

#include <hidef.h> /* for EnableInterrupts macro */
#include <string.h>
#include "device_header.h"
#include "pub_def.h"
#include "simple_mac.h"
#include "mc13192_hw_config.h"
#include "SCI.h"
#include "flash_utility.h"
#include "main_valve.h"

/* Global Variables */
UINT8 gu8RTxMode; /* needed for SMAC, application can read this variable */
INT8  gi8AppStatus = 0;
UINT8 sprinkling = 0; /* Set to '1' when sprinkler is running */
UINT8 pattern[POSITIONS]; /* Distances required for our watering pattern */
UINT16 input_last = 0; /* 'input_*' are for our flow meter frequency */
UINT16 input_current = 0;
UINT16 input_period = 0;

/* Main program loop */
void main (void)
{
       tRxPacket sRxPacket;
       tTxPacket sTxPacket;
       UINT8 au8RxDataBuffer[96]; /* Pattern packet should be 74 bytes */
       UINT8 au8TxDataBuffer[20];

       /*Initialize the packet.*/
       sTxPacket.u8DataLength = 0;
       sTxPacket.pu8Data = &au8TxDataBuffer[0];
       sRxPacket.u8DataLength = 0;
       sRxPacket.pu8Data = &au8RxDataBuffer[0];
       sRxPacket.u8MaxDataLength = 100;
       sRxPacket.u8Status = 0;

#ifdef BOOTLOADER_ENABLED
       boot_init(); //Initialize the bootloader.
       boot_call(); //Check for user request to run bootloader.
                                  //App will not return, if Bootloader is requested.
#endif BOOTLOADER_ENABLED

       MCUInit ();
       MC13192Init ();
       MLMESetMC13192ClockRate(0);  /* Set initial Clk speed to 16MHz */
       UseExternalClock();    /* switch clock sources to 8MHz CPU bus*/
```

```
	SCIInit ();

	/*********************************************************************
	To adjust output power call the MLME_MC13192_PA_output_adjust() with:

	MAX_POWER     (+3 to +5dBm)
	NOMINAL_POWER (0 dBm)
	MIN_POWER     ~(-16dBm)

	or somewhere custom ? (0-15, 11 (NOMINAL_POWER) being Default power)

	*********************************************************************/
	MLMEMC13192PAOutputAdjust(MAX_POWER);    /*Set MAX power setting*/
	/* include your start up code here */
	EnableInterrupts; /* Turn on system interrupts */
	MLMESetChannelRequest(0);

	/* Initial controller setup */
	valve_init ();

	gi8AppStatus = INITIAL_STATE;

	for (;;)
	{
		LED2 ^= 1;

		switch (gi8AppStatus)
		{
			case INITIAL_STATE:
				gi8AppStatus = LISTENING;
				break;
			case RESET_STATE:
				MC13192Init ();
				gi8AppStatus = LISTENING;
				break;
			case LISTENING:
				MLMERXEnableRequest (&sRxPacket, 0);
				LOW_POWER_WHILE();
				break;
			case SPRINKLING:
				sprinkler_run ();
				break;
			case STOP_SPRINKLING:
				sprinkler_stop ();
				gi8AppStatus = LISTENING;
				break;
			default:
				break;
		}
	}

	return;
}

/* MC13192 Rx interrupt handler */
void MCPSDataIndication(tRxPacket *sRxPacket)
{
	if (sRxPacket->u8Status == TIMEOUT)
	{

	/* Put timeout condition code here */

	}
```

```
	if (sRxPacket->u8Status == SUCCESS)
	{
		UINT8 byte1, byte2;

		byte1 = sRxPacket->pu8Data[0];
		byte2 = sRxPacket->pu8Data[1];

		/* Check for packet type */
		if (!byte1 && !byte2)
			gi8AppStatus = SPRINKLING;
		else if (byte1 == 1 && byte2 == 1)
			gi8AppStatus = STOP_SPRINKLING;
		else
		{
			/* Skip the first byte, it's just a
			 * packet-type indicator. */
			flash_pattern_store (sRxPacket->pu8Data + 1);
		}
	}
}

/* Notification that the MC13192 has been reset */
void MLMEMC13192ResetIndication (void)
{
	gi8AppStatus = RESET_STATE;

	return;
}

/* One-time setup of valve registers */
void valve_init (void)
{
	/* Read our desired watering pattern */
	flash_pattern_read (pattern);

	/* Enable our Analog-to-Digital converter */
	ATD1PE = 0x01; /* enable ADC channel AD0 */
	ATD1C=0xE1; /* Prescale value of 4, 8-bit data, left-justified */

	/* Setup flash interface */
	FSTAT_FACCERR = 1;
	FSTAT_FPVIOL = 1;
	FCDIV = 0x27; /* Bus speed of 8MHz, FCLK of 200kHz */

	/* Enable the PWM on TPM1 */
	TPM1SC = 0x15; /* Use bus clock with divisor of 32 */
	TPM1MODH = 0x00;
	TPM1MODL = 0xff;

	/* Edge-select PWM */
	TPM1C0SC = 0x28;
	TPM1C0VH = 0x00;
	TPM1C0VL = 0x01;

	/* Enable input capture on TPM2 */
	TPM2SC = 0x0f; /* System clock with 128 divider */

	TPM2C0SC = 0x44; /* Interrupt-based input capture */

	/* FIXME: Debugging code */
	LED2 = LED_OFF;
	LED2DIR = 1;
```

```
	LED4 = LED_OFF;
	LED4DIR = 1;

	return;
}

/* Read in the current voltage from the position sensor */
UINT8 position_read (void)
{
	UINT8 pos;

	/* Read pressure sensor on AD0 */
	ATD1SC = 0x00;
	/* Wait until the Conversion Complete flag is set */
	while ((ATD1SC & 0x80) != 0x80);
	pos = ATD1RH;

	return pos;
}

/* Given the analog value, determin which of the 72
 * discrete positions our sprinkler is in */
UINT8 compute_position (UINT8 value)
{
	UINT16 position = 0;

	/* values < 6 or > 201 are the same position, 0 */
	if (!(value < 6 || value > 201))
	{
		position = value - 6;
		/* Acrobatics to avoid float-point arithmetic */
		position *= 100;
		position /= 275;
		position++;
	}

	return (position & 255);
}

UINT16 compute_flow (UINT8 distance)
{
	UINT16 flow;

	/* Our flow vs distance equation is
	 * 0.1542x - 5.7073, but we need to multiply everything
	 * by 1000 to avoid floating-point crack rock. */
	flow = ((distance * 1000) + 5707) / 154;

	return flow;
}

/* Read in the time of the capture event
 *  and compare it with the previous time */
interrupt void input_capture (void)
{
	UINT8 high, low, status;

	high = TPM2C0VH;
	low = TPM2C0VL;

	input_last = input_current;
	/* Combine two 8-bit ints into a 16-bit int */
	input_current = (high << 8) + low;
```

```
	/* Make sure neither is 0, as this means they are uninitalized */
	if (input_last && input_current)
		input_period = input_current - input_last;

	/* Clear the interrupt */
	status = TPM2C0SC;
	TPM2C0SC_CH0F = 0;

	return;
}

void sprinkler_run (void)
{
	UINT8 i, modulo, current;

	LED4 = LED_ON;

	/* Just barely open the valve */
	TPM1C0VH = 0x00;
	TPM1C0VL = 0x78;
	current = TPM1C0VH;
	current = TPM1C0VL;
	sprinkling = 1;

	/* Loop until we change @sprinkling in the
	 * sprinkler_stop() method */
	while (sprinkling)
	{
		UINT8 position, value, distance;
		UINT8 output_current_low, output_current_high;
		UINT16 flow_desired;

		/* Figure out the sprinkler's position */
		value = position_read ();
		position = compute_position (value);

		TPM1C0VH = 0x00;
		TPM1C0VL = pattern[position];

		current = TPM1C0VH;
		current = TPM1C0VL;

		/* Determine desired distance */
		distance = pattern[position];

		/* Compute needed water flow */
		flow_desired = compute_flow (distance);
		if (input_period)
		{
			UINT8 temp_high, temp_low;

			temp_low = input_period & 255;
			temp_high = (input_period >> 8) & 255;

			SCIStartTransmit (temp_high);
			SCIStartTransmit (temp_low);
		}
		if (flow_desired)
		{
			UINT8 temp_high, temp_low;

			temp_low = flow_desired & 255;
```

```
                temp_high = (flow_desired >> 8) & 255;

                SCIStartTransmit (temp_high);
                SCIStartTransmit (temp_low);
        }

        /* If our current water flow is more than
         * needed, slightly close the valve.  If it
         * is less than needed, slightly open the valve.
         * If we're really close, don't screw with
         * anything. */

        output_current_high = TPM1C0VH;
        output_current_low = TPM1C0VL;
        SCIStartTransmit (output_current_high);
        SCIStartTransmit (output_current_low);
        if (input_period < (flow_desired - 10))
        {
                /* Slightly close valve */
                 output_current_low -= 0x0A;
        }
        else if (input_period > (flow_desired + 10))
        {
                /* Slightly open valve */
                output_current_low += 0x0A;
        }
        TPM1C0VH = output_current_high;
        TPM1C0VL = output_current_low;
        SCIStartTransmit (output_current_high);
        SCIStartTransmit (output_current_low);

        /* Blink the light as a debugging indicator */
        LED4 ^= 1;

        /* Wait for valve to adjust */
        for (i = 0; i < 64; i++)
                delay (0x0FFF);
    }

    /* Close the valve */
    TPM1C0VH = 0x00;
    TPM1C0VL = 0x10;
    modulo = TPM1C0VH;  /* We need to read each register for */
    modulo = TPM1C0VL;  /* our 16-bit change to take effect. */
                        /* Silly 8-bit MCU... */
    return;
}

void sprinkler_stop (void)
{
    sprinkling = 0;
    LED4 = LED_OFF;

    return;
}

/* Simple delay loop */
void delay (INT16 count)
{
    INT16 i;

    for (i=0; i < count; i++);
```

```
	return;
}
```

Source code for flash_utility.c

```
/*****************************************************************************
flash_utility.c

Purpose: Handle reading and writing to on-board flash memory.

Author: Todd Kulesza

Application Note: Based on SMAC Application Template
*****************************************************************************/

#include "flash_utility.h"

/* Store @distances into flash memory */
void flash_pattern_store (UINT8 distances[])
{
	int i;

	/* Erase the block */
	FlashErase1 (0xf200);

	/* Program our pattern into flash */
	for (i = 0; i < POSITIONS; i++)
	{
		/* Our offset in FLASH is 0xf200 */
		FlashProg1 (0xf200 + i, distances[i]);
	}

	return;
}

/* Read @distances from flash memory */
void flash_pattern_read (UINT8 distances[])
{
	int i;

	for (i = 0; i < POSITIONS; i++)
	{
		memcpy (distances + i, 0xf200 + i, 1);
	}

	return;
}
```

Source code for flash_utility.asm

```
;*******************************************************************
; HEADER_START
;
;         $File Name: flash_utility.asm$
;      Project:        ValveController
;      Description:    Code for writing/erasing flash memory
;      Platform:       HCS08
;      $Version: 1.0.0.0$
;      $Date: Nov-15-2005$
;      $Last Modified By: Todd Kulesza$
;
;             Based on code provided by Freescale's HCS08 Reference Manual
```

```
;
; HEADER_END

       include "9S08GB60.inc"

       XDEF FlashErase1
       XDEF FlashProg1

CodeSection: SECTION

;*******************************************************************
;* FlashErase1 - erases one page of FLASH (512 bytes)
;*
;* On entry... H:X - points at a location in the page to be erased
;*
;* Calling convention:
;* jsr FlashErase1
;*
;* Uses: DoOnStack which uses SpSub
;* Returns: H:X unchanged and A = FSTAT shifted left by 2 bits
;* Z=1 if OK, Z=0 if protect violation or access error
;* uses 32 bytes of stack space + 2 bytes for BSR/JSR used to call it
;*******************************************************************

FlashErase1:
       psha ;adjust sp for DoOnStack entry
       lda #(mFPVIOL+mFACCERR) ;mask
       sta FSTAT ;abort any command and clear errors
       lda #mPageErase ;mask pattern for page erase command
       bsr DoOnStack ;finish command from stack-based sub
       ais #1 ;deallocate data location from stack
       rts ;Z = 0 means there was an error
;********************

;*******************************************************************
;* FlashProg1 - programs one byte of FLASH
;* This routine waits for the command to complete before returning.
;* assumes location was blank. This routine can be run from FLASH
;*
;* On entry... H:X - points at the FLASH byte to be programmed
;* A holds the data for the location to be programmed
;*
;* Calling convention:
;* jsr FlashProg1
;*
;* Uses: DoOnStack which uses SpSub
;* Returns: H:X unchanged and A = FSTAT shifted left by 2 bits
;* Z=1 if OK, Z=0 if protect violation or access error
;* uses 32 bytes of stack space + 2 bytes for BSR/JSR used to call it
;*******************************************************************
FlashProg1:
       psha ;temporarily save entry data
       lda #(mFPVIOL+mFACCERR) ;mask
       sta FSTAT ;abort any command and clear errors
       lda #mByteProg ;mask pattern for byte prog command
       bsr DoOnStack ;execute prog code from stack RAM
       ais #1 ;deallocate data location from stack
       rts ;Z = 0 means there was an error
;********************

;*******************************************************************
;* DoOnStack - copy SpSub onto stack and call it (see also SpSub)
;* Deallocates the stack space used by SpSub after returning from it.
```

```
;* Allows flash prog/erase command to execute out of RAM (on stack)
;* while flash is out of the memory map.
;* This routine can be used for flash byte-program or erase commands
;*
;* Calling Convention:
;* psha ;save data to program (or dummy
;* ; data for an erase command)
;* lda #(mFPVIOL+mFACCERR) ;mask
;* sta FSTAT ;abort any command and clear errors
;* lda #mByteProg ;mask pattern for byte prog command
;* jsr DoOnStack ;execute prog code from stack RAM
;* ais #1 ;deallocate data location from stack
;* ; without disturbing A or CCR
;*
;* or substitute #mPageErase for page erase
;*
;* Uses 29 bytes on stack + 2 bytes for BSR/JSR used to call it
;* returns H:X unchanged and A=0 and Z=1 if no flash errors
;*****************************************************************
DoOnStack:
       pshx
       pshh ;save pointer to flash
       psha ;save command on stack
       ldhx #SpSubEnd ;point at last byte to move to stack
SpMoveLoop:
       lda ,x ;read from flash
       psha ;move onto stack
       aix #-1 ;next byte to move
       cphx #SpSub-1 ;past end?
       bne SpMoveLoop ;loop till whole sub on stack
       tsx ;point to sub on stack
       tpa ;move CCR to A for testing
       and #$08 ;check the I mask
       bne I_set ;skip if I already set
       sei ;block interrupts while FLASH busy
       lda SpSubSize+6,sp ;preload data for command
       jsr ,x ;execute the sub on the stack
       cli ;ok to clear I mask now
       bra I_cont ;continue to stack de-allocation
I_set:
       lda SpSubSize+6,sp ;preload data for command
       jsr ,x ;execute the sub on the stack
I_cont:
       ais #SpSubSize+3 ;deallocate sub body + H:X + command
       ;H:X flash pointer OK from SpSub
       lsla ;A=00 & Z=1 unless PVIOL or ACCERR
       rts ;to flash where DoOnStack was called
;********************

;*****************************************************************
;* SpSub - This variation of SpSub performs all of the steps for
;* programming or erasing flash from RAM. SpSub is copied onto the
;* stack, SP is copied to H:X, and then the copy of SpSub in RAM is
;* called using a JSR 0,X instruction.
;*
;* At the time SpSub is called, the data to be programmed (dummy data
;* for an erase command), is in A and the flash address is on the
;* stack above SpSub. After return, PVIOL and ACCERR flags are in bits
;* 6 and 5 of A. If A is shifted left by one bit after return, it
;* should be zero unless there was a flash error.
;*
;* Uses 24 bytes on stack + 2 bytes if a BSR/JSR calls it
;*****************************************************************
```

```
SpSub:
       ldhx SpSubSize+4,sp ;get flash address from stack
       sta 0,x ;write to flash; latch addr and data
       lda SpSubSize+3,sp ;get flash command
       sta FCMD ;write the flash command
       lda #mFCBEF ;mask to initiate command
       sta FSTAT ;[pwpp] register command
       nop ;[p] want min 4~ from w cycle to r
ChkDone:
       lda FSTAT ;[prpp] so FCCF is valid
       lsla ;FCCF now in MSB
       bpl ChkDone ;loop if FCCF = 0
SpSubEnd:
       rts ;back into DoOnStack in flash
SpSubSize:    equ          (*-SpSub)
;*********************
```

Appendix C

Source code for SprinklerGUI.vb

```
Public Class SprinklerForm
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents Label2 As System.Windows.Forms.Label
    Friend WithEvents txtLength As System.Windows.Forms.TextBox
    Friend WithEvents picLawn As System.Windows.Forms.PictureBox
    Friend WithEvents cbPattern As System.Windows.Forms.ComboBox
    Friend WithEvents btnUpdate As System.Windows.Forms.Button
    Friend WithEvents lblLMax As System.Windows.Forms.Label
    Friend WithEvents lblWMin As System.Windows.Forms.Label
    Friend WithEvents lblLMin As System.Windows.Forms.Label
    Friend WithEvents lblWMax As System.Windows.Forms.Label
    Friend WithEvents rtbFeedback As System.Windows.Forms.RichTextBox
    Friend WithEvents btnReset As System.Windows.Forms.Button
    Friend WithEvents btnClear As System.Windows.Forms.Button
    Friend WithEvents Label4 As System.Windows.Forms.Label
    Friend WithEvents cbDebug As System.Windows.Forms.CheckBox
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        Me.cbPattern = New System.Windows.Forms.ComboBox
        Me.Label1 = New System.Windows.Forms.Label
        Me.Label2 = New System.Windows.Forms.Label
        Me.txtLength = New System.Windows.Forms.TextBox
        Me.btnUpdate = New System.Windows.Forms.Button
        Me.picLawn = New System.Windows.Forms.PictureBox
        Me.lblLMax = New System.Windows.Forms.Label
        Me.lblWMin = New System.Windows.Forms.Label
        Me.lblLMin = New System.Windows.Forms.Label
        Me.lblWMax = New System.Windows.Forms.Label
        Me.rtbFeedback = New System.Windows.Forms.RichTextBox
```

```
        Me.btnReset = New System.Windows.Forms.Button
        Me.btnClear = New System.Windows.Forms.Button
        Me.Label4 = New System.Windows.Forms.Label
        Me.cbDebug = New System.Windows.Forms.CheckBox
        Me.SuspendLayout()
        '
        'cbPattern
        '
        Me.cbPattern.Items.AddRange(New Object() {"Square", "Circle",
"Latcha Design"})
        Me.cbPattern.Location = New System.Drawing.Point(8, 32)
        Me.cbPattern.Name = "cbPattern"
        Me.cbPattern.Size = New System.Drawing.Size(152, 21)
        Me.cbPattern.TabIndex = 0
        '
        'Label1
        '
        Me.Label1.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.0!,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label1.Location = New System.Drawing.Point(8, 8)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(136, 24)
        Me.Label1.TabIndex = 1
        Me.Label1.Text = "Select Pattern:"
        Me.Label1.TextAlign = System.Drawing.ContentAlignment.MiddleLeft
        '
        'Label2
        '
        Me.Label2.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.0!,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label2.Location = New System.Drawing.Point(200, 8)
        Me.Label2.Name = "Label2"
        Me.Label2.Size = New System.Drawing.Size(112, 24)
        Me.Label2.TabIndex = 2
        Me.Label2.Text = "Radius"
        Me.Label2.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        '
        'txtLength
        '
        Me.txtLength.Location = New System.Drawing.Point(208, 32)
        Me.txtLength.Name = "txtLength"
        Me.txtLength.Size = New System.Drawing.Size(96, 20)
        Me.txtLength.TabIndex = 3
        Me.txtLength.Text = ""
        '
        'btnUpdate
        '
        Me.btnUpdate.Location = New System.Drawing.Point(384, 24)
        Me.btnUpdate.Name = "btnUpdate"
        Me.btnUpdate.Size = New System.Drawing.Size(96, 48)
        Me.btnUpdate.TabIndex = 6
        Me.btnUpdate.Text = "Execute"
        '
        'picLawn
        '
        Me.picLawn.Location = New System.Drawing.Point(96, 144)
        Me.picLawn.Name = "picLawn"
        Me.picLawn.Size = New System.Drawing.Size(400, 400)
        Me.picLawn.SizeMode = System.Windows.Forms.PictureBoxSizeMode.StretchImage
        Me.picLawn.TabIndex = 8
        Me.picLawn.TabStop = False
        '
        'lblLMax
```

```
        '
        Me.lblLMax.BackColor = System.Drawing.Color.Transparent
        Me.lblLMax.Location = New System.Drawing.Point(440, 104)
        Me.lblLMax.Name = "lblLMax"
        Me.lblLMax.Size = New System.Drawing.Size(56, 16)
        Me.lblLMax.TabIndex = 9
        Me.lblLMax.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        '
        'lblWMin
        '
        Me.lblWMin.BackColor = System.Drawing.Color.Transparent
        Me.lblWMin.Location = New System.Drawing.Point(24, 144)
        Me.lblWMin.Name = "lblWMin"
        Me.lblWMin.Size = New System.Drawing.Size(64, 24)
        Me.lblWMin.TabIndex = 10
        Me.lblWMin.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        '
        'lblLMin
        '
        Me.lblLMin.Location = New System.Drawing.Point(88, 104)
        Me.lblLMin.Name = "lblLMin"
        Me.lblLMin.Size = New System.Drawing.Size(40, 16)
        Me.lblLMin.TabIndex = 11
        Me.lblLMin.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        '
        'lblWMax
        '
        Me.lblWMax.Location = New System.Drawing.Point(24, 520)
        Me.lblWMax.Name = "lblWMax"
        Me.lblWMax.Size = New System.Drawing.Size(64, 24)
        Me.lblWMax.TabIndex = 12
        Me.lblWMax.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        '
        'rtbFeedback
        '
        Me.rtbFeedback.Location = New System.Drawing.Point(520, 136)
        Me.rtbFeedback.Name = "rtbFeedback"
        Me.rtbFeedback.Size = New System.Drawing.Size(216, 408)
        Me.rtbFeedback.TabIndex = 14
        Me.rtbFeedback.Text = ""
        '
        'btnReset
        '
        Me.btnReset.Location = New System.Drawing.Point(536, 24)
        Me.btnReset.Name = "btnReset"
        Me.btnReset.Size = New System.Drawing.Size(96, 48)
        Me.btnReset.TabIndex = 15
        Me.btnReset.Text = "Reset"
        '
        'btnClear
        '
        Me.btnClear.Location = New System.Drawing.Point(576, 576)
        Me.btnClear.Name = "btnClear"
        Me.btnClear.Size = New System.Drawing.Size(104, 32)
        Me.btnClear.TabIndex = 16
        Me.btnClear.Text = "Clear Feedback"
        '
        'Label4
        '
        Me.Label4.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.0!,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label4.Location = New System.Drawing.Point(520, 112)
        Me.Label4.Name = "Label4"
```

```
        Me.Label4.Size = New System.Drawing.Size(160, 24)
        Me.Label4.TabIndex = 17
        Me.Label4.Text = "Feedback"
        '
        'cbDebug
        '
        Me.cbDebug.Location = New System.Drawing.Point(584, 552)
        Me.cbDebug.Name = "cbDebug"
        Me.cbDebug.Size = New System.Drawing.Size(88, 16)
        Me.cbDebug.TabIndex = 18
        Me.cbDebug.Text = "Debug Mode"
        '
        'SprinklerForm
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(752, 629)
        Me.Controls.Add(Me.cbDebug)
        Me.Controls.Add(Me.Label4)
        Me.Controls.Add(Me.btnClear)
        Me.Controls.Add(Me.btnReset)
        Me.Controls.Add(Me.rtbFeedback)
        Me.Controls.Add(Me.lblWMax)
        Me.Controls.Add(Me.lblLMin)
        Me.Controls.Add(Me.lblWMin)
        Me.Controls.Add(Me.lblLMax)
        Me.Controls.Add(Me.picLawn)
        Me.Controls.Add(Me.btnUpdate)
        Me.Controls.Add(Me.txtLength)
        Me.Controls.Add(Me.Label2)
        Me.Controls.Add(Me.Label1)
        Me.Controls.Add(Me.cbPattern)
        Me.Name = "SprinklerForm"
        Me.Text = "Smart Sprinkler User Interface"
        Me.ResumeLayout(False)

    End Sub

#End Region

    Dim distances(72) As Byte

    Private Sub btnUpdate_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnUpdate.Click
        'Check which pattern is selected, and call the appropriate calculation
function.
        If (cbPattern.SelectedIndex = 0) Then
            lblLMin.Text = "0 ft"
            lblLMax.Text = txtLength.Text + " ft"
            lblWMin.Text = "0 ft"
            lblWMax.Text = CStr(CInt(txtLength.Text) * 0.707) + " ft"
            rtbFeedback.AppendText("Radius accepted as: " + txtLength.Text + " feet."
+ ControlChars.NewLine)
            'Converts the radius to a double variable and calls the calculation.
            performCalcSquare((CDbl(txtLength.Text) * 0.707))
        ElseIf (cbPattern.SelectedIndex = 1) Then
            lblLMin.Text = "0 ft"
            lblLMax.Text = txtLength.Text + " ft"
            lblWMin.Text = "0 ft"
            lblWMax.Text = txtLength.Text + " ft"
            rtbFeedback.AppendText("Radius accepted as: " + txtLength.Text + " feet."
+ ControlChars.NewLine)
            'Converts the radius to an integer variable and calls the calculation.
            performCalcCircle(CInt(txtLength.Text))
```

```
        ElseIf (cbPattern.SelectedIndex = 2) Then
            lblLMin.Text = "0 ft"
            lblLMax.Text = CStr(2 * CInt(txtLength.Text) * 0.707) + " ft"
            lblWMin.Text = "0 ft"
            lblWMax.Text = CStr(2 * CInt(txtLength.Text) * 0.707) + " ft"
            rtbFeedback.AppendText("Radius accepted as: " + txtLength.Text + " feet."
+ ControlChars.NewLine)
            'Converts the radius to a double variable and calls the calculation.
            performCalcLatcha(CDbl(txtLength.Text) * 0.707, CDbl(txtLength.Text) / 2)
        End If



    End Sub

    Private Function performCalcSquare(ByVal length As Double)
        'x is the loop variable, p is the degree value, o is the converted radian
value.
        'These are all based off of the equations listed in the report.
        Dim x As Integer
        Dim p As Double
        Dim o As Double
        p = 0
        For x = 1 To 72
            o = p * Math.PI / 180
            If x >= 1 And x <= 10 Then
                distances(x) = CByte(Math.Round(length / Math.Cos(o)))
            End If
            If x >= 11 And x <= 19 Then
                distances(x) = CByte(Math.Round(length / Math.Cos(o - (Math.PI / 4))))
            End If
            If x >= 20 And x <= 28 Then
                distances(x) = CByte(Math.Round(length / Math.Cos(o - (Math.PI / 2))))
            End If
            If x >= 29 And x <= 37 Then
                distances(x) = CByte(Math.Round(length / Math.Cos(Math.PI - o)))
            End If
            If x >= 38 And x <= 46 Then
                distances(x) = CByte(Math.Round(length / Math.Cos(o - Math.PI)))
            End If
            If x >= 47 And x <= 55 Then
                distances(x) = CByte(Math.Round(length / Math.Cos((Math.PI * 3 / 2) -
o)))
            End If
            If x >= 56 And x <= 64 Then
                distances(x) = CByte(Math.Round(length / Math.Cos(o - (Math.PI * 3 /
2))))
            End If
            If x >= 65 And x <= 72 Then
                distances(x) = CByte(Math.Round(length / Math.Cos((Math.PI * 2) - o)))
            End If
            p = p + 5
        Next
        serialO()
    End Function

    Private Function performCalcCircle(ByVal length As Integer)
        'Simple straight-through input to output. No calculations necessary.
        Dim x As Integer
        For x = 1 To 72
            distances(x) = CByte(length)
        Next
        serialO()
    End Function
```

```
    Private Function performCalcLatcha(ByVal length As Double, ByVal width As Double)
        'Same functionality as the square with different equations.
        Dim x As Integer
        Dim p As Double
        Dim o As Double
        p = 0
        For x = 1 To 72
            o = p * Math.PI / 180
            If x >= 1 And x <= 10 Then
                distances(x) = CByte(Math.Round(width / Math.Cos((Math.PI / 4) - o)))
            End If
            If x >= 11 And x <= 19 Then
                distances(x) = CByte(Math.Round(width / Math.Cos(o - (Math.PI / 4))))
            End If
            If x >= 20 And x <= 28 Then
                distances(x) = CByte(Math.Round(length / Math.Cos(o - (Math.PI / 2))))
            End If
            If x >= 29 And x <= 37 Then
                distances(x) = CByte(Math.Round(length / Math.Cos((Math.PI) - o)))
            End If
            If x >= 38 And x <= 46 Then
                distances(x) = CByte(Math.Round(width / Math.Cos((Math.PI * 5 / 4) -
o)))
            End If
            If x >= 47 And x <= 55 Then
                distances(x) = CByte(Math.Round(width / Math.Cos(o - (Math.PI * 5 /
4))))
            End If
            If x >= 56 And x <= 64 Then
                distances(x) = CByte(Math.Round(length / Math.Cos(o - (Math.PI * 3 /
2))))
            End If
            If x >= 65 And x <= 72 Then
                distances(x) = CByte(Math.Round(length / Math.Cos((Math.PI * 2) - o)))
            End If
            p = p + 5
        Next
        serialO()
    End Function

    Private Function serialO()
        'Create a new Rs232 object for using com port 1 and output the distances
array.
        Dim scom As New Rs232
        Dim y As Integer
        rtbFeedback.AppendText("Opening com port..." + ControlChars.NewLine)
        Try
            'Open the com port at baud rate 38400 bps, no parity, 8 data bits, and 1
stop bit. No buffer.
            scom.Open(1, 38400, 8, Rs232.DataParity.Parity_None,
Rs232.DataStopBit.StopBit_1, 0)
            rtbFeedback.AppendText("Downloading data." + ControlChars.NewLine)
            'If the debug mode is selected, display the values being written inside
the rich text box.
            If cbDebug.Checked = True Then
                For y = 1 To 72
                    'Write the ascii binary number to the port that represents the
decimal number.
                    scom.Write(Chr(distances(y)))
                    rtbFeedback.AppendText(distances(y).ToString + " ")
                Next
                rtbFeedback.AppendText(ControlChars.NewLine)
            ElseIf cbDebug.Checked = False Then
                For y = 1 To 72
```

```
                    scom.Write(Chr(distances(y)))
                Next
            End If
            rtbFeedback.AppendText("Success! Closing port..." + ControlChars.NewLine)
            scom.Close()
            rtbFeedback.AppendText("Done." + ControlChars.NewLine)
        Catch
            'If there is an error in transmission, alert the user.
            rtbFeedback.AppendText("Error in transmission. Retrying..." +
ControlChars.NewLine)
        End Try
    End Function

    Private Sub cbPattern_SelectedIndexChanged(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles cbPattern.SelectedIndexChanged
        'Change the picture based on the selection of the user.
        If cbPattern.SelectedIndex = 0 Then
            picLawn.Visible = True
            picLawn.Image = Image.FromFile("circle2square2.png")
            rtbFeedback.AppendText("Square selected." + ControlChars.NewLine)
        ElseIf cbPattern.SelectedIndex = 1 Then
            picLawn.Visible = True
            picLawn.Image = Image.FromFile("circle.png")
            rtbFeedback.AppendText("Circle selected." + ControlChars.NewLine)
        ElseIf cbPattern.SelectedIndex = 2 Then
            picLawn.Visible = True
            picLawn.Image = Image.FromFile("latcha.png")
            rtbFeedback.AppendText("Latcha Design selected." + ControlChars.NewLine)
        End If

    End Sub

    Private Sub btnReset_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnReset.Click
        'Resets all the inputs and the data values in the array.
        txtLength.Clear()
        lblWMin.Text = ""
        lblWMax.Text = ""
        lblLMin.Text = ""
        lblLMax.Text = ""
        picLawn.Visible = False
        cbPattern.Text = ""
        Dim x As Integer
        For x = 0 To 72
            distances(x) = 0
        Next
        rtbFeedback.AppendText("Form reset successfully." + ControlChars.NewLine)
    End Sub

    Private Sub btnClear_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnClear.Click
        'Clears the feedback textbox.
        rtbFeedback.Clear()
    End Sub

    Private Sub cbDebug_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cbDebug.CheckedChanged
        'Alerts the user to the debug mode on/off.
        If cbDebug.Checked = True Then
            rtbFeedback.AppendText("Debug mode on." + ControlChars.NewLine)
        Else
            rtbFeedback.AppendText("Debug mode off." + ControlChars.NewLine)
        End If
```

```
    End Sub
End Class
```

Appendix D

Gray cells are directly measured, white cells are calculated.

Test Data for Graphs 1-3

psi	kPa	related flowrate (m/s)	related flowrate (gpm)	feet	meter	Velocity	x velocity	y velocity
2.50	17.23	9.1812E-05	1.46	16.00	4.8800	5.1580	4.4762	2.5629
5.00	34.45	1.0451E-04	1.66	19.00	5.7950	5.8713	5.0952	2.9173
7.50	51.68	1.1721E-04	1.86	23.50	7.1675	6.5846	5.7142	3.2717
8.00	55.12	1.1974E-04	1.90	22.00	6.7100	6.7272	5.8380	3.3426
10.00	68.90	1.2990E-04	2.06	23.00	7.0150	7.2979	6.3332	3.6262
12.00	82.68	1.4006E-04	2.22	27.00	8.2350	7.8685	6.8285	3.9097
15.00	103.35	1.5530E-04	2.46	31.00	9.4550	8.7245	7.5713	4.3350
17.50	120.58	1.6799E-04	2.66	32.00	9.7600	9.4378	8.1903	4.6895
20.00	137.80	1.8069E-04	2.86	34.00	10.3700	10.1511	8.8093	5.0439
22.50	155.03	1.9339E-04	3.07	35.00	10.6750	10.8644	9.4283	5.3983
25.00	172.25	2.0608E-04	3.27	37.00	11.2850	11.5777	10.0474	5.7527
26.00	179.14	2.1116E-04	3.35	38.00	11.5900	11.8630	10.2950	5.8945

Test Data for Graph 4

hz	x feet	meters	gpm	m3/s	V0
55	3.5	1.0668	0.9056	5.7134E-05	2.960327
91	7.5	2.286	1.4852	9.3701E-05	4.854988
92	6.5	1.9812	1.5013	9.4717E-05	4.907617
92	7	2.1336	1.5013	9.4717E-05	4.907617
92	7	2.1336	1.5013	9.4717E-05	4.907617
109	9.5	2.8956	1.775	1.1198E-04	5.802319
110	10	3.048	1.7911	1.1300E-04	5.854948
111	10	3.048	1.8072	1.1402E-04	5.907578
112	11	3.3528	1.8233	1.1503E-04	5.960207
134	15	4.572	2.1775	1.3738E-04	7.118056
135	13	3.9624	2.1936	1.3839E-04	7.170685
136	15.5	4.7244	2.2097	1.3941E-04	7.223315
137	15	4.572	2.2258	1.4043E-04	7.275944
137	15	4.572	2.2258	1.4043E-04	7.275944
159	19	5.7912	2.58	1.6277E-04	8.433793
160	20	6.096	2.5961	1.6379E-04	8.486422
160	20.5	6.2484	2.5961	1.6379E-04	8.486422
160	20.5	6.2484	2.5961	1.6379E-04	8.486422
161	21.5	6.5532	2.6122	1.6480E-04	8.539052
162	20	6.096	2.6283	1.6582E-04	8.591681
173	22.5	6.858	2.8054	1.7699E-04	9.170605

175	20.5	6.2484	2.8376	1.7902E-04	9.275864
175	21.5	6.5532	2.8376	1.7902E-04	9.275864
176	23	7.0104	2.8537	1.8004E-04	9.328494
177	22	6.7056	2.8698	1.8106E-04	9.381123
177	22.5	6.858	2.8698	1.8106E-04	9.381123
179	20.5	6.2484	2.902	1.8309E-04	9.486382
183	25	7.62	2.9664	1.8715E-04	9.6969
188	27	8.2296	3.0469	1.9223E-04	9.960048
192	23	7.0104	3.1113	1.9629E-04	10.17057
193	23	7.0104	3.1274	1.9731E-04	10.2232
193	25.5	7.7724	3.1274	1.9731E-04	10.2232
193	27	8.2296	3.1274	1.9731E-04	10.2232
195	22.5	6.858	3.1596	1.9934E-04	10.32845
195	25	7.62	3.1596	1.9934E-04	10.32845
195	26.5	8.0772	3.1596	1.9934E-04	10.32845
196	25	7.62	3.1757	2.0035E-04	10.38108
196	28	8.5344	3.1757	2.0035E-04	10.38108
198	25	7.62	3.2079	2.0239E-04	10.48634
199	26.5	8.0772	3.224	2.0340E-04	10.53897
202	25	7.62	3.2723	2.0645E-04	10.69686
204	26.5	8.0772	3.3045	2.0848E-04	10.80212
205	27.5	8.382	3.3206	2.0950E-04	10.85475
205	27.5	8.382	3.3206	2.0950E-04	10.85475
206	26	7.9248	3.3367	2.1051E-04	10.90738
213	31	9.4488	3.4494	2.1762E-04	11.27578
214	27.5	8.382	3.4655	2.1864E-04	11.32841
214	28.5	8.6868	3.4655	2.1864E-04	11.32841
216	28	8.5344	3.4977	2.2067E-04	11.43367
218	26	7.9248	3.5299	2.2270E-04	11.53893
224	28	8.5344	3.6265	2.2880E-04	11.85471
225	29	8.8392	3.6426	2.2981E-04	11.90734
228	26.5	8.0772	3.6909	2.3286E-04	12.06523
228	28	8.5344	3.6909	2.3286E-04	12.06523
228	31	9.4488	3.6909	2.3286E-04	12.06523
229	30	9.144	3.707	2.3387E-04	12.11786
229	31.5	9.6012	3.707	2.3387E-04	12.11786
231	24	7.3152	3.7392	2.3591E-04	12.22312
231	28	8.5344	3.7392	2.3591E-04	12.22312
233	31	9.4488	3.7714	2.3794E-04	12.32837
233	31.5	9.6012	3.7714	2.3794E-04	12.32837
233	32	9.7536	3.7714	2.3794E-04	12.32837
233	32.5	9.906	3.7714	2.3794E-04	12.32837
234	25	7.62	3.7875	2.3895E-04	12.381
235	28	8.5344	3.8036	2.3997E-04	12.43363
235	29	8.8392	3.8036	2.3997E-04	12.43363
235	31.5	9.6012	3.8036	2.3997E-04	12.43363

237	30.5	9.2964	3.8358	2.4200E-04	12.53889
240	29.5	8.9916	3.8841	2.4505E-04	12.69678
240	29.5	8.9916	3.8841	2.4505E-04	12.69678
241	29.5	8.9916	3.9002	2.4606E-04	12.74941
243	29.5	8.9916	3.9324	2.4810E-04	12.85467

Calculated Data for Graph 6

t	x theory (1.0668)	y theory	x theory (3.048)	y theory	x theory (5.7912)	y theory	x theory (7.0104)	y theory	x theory (8.5344)	y theory	x theory (9.906)	y theory
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0500	0.1350	0.0617	0.2557	0.1354	0.3635	0.1986	0.4020	0.2210	0.4472	0.2473	0.5271	0.2959
0.1000	0.2845	0.0990	0.5110	0.2463	0.7236	0.3726	0.8001	0.4174	0.8898	0.4700	1.0409	0.5674
0.1500	0.4502	0.1117	0.7661	0.3327	1.0804	0.5222	1.1944	0.5893	1.3278	0.6682	1.5419	0.8143
0.2000	0.6338	0.0998	1.0208	0.3946	1.4339	0.6472	1.5848	0.7367	1.7613	0.8419	2.0302	1.0366
0.2500	0.8372	0.0635	1.2753	0.4319	1.7841	0.7477	1.9715	0.8595	2.1904	0.9911	2.5062	1.2345
0.3000	1.0626	0.0026	1.5294	0.4447	2.1310	0.8236	2.3545	0.9578	2.6150	1.1157	2.9703	1.4078
0.3500	1.3123	-0.0828	1.7833	0.4330	2.4748	0.8751	2.7338	1.0316	3.0353	1.2158	3.4227	1.5566
0.4000			2.0368	0.3967	2.8153	0.9020	3.1094	1.0809	3.4512	1.2914	3.8638	1.6809
0.4500			2.2901	0.3359	3.1527	0.9043	3.4814	1.1057	3.8629	1.3425	4.2937	1.7806
0.5000			2.5430	0.2506	3.4869	0.8822	3.8498	1.1059	4.2703	1.3690	4.7129	1.8559
0.5500			2.7957	0.1408	3.8181	0.8355	4.2147	1.0816	4.6735	1.3710	5.1215	1.9065
0.6000			3.0481	0.0065	4.1462	0.7643	4.5760	1.0328	5.0725	1.3485	5.5198	1.9327
0.6500			3.3001	-0.1524	4.4712	0.6686	4.9339	0.9594	5.4675	1.3015	5.9081	1.9344
0.7000					4.7932	0.5484	5.2883	0.8615	5.8583	1.2299	6.2866	1.9115
0.7500					5.1123	0.4036	5.6393	0.7391	6.2452	1.1339	6.6557	1.8641
0.8000					5.4283	0.2343	5.9869	0.5922	6.6280	1.0132	7.0154	1.7922
0.8500					5.7415	0.0405	6.3312	0.4208	7.0070	0.8681	7.3661	1.6957
0.9000					6.0517	-0.1778	6.6721	0.2248	7.3820	0.6984	7.7080	1.5747
0.9500							7.0097	0.0043	7.7531	0.5043	8.0413	1.4292
1.0000							7.3441	-0.2407	8.1205	0.2856	8.3662	1.2592
1.0500									8.4840	0.0423	8.6830	1.0646
1.1000									8.8438	-0.2254	8.9917	0.8456
1.1500											9.2927	0.6020
1.2000											9.5862	0.3338
1.2500											9.8723	0.0412
1.3000											10.1511	-0.2760
1.3500												
1.4000												
1.4500												
1.5000												
1.5500												
1.6000												
1.6500												
1.7000												
1.7500												
1.8000												
1.8500												
1.9000												
1.9500												
2.0000												

Appendix E

The purpose of this appendix is to provide equations, data, and specifications pertinent to the function of the flow meter in the WaterBoy system.

Continuity equation

The steady flow velocity of water through the system can be modeled by applying the conservation of mass (continuity) equation between points 1 and 2 of the control volume shown above. Here it is expressed as a net rate into the system:

$$0 = \frac{\partial}{\partial t}\int_{C\forall} \rho d\forall + \int_{CS} \rho\left(\vec{V}\cdot\hat{n}\right)dA \Rightarrow \text{where } \frac{\partial}{\partial t}\int_{C\forall} \rho d\forall = 0 \text{ ; for steady flow}$$

$$0 = -\int_{CS} \rho\left(\vec{V}\cdot\hat{n}\right)dA = -\int_{CSi} \rho\left(\vec{V}\cdot\hat{n}\right)dA - \int_{CSo} \rho\left(\vec{V}\cdot\hat{n}\right)dA$$

$$0 = \sum(\rho AV)_{in} - \sum(\rho AV)_{out} = Net\ rate\ in \Rightarrow \sum(\rho AV)_{in} = \sum(\rho AV)_{out}$$

The velocity at any location of the system can then be determined if the cross sectional area at that location and the mass flow rate are known. This actual velocity is given by,

$$V_2 = \dot{m} \Big/ \left(\rho_{H2O} A_2\right) = \left(\frac{A_1}{A_2}\right) V_1$$

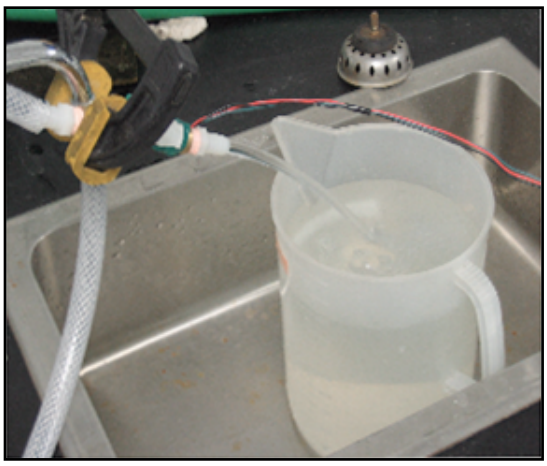
$$Where\ A_1 = Area\ of\ inlet\ location\ downstream\ of\ meter$$

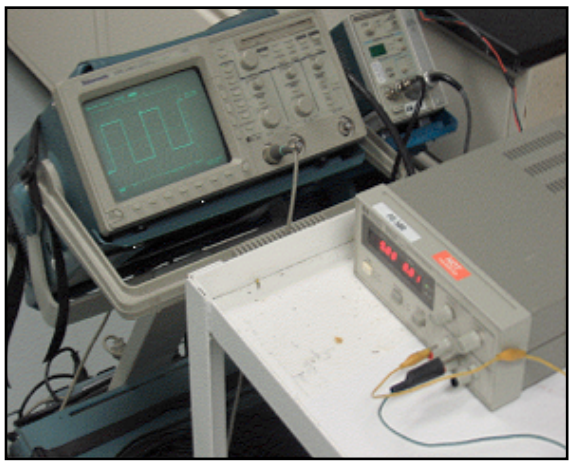
$$A_2 = Area\ of\ exit\ location$$

$$V_1 = Velocity\ @\ inlet\ calculated\ by\ continuity$$

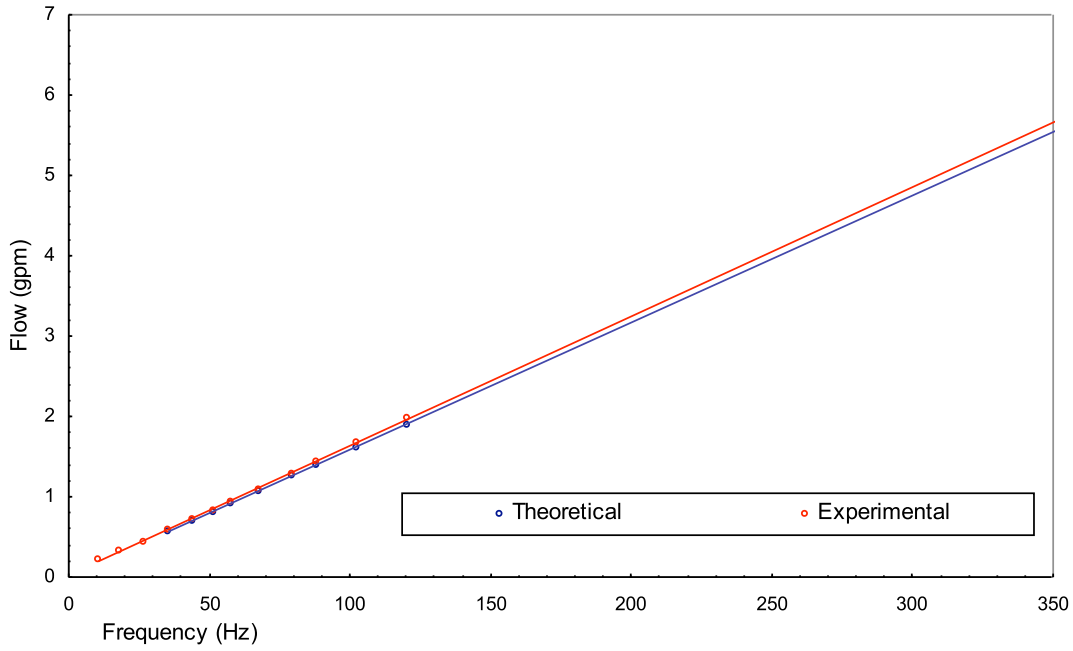
Test Setup and Results

Experiments were conducted to verify the linear relationship between flow and output frequency. Leads were connected to the terminals as prescribed by the specification sheet (please see appendix), and the meter was wrapped in water proof tape. Both flow ports were then mated with ½ inch barbed tube connectors and sealed using Teflon tape. The inlet barbed tube connector was then affixed into a length of braided ½" PVC hose. The inlet to this hose was plugged into the faucet and a steady flow was established.





The figures above show the experimental setup. The flow meter was secured by clamping it to the faucet. Positioned nearby were the oscilloscope, and the 15V power source. After adjusting the source to approximately 9V, a flow of water was then established through the meter, generating the square wave shown in the figure to the right. Flow was then timed over a one minute interval and collected in a large bucket. After the given time interval had elapsed, the bucket and its contents were weighed. Using the density of water, the volume was solved, and then divided by the time interval to obtain the flow. Measurements were taken over a range of flows, incremented by monitoring the change in frequency of the square wave on the oscilloscope. Starting with about 10Hz, twelve points were recorded, ending at about 120Hz, which was the maximum flow obtainable from this faucet.



Theoretical data was obtained using minimum and maximum flow data specified by the specification sheet. Assuming a linear relationship between flow and output signal, these minimum and maximum flow points were fitted with a trend line, the slope of which was then used to calculate the flow for each frequency generated experimentally. Theoretical data was then compared to experimental in the plot above.

As expected, the experimental data lines up very nicely when compared to the theoretical trend line. Thus we can assume with confidence that our flow meter will provide accurate data which we can use to predict spraying distance, so long as no loss of flow is present upstream of the meter before leaving the nozzle. Data is tabulated below.

Test 1	Theoretical		Experimental					
Point	Flow	Frequency	Collected Water weight + Bucket weight	Bucket Weight	Water Weight	Volume	Volume	Flow
#	gpm	Hz	g	g	kg	m^3	Gal	GPM
1	0.17	10.75	1136.8	303.9	0.8329	0.00083	0.2203	0.22
2	0.28	17.70	1522.4	307.7	1.2147	0.00122	0.3212	0.32
3	0.42	26.50	1946.8	306.5	1.6403	0.00164	0.4338	0.43
4	0.56	35.50	2557.0	305.3	2.2517	0.00225	0.5955	0.6
5	0.69	44.00	2982.9	307.7	2.6752	0.00268	0.7075	0.71
6	0.81	51.20	3448.8	306.4	3.1424	0.00315	0.8311	0.83
7	0.91	57.70	3869.4	307.6	3.5618	0.00357	0.9420	0.94
8	1.06	67.40	4445.6	307.1	4.1385	0.00414	1.0945	1.09
9	1.25	79.50	2747.4	308.2	2.4392	0.00244	0.6451	1.29
10	1.40	88.40	3020.0	307.7	2.7123	0.00272	0.7173	1.43
11	1.62	102.50	3462.3	305.8	3.1565	0.00316	0.8348	1.67
12	1.90	120.50	4061.8	307.0	3.7548	0.00376	0.9930	1.99

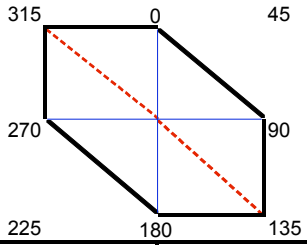
Specification Sheet

Appendix F

Enter radius (r) = **30** feet

then x = 21 feet

then y = 15 feet



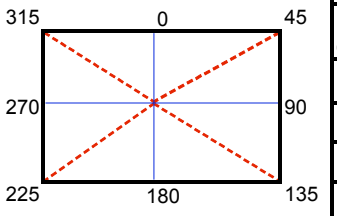
Both low and hi readings reflect 0 or 360 degrees, the change occurs virtually instantaneous		
enter position sensor lowest reading	0.14	V
enter position sensor highest reading	4.9	V
volts/division (72 divisions)	0.0657778	V
volts/degree (360 degrees)	0.0131556	V

0 to 45 degrees -	$d_n = y * (\cos (45 - \_))^{(-1)}$
45 to 90 degrees -	$d_n = y * (\cos (\_ - 45))^{(-1)}$
90 to 135 degrees -	$d_n = x * (\cos (\_ - 90))^{(-1)}$
135 to 180 degrees -	$d_n = x * (\cos (180 - \_))^{(-1)}$
180 to 225 degrees -	$d_n = y * (\cos (225 - \Theta))^{(-1)}$
225 to 270 degrees -	$d_n = y * (\cos (\_ - 225))^{(-1)}$
270 to 315 degrees -	$d_n = x * (\cos (\_ - 270))^{(-1)}$
315 to 360 degrees -	$d_n = x * (\cos (360 - \_))^{(-1)}$

where Θ is the angle corresponding to a circle of radius ®
x and y are the adjacent fixed line and d_n is the changing hypotenuse

dn	positions where dn is the same for parallel portion of design				dn	positions where dn is the same for the corner portions of design			
21.213	1	19	37	55	21.213	19	37	55	1
19.581	2	18	38	54	21.291	20	36	56	72
18.312	3	17	39	53	21.537	21	35	57	71
17.321	4	16	40	52	21.958	22	34	58	70
16.551	5	15	41	51	22.571	23	33	59	69
15.963	6	14	42	50	23.403	24	32	60	68
15.529	7	13	43	49	24.491	25	31	61	67
15.231	8	12	44	48	25.893	26	30	62	66
15.057	9	11	45	47	27.688	27	29	63	65
15.000	10		46		29.995	28		64	

Appendix G



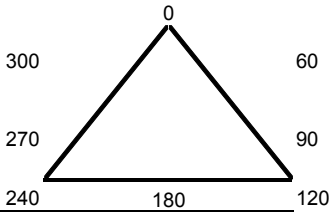
Assume radius(r) = 30 ft.	Assume x = 21.2 ft.	
Both low and hi readings reflect 0 or 360 degrees, the change occurs virtually instantaneous		
enter position sensor lowest reading	0.124	V
enter position sensor highest reading	4.85	V
volts per division (72 divisions)	0.0656389	V
volts per degree (360 degrees)	0.0131278	V

0 to 45 degrees -	$d_n = x * (\cos \Theta)^{-1}$
45 to 90 degrees -	$d_n = x * (\cos (\_ - 45))^{(-1)}$
90 to 135 degrees -	$d_n = x * (\cos (\_ - 90))^{(-1)}$
135 to 180 degrees -	$d_n = x * (\cos (180 - \_))^{(-1)}$
180 to 225 degrees -	$d_n = x * (\cos (\Theta - 180))^{(-1)}$
225 to 270 degrees -	$d_n = x * (\cos (270 - \_))^{(-1)}$
270 to 315 degrees -	$d_n = x * (\cos (\_ - 270))^{(-1)}$
315 to 360 degrees -	$d_n = x * (\cos (360 - \_))^{(-1)}$

where Θ is the angle corresponding to the radius of a circle
x and y are the adjacent fixed line and d_n is the changing hypotenuse

dn	positions where dn is the same									
21.210	1	19			37		55		1	
21.291	2	18		20	36	38	54	56	72	
21.537	3	17		21	35	39	53	57	71	
21.958	4	16		22	34	40	52	58	70	
22.571	5	15		23	33	41	51	59	69	
23.403	6	14		24	32	42	50	60	68	
24.491	7	13		25	31	43	49	61	67	
25.893	8	12		26	30	44	48	62	66	
27.688	9	11		27	29	45	47	63	65	
29.995	10			28		46		64		

Appendix H



degrees between (d_n & d_{n+1})	divisions per 60 degrees	circle radius(r) = 30 ft.
5	12	x = 15 ft

where r is the distance from the center of the circle to the outer most point of the triangle in 3 places - 30 feet assumed here

where x is the distance from the center of the circle to the shortest distance on the triangle in 3 places where the angle is perpendicular to the outer line of the triangle - 15 feet for this example

Both low and hi readings reflect 0 or 360 degrees, the change occurs virtually instantaneous

enter position sensor lowest reading	0.124	V
enter position sensor highest reading	4.85	V
volts per division (72 divisions)	0.065638889	V
volts per degree (360 degrees)	0.013127778	V

for 0 to 60 degrees -	$d_n = x * (\cos (60 - \Theta))^{-1}$
for 60 to 120 degrees -	$d_n = x * (\cos (\_ - 60))^{(-1)}$
for 180 to 240 degrees -	$d_n = x * (\cos (180 - \_))^{(-1)}$
for 180 to 240 degrees -	$d_n = x * (\cos (\Theta - 180))^{(-1)}$
for 240 to 300 degrees -	$d_n = x * (\cos (300 - \_))^{(-1)}$
for 300 to 360 degrees -	$d_n = x * (\cos (\_ - 300))^{(-1)}$

where Θ is the angle corresponding to a circle of radius ®
x is the adjacent fixed line and d_n is the changing hypotenuse

dn	positions where dn is the same					
30.000	1	25		49		73
26.152	2	24	26	48	50	72
23.336	3	23	27	47	51	71
21.213	4	22	28	46	52	70
19.581	5	21	29	45	53	69
18.312	6	20	30	44	54	68
17.321	7	19	31	43	55	67
16.551	8	18	32	42	56	66
15.963	9	17	33	41	57	65
15.529	10	16	34	40	58	64
15.231	11	15	35	39	59	63
15.057	12	14	36	38	60	62
15.000	13		37		61	

Appendix I

Extreme Connectivity

1319x Developer's Starter Kit

Overview

Freescale Semiconductor's 1319x Developer's Starter Kit is a cost-effective, reusable development kit used to implement wireless network designs compatible with the IEEE® 802.15.4 standard. The kit contains all the hardware, software and documentation necessary to create proprietary and standards-based peer-to-peer and star networks.

The development kit contains two Sensor Application Reference Design (SARD) boards that utilize Freescale's 1319x 2.4 GHz transceiver, MC9S08GT60 microcontroller unit, and MMA6261Q 1.5g X-Y-axis and MMA1260D 1.5g Z-axis accelerometers. The boards can either be powered by a 9-volt battery or by an external power supply (both included). Also included in the development kit are Freescale's IEEE 802.15.4 media access controller (MAC) and example Simple MAC (SMAC) software. The SMAC software, which is available as source code and can be targeted to any processor, allows for development of non-standards-based proprietary point-to-point and star networks. The IEEE 802.15.4 MAC software allows for standards-based peer-to-peer and star network topologies. The boards come programmed with a wireless accelerometer demonstration based on the SMAC software and can be reprogrammed through the serial port, or by using a USB MULTILINK debugger/programmer.

Freescale's ZigBee™-compliant family of 2.4 GHz transceivers offers development of cost-effective, low-power and low-data-rate RF connectivity solutions. The 1319x allows for development of simple proprietary point-to-point and networks. The MC13192 supports proprietary development of standards-based IEEE 802.15.4 network topologies. The MC13193 is our ZigBee RF transceiver and allows for the development of sophisticated ZigBee mesh networks.

Freescale's 1319x family of transceivers offers affordable wireless connectivity for battery-powered monitor and control applications, such as home and industrial automation, industrial control and personal health care.

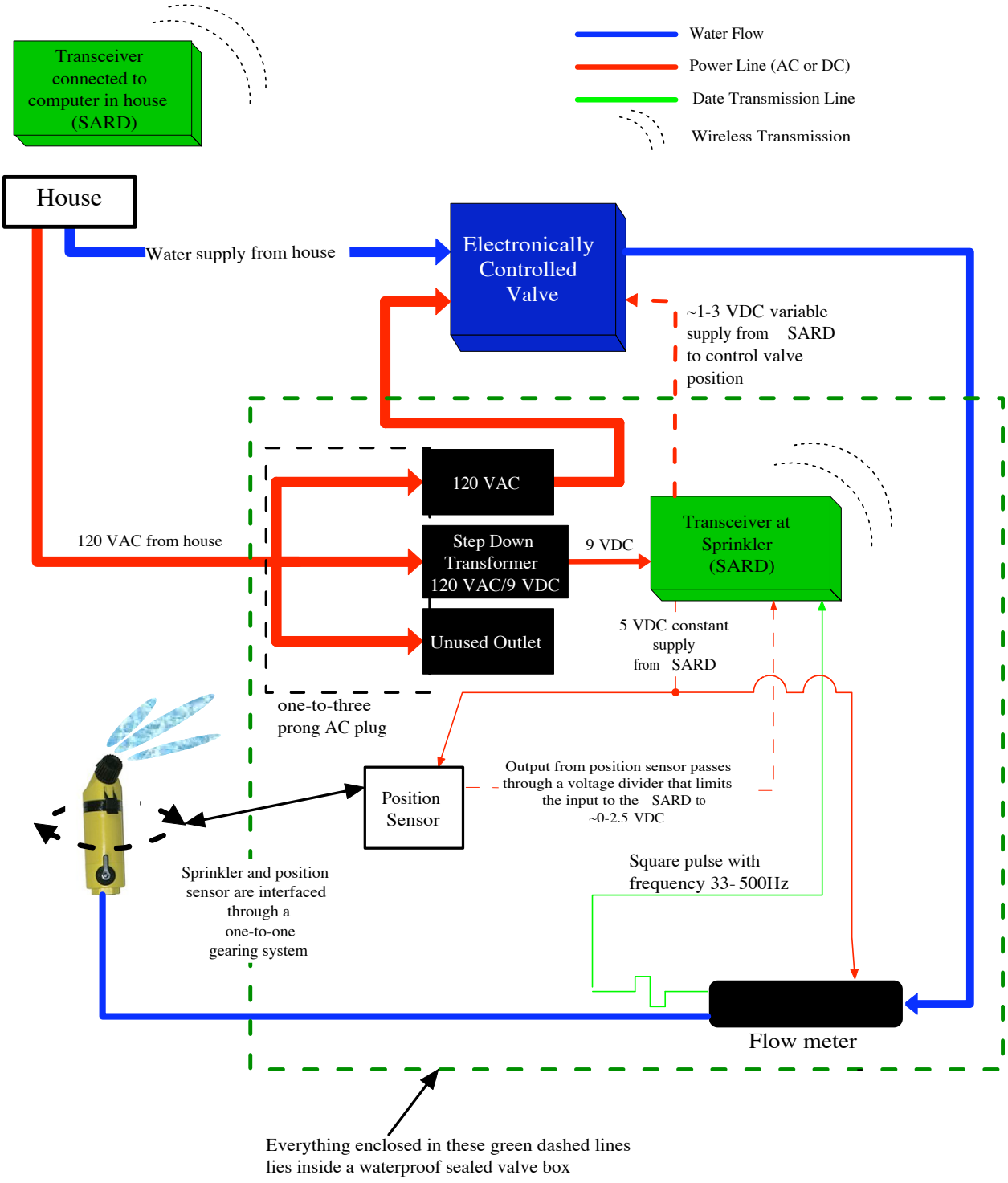
Freescale combines the development board, CodeWarrior™ Development Studio and preprogrammed sample applications to create a comprehensive, scalable solution for easy, fast and efficient wireless network development.

Developer's Starter Kit Features

> Two 2.4 GHz wireless nodes compatible with the IEEE 802.15.4 standard
  - MC13192 2.4 GHz RF data modem
  - MC9S08GT60 low-voltage, low-power 8-bit MCU for baseband operations
  - Integrated MMA6261Q 1.5g X-Y-axis and MMA1260D 1.5g Z-axis accelerometers
  - Printed transmit-and-receive antennae
  - Onboard expansion capabilities for external application-specific development activities
  - Onboard background debug module (BDM) port for MCU Flash reprogramming and in-circuit hardware debugging
  - RS-232 port for monitoring and Flash programming
  - LEDs and switches for demonstration, monitoring and control
  - Connections for 9-volt battery or external power supply
> Hardware supports Freescale's IEEE 802.15.4 MAC and example SMAC software
> Preprogrammed accelerometer demonstration and additional downloadable sample applications
> Metrowerks' CodeWarrior™ Development Studio for HCS08 MCUs
> USB MULTILINK BDM Debugger/Programmer (included with the 13192DSK-BDM only)

freescale semiconductor

Appendix J


Transceiver connected to computer in house (SARD)
Water Flow
Power Line (AC or DC)
Date Transmission Line
Wireless Transmission
House
Water supply from house
Electronically Controlled Valve
~1-3 VDC variable supply from SARD to control valve position
120 VAC
120 VAC from house
Step Down Transformer 120 VAC/9 VDC
9 VDC
Transceiver at Sprinkler (SARD)
Unused Outlet
5 VDC constant supply from SARD
one-to-three prong AC plug
Position Sensor
Output from position sensor passes through a voltage divider that limits the input to the SARD to ~0-2.5 VDC
Square pulse with frequency 33- 500Hz
Sprinkler and position sensor are interfaced through a one-to-one gearing system
Flow meter
Everything enclosed in these green dashed lines lies inside a waterproof sealed valve box

Bibliography

[1] MSDN Library. http://www.microsoft.com/downloads/details.aspx?FamilyID=075318ca-e4f1-4846-912c-b4ed37a1578b&DisplayLang=en. 2005.

[2] Gems Sensors. http://www.gemssensors.com/. 2005.

[3] Ron Woodward. 555 Timer RC Servo Control diagram. Wolfstone Technical Base. http://wolfstone.halloweenhost.com/TechBase/svoint_RCServos.html#Avail

[4] US Census Bureau. Statistics of US Businesses, 2002. http://www.census.gov/csd/susb/usalli02.xls

[5] Pro Garden Biz http://www.progardenbiz.com/currentissue/Feature1.html

[6] According to the Michigan Nursery and Landscape Association. http://www.mnla.org/

[7] Michigan Green Industry Association. http://www.landscape.org/landsculptor.htm

[8] American Water Works Association – Michigan Section Water Legacy Act. http://www.mi-water.org/awwa/Whats_New_Archives/MIAWWA%20Water%20Policy%20Position%20Statement%20031405a.pdf#search='Water%20Legacy%20Act%2C%20mi'

[9] Accurain Website http://www.accurain.com/index.html

[10] Toro Website. http://www.toro.com/apps/distributor/distlocator.jsp

[11] Hydroturf International Website. http://www.hydroturfinternational.com/docs/to_product.html