Capacitive Rain Sensor for Automatic Wiper Control

Final Report
ECE 480: Design Team 6

Eric Alexander Otte

Arslan Qaiser

Ishaan Sandhu

Anuar Tazabekov

Danny (Dongho) Kang

Project Sponsor: *Hyundai-Kia America Technical Center, Inc. (HATCI)*

Sponsor Representatives: *Mr. Jeff Shtogrin & Mr. Daniel D. Vivian*

MSU Facilitator: *Dr. John R. Deller*

Executive Summary

While technological advances have worked to increase the safety and convenience of modern vehicles, the fact remains that drivers today have more distractions than ever before. The prevalence of cell phones, MP3 players, and in-dash navigation systems have lead to a multitude of potentially dangerous diversions literally at the driver's fingertips. One feature designed to ease the burden on vehicle operators is the automatic rain-sensing wiper system, which detects rain on the windshield and turns on the automobile's wipers accordingly.

Hyundai-Kia America Technical Center (HATCI) tasked ECE 480 Design Team 6 with developing a new rain-sensing system for wiper control based on capacitive sensor technology. Current systems utilize an optical sensor for rain detection, which have a small sensing area and are prone to false detection of rain causing inappropriate wiper operation. Design Team 6 has developed a solution to this problem in the form of an integrated capacitive rain sensor system. Capacitive-sensing relies on interactions with an emitted electric field to determine the presence of an object. Design Team 6's capacitive rain sensor utilizes this principle to accurately detect varying levels of rain on the windshield, while preventing false positives from objects such as dirt and human hands. The sensor unit mounts discretely to the interior of the windshield while providing increased detection area, improved accuracy, and a lower cost than the preexisting optical unit.

Acknowledgments

Design Team 6 would like to sincerely thank all of those who assisted in the development of the capacitive rain sensor. Special thanks to:

- **Dr. Shantanu Chakrabartty:** for always being willing and ready to help, and providing extremely helpful guidance during the early development phase of the project.
- **Mrs. Roxanne Peacock:** for assisting in the ordering of countless small part orders, and for always finding the better deal.
- **Mr. Jeff Shtogrin:** for his commitment to Design Team 6 as the HATCI representative, and for his dedication in driving to East Lansing to hold weekly meetings at 8:00 am.
- **Dr. John R. Deller:** for his guidance and effort in grading the plethora of ECE 480 assignments as the project's MSU facilitator.
- **Mr. Brian Wright & Mr. Gregg Mulder:** for their effort in fabricating prototype PCB designs and assisting in the soldering of surface-mount components.

Table of Contents

CHAPTER ONE: INTRODUCTION & BACKGROUND

1.1. Introduction

Over the past two decades, the automotive industry has aggressively researched ways to exploit modern computing and electronic advances in the development of safety, reliability, and entertainment technologies for vehicles. Previously remarkable and uncommon features such as auto-dimming mirrors and rear-view cameras have become standard in the modern era. Today consumers expect their automobiles to be able to connect to their MP3 players, provide GPS-assisted visual directions, and allow hands-free phone calls via Bluetooth technology. While these features have improved the driving experience for many, they also imply the increasingly common interaction between driver and electronic gadgetry during vehicle operation. These interactions can be a dangerous distraction for the driver, who must take his/her eyes off the road to attend to a device.

With drivers exposed to an ever increasing number of distractions, automatic rain-sensing wiper systems become an even more appealing feature, as they work to minimize the time the driver must take his/her hands off the wheel. These systems detect droplets of rain on the windshield and automatically turn on and adjust the wiper system in accordance to the level of precipitation. Current rain-sensing systems use an optical sensor to detect the presence of water on the windshield, and relay wiper control data to the vehicle's body control module (BCM). Unfortunately, these optical rain sensors suffer from a small sensing area, are prone to false-positives, and are too expensive to be included as standard equipment in most vehicles.

At the beginning of Michigan State University's spring semester in 2010, HATCI tasked ECE 480 Design Team 6 with designing a new automatic rain-sensing system utilizing recent advances in capacitive sensor technology. Design Team 6 has delivered on their proposal with a compact, highly accurate, and cost effective capacitive rain sensor system. This sensor has been designed to be able to easily replace optical units, as it mounts in the same location of the vehicle, on the interior of the windshield, and relays the same control signals to the BCM of the automobile. Sigma-Delta capacitance-to-digital converter circuits from Analog Devices convert minute changes in capacitance from the sensor traces into a 24-bit digital output signal, which is then processed by an on-board microprocessor to

determine appropriate wiper action. The sensor improves upon the preexisting optical unit in detection area, reliability, package size, and most importantly, cost. Design Team 6 estimates that a production-level run of capacitive rain sensors will cost under $12 per unit, significantly less than the optical sensor at $18 per unit.

1.2. Background

Many attempts have been made at constructing an effective, reliable, and cheap rain detection and wiper control system for vehicles. A perfect system could subtract one more task from the driver's workload, and allow them to better keep their eyes on the road and hands on the wheel during foul weather. Despite this, automatic rain-sensing wiper systems are relatively uncommon in modern vehicles for a number of reasons. They are often too expensive, too unsightly, or too unreliable to be desired in new automobiles. While a number of different design approaches have been made to improve upon these issues, none have been successful enough for the technology to become widely adapted in new vehicles.

By far the most common rain detection method, and the one currently employed by Hyundai vehicles, is the use of an optical sensor. These optical sensors function by transmitting an infrared beam at an angle through the windshield and measuring the reflection to determine the presence of water. This is a relatively difficult task, requiring complex circuitry and precision manufacturing. Optical sensors are thus somewhat expensive and can produce false readings when dirt or other particles on the windshield cause a reflection mimicking that of rain. Because it relies on an infrared beam for detection, the optical sensor also suffers from a very small sensing area on the windshield, limiting its effectiveness in rapidly responding to light rain. In addition, the sensor housing is physically bulky, reducing its appeal in luxury vehicles.

These issues can largely be mitigated by using a capacitive sensor rather than an optical one. Instead of sending an infrared beam through the windshield glass, a capacitive sensor works by emitting an electric field which can pass through the glass to interact with objects resting on it. Because water and other objects such as dirt or rocks interfere with the electric field in very different ways, the sensor will be less likely to be fooled if designed correctly. Unlike a standard capacitor, which confines the electric field lines between two conductors in a tight package, a capacitive sensor allows the field lines

to spread out, and is designed to maximize the fringing of the electric field lines away from the conductors. These electric field lines are known as “fringe fields”, and are vital to the operation of a capacitive sensor. Because they extend away from the conductors, which are typically just copper traces laid out flat on a printed circuit board (PCB), the fringe fields can be interacted with by other objects. When conductive or dielectric objects interfere with these fields, it changes the capacitance of the capacitive sensor, as seen in Figures 1 and 2. This change in capacitance can then be detected via circuitry and used to modulate an output signal. Capacitive sensors can detect the presence, position, and type of conductive or dielectric material interfering with their fringe fields. When multiple capacitive sensors are connected in an array, they can also be used to detect movement of a conductive or dielectric object. This effect is most commonly seen in capacitive touch pads, such as on popular products like the iPod Touch from Apple.

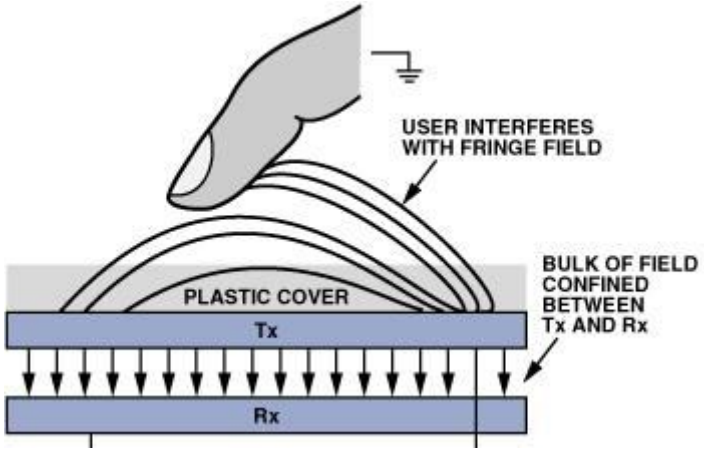


Figure 1: Finger interfering with fringe fields

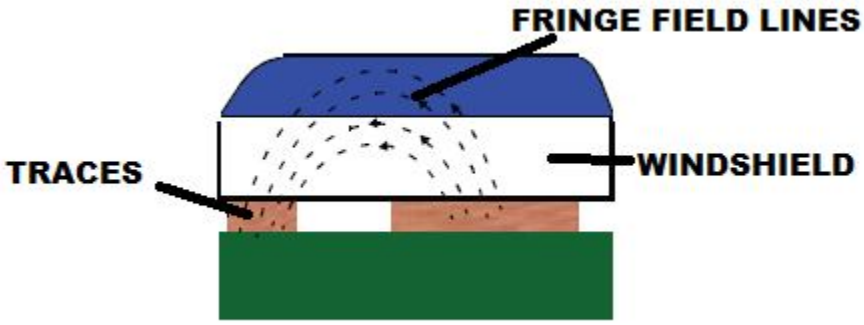


Figure 2: Fringe field lines extending from sensor traces through windshield

The electric field is created by applying an alternating current (AC) voltage to one of the conductors forming the sensor traces. A typical button sensor requires only two conductors, which never physically connect but are separated by a small distance and patterned into shapes. Depending on the application of the sensor, the sensor traces can take on a variety of different sizes and patterns. The layout of the traces is often designed to maximize the fringing fields over a given area. The traces, along with the materials surrounding them, also form the base capacitance of the system, typically along the order of 2 – 20 pico-Farads (pF) in magnitude. Base capacitance should be minimized when possible, as the change in capacitance resulting from fringe field interference is often less than 0.5 pF, and detection is easiest when the changing capacitance value is close to the base value.

The idea to use capacitive-sensing to detect rain on a windshield is not entirely new, as seen in United States Patent US6094981, among others. However, technical limitations have largely prevented such designs from being commercially viable. With advances in modern integrated circuits over the past decade, however, this problem can now be avoided under the proper design. HATCI had previously been contracted with Enterprise Electronics to design a capacitive sensor for this application, but development was halted. PREH, located out of Germany, have been able to create an accurate multifunction device which includes a capacitive rain sensor, but also includes other features such as temperature and humidity sensors. These extra features were deemed not necessary for Hyundai vehicles, and the overall cost of the system was far too expensive to be a practical alternative to optical designs.

Design Team 6 has developed a stand-alone capacitive rain sensor system that is both reliable and affordable. Unlike the design from PREH, this sensor is a compact unit solely dedicated to the task of detecting rainwater on the windshield and controlling the wipers accordingly. This allows the design to contain few parts, take up a small volume, and perform its job extremely well. It is significantly cheaper than the current optical sensor, with an estimated $11.40 per unit cost, down from $18 for the optical unit. It attaches to the interior of the windshield in the same location as the optical unit, but takes up less volume in the prototype unit and could be further refined if necessary in the production model for aesthetic purposes. Most importantly, the new sensor utilizes highly accurate 24-bit capacitance-to-digital converters and an on-board microcontroller to allow extreme accuracy and prevention of false-positives, improving the reliability of the device. These improvements in cost

and functionality will enable Hyundai to integrate the product into more vehicles in the future, further improving vehicle safety in the modern era.

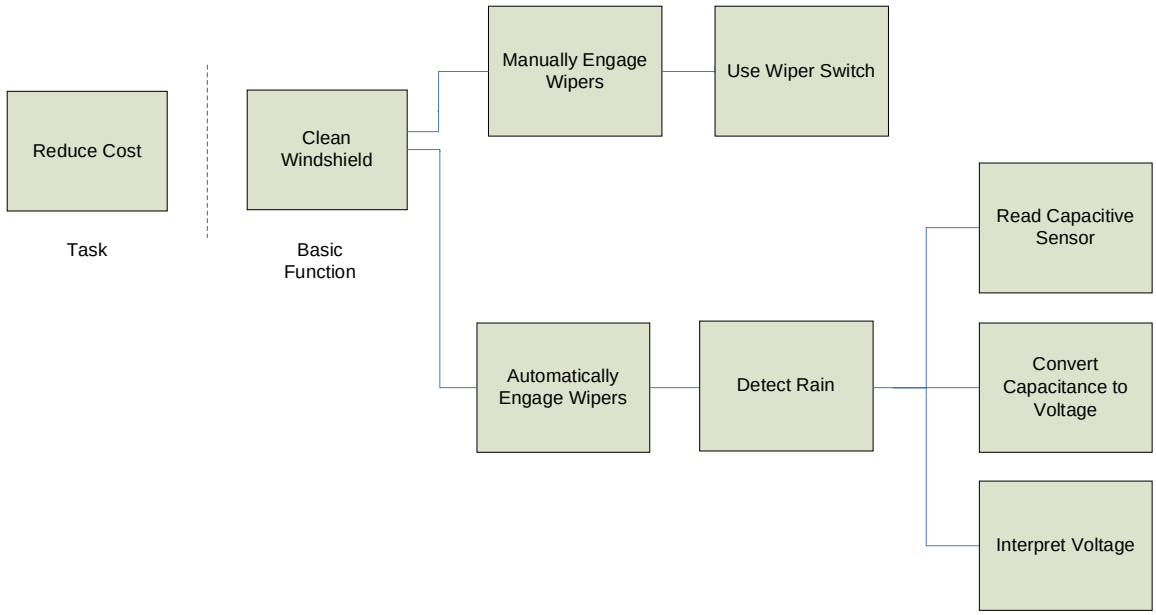
CHAPTER TWO: EXPLORING THE SOLUTION SPACE

2.1. Design Specifications

In designing the capacitive rain sensor, the following design specifications were provided by the sponsor:

- **Functionality**
    - Detect and report the presence of one drop of water placed on top of a 6mm thick glass windshield above the sensor trace area
- **Accuracy**
    - Must not falsely trigger the wipers when a hand is placed in proximity of the sensor trace area
    - Provide at least two different output signal levels depending on the amount of rain present on the windshield
    - Be shielded from the vehicle interior to avoid interference; only water on the windshield should activate the wipers, not objects or circuits inside the vehicle
    - Maintain all performance characteristics across the temperature range from 33 – 120 degrees Fahrenheit
- **Compatibility**
    - Device fits in existing Hyundai optical rain sensor housing area (1250 mm^2)
    - Device mounts to interior of windshield via adhesive
    - Device can operate on vehicle's 12 V power supply
- **Cost**
    - Estimated production cost less than $12 / unit

2.2. FAST Diagram



Design Team 6 utilized the FAST Diagram shown above to help decompose the capacitive rain sensor system into its primary tasks and components. The top branch describes the typical method of cleaning the windshield using a manual switch. The lower branch describes using an automatic rain-sensing system in which it is necessary to detect the rain. This is accomplished by monitoring the capacitive sensor, converting the change in capacitance to a corresponding change in voltage, and interpreting this changing voltage signal to determine wiper action.

2.3 Conceptual Design Descriptions

Unlike some of the more open-ended projects in ECE 480, HATCI provided Design Team 6 with a specific solution to the problem of detecting rain on the windshield through the use of capacitive-sensing. The most critical component in the design was a circuit to monitor the capacitance of the sensor traces and modulate an output signal correspondingly. The conceptual design descriptions thus represent a number of different variations on this critical component.

2.3.1. Design a Capacitive-Sensing Circuit

The first proposed design was to build a capacitive sensing circuit from basic components such as op-amps, comparators, and passive components. Due to experience in analog circuitry, the team realized that the capacitive sensor traces form a variable capacitor that changes as objects interfere with the

fringe fields. Many circuits exist that utilize the time-constant principle of an RC circuit to produce an output waveform. An astable RC-multivibrator circuit, as seen in Figure 3, produces a square wave output with a frequency varying with respect to the capacitance of the sensor traces when they are used as the capacitor. This varying square wave could then be interpreted by a microcontroller, and compared to known responses from rain to determine appropriate wiper action.

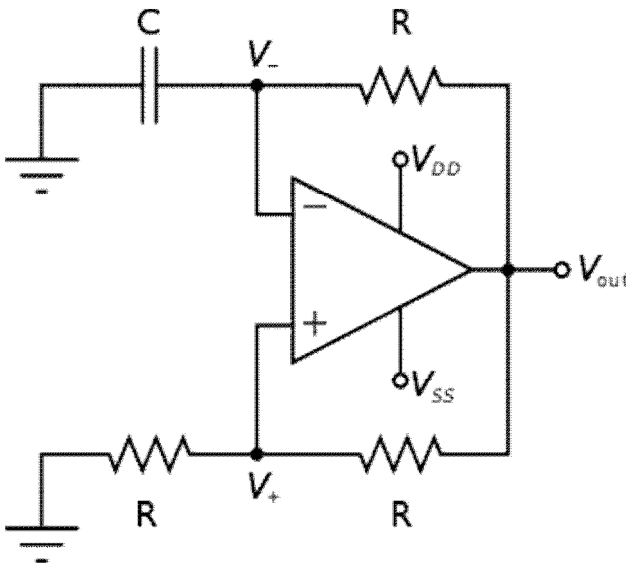


Figure 3: Astable multivibrator capacitive-sensing circuit (C1 = Csensor)

It was determined that this circuit would not provide the high level of accuracy needed to determine the presence of minute amounts of rain through a 6-8 mm thick glass windshield. Considering that the base capacitance (steady state) of the sensor traces was going to be around 5 – 15 pF, and that the changing capacitance from rain was expected to be between 0.1 – 0.5 pF, this would result in a change in a very small change in output frequency. This would be difficult to differentiate by a microcontroller and would also be highly prone to errors from noise. In addition, designing a capacitive-sensing circuit, when highly accurate dedicated circuits were available on the market, was a risk that would not only lower the accuracy of our product but take precious development time.

2.3.2. Microcontroller Capacitive-Sensing Modules

Many recently released microcontrollers include specific hardware modules for capacitive sensing. For example, Cypress Semiconductor has a popular “CapSENSE” module, and Microchip has the appropriately named “Capacitive Sensing Module”. These modules were investigated as potential

solutions to the capacitance sensing circuit. This method would simplify the system, as the microcontroller could perform two tasks – monitoring the capacitive sensor traces, and processing the change in capacitance to determine wiper action. Since a microcontroller would still be needed for processing if a separate circuit were used to monitor the sensor traces, it would be convenient to have the microcontroller perform both tasks. Unfortunately, these hardware modules are primarily designed for human touch applications, and it was determined that they would not possess the extreme accuracy needed for the product, and offered by other, stand-alone circuits such as the Analog Devices AD7745. Human touch applications are relatively easy work when compared to a rain sensing application, as the covering of the sensor traces is often only 1 – 2 mm thick instead of the 6 – 8 mm of glass covering a standard windshield. Furthermore, the change in capacitance to a sensor from a human finger is much larger than a change in capacitance from a few raindrops. Thus, while the capacitive-sensing modules would be a very convenient solution to any human interface application, they don't provide the accuracy needed for reliably detecting rain through a windshield.

2.3.3. Analog Devices Capacitance-to-Digital Converter

Stand-alone integrated circuits often off better performance than integrated modules. Analog Devices' offers a series of highly regarded capacitance-to-digital converters (CDCs). These chips offer industry leading accuracy in a variety of different configurations for applications requiring only one sensor to ones requiring up to 14 sensors. A table of all Analog Devices CDCs is illustrated in Table 1. Since measuring only one sensor on the windshield, which can be thought of as a “button sensor”, only one channel of conversion was required. This narrowed our search down to either the AD7151/AD7153 low-power, 12-bit, one-channel CDCs or the AD7745/AD7747 24-bit, one-channel CDCs. Power consumption was of little importance to the design, as it would be low regardless and the device would be running off of the vehicle's power system. The AD7151/53 12-bit CDCs only cost approximately $1.75 per, while the AD7745/47 cost closer to $4.50 per. However, the AD7745/47 offer 24-bits of accuracy on capacitance readings from the sensor, while the cheaper, low-power AD7151/53 offer only 12-bits. As performance was the most critical criteria, the decision was made to focus on the AD7745/47 CDCs from Analog Devices.

Part# Results: 13	Resolution (Bits)	T-Put Rate	# Chan	Supply V	Pwr Diss	Input Base C (pF Max)	Cin Range (pF)	Pkg Type	Price* (1000 pcs.)
AD7153	12	200SPS	1	Single(+2.7 to +3.6)	432µW	5	±2	SOP	$1.77
AD7151	12	100SPS	1	Single(+2.7 to +3.6)	0.3mW	10	+4	SOP	$1.37
AD7152	12	200SPS	2	Single(+2.7 to +3.6)	432µW	5	±2	SOP	$1.97
AD7156	12	100SPS	2	Single(+1.8 to +3.6)	306µW	10	+4	CSP	$1.25
AD7150	12	200SPS	2	Single(+2.7 to +3.6)	0.43mW	10	+4	SOP	$1.37
AD7148	16	4SPS	8	Multi(+3.3Anlg, +3.3Dig)	3.3mW	20	±8	CSP	$1.21
AD7143	16	43.5SPS	8	Single(+2.6 to +3.6)	3.6mW	20	±2	CSP	$1.25
AD7147A	16	111SPS	13	Single(+3); Single(+3.3)	3.6mW	20	±8	CSP	$1.25
AD7147	16	111SPS	13	Single(+3); Single(+3.3)	3.6mW	20	±8	CSP	$1.28
AD7142	16	27.8SPS	14	Single(+2.6 to +3.6); Single(+2.7); Single(+3); Single(+3.3)	3.6mW	20	±2	CSP	$1.37
AD7745	24	90SPS	1	Single(+2.7 to +5.25)	4.25mW	17	±4	SOP	$4.66
AD7747	24	45.5SPS	1	Single(+2.7 to +5.25)	4.25mW	17	±8	SOP	$4.66
AD7746	24	90SPS	2	Single(+2.7 to +5.25)	4.25mW	17	±4	SOP	$5.01

Table 1: Analog Devices capacitance-to-digital converter circuits

These circuits are designed for one channel of conversion, enabling one single-ended capacitive “button sensor” or two differentially operated capacitive “button sensors” to be monitored. The term “button sensor” simply indicates that the capacitive sensor is taking only one series of measurements over the single capacitor formed by the sensor traces. It does not indicate that the sensor is to be used as a human-interface button, although it potentially could be. It is useful to use the term “button sensor” to differentiate a single point calculation as opposed to a “slider” or array of sensors, which are integrated together to perform analysis of moving objects. Both the AD7745 and AD7747 operate on either 3.7 V or 5 V DC, and have a built in excitation source generator, which is a 32 kHz square wave with peak-to-peak amplitude equal to the operating voltage (Vdd). This excitation source is connected to one conductor of the capacitive sensor traces, and the other conductor is tied to the “Cin” pin.

The primary difference between the AD7745 and the AD7747 is that the AD7745 is designed for floating capacitive sensor traces, while the AD7747 is designed for sensors in which one trace is grounded. Because the AD7747's sensor capacitance is between the excitation conductor and a grounded conductor, any extraneous capacitance between the excitation pin and ground will accumulate as a parasitic capacitance, making the base capacitance of the sensor appear larger than it should be. Since the capacitive sensor base capacitance is only between 5 – 15 pF, any additional parasitics can easily dominate the base capacitance of the system, leading to errors. Alternatively, the AD7745 is designed for floating capacitive sensors, in which the “Cin” conductor is not grounded but is

instead floating. Only capacitance formed between the “Cin” trace and the excitation trace add to the base capacitance of the sensor; any capacitance to ground does not increase the effective capacitance of the sensor. Parasitics to ground can form very easily through shielded cables or on PCB layouts, so this makes the AD7745 design more robust. The decision was made early on to focus on implementing the design with the AD7745, with the AD7747 as an alternative if problems arose.

2.4. Feasibility Matrix

Design Factor	**Weight**	**Self-Designed Circuit**	**uC Capacitive Sensing Modules**	**Analog Devices CDC (AD7745)**
Accuracy	5	2	2	5
Cost	4	3	5	2
Ease of Manufacture	3	2	5	5
Development Time	2	1	4	4
Additional Features	2	1	3	4
Power Consumption	1	2	5	4
TOTAL		*34*	*64*	*68*

Table 2: Feasibility Matrix

The Feasibility Matrix is a useful development tool allowing for quick comparison between a number of different design schemes based on weighted design factors. Design Team 6 concluded that accuracy was the most important design factor, as the capacitive rain sensor would be useless if it could not accurately detect a change in capacitance caused by rain through a 6 – 8 mm glass windshield. Beyond this, cost was evaluated as the second most critical factor, as one of the primary project goals was to develop a sensor with an estimated production cost less than the current optical sensor. Through the Feasibility Matrix, Design Team 6 compared the three proposed designs and determined that the use of a stand-alone capacitance-to-digital converter from Analog Devices, the AD7745, would provide the best solution for a capacitance monitoring circuit.

2.5. Proposed Design Solution

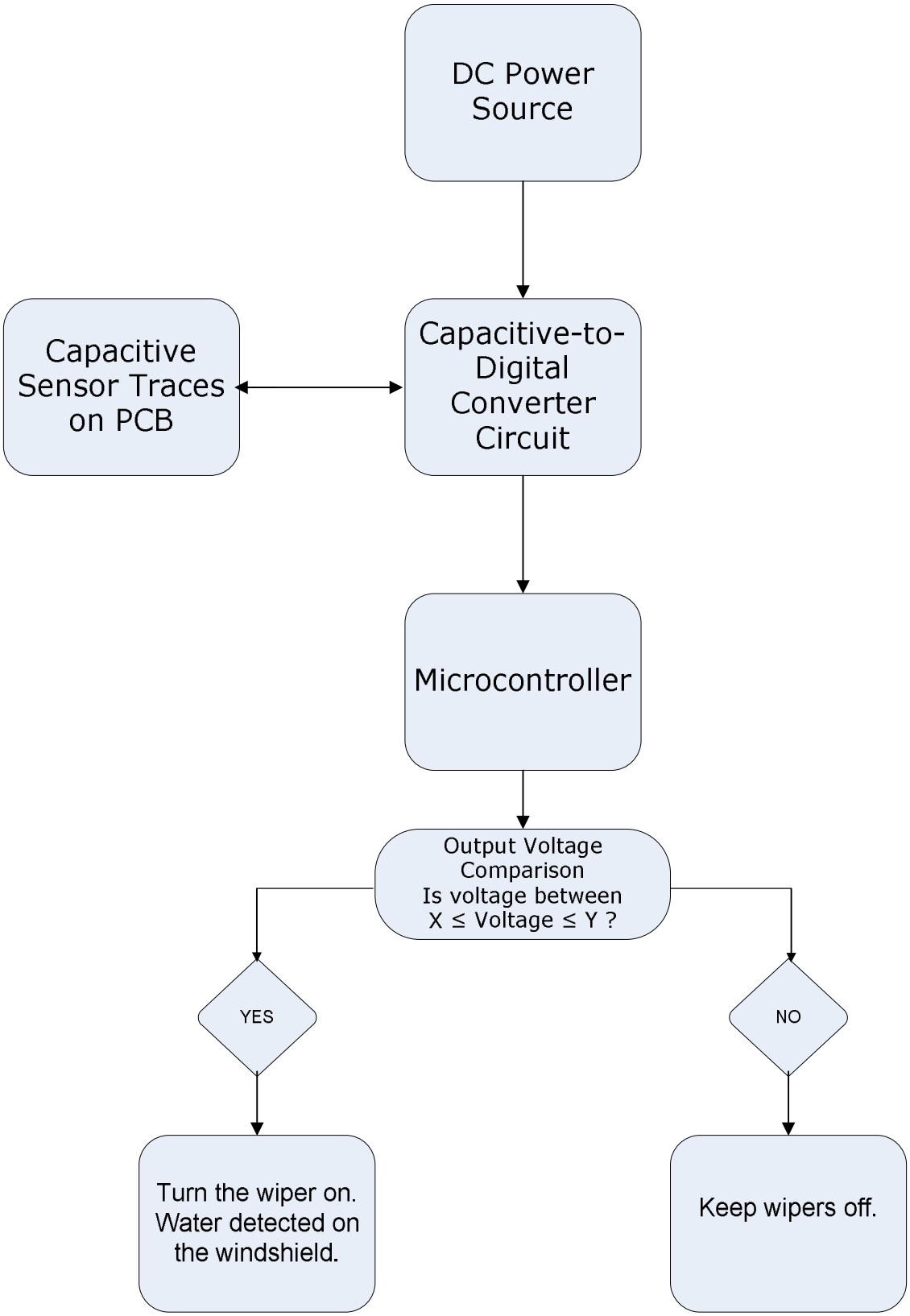


Figure 4: Block diagram of the proposed capacitive rain sensor system

Design Team 6 has developed an accurate and inexpensive capacitive rain-sensing system utilizing the block diagram architecture shown in Figure 4. This device has four primary components: a capacitance monitoring circuit, a microcontroller, a voltage regulator, and the sensor traces. These components are mounted on a stack of two two-layer PCBs which are neatly housed in a plastic enclosure and mounted to the interior of the windshield. The lower PCB contains the sensor traces, which adhere directly to the windshield, on one side and a wire connector on the other side. The upper PCB mounts approximately 1 cm above the lower and contains a protective ground shield on the bottom layer, and the surface-mount components and connectors on the top layer. The device layout is illustrated in Figure 5. The prototype to be displayed at Design Day contains the microcontroller in a separate housing to allow it to interface with a laptop, which will display the wiper operation and sensor data through a computer program. A fully functioning wiper system for display purposes was not realistic, however, an actual Hyundai windshield will be on display with the sensor mounted to it. Production-level prototypes will have the microcontroller on the windshield-mounted unit itself, and these circuits will be on display at Design Day to give viewers a better image of how the final product will look.

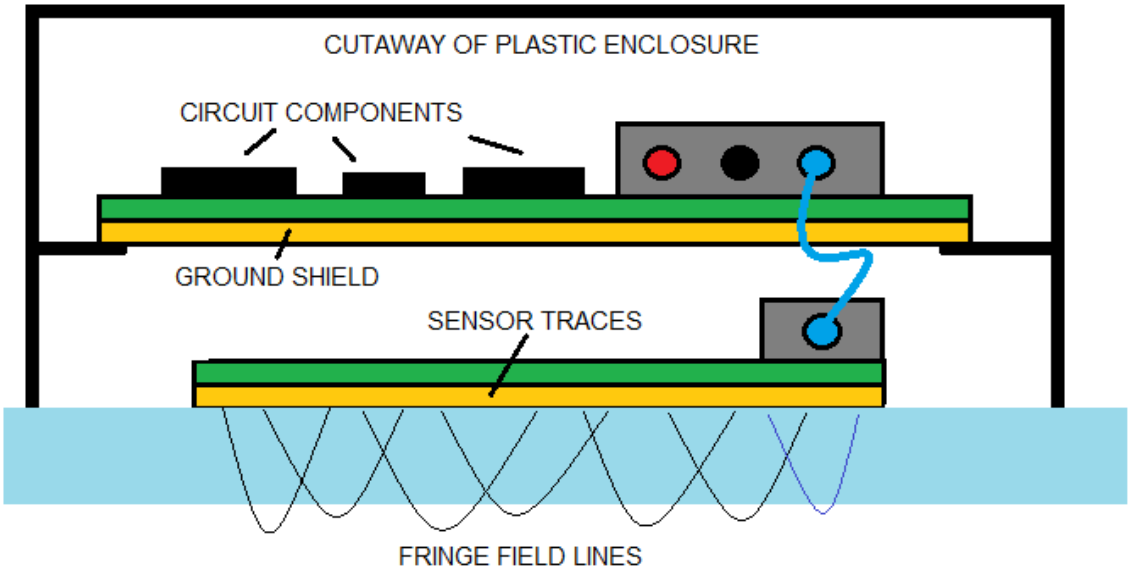


Figure 5: Cutaway view of plastic enclosure displaying two layer design

2.5.1. Capacitance Monitoring Circuitry: Analog Devices AD7745

As described in sections 2.3.3 and 2.4, the Analog Devices AD7745 was chosen as the capacitance monitoring circuit. The AD7745 interfaces with both the capacitive sensor traces and the PIC microcontroller processor. Its primary role is to sample the changing capacitance of the sensor traces and output that data as a digital signal to the microcontroller for processing. The AD7745 communicates with the microcontroller via a two-wire I2C standardized communication system. This is a Master/Slave system with a Master-generated clock line and bidirectional data line. The AD7745 is powered by the 5 V DC output from the ADP3301 linear voltage regulator. It produces a 32 kHz, 5 V square wave excitation signal to be routed to one of the sens\or traces, and the other sensor trace connects to the “Cin” pin. The AD7745 comes standard in a 16-pin surface-mount (TSSOP-16) package.

2.5.2. Microcontroller: Microchip PIC18F4520/PIC16F1826

A microcontroller is necessary in the design to control the AD7745 and process the incoming capacitance data. The PIC18F4520 was selected for use in the prototype display unit due to its free availability in the MSU ECE 480 lab. For production-level prototypes, the very similar but smaller PIC16F1826 will be used, as it contains only 18 pins as opposed to the 40 on the PIC18F4520. The

PIC18F/16F is a popular and affordable 8-bit microcontroller which runs off a 5 V power supply and comes in a DIP or surface-mount package. It can be programmed using the C programming language to perform a wide variety of tasks, and has 3.5 kB of program memory. In the capacitive rain sensor, the PIC serves as the Master in the I2C communication system with the AD7745. It is responsible for configuring the AD7745 into the correct operating state, polling it for capacitive and other data, and interpreting that data by comparing it to known capacitance values gained through extensive testing of the device. If the incoming capacitive data falls into a certain range over a certain number of samples, the PIC will output a signal instructing the wipers to engage. Furthermore, the PIC can differentiate between varying levels of rain to adjust the speed of the wipers, and prevent false positives by ignoring capacitance values outside the range of rain.

2.5.3. Capacitive Sensor Traces: Custom Design

The capacitive sensor trace layout is critical to the performance of the capacitive sensor system. The shape and spacing of the two traces forming the capacitive sensor are directly related to the electric field lines produced when the excitation voltage is applied. As the rain to be detected is present through 6 – 8 mm of glass, the sensor traces should be designed as to maximize the fringing fields away from the plane of the PCB. Glass has a relatively high dielectric constant of around 4.5, allowing easy transmission of electric fields through it. Nonetheless, 6 – 8 mm is a very large distance away from the sensor traces to have to measure, as most capacitive touchscreens have an overlay thickness of only 1 – 2 mm. The software COMSOL was used to model a variety of different sensor layout designs, where parameters such as trace patterns, conductor width, conductor spacing, and total sensor size could be adjusted to find the perfect layout for the system. These parameters had a large impact on the total system capacitance, which had to be less than 16 pF due to the range of the AD7745 CAPDAC, and the shape and strength of the fringe fields. Using COMSOL, an exact sensor trace pattern was decided upon, and empirical results mirrored that of the software's predictions.

2.5.4. Voltage Regulator: Analog Devices ADP3301-5

In a vehicle, the typical battery voltage can range from 11 – 13.5 V depending on the strength of the battery and the operating state of the vehicle and the alternator. Both the AD7745 and PIC microcontroller require 5 V DC for operation, so a reliable voltage regulator was required to scale the vehicle power supply voltage to 5 V. The Analog Devices ADP3301-5 is a linear voltage regulator which

can accept up to 14 V of input voltage, and outputs a preset 5 V DC. It can source up to 100 mA of current, more than enough for the entire device. It offers high linearity, a wide operating temperature range, and is available in a surface-mount package. The ADP3301-5 requires a capacitor on the output pin of at least 0.47 uF in magnitude for proper operation.

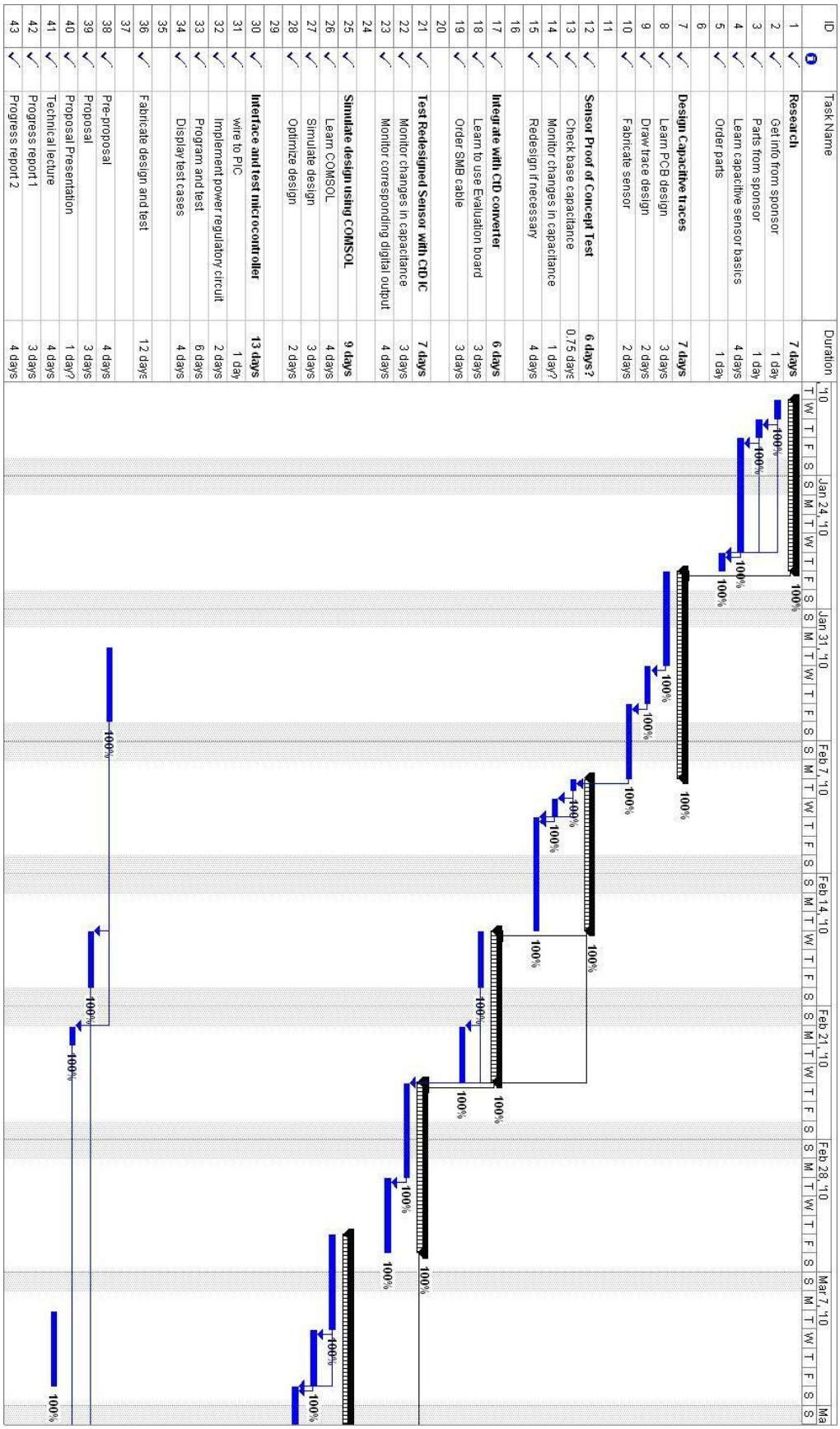
2.6 Estimated Production Cost Per Unit

Component	**Cost**
PIC16F1826 Microcontroller	$0.90
AD7745 CDC	$4.50
ADP3301-5 Voltage Regulator	$1.40
Surface-mount Passive Components	$0.10
Two-layer PCB Production and Assembly	$4.00 (estimate)
Plastic Enclosure	$0.50 (estimate)
TOTAL	*$11.40 (estimate)*

Table 3: Estimated production cost per unit for capacitive rain sensor

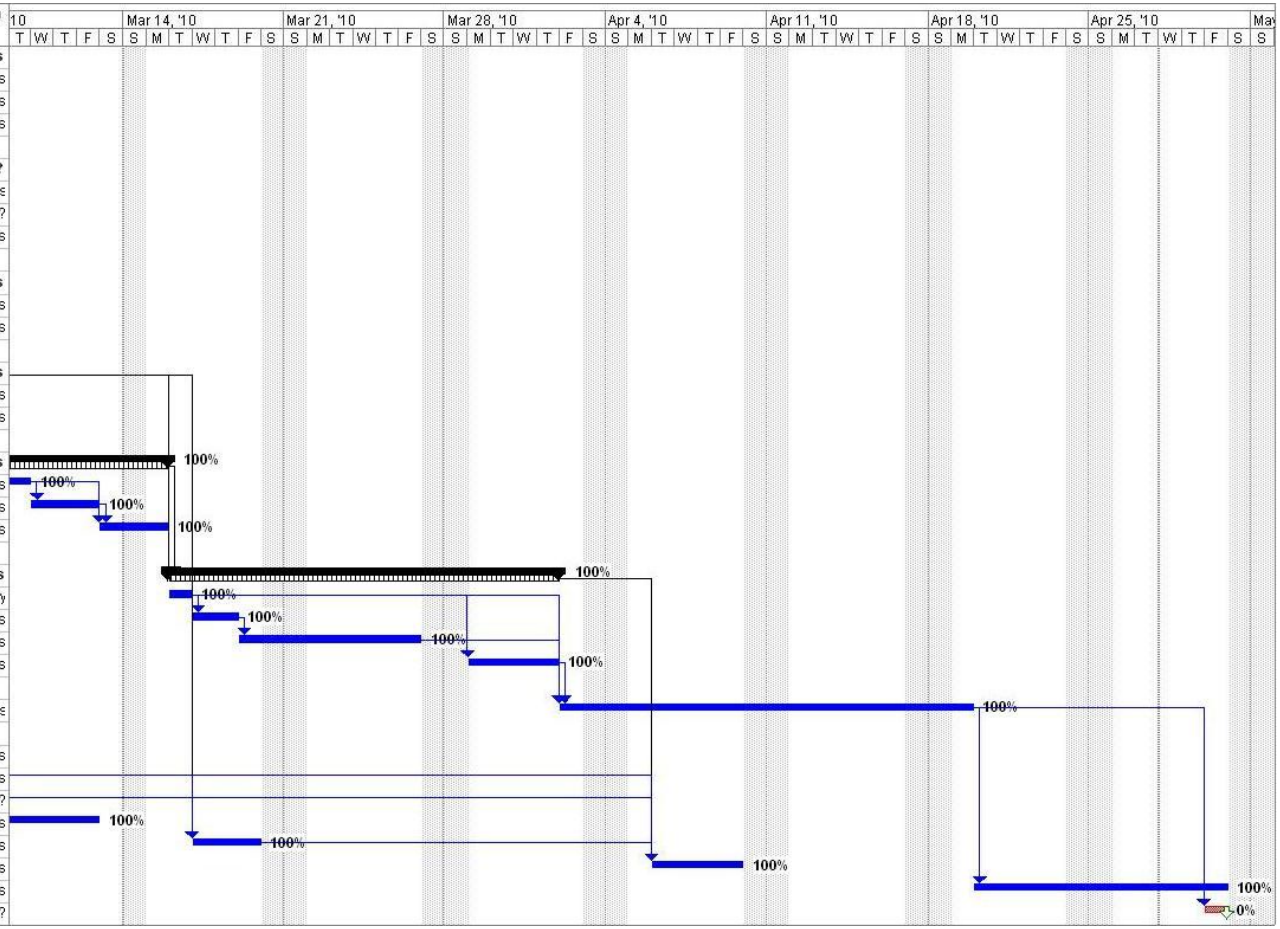
Design Team 6 was tasked with developing a capacitive rain sensor system less expensive than the current optical unit, which costs around $18 per unit. As Table 3 indicates, the estimated production cost of the new capacitive sensor is only $11.40 per unit, a savings of approximately $6600 per 1000 vehicles.

2.7. Gantt Chart



ID	Task Name	Duration
1	**Research**	**7 days**
2	Get info from sponsor	1 day
3	Parts from sponsor	1 day
4	Learn capacitive sensor basics	4 days
5	Order parts	1 day
6		
7	**Design Capacitive traces**	**7 days**
8	Learn PCB design	3 days
9	Draw trace design	2 days
10	Fabricate sensor	2 days
11		
12	**Senson Proof of Concept Test**	**6 days?**
13	Check base capacitance	0.75 days
14	Monitor changes in capacitance	1 day?
15	Redesign if necessary	4 days
16		
17	**Integrate with CtD converter**	**6 days**
18	Learn to use Evaluation board	3 days
19	Order SMB cable	3 days
20		
21	**Test Redesigned Sensor with CtD IC**	**7 days**
22	Monitor changes in capacitance	3 days
23	Monitor corresponding digital output	4 days
24		
25	**Simulate design using COMSOL**	**9 days**
26	Learn COMSOL	4 days
27	Simulate design	3 days
28	Optimize design	2 days
29		
30	**Interface and test microcontroller**	**13 days**
31	wire to PIC	1 day
32	Implement power regulatory circuit	2 days
33	Program and test	6 days
34	Display test cases	4 days
35		
36	Fabricate design and test	12 days
37		
38	Pre-proposal	4 days
39	Proposal	3 days
40	Proposal Presentation	1 day?
41	Technical lecture	4 days
42	Progress report 1	3 days
43	Progress report 2	4 days

ID	Task Name	Duration
7	**Design Capacitive traces**	**7 days**
8	Learn PCB design	3 days
9	Draw trace design	2 days
10	Fabricate sensor	2 days
11		
12	**Sensor Proof of Concept Test**	**6 days?**
13	Check base capacitance	0.75 days
14	Monitor changes in capacitance	1 day?
15	Redesign if necessary	4 days
16		
17	**Integrate with CtD converter**	**6 days**
18	Learn to use Evaluation board	3 days
19	Order SMB cable	3 days
20		
21	**Test Redesigned Sensor with CtD IC**	**7 days**
22	Monitor changes in capacitance	3 days
23	Monitor corresponding digital output	4 days
24		
25	**Simulate design using COMSOL**	**9 days**
26	Learn COMSOL	4 days
27	Simulate design	3 days
28	Optimize design	2 days
29		
30	**Interface and test microcontroller**	**13 days**
31	wire to PIC	1 day
32	Implement power regulatory circuit	2 days
33	Program and test	6 days
34	Display test cases	4 days
35		
36	Fabricate design and test	12 days
37		
38	Pre-proposal	4 days
39	Proposal	3 days
40	Proposal Presentation	1 day?
41	Technical lecture	4 days
42	Progress report 1	3 days
43	Progress report 2	4 days
44	Free time	9 days
45	Design day	1 day?



CHAPTER THREE: TECHNICAL DETAILS

3.1. Sensor Trace Design

The capacitive sensor traces are formed by two copper conductors, closely spaced, laid out flat on a PCB. This PCB adheres directly to the interior of the windshield with the use of 3M 468MP adhesive transfer tape. This tape is non-conductive and has been recommended for similar applications (see Reference 6). Capacitive measurements are taken over the sensor trace area on the windshield, therefore, only rain hitting this area will be detected. This still offers a larger detection area than the current optical system, however. Since the purpose of the sensor is to detect between rain, no rain, and other objects on the windshield, no data is required about the movement of the objects, just the presence of them. This negates the need for a complex trace layout such as a slider or a touch-pad, which are used to track movement, typically a human finger. Therefore, a single button sensor design forming one capacitor to be measured was used, requiring only the two traces mentioned previously. The layout of these traces can take on a variety of different shapes and sizes. Examples of sensor trace layouts are illustrated in Figure 6.

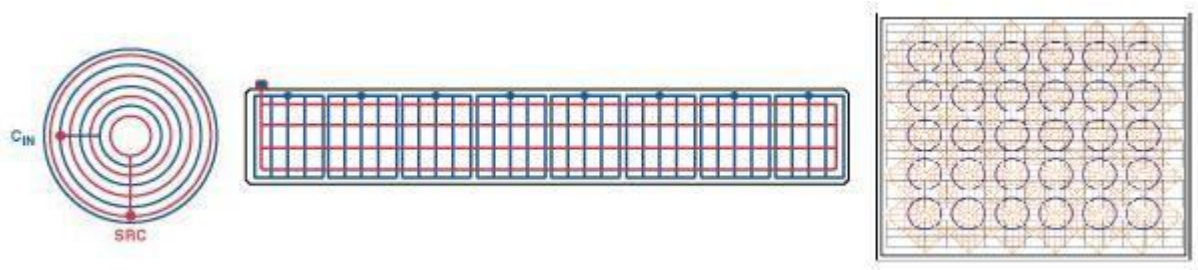


Figure 6: From left to right – a button sensor, slider, and touch-pad trace layouts

For a button sensor with two conductors, the primary design variables to consider are the pattern of the two conductors, the width of the conductors, the spacing between the conductors, and the overall size of the sensor layout. All of these have a substantial impact on the total capacitance of the system, as well as the distribution of fringe field lines. Typical patterns include concentric circles, parallel lines, or interweaving “fingers”. The size of the sensor layout is chosen to match the system environment. If the sensor is to detect a human finger touch, the overall size should be close to the size of a fingertip. For the capacitive rain sensor, the size was chosen to be as large as possible without extending beyond

the size constraints provided by HATCI. Due to the AD7745 CDC's ability to only null out the base capacitance up to 16 pF, the size also had to be adjusted so the sensor did not exceed that value.

The spacing between the conductors is critical to the overall capacitance of the sensor, as well as the distribution of fringe field lines. A continuous spacing of 0.25 - 1 mm between conductors is common, as this typically provides a good combination of large fringing fields and small base capacitance. The thickness of the windshield overlay presented a considerable design challenge, and because of this the fringe fields took primary concern. If the fringe field lines did not extend all the way through the glass, the change in capacitance from any object on the windshield would be much smaller than if the lines did extend all the way. However, as the sensor traces move closer together in a design, the capacitance will increase, so a balance must be struck. The ideal sensor trace layout for the capacitive rain sensor is the one that produces the farthest extending fringe fields and covers the largest area, while minimizing the sensor capacitance (maximum of 16 pF due to AD7745).

Assuming an effective sensor design, care must also be taken in the materials surrounding the trace area. The dielectric constant of a material, ε, is a measure of the material's ability to transmit an electric field. Higher values of ε indicate a better transmission of electric fields. The dielectric constant of air is 1, standard FR4 PCB material is around 4, and glass is approximately 4.5 – 6. Windshield glass also contains a thin layer of plastic wedged between the two panes of glass, but results indicated that this had little effect on the system. Because of air's poor dielectric constant, no air gaps can be present between the sensor trace area and the windshield, as any air gap would weaken the field above it.

3.1.1. COMSOL Multiphysics

COMSOL Multiphysics is a power scientific tool that allows the use of visual environments to model and implement engineering problems. The software uses Partial Differential Equations (PDEs) to solve for complicated models. It is basically a computer program that allows the modeling and simulation of a wide variety of physical phenomena. Technical problems relating to the field of: acoustics, electromagnetics, heat transfer, fluid dynamics, structural mechanics and MEMs (Micro Electro Mechanical Systems) can be modeled and studied using a rich and interactive user environment.

Even though the software allows modeling of complex applications, it does not require an in-depth knowledge of numerical or mathematical analysis. It is possible to build models by simply defining the physical parameters like area, length, width, fluxes, and constraints rather than defining the equations. Once the parameters are defined and the sub-domain and boundary conditions are set, COMSOL automatically compiles a set of PDEs to represent the entire model. Due to the simple user interface and easy modeling, COMSOL was chosen to model the capacitive sensor and observe the base capacitance before actually fabricating the PCB design. There was also a time constraint and the team did not have enough time to explore other alternatives.

Another reason the team opted to use COMSOL was because of the availability of the software. Prior to using this software, the team had designed a capacitive sensor on Eagle PCB Design to verify if the design works. The sensor worked surprisingly well for our first try but a more accurate design was needed as required by our sponsor. That is why COMSOL was used to design the capacitive sensor model and then compare various models to see which one is more accurate. A couple of designs were laid out and the best one was chosen based on the COMSOL results. In the following, only four designs will be discussed to give an overview of how COMSOL was used to optimize the design of the capacitive sensor.

For more information on COMSOL, please refer to Appendix F.

Design 1:

The first capacitive sensor that was designed was on Eagle PCB Design. The sensor was approximately designed and the area was kept under 1200 mm^2 as specified by the sponsor. The purpose of this initial design was to monitor the capacitance values and if whether it changes. The following pictures show the sensor trace layouts and the actual fabricated design.

Figure 7: Design 1 trace layout on Eagle PCB Design

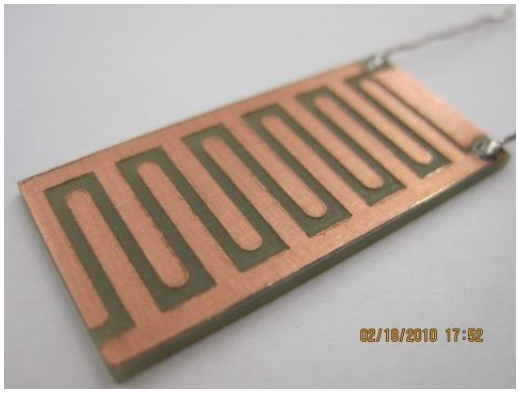


Figure 8: Design 1 fabrication

Next, COMSOL was used to model this design. The 2D model is shown in Figure 9 below. It highlights some of the important terms: trace gap, trace width and trace length that will be used to explain the results.

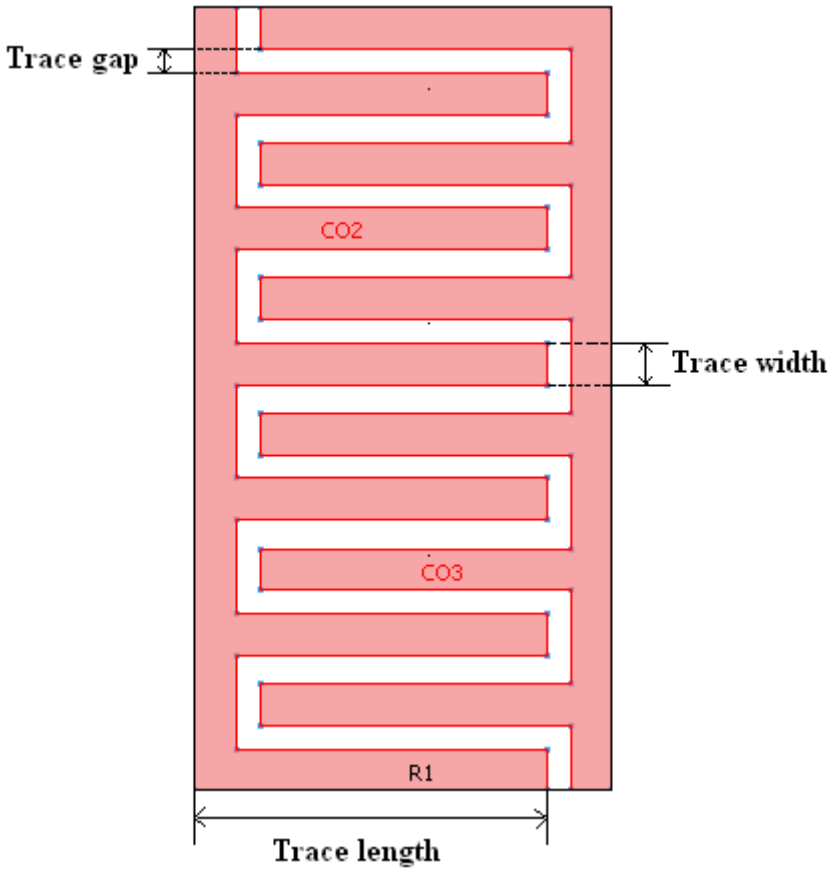


Figure 9: 2D trace on COMSOL

The above 2D trace is then extruded into 3D geometry. A 5mm layer of windshield is added on top of the sensor traces. This is the Subdomain 2. A 1mm layer of silicon substrate is added to the bottom of the traces. This is the Subdomain 1. The windshield is typically made out of glass of dielectric in range 4.6 to 14.5. The dielectric constant of each material was defined as follows.

Settings	Subdomain 1 (Silicon)	Subdomain 2 (Glass)
ϵ_r	2.1	~10.0

The following figure 10 shows the 3D geometry of the sensor traces along with the windshield and substrate layers.

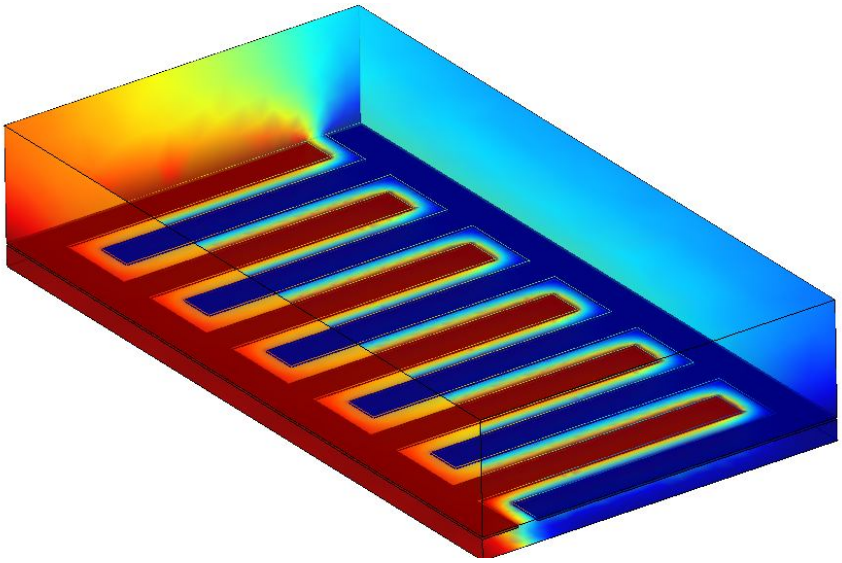


Figure 10: 3D trace on COMSOL – Design 1

In order to determine the sensitivity of the sensor, it was important to plot the fringe field lines that penetrate through the windshield. The intent was to see how many fringe field can penetrate and go through the 5mm windshield. A 10mm layer of air was added to observe the fringe fields past the windshield.

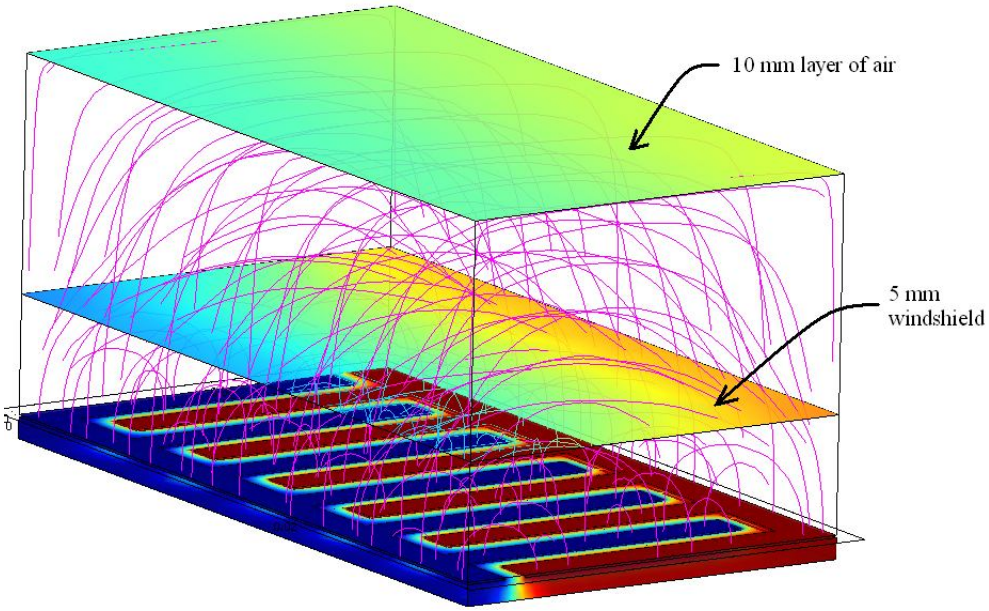


Figure 11: 3D trace on COMSOL with fringe field lines– Design 1

Figure 12 below shows a side view of the sensor with the fringe fields. This picture gives a better idea of the extent to which the fringe fields penetrate. The fringe fields under the windshield do not contribute to any change in capacitance.

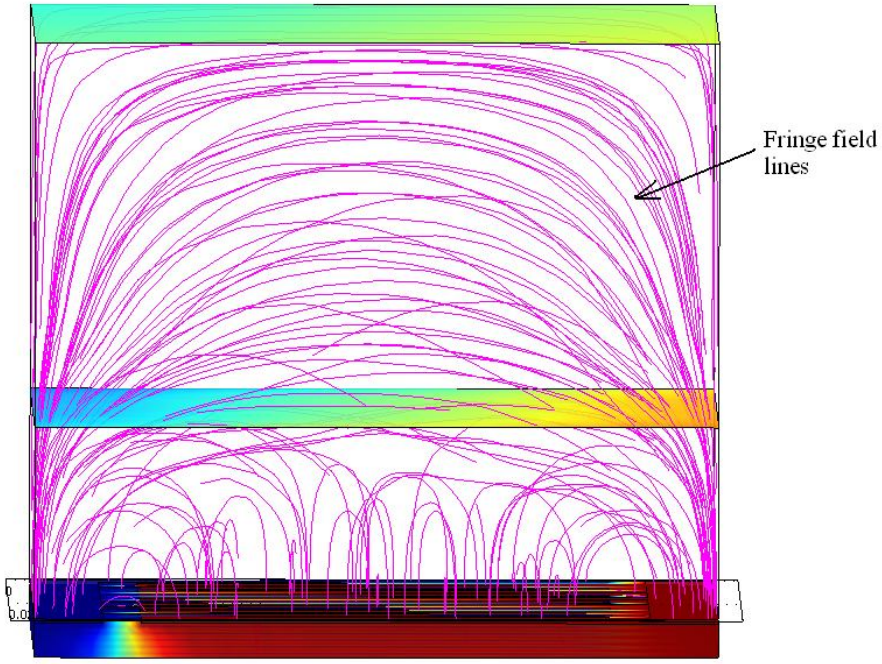


Figure 12: 3D trace on COMSOL with fringe field lines alternate view – Design 1

Results:

The results of Design 1 are summarized in the following table:

Trace characteristic	Measurement
Trace width	1.778 mm
Trace gap	1 mm
Trace length	14.5 mm
Trace Area	800 mm^2
Simulated Base Capacitance	11.4873 pF
Measured Base Capacitance	9.5620 pF

Table 4: Design 1 results

The desired base capacitance should be around 16 pF because the capacitance to digital converter has a maximum offset of 16 pF. This means that the converter is able to nullify the base capacitance as long as it is less than 16 pF. This is effectively used to remove any capacitance changes due to noise and electromagnetic radiation.

Problems:

There was some inconsistency in the measured and simulated value of capacitance. This is because at the time of simulation, the values of dielectric constants were not confirmed. The correct values are accounted for in the next three design cases. Design 2 looks at changes in trace length and trace width to see the effect on the base capacitance.

Design 2:

Since COMSOL is able to give an estimated base capacitance, the second design was not fabricated. This design has a larger area, longer trace length and longer trace width. The purpose of this design is to show the effect on base capacitance with these parameter changes. The 3D geometry is shown below:

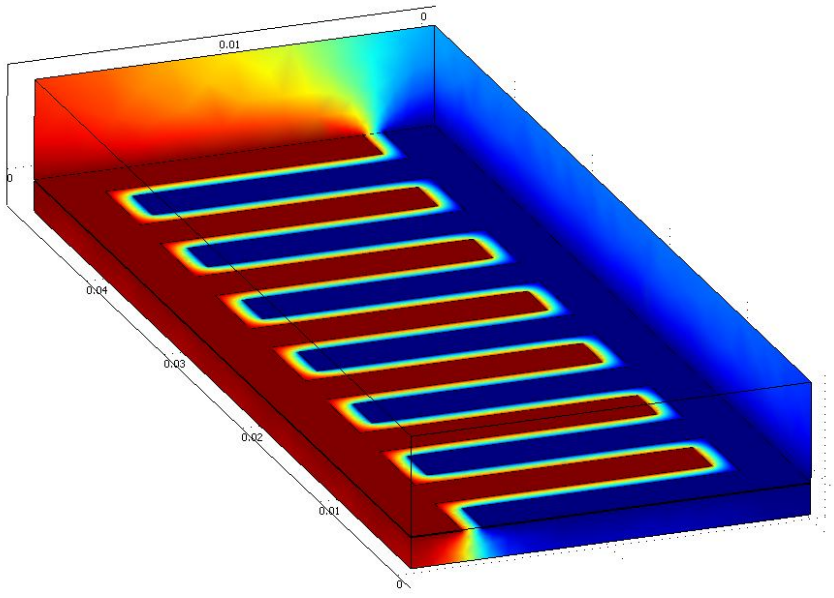


Figure 13: 3D trace on COMSOL – Design 2

Results:

The results of Design 2 are summarized in the following table:

Trace characteristic	Measurement
Trace width	2.54 mm
Trace gap	1 mm
Trace length	16.3 mm
Trace Area	963.43 mm^2
Simulated Base Capacitance	17.9301 pF

Table 5: Design 2 results

Comments:

The results of this design were as expected. The capacitance of the sensor is given by $C = \frac{A \cdot \varepsilon}{d}$, where A is the sensing area, ϵ is the dielectric permittivity $\left(8.85 \times 10^{-12}\, F/m\right)$ and *d* is the trace gap. Since the sensing area has increased compared to Design 1 so did the value of base capacitance. This is outside the required range of 16 pF and so no further analysis was done for this design. In the next design, the effect of decreasing trace gap is observed.

Design 3:

In this design, the effect of changing trace gap is observed on the base capacitance. Compared to Design 2, the trace length and trace width are kept the same and only the trace gap is changed. The reason for doing so is to study the changes in base capacitance due only to the trace gap.

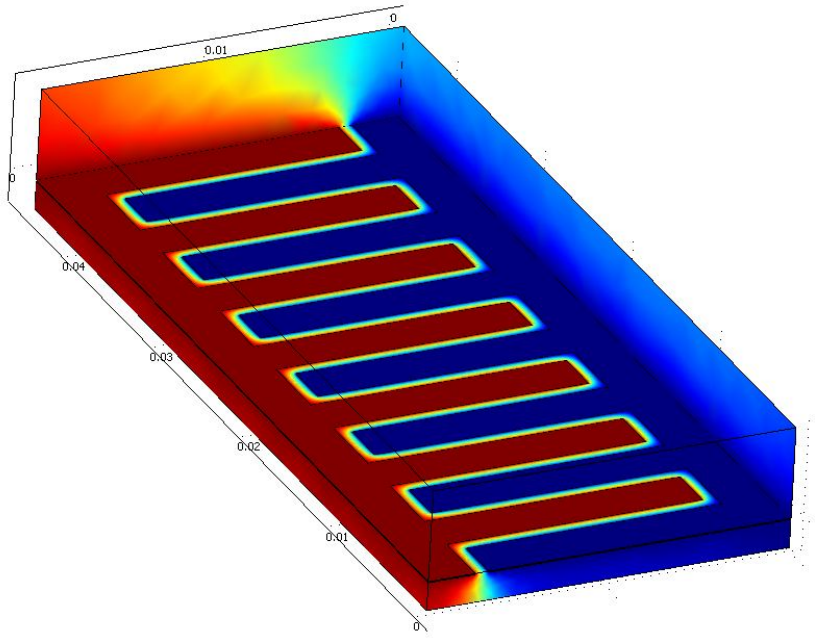


Figure 14: 3D trace on COMSOL – Design 3

Results:

The results of Design 3 are summarized in the following table:

Trace characteristic	Measurement
Trace width	2.54 mm
Trace gap	0.7 mm
Trace length	16.3 mm
Trace Area	872.656 mm^2
Simulated Base Capacitance	20.5280 pF

Table 6: Design 3 results

Comments:

The results of this design were as expected. The capacitance is given by

$C = \frac{A \cdot \varepsilon}{d}$. Since the trace gap, *d* has decreased compared to Design 2, the capacitance should

increase. This is exactly what was observed. The capacitance increased from 17.9301 pF to 20.5280 pF. This is outside the required range of 16 pF and so no further analysis was done for this design. In the next design is similar to Design 2 except that the area is decreased from 963 mm^2 to 913 mm^2. In order to do this, one of the traces was deleted.

Design 4:

This is the final design. Out of the above three designs, the closest one to 16 pF is Design 2 with a base capacitance of 17.9301 pF. In Design 2, the number of traces of both comb drives was seven. However in Design 4, one of the traces was deleted thus reducing the overall area. Expected result is a decrease in the base capacitance. The following figure shows that the left comb (red) has seven traces while the right trace (blue) has six traces.

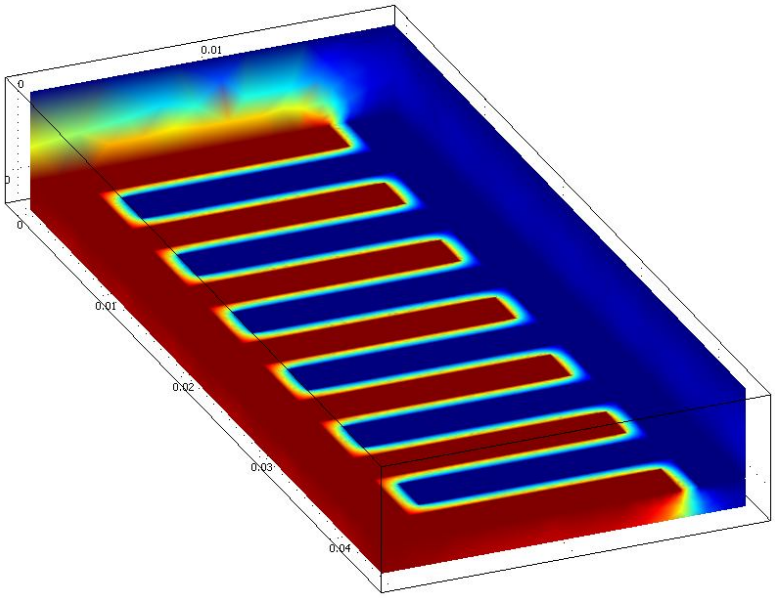


Figure 15: 3D trace on COMSOL – Design 4

In order to further analyze this design, the fringe field lines were plotted to observe the penetration effect through the windshield. The following figure shows the 3D trace design with fringe fields.

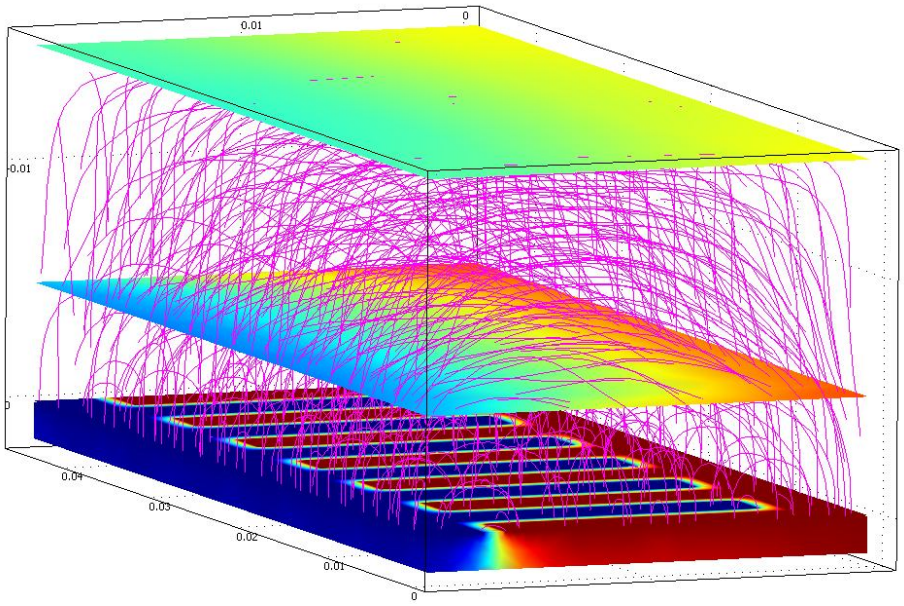


Figure 16: 3D trace on COMSOL with fringe fields – Design 4

An alternate view is shown in Figure 17 below. This gives a better idea of how many fringe field lines penetrate through the windshield.

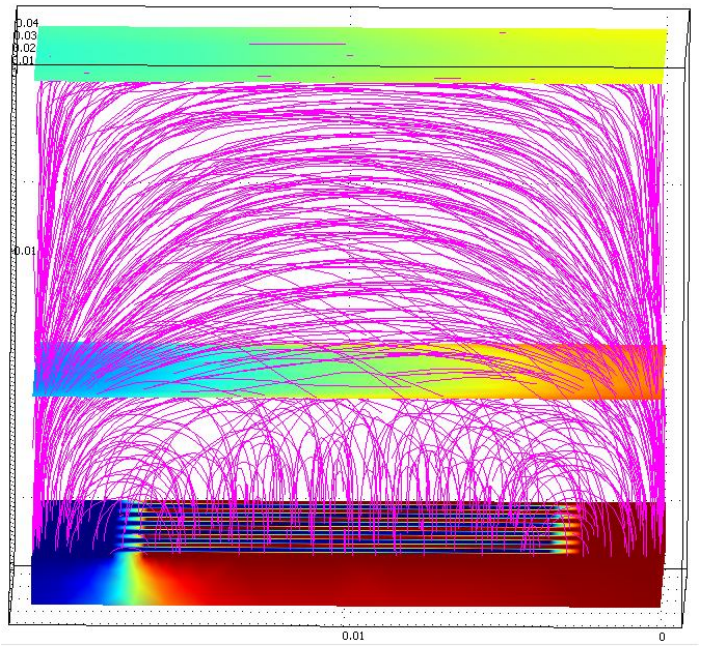


Figure 17: 3D trace on COMSOL with fringe field alternate view – Design 4

Results:

The results of Design 4 are summarized in the following table:

Trace characteristic	Measurement
Trace width	2.54 mm
Trace gap	1 mm
Trace length	16.3 mm
Trace Area	913.036 mm^2
Simulated Base Capacitance	16.581 pF

Table 7: Design 4 results

Comments:

The results of this design were as expected. The overall base capacitance decreased from 17.9301 pF to 16.581 pF. This is close to 16 pF and so no further analysis was done for this design. The next design is similar to Design 2 except that the area is decreased from 963 mm^2 to 913 mm^2. In order to do this, one of the traces was deleted.

Comparison to Design 1:

A comparison of figure 12 and figure 17 clearly shows that Design 4 has more concentrated fringe field lines. Not only that, but more fringe fields penetrate through the 5mm windshield and that increases the sensitivity of the sensor. This is because any change in capacitance will be larger in Design 4 because of interference with a larger number of fringe field lines. The fringe field lines not only penetrate the windshield but also go through the 10mm layer of air on top. Clearly, the penetration of fringe fields in Design 4 is more than that of Design 1.

	Design 1	Design 4
Base capacitance	11.143 pF	16.581 pF
Change in capacitance with touch (one trace)	~ 1pF	819.9 fF

Change in capacitance with touch (whole surface)	1000 fF	2.1651 pF
Change in capacitance with water (min)	~ 11 fF	20 fF
Change in capacitance with water (max)	~ 200 fF	497.3 fF

Table 8: Comparison of Design 1 and Design 4

3.2. Analog Devices AD7745 CDC

As described in section 2.5.1, the AD7745 is the mediate between the capacitive sensor traces and the microcontroller. The core of the AD7745 is a 24-bit Sigma-Delta architecture ADC which is modified to convert capacitance directly to a corresponding digital signal. A simplified diagram of this capacitance-to-digital converter in the AD7745 can be seen in Figure 18, and a more detailed circuit schematic of the Sigma-Delta CDC is displayed in Figures 19 and 20. At a high level, the Sigma-Delta CDC functions by balancing charge through two capacitors – the variable sensor capacitor, Csensor, and an internal reference capacitor. The capacitors are switched between a fixed input voltage to charge them, and then discharge through an integrator. This can be thought of as a charge amplifier, as illustrated in Figure 19, which produces a voltage proportional to the total charge. As Csensor increases, more charge is pumped into the integrator from that branch because:

$$Q = C * V$$

With increasing Csensor, the output of the integrator will grow larger. This is fed to a comparator to produce a series of zeros and ones, which vary with the charge needed to balance the feedback loop. The feedback loop connects only to the reference capacitor, so as Csensor increases, the output voltage increases, which is fed back to the reference side and increases the charging voltage of that capacitor to balance the two branches. The feedback signal is also fed through a third-order digital filter to produce the digital result which can then be output to a microcontroller for processing. The Vref(+) and Vref(-) signals are reference voltage signals supplied by an internal temperature sensor to

prevent drift from temperature variations.

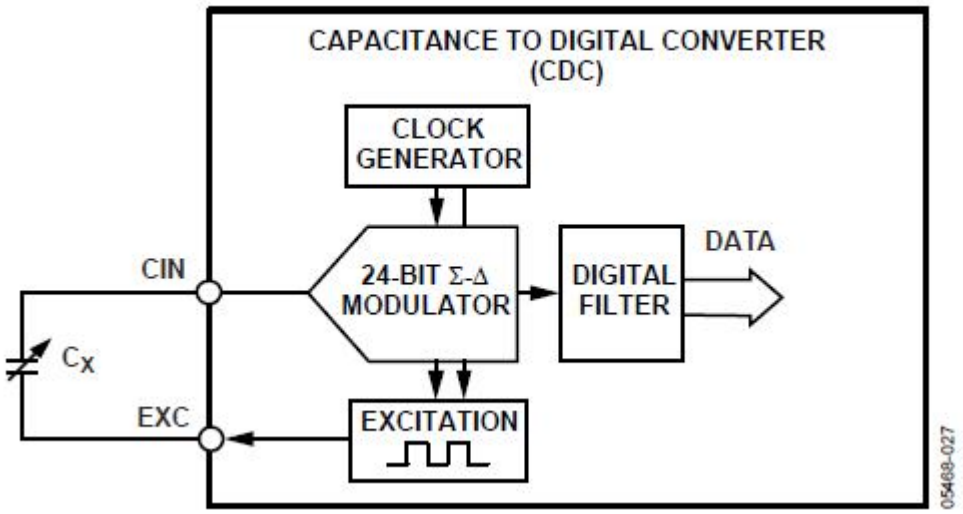


Figure 18: AD7745 CDC architecture

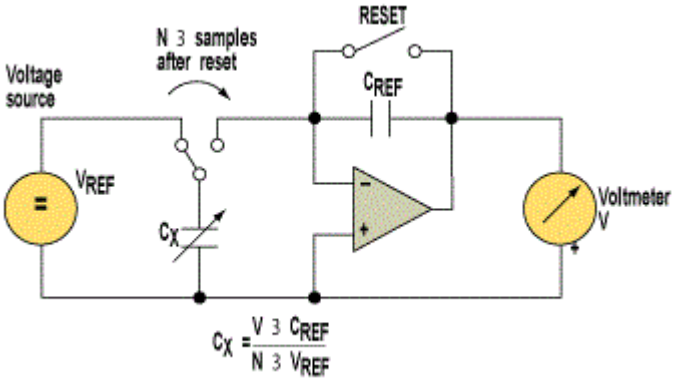


Figure 19: Charge amplifier circuit of which the Sigma-Delta CDC is roughly based on

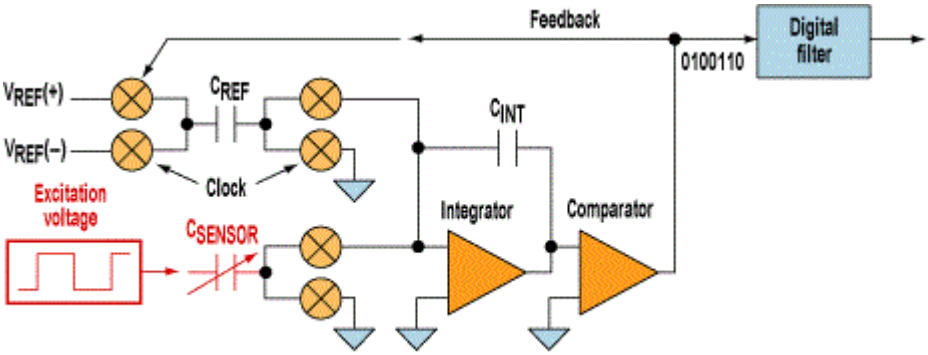


Figure 20: Detailed circuit schematic of a Sigma-Delta CDC

The AD7745 can measure up to +/- 4.096 pF changing capacitance, and outputs the result as a 24-bit digital signal. It can, however, accept up to approximately 16 pF of unchanging base capacitance from the sensor traces. This base capacitance can then be nulled to approximately 0 pF using the on-board CAPDAC. The CAPDAC can be thought of as a programmable negative capacitance value which can be added to the "Cin" pin to null the base capacitance to around 0 pF. The AD7745 will then be able to measure the full range of +/- 4.096 pF of changing capacitance from there. If one were not to use the CAPDAC, and had a base sensor capacitance value of well over 4.096 pF, the data output would be a constant reading of "4.096 pF" and the sensor would be useless.

The AD7745 interfaces with the PIC microcontroller using the I2C communication system, in which the AD7745 is the Slave and the PIC the Master. I2C stands for "Inter-Integrated Circuit", and is technically a multi-master serial single-ended computer bus. The Master can control multiple Slave devices, although only one is used in this design. The I2C system contains only two wires, a clock line known as "SCL" and a data line known as "SDA". The lines are open-drain type and require pull-up resistors. The SCL line is generated by the Master and used to synchronize the two devices, while the SDA line transmits data bit by bit bidirectionally but is controlled by the Master. Standard operating frequency is 100 kHz.

The AD7745 contains 19 eight-bit registers, many of which must be set to configure the CDC into the correct operating mode for the rain sensor system. As the PIC microcontroller is the Master, it is responsible for writing the correct hex codes into the registers. The PIC is programmed to perform an initialization sequence upon start-up (see 3.3 for more details).

The AD7745 is first reset to clear any data or settings. The excitation signal is then setup to be full strength of Vdd. Next, the CAPDAC's are set to null out the base capacitance of the system close to 0. The capacitive channel is setup to put it into single-ended mode at a sample update rate of 62 ms. After this, the temperature channel is setup to take temperature measurements every 62 ms as well. Finally, the AD7745 is placed into continuous conversion mode, where it will start producing 24-bit capacitive data readings approximately every 62 ms. The data is stored in three registers, each of

eight-bits, and must be read sequentially to ensure that no data corruption occurs. After the initialization sequence is complete, the PIC polls the status register of the AD7745 to determine when a capacitive data sample is available to be read. When the status register bit goes high, the PIC reads from the three capacitive data registers sequentially and stores the data for processing. Temperature data can be read in the same way. If the initial starting temperature is below 32 degrees Fahrenheit, the system will shut off as the product is not intended to work in sub-freezing conditions, as per the sponsor HATCI.

3.3. PIC Programming and Interface

3.3.1. The I²C Interface

In order for the PIC to utilize its I^2C capabilities, the following header file must be included.

```
#include <i2c.h>
```

This header file allows the microcontroller to use all available I^2C subroutines (shown below). In order to use these subroutines, the communication lines must be initialized.

```
OpenI2C( MASTER, SLEW_OFF);
SSPADD = 0x3F;
```

The OpenI2C function sets the microcontroller to the master device (the AD7745 is configured as a slave peripheral device) and the SLEW_OFF command lets the microcontroller know to disable all I^2C slewing. Once the communication lines are opened, the start routine must be used to begin communication.

```
StartI2C();
      IdleI2C();
```

Once the communication is started, the master (the PIC) must choose a peripheral device address to communication with.

```
WriteI2C(0xXX);
      IdleI2C();
```

The ‘XX’ must be replaced by the unique device address of the AD7745. The next write will choose a register from the peripheral device in order to send/receive data. If the microcontroller must receive data, then the read routine is used.

```
unsigned char StatusReg = '0xFF';
StatusReg = ReadI2C();
      IdleI2C();
```

The unsigned character, StatusReg in this case, is needed to store the read data. If the consecutive data is to be read, the microcontroller acknowledges the received data and prompts to receive again.

```
AckI2C();
      IdleI2C();
```

If, on the other hand, the microcontroller is done receiving data, then it will send a not acknowledge signal alerting the peripheral device that consecutive reading is done.

```
NotAckI2C();
      IdleI2C();
```

Once all reading and writing is finished, the I^2C communcation must be stopped.

```
StopI2C();
```

This is the only I^2C routine that does not require the idle function to follow because it is the end of the current I^2C communication. The last routine used is restart, which clears the data on the communication line, but allows for additional communication to follow.

```
RestartI2C();
      IdleI2C();
```

3.3.2. The Serial Interface on the PIC

Much like the I^2C header file, the usart header file is required for serial communication between the

microcontroller and the PC. One key difference is that both the PC and the microcontroller must be coded to handle the serial communication. The following section describes the software on the microcontroller end and the next section describes the software on the PC.

```
#include <usart.h>
```

The header file allows all necessary serial functions to be utilized by the microcontroller. In the main function of the microcontroller, the serial communication line must be opened.

```
//Initialize the Serial communication interface
OpenUSART (USART_TX_INT_OFF & USART_RX_INT_ON &
           USART_ASYNCH_MODE & USART_EIGHT_BIT &
           USART_CONT_RX & USART_BRGH_LOW, 63);
RCONbits.IPEN = 1;      // Enable interrupt priority
IPR1bits.RCIP = 1;      // Make receive interrupt high priority
INTCONbits.GIEH = 1;    // Enable all high priority interrupts
```

In the OpenUSART routine, the various predefined constants (reference the header file) initialize the serial connection based on various different variables. In our particular application, we utilize interrupts which must be enabled as illustrated above. Once opened, the connection can be used.

```
unsigned char cc;
cc = getcUSART();
while(BusyUSART());
```

The unsigned character is temporary and holds the value of an interrupt character on the USART line (received via getc). The Busy function functions very similarly to the Idle function from the I^2C interface and halts all subsequent program operation until the communication is finished. This helps prevent against control and data hazards. Once the appropriate amount of data has been read from the serial line, the microcontroller can respond, or put data on the line.

```
putcUSART (CapHIGH);
while(BusyUSART());
```

The putc function puts the character value in parenthesis (CapHIGH in this case) onto the serial line to be read by the PC.

3.3.3. The Serial Interface for the Visual Basic Application

In order for the microcontroller to communicate with the Visual Basic application, a serial interface must be used. Appendix X has the wiring diagram for the MAX232 serial communcation chip. In order for the application to allow serial communication, a couple things need to be considered. First, the CRs232.vb file must be included. Second, the port, baud rate, data bit size, stop bit, parity, and timeout need to be configured. Below is a snippet of code from the Main.vb file, which is refrenced in Appendix X. The code must be included in the form's Load event:

```
With setRs232
      .Port = 3
      .BaudRate = 2400
      .DataBit = 8
      .StopBit = Rs232.DataStopBit.StopBit_1
      .Parity = Rs232.DataParity.Parity_None
      .Timeout = 10000
End With
setRs232.Open()
```

In our case, the USB-to-Serial connector is set to COM3, or port 3. The baud rate is determined by the speed of the clock used by the microcontroller. The data bit size is determined by the amount of data on the serial line at any one time. The stop bit is set to whatever character is desired to terminate serial based communication.

After the initialization of the connection, the third step would be to trigger an interrupt by sending a character over the serial line.

```
setRs232.Write("X")
```

The Write command sends a single character, 'X' in this case, over the serial line to interrupt the microcontroller to perform a certain action. If the command is for the microcontroller to return data, the Read command must be used where the integer in the parenthesis represents the number of bits

to recieve

```
setRs232.Read(1)
```

3.3.4. The Visual Basic Application GUI

In order to visually interpret the capacitance data, we have created a Visual Basic application involving two main forms. The first form is loaded when the program is loaded, and is represented by Figure 21 below. The username and password fields must be correct for the AD7745 to initialize.

Figure 21: Visual Basic Login GUI

Once the user presses the 'Login' button, the graphical user interface in Figure 22 will load. At first, the graph will be blank, the Data textboxes will be blank and no LED pictures will be shown under the Interpretation field. After inputting a sampling time and pressing 'Start', the user can see the current capacitance, temperature, change in capacitance and interpretation of the capacitance value. The animation of the windshield automatically increases/decreases in speed depending on what is sensed.

Figure 22: Visual Basic Main GUI

The microcontroller setup, as pictured below, is housed in a small black control unit with an output for serial connectivity as well as a simple on-off switch. All components from Figure 23 are soldered onto a small two layered breadboard and placed inside the black box pictured in Figure 24.

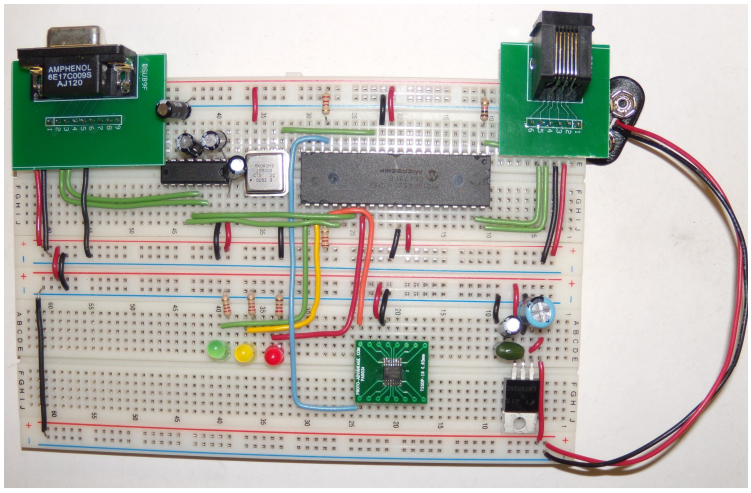


Figure 23: Circuit Connectivity

All of the components in the Protoboard above fit in the box below:

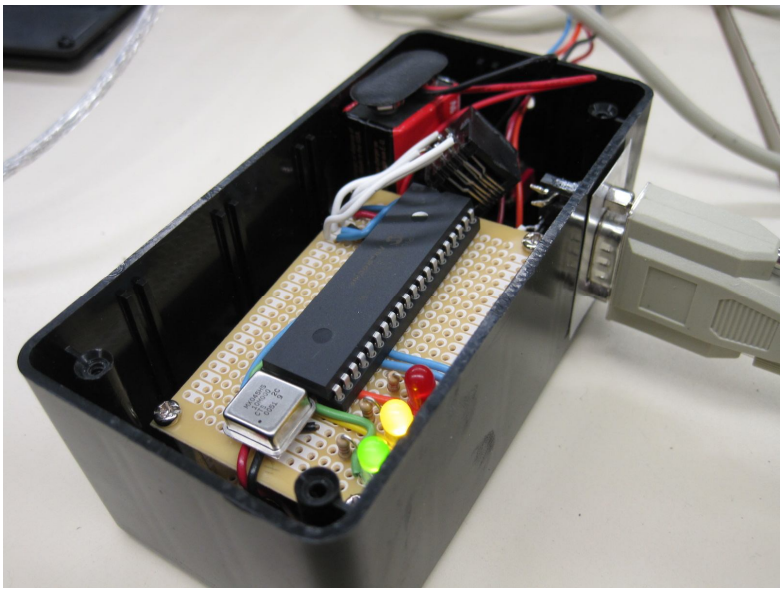


Figure 24: Boxed Control Unit

3.4. ADP3301-5 Voltage Regulator

The capacitive rain sensor is designed to operate on a vehicle's power supply, which can range from 11.5 – 13.5 V DC depending on the strength of the battery and the current state of the vehicle and alternator. The PIC and AD7745 both operate on 5 V, so a voltage regulator was required to regulate the changing vehicle voltage to a steady 5 V. It also had to meet the current requirements of the two components. The AD7745 uses only 0.7 mA of current, while the PIC16F1826 uses approximately 5 – 10 mA depending on its operation state. The Analog Devices ADP3301-5 is a fixed 5 V DC output, up to 14 V DC input, linear voltage regulator capable of supplying up to 100 mA of current, more than enough for the device. The ADP3301-5 produces no high frequency switching noise to possibly interfere with the sensor. It comes in an 8-lead SOIC surface-mount package which is mounted close to the other components on the top layer of the upper PCB. A bypass surface-mount capacitor of 0.47 uF is present on the voltage input pin to increase voltage stability at the input. The ADP3301-5 requires a capacitor of at least 0.47 uF on the output pin for proper operation, and an additional surface-mount capacitor is mounted onto the PCB for this purpose.

3.5. PCB Layout Design

The layout and geometry of the PCBs were critical to the functionality of the sensor. Initially, a four-layer PCB was considered, as this would reduce the complexity and cost of assembly of the device. In this case, the bottom layer would contain the sensor traces, the second layer would be empty, the third layer would contain a ground plane (ground shield), and the top layer would contain all surface-mount components and connectors. This design was not pursued because of the worry that the four-layer PCB would not be flexible enough to curve to fit the geometry of the windshield, concerns that the ground shield would be too close to the sensor trace area, and the difficulty of layout out four component traces on only one layer.

Instead of a four-layer PCB, a two-layer design was used with the PCB cut into half and mounted vertically on top of each other with a spacing of 1 cm in between. The bottom PCB contains the sensor traces adhered directly to the interior of the windshield, and a connector on the top layer. The top PCB contains a ground shield on the bottom layer, and the surface-mount components and connectors on the top layer. The top PCB mounts above the bottom PCB by fitting into the small plastic enclosure which covers both devices.

ExpressPCB offered the most inexpensive prototype PCB production service. Using the “Miniboard” option, fixed size 3.8” x 2.5” PCBs can be produced at $51 for three boards. ExpressPCB requires that their software, PCBLayout, be used in designing the PCBs. This software offers industry standard features and an intuitive user interface, so that layout out the board designs was very straightforward. Parts can be selected using the built-in part finder tool, which places the pad geometry of the selected part on the design. If the part to be used is not in the catalog, a custom pad geometry can be constructed. The design includes a total of three integrated circuits, three terminal block connectors, and four passive components, all of which are surface-mount. The sensor trace design also had to be placed onto the PCB. Because the board was intended to be cut in half, the bottom PCB was placed on one side of the design, and the upper PCB on the other. See Appendix D for figures relating to the PCB layout.

CHAPTER FOUR: TEST DATA

4.1. AD7746 Evaluation Board Testing

In the early design stages of the project, once the decision to utilize the AD7745 was made, an evaluation board for the AD7746 was ordered to allow for rapid prototyping of sensor trace designs. The AD7746 evaluation board contains the AD7746, which is the exact same as the AD7745 except for it allows for two channels (two sensors) to be measured instead of one. The evaluation board contains built in circuitry to allow the board to connect directly to a laptop, and includes a CD with software to run a program allowing all data from the AD7746 to be displayed visually on the laptop. Capacitive sensor traces can be connected to the input pins of the AD7746 and performance can be judged through use of the software program. This allows for easy and rapid testing of different sensor trace layouts to determine best performance.

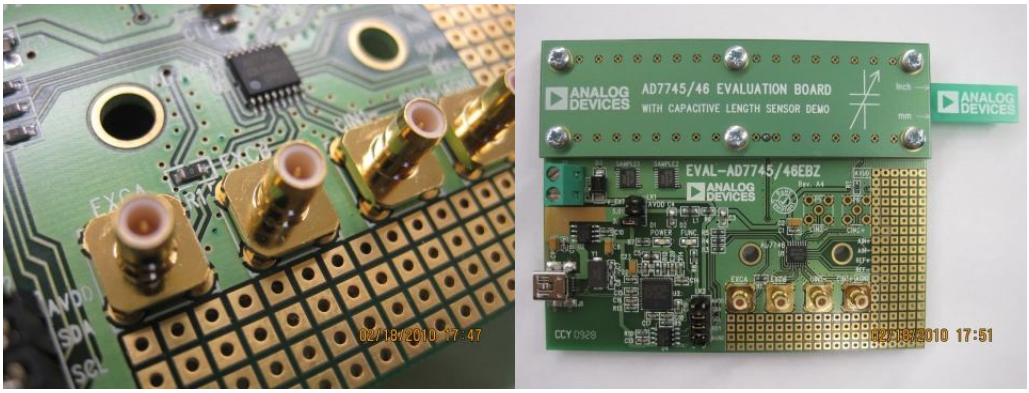


Figure 25: Evaluation Board

Design Team 6 utilized the MSU ECE Shop's capabilities to construct simple two-layer PCBs for performance comparison between sensor designs. Using the COMSOL software, sensor trace layouts were created and analyzed. These were then transferred to a PCB layout using the software EAGLE. They were then provided to the ECE shop, which produced small sensor trace PCBs for use in testing, as seen in Figure 26. These test PCBs were then connected to the evaluation board, and adhered to a small test piece of windshield glass. Objects could then be placed on the test piece of glass and the performance results, such as the change in capacitance, were displayed on the laptop. Using this process, the team was able to analyze real-world performance compared to predicted performance in

COMSOL. The design predicted to work the best in COMSOL also performed the best in testing, and this sensor trace design was chosen as the project moved forward and was used in the PCB layouts from ExpressPCB.

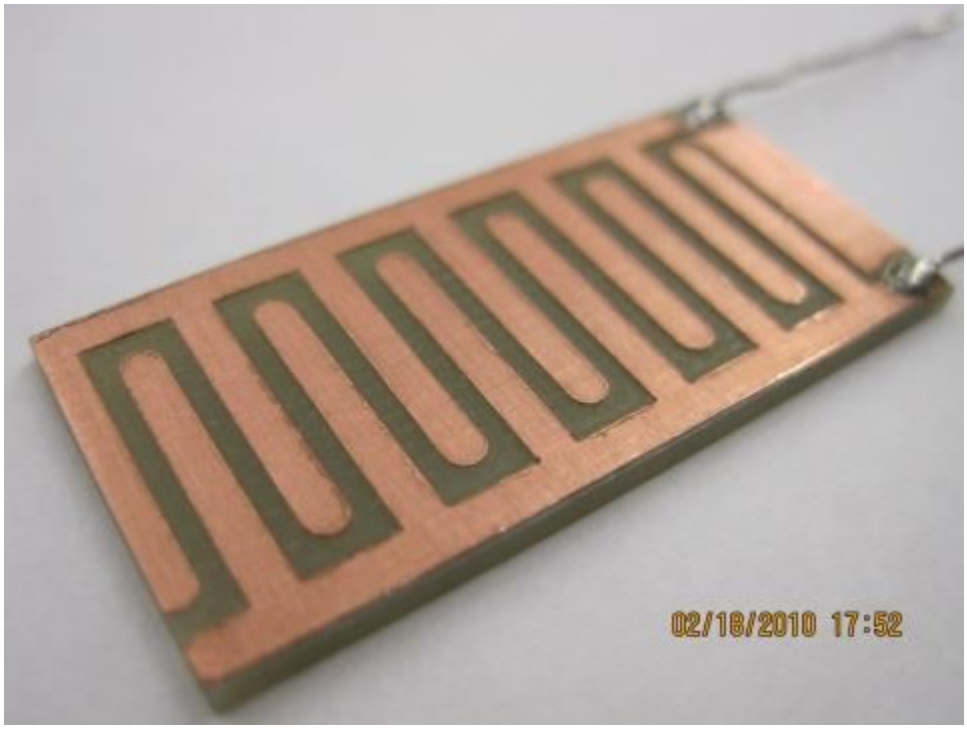


Figure 26: Prototype sensor trace PCB produced by MSU ECE Shop

4.2. PIC I2C Interface & Initialization Testing

Once the sensor trace layout was decided upon, the next step was to begin testing with the AD7745 and the PIC microcontroller without the evaluation board present. This was done by using a standard protoboard with the PIC and AD7745 mounted, and a serial and USB connector hooked up to the PIC to allow for programming and data transfer to a computer. Because the AD7745 is a surface-mount component, a TSSOP-to-DIP adapter was required to be able to be able to mount it in the protoboard.

An image of the protoboard used for testing purposes is shown in Figure 28. The AD7745 and PIC were connected as would be in the final design, as seen in the schematic diagram in Appendix C, except for the addition of the USB and Serial-port connectors to the protoboard, which interface with the PIC microcontroller. The prototype sensor trace PCB was connected to the AD7745 using short cables, and

adhered to the test piece of glass so that meaningful data could be produced.

Before the PIC could communicate with the AD7745, the I2C communication system had to be integrated into the programming of the PIC. The software used to program the PIC was Microchip's MPLab IDE. The PIC can be programmed using C code, which can be entered into the computer running the MPLab IDE software, and then transferred to the PIC using the ICD2 debugger/programmer which connects to the PIC using a USB interface mounted on the protoboard. The MPLab IDE software library includes a ".h" code file containing all of the standard I2C communication commands, such as "Start", "Stop", "Ack", "NotAck", "Idle", etc. Using these commands, the initialization sequence described in section 3.2 was built in C code and programmed into the PIC. The purpose of this initial testing sequence was to determine if the I2C communication system was functioning correctly, and the data values were in fact being written to the AD7745's registers.

Initially, the system did not work. This was determined by writing a hex value to a register, and then reading the value of the register. If they did not match, then the communication failed. The problem was determined to be with the "SSPADD" register value of the PIC, which determines the frequency of the I2C communication line. The system is designed to operate at a standard frequency of 100 kHz. This frequency is determined by the hex value written to the SSPADD register. This register counts down from the programmed hex value twice for every cycle, and so is dependent on the clock frequency of the PIC. The datasheet of the PIC had an incorrect formula for determining the value of the SSPADD. Once this problem was corrected, the I2C communication system worked flawlessly.

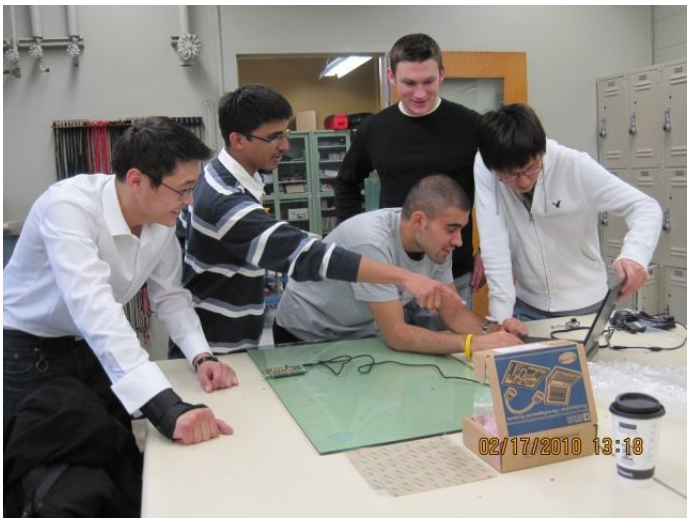


Figure 27: Group working on project

Testing of the AD7745 initialization sequence indicated that all register values were correctly set, and the device was producing capacitive data. The PIC was receiving this data approximately every 62 ms as predicted. Unfortunately, the data could only be seen through the MPLab IDE software's "Watch" feature which could only display one value at a time and did not update. To display the streaming capacitance data, the Visual Basic program had to be created. This program is detailed in section 3.3. This program runs on a laptop or desktop computer and allows for a visual representation of the data on the PIC. The computer connects to the protoboard using a Serial-port, which is connected to the PIC. The Visual Basic program displays the current capacitance data, as well as a graph that plots the capacitive data over a series of samples. In addition, the current temperature reading is displayed, and an animation of a windshield is also shown on the screen. If the capacitive reading falls in the correct range for light mist, rain, or heavy rain, the windshield animation responds accordingly. If a foreign object such as a hand is placed over the sensor area, the windshield animation does not trigger.

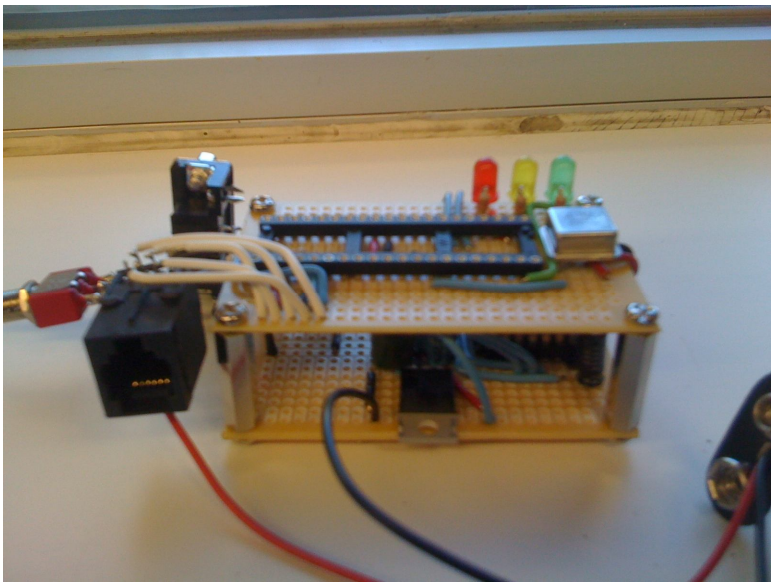


Figure 28: Prototype used for testing

The majority of testing was performed to determine the data ranges for varying stimuli on the windshield. Because the PIC compares incoming capacitance data to these data ranges to determine wiper action, the system can not function before these ranges are determined. These ranges were found by taking a number of data points for each data range. Data ranges had to be determined for each rain sensor setting: clean windshield, mist, rain, downpour, leaf, and hand. Appropriate levels of mist, rain, etc were applied to the test windshield glass, data recorded, and the glass cleared before repeating. Capacitance changes as small as only 10 fF were able to be measured accurately by the AD7745 system, supporting the choice to utilize the high accuracy of the 24-bit CDC.

4.3. Final Prototype Testing

Once all the working code and Visual Basic program was complete, the PCB layout was ordered from ExpressPCB. Three identical PCBs were received, although one had some of the component pads soldered together and had to be scrapped. The other two were cut into their respective sizes using a fine-tooth saw. The surface-mount components were then soldered onto the boards, and wires fit into the terminal blocks connecting the upper PCB to the lower, and the upper PCB to the “black box” containing the PIC18F4520 and laptop-interface equipment. The lower PCB was adhered to the Hyundai test windshield using the 468-MP adhesive. A plastic enclosure from RadioShack was placed around the lower PCB, and the upper PCB could then be placed in the plastic enclosure to rest about 1 cm above the lower.

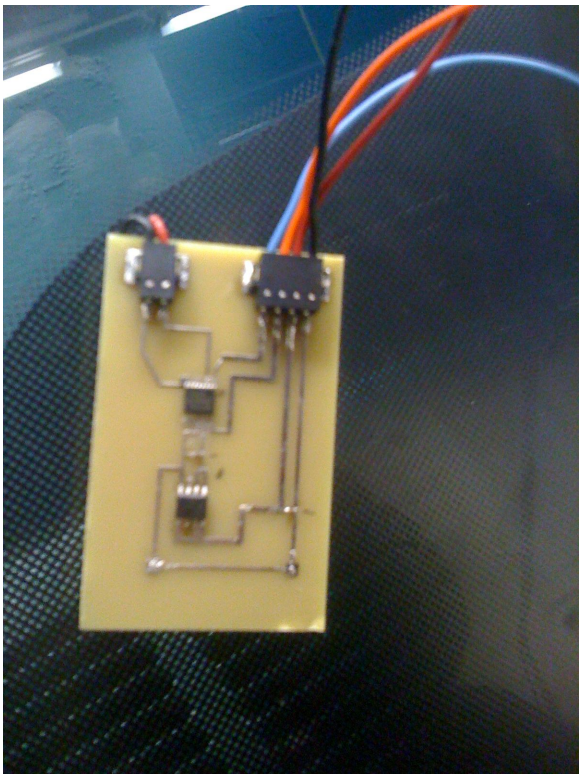


Figure 29: Upper PCB

Because of the new PCB layout and device structure, the capacitance change for each rain sensor setting would change by a small amount. Due to this, new data points had to be taken to program into the PIC for accurate operation. Between 50 – 100 data test points were taken for each rain sensor setting for the final prototypes. A complete compilation of test data can be found in Appendix D. Results indicated that the system was working precisely as designed. The sensor had no difficulty determining rain on the windshield, and could very reliably differentiate between the different rain sensor settings. The Visual Basic program worked flawlessly, displaying moving wipers in response to different stimuli on the windshield.

CHAPTER FIVE: CONCLUSION

5.1. Conclusion

Design Team 6 has developed an accurate and cost-effective capacitive rain sensor system, improving on nearly all of the faults of the current optical sensor. Testing confirms that it can accurately detect varying levels of rain through a 6 mm thick vehicle windshield, and uses this data to turn the wipers on to three different settings depending on the amount of rain present on the windshield. Furthermore, the sensor can differentiate between rain and other objects, such as leaves and human hands, that are placed above the sensor, thus preventing false positives and inappropriate wiper operation. The entire production-level unit is self contained in a compact plastic enclosure mounting near the rear view mirror. This enclosure is smaller than the optical sensor unit, yet still provides a substantially larger sensing area for better detection of sparse rain. The system contains only three integrated circuits and four passive components, thus allowing for efficient assembly, low complexity, and easy repair. Slight refinements will be needed to incorporate the capacitive rain sensor into an actual vehicle, as the control signals to be sent to the vehicle's BCM are not implemented at this time, as this was not a sponsor requirement. However, these control signals can be easily generated by the on-board PIC microcontroller and output to the BCM, and should require little additional design effort. The final estimated production cost of the device is $11.40 per unit, which is a savings of approximately $6.50 per unit over the optical design.

Appendix

Appendix A. Team Member Technical Roles

A.1. Danny Kang – Manager

Danny's technical contributions to the project include assisting with the PCB layout design, performing test data collection, and systems-level research.

Danny worked with Eric Otte in designing the PCB layouts which would be ordered through ExpressPCB. Danny determined that the Flex-PCB, which the team had initially planned on using for the sensor trace layout material, would not be financially feasible. Danny researched the ExpressPCB software and assisted in layout out the design with their PCB layout software. In order to connect the components, each component's layout had to be custom designed. Danny researched and designed these custom component layouts.

Beside to design PCB layout, Danny Kang also supported many different technical portions. Early on in the project, he was talking with Professor Tim Hogan about an initial sensor trace and get idea about to use COMSOL software. He also researched about the components what team need.

Danny also focuses on the communication between sponsors, Jeff Shtogrin and HATCI. When the team needed parts or technical advice from professional engineer, he found part from HATCI and brought from there, and arranged the meeting with specialist.

A.2. Eric Otte – Document Preparation

Eric's technical contribution to the project was in systems-level design, capacitance-to-digital converter research, utilization of the AD7745 including programming the initialization sequence, capacitive sensor trace design research, final PCB design, layout, and device structure, as well as general construction, troubleshooting and data sampling.

During the initial weeks of the project, a general design solution was unclear. Eric took a lead role in researching current information on capacitive sensors, and how they could be utilized in a rain sensing application. Using this research data, Eric proposed the use of a capacitance-to-digital converter (CDC) integrated circuit, along with the use of a microcontroller to process the capacitance readings. Furthermore, Eric analyzed current CDCs on the market and determined that the Analog Devices AD7745 24-bit CDC offered the best performance available, which would be necessary since trying to detect rain through 6 – 8 mm of glass. He also researched standard sensor trace layout designs, which were utilized by Arslan Qaiser when simulating designs in COMSOL.

After obtaining the AD7745 and PIC microcontrollers, Eric designed the initialization sequence for the AD7745 in C programming code, which programmed various registers of the AD7745 to configure it into the proper operating mode. He also designed the basic process of the PIC polling the status register of the AD7745 to determine when data was ready and could be transferred.

Eric was responsible for the PCB layout design of the boards ordered through ExpressPCB. He first created schematic diagrams illustrating pin connections for the components, and then used this to layout the board using ExpressPCB PCB layout software. He determined that since the ExpressPCB Miniboard service required 3.8 x 2.5" PCB's, the layout could be divided into two sections and cut in half upon receiving the boards, and the PCBs stacked in an enclosure for the final prototype. Additionally, Eric worked extensively in constructing and testing all prototype devices.

A.3. Ishaan Sandhu – Presentation Preparation

As designated in the proposal Ishaan Sandhu was responsible for microcontroller integration. His main focus was to allow the microcontroller to perform the detection of the capacitance change when an object interfered with the fringing fields of the capacitive sensor and writing necessary code to distinguish those changes from each other.

Ishaan wrote a microcontroller program using MPLAB IDE for our microcontroller and the GUI program necessary to detect the change in capacitance and output the

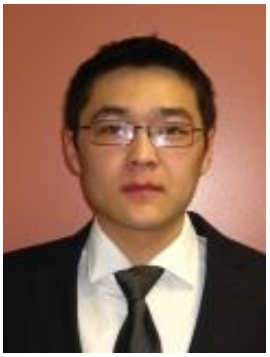
different levels of output for different objects being introduced to the sensor in Microsoft Visual Studio. He also created the graph that outputs the change in capacitance throughout predefined time intervals along with the animation for our output sequence. For example if you have the water pattern 'mist' the windshield wipers animation will move slow. Conversely, if you have 'downpour' the wipers will move very fast in order to clear the rain from the windshield.

The biggest progress to the project was made when the team actually got their final design of the sensor fabricated. Before assembling the final design Mr. Sandhu , Mr. Tazabekov and Mr. Qaiser did some verification testing, and after the results matched the COMSOL predicted base capacitance they started assembling the final design of the project. They build the circuit for driving the rain sensor and created the adequate casing for it, in order to prevent it from being accidentally damaged. In the end they mounted the sensor to the windshield and verified that sensor works before they start their final testing.

During final testing, Mr. Qaiser, Mr. Tazabekov and Mr. Sandhu tested 3 different rain patterns: mist, rain and downpour, and also figured out the range of change in capacitance for the 'leaf' case as well as for 'finger/hand' case. They took 100 data points for each case in order to make sure that the range is correct.

As Mr. Sandhu was responsible for the microcontroller integration for the final design, he had to correct some of the values of the rain patterns. The final testing included five different cases, compared to four at the first testing, so the final code had to be modified, based on the final testing results.

A.4. Anuar Tazabekov – Webmaster



As designated in the proposal Mr. Anuar Tazabekov was responsible for identifying and providing the required power supply to the whole system, and also to teams's microprocessor and capacitance-to-digital circuits, as well as maintaining the suitable current and voltage supply to keep the system from overheating. As the design process kept evolving and the first capacitance sensor was build in the ECE Shop, Mr. Tazabekov and Mr. Qaiser were the people

responsible for testing the design, and making sure that the concept is actually working. They took hundreds of data points and developed a range of change in capacitance that will be matching a certain rain case.

The biggest progress to the project was made when the team actually got our final design of the sensor fabricated. Before assembling the final design Mr. Tazabekov, Mr. Qaiser and Mr. Sandhu did some verification testing, and after the results matched the COMSOL predicted capacitance they started assembling the final design of the project. They build the circuit for driving the rain sensor and created the adequate casing for it, in order to prevent it from being accidentally damaged. In the end they mounted the sensor to the windshield and verified that sensor works before they start their final testing.

At the final testing, Mr. Tazabekov, Mr. Qaiser and Mr. Sandhu tested 3 different rain patterns: mist, rain and downpour, and also figured out the range of change in capacitance for the "leaf" case as well as for "finger/hand" case. They took 100 data points for each case in order to make sure that the range is correct.

As Mr. Tazabekov was responsible for the power supply for microcontroller and capacitance-to-digital controller he wanted to make sure that our circuit matched the real car scenario, where the whole car system is being driven by the 12 volt car battery. He had to step down the supplied 12 volts to the 5 volts, since the microcontroller and capacitance-to-digital circuits required 5 volt power supply. In order to perform that function, the voltage regulator circuit was built. After that checks for overheating were made, and as a result it was obvious that the circuit will not overheat.

A.5. Arslan Qaiser – Lab Coordinator

Arslan Qaiser was mainly responsible for the design and testing of the capacitive sensor interface. Along with fellow teammate Ishaan Sandhu, he was involved in the building and testing of the initial prototype. Once the initial prototype was complete, his task was to optimize the capacitive sensor and increase the sensitivity. This was an integral part of the project because the sensor needed to

be sensitive enough to detect and distinguish between various levels of water and other objects like leaf and hand. In order to achieve this goal, Arslan used a 3D modeling software called COMSOL Multiphysics which is capable of implementing a capacitive sensor in a visually friendly user interface.

Arslan helped design the initial prototype by using Eagle PCB Designer with the help of Ishaan Sandhu. This was the first attempt at the design of the capacitive sensor and the purpose was to prove that the capacitive sensor can sense change in capacitance. Using an LCR meter, it was proved that the capacitive sensor works.

In order to optimize the capacitive sensor design, Arslan used the 3D modeling software, COMSOL. He spent several weeks researching and understanding the software. There was not enough information on the internet so Arslan had to use different examples provided with the software to get a general idea of how to design a capacitive sensor. The software was intimidating at the beginning but once the design was complete, it was well worth the time and effort. He was able to optimize the design and proved that it was more sensitive compared to the initial design.

Once the final sensor design was fabricated, Arslan along with Ishaan Sandhu and Anuar Tazabekov helped put the final prototype together. This involved adhere the sensor to the windshield, connect the sensor to the microcontroller interface, and make an enclosure for the sensor and for the final prototype.

In addition to these contributions, Arslan worked together with the team effectively and was heavily involved in testing the final prototype. He spent several days testing the sensor and made sure that there is no false reading that would trigger the sensor. Each test case; mist, rain, downpour, leaf and hand required very accurate measurement. To do this, a hundred test points were taken for each case and then averaged to give a fixed capacitance range for each test case.

Appendix B. Code

B.1. Microcontroller Code (Link.c)

```
/*****************************************************************
 *  Name:    ECE 480 - Team 6
     *
 *  What:    Link.c
           *
 *  When:    04/27/10
     *
 *  Purpose: This file allows our Microcontroller to communicate *
 *                with our VB App in order to display the capacitance *
 *                from our AD7745 as well as the corresponding action *
 *****************************************************************/

#include <p18cxxx.h>                        //Include file for PIC18F4520
#include <usart.h>                                //Include file for serial
communication
#include <i2c.h>                            //Include file for I2C communication
#pragma config LVP=OFF
#pragma config WDT=OFF
void rx_handler (void);                     //Declare the ISR function (interrupts)
void initialize (void);                     //Declare the Initialize subroutine for
the C-D Converter
void get_data (void);                       //Declare the retrieve data subroutine
for the C-D Converter

long int loop;

unsigned char StatusReg = '0xFF';   // Creates a byte for use in analyzing the
status register of the AD7745
unsigned char CapDacValue = '0x00'; // Temporary byte for reading the CAPDAC value
to ensure proper operation

unsigned char CapHIGH = '0x00';     // Byte to receive MSB byte capacitive data
unsigned char CapMED = '0x00';            // Byte to receive the next (middle)
capacitive data
unsigned char CapLOW = '0x00';            // Byte to receive the last (LSB byte)
capacitive data

unsigned char VtHIGH = '0x00';            // Byte to receive MSB byte capacitive
data
unsigned char VtMED = '0x00';             // Byte to receive the next (middle)
capacitive data
unsigned char VtLOW = '0x00';             // Byte to receive the last (LSB byte)
capacitive data

/*-------------------------------------------------------------------
    Name:    main()
    Input:   none
    Output:  none
    Purpose: Give the program a starting point
  -------------------------------------------------------------------*/
void main()
{
      TRISC = 0x00;                               //turn on tri-state register and
make all output pins
      PORTC = 0x00;                               //make all output pins LOW
```

```
      OpenI2C( MASTER, SLEW_OFF);         //Open the I2C connection and allow for
communication
      SSPADD = 0x3F;                            //Configuration register based on
10 MHz Clock

      //Initialize the Serial communication interface
      OpenUSART (USART_TX_INT_OFF & USART_RX_INT_ON &
                  USART_ASYNCH_MODE & USART_EIGHT_BIT &
                  USART_CONT_RX & USART_BRGH_LOW, 63);
      RCONbits.IPEN = 1;                        // Enable interrupt priority
      IPR1bits.RCIP = 1;                        // Make receive interrupt high
priority
      INTCONbits.GIEH = 1;                // Enable all high priority interrupts
      ADCON1 =0x00;                             //set VREF+ to VDD and VREF- to GND
(VSS)
      TRISD = 0x04;
      PORTDbits.RD0 = 0;                        //GREEN
      PORTDbits.RD1 = 0;                        //YELLOW
      PORTDbits.RD3 = 0;                        //RED
      while(1);
}

/*-------------------------------------------------------------------
    Name:    rx_int()
    Input:   void
    Output:  none
    Purpose: Default interrupt code
  -------------------------------------------------------------------*/
#pragma code rx_interrupt = 0x8
void rx_int (void)
{
      _asm goto rx_handler _endasm
}
#pragma code
#pragma interrupt rx_handler

/*-------------------------------------------------------------------
    Name:    rx_handler()
    Input:   void
    Output:  none
    Purpose: Handle all interupts from the VB App
  -------------------------------------------------------------------*/
void rx_handler (void)
{
      unsigned char cc;                   //Temporary variable to handle interrupts
      cc = getcUSART();                         //get a single character off the
USART line
      while(BusyUSART());
      PORTDbits.RD1 = 1;
      if(cc == 'i')                             //The VB App sends 'i' over the
Serial line to
      {                                               //initialize the AD7745
            PORTDbits.RD0 = 1;
            initialize();                       //Initialize the AD7745 (also works
as a reset)
            while(BusyUSART());
            putcUSART (CapHIGH);          //put the Highest 2/6 charachters on the
USART line
            while(BusyUSART());
```

```
            putcUSART (CapMED);           //put the Middle 2/6 charachters on the
USART line
            while(BusyUSART());
            putcUSART (CapLOW);           //put the Lowest 2/6 charachters onthe
USART line
            while(BusyUSART());
      }
      else if(cc = 'g')
      {
            get_data();                         //Gets the capacitance and
temperature data from the AD7745
            while(BusyUSART());
            putcUSART (StatusReg);        //put a single character on the USART
line
            while(BusyUSART());
            putcUSART (CapHIGH);          //put a single character on the USART
line
            while(BusyUSART());
            putcUSART (CapMED);           //put a single character on the USART
line
            while(BusyUSART());
            putcUSART (CapLOW);           //put a single character on the USART
line
            while(BusyUSART());
            putcUSART (VtHIGH);           //put a single character on the USART
line
            while(BusyUSART());
            putcUSART (VtMED);                  //put a single character on the
USART line
            while(BusyUSART());
            putcUSART (VtLOW);                  //put a single character on the
USART line
            while(BusyUSART());
      }
      PORTDbits.RD1 = 0;
}

/*-------------------------------------------------------------------
    Name:    initialize()
    Input:   void
    Output:  none
    Purpose: Initializes the AD7745 and get the base capacitance
                  value to be sent to the Visual Basic program for
                  interpretation
  -------------------------------------------------------------------*/
void initialize (void)
{
      StartI2C();                               // Start Sequence
            IdleI2C();
      WriteI2C(0x90);                     // AD7745 Address + WRITE
            IdleI2C();
      WriteI2C(0xBF);                     // RESET COMMAND TO AD7745
            IdleI2C();
      StopI2C();                                // Stop Sequence

      // Delays 210 us after the AD7745 Reset
      for (loop=0x001; loop<0x834; loop++);

      StartI2C();
```

```
            IdleI2C();
      WriteI2C(0x90);
            IdleI2C();
      WriteI2C(0x09);                  // Sets ADDRESS POINTER REGISTER of
AD7745 to EXC SETUP REG
            IdleI2C();
      WriteI2C(0x4B);                  // Sets EXC SETUP REG to use amplitude +-
Vdd/2, EXCON=1, and use EXCA
            IdleI2C();
      StopI2C();

      StartI2C();
            IdleI2C();
      WriteI2C(0x90);
            IdleI2C();
      WriteI2C(0x07);                  // Sets ADDRESS POINTER REGISTER of
AD7745 to CAP SETUP REG
            IdleI2C();
      WriteI2C(0x80);                  // Sets CAP SETUP REG to enable
capacitive channel 1 in single-ended mode. CHOP disabled
            IdleI2C();
      StopI2C();

      StartI2C();
            IdleI2C();
      WriteI2C(0x90);
            IdleI2C();
      WriteI2C(0x08);                  // Sets ADDRESS POINTER REGISTER of
AD7745 to VOLTAGE SETUP REG
            IdleI2C();
      WriteI2C(0x81);                  // Sets VOLTAGE SETUP REG to enable
capacitive channel 1 in single-ended mode. CHOP disabled
            IdleI2C();
      StopI2C();

      StartI2C();
            IdleI2C();
      WriteI2C(0x90);
            IdleI2C();
      WriteI2C(0x0B);                  // Sets ADDRESS POINTER REGISTER of
AD7745 to CAP DAC A REG
            IdleI2C();
      WriteI2C(0xDA);                  // Sets CAP DAC A REG to full value of
capacitance ~16 pF
            IdleI2C();
      StopI2C();

      StartI2C();
            IdleI2C();
      WriteI2C(0x90);
            IdleI2C();
      WriteI2C(0x0A);                  // Sets ADDRESS POINTER REGISTER of
AD7745 to CONFIGURATION REG
            IdleI2C();
      WriteI2C(0xA1);                  // Sets CONFIGURATION REG to enable
CONTINUOUS CONVERSION at 62 ms conversion time
            IdleI2C();
      StopI2C();
```

```
      while(StatusReg!=0x00) get_data();
}

/*-------------------------------------------------------------------
    Name:    get_data()
    Input:   void
    Output:  none
    Purpose: Get the capacitance and temperature data from the
                  AD7745 Chip to be sent to the Visual Basic Program
  -------------------------------------------------------------------*/
void get_data (void)
{
      StartI2C();                                 //---------- THIS SEQUENCE POLLS
THE STATUS REGISTER
            IdleI2C();
      WriteI2C(0x90);                     // AD7745 Address + WRITE
            IdleI2C();
      WriteI2C(0x00);                     // Sets ADDRESS POINTER REGISTER of
AD7745 to STATUS REG
            IdleI2C();
      RestartI2C();                             // Restart command to begin reading
            IdleI2C();
      WriteI2C(0x91);                     // AD7745 Address + READ
            IdleI2C();
      StatusReg = ReadI2C();              // Reads the STATUS REG value and stores
it in char "StatusReg"
            IdleI2C();
      NotAckI2C();                              // Since reading only ONE BYTE, NO
ACKNOWLEDGE is sent back to prevent auto-increment of address pointer
            IdleI2C();
      StopI2C();
      for(loop = 1; loop < 4000; loop++);
      if (StatusReg>0x07 && StatusReg<0x10)
      {
            // If the EXCERR bit goes high in the Status Register, the EXC cannot
be driven
            // We want to then stop looping and perhaps turn on a red LED to
indicate an error
            PORTDbits.RD3 = 1;      //RED
      }
      else if ((StatusReg == 0x00)||(StatusReg == 0x01)||(StatusReg == 0x02)) //
Capacitive data is ready to be read
      {
            if(StatusReg == 0x00)
            {
                  StartI2C();
                        IdleI2C();
                  WriteI2C(0x90);         // AD7745 Address + WRITE
                        IdleI2C();
                  WriteI2C(0x01);         // Sets ADDRESS POINTER REGISTER of
AD7745 to CAP DATA REGISTER 1
                        IdleI2C();
                  RestartI2C();
                        IdleI2C();
                  WriteI2C(0x91);         // AD7745 Address + READ
                        IdleI2C();
                  CapHIGH = ReadI2C();
                        IdleI2C();
                  AckI2C();
```

```
                IdleI2C();
            CapMED = ReadI2C();
                IdleI2C();
            AckI2C();
                IdleI2C();
            CapLOW = ReadI2C();
                IdleI2C();
            AckI2C();
                IdleI2C();
            VtHIGH = ReadI2C();
                IdleI2C();
            AckI2C();
                IdleI2C();
            VtMED = ReadI2C();
                IdleI2C();
            AckI2C();
                IdleI2C();
            VtLOW = ReadI2C();
                IdleI2C();
            NotAckI2C();
                IdleI2C();
            StopI2C();
        }
        else if(StatusReg == 0x01)
        {
            StartI2C();
                IdleI2C();
            WriteI2C(0x90);         // AD7745 Address + WRITE
                IdleI2C();
            WriteI2C(0x04);         // Sets ADDRESS POINTER REGISTER of
AD7745 to CAP DATA REGISTER 1
                IdleI2C();
            RestartI2C();
                IdleI2C();
            WriteI2C(0x91);         // AD7745 Address + READ
                IdleI2C();
            VtHIGH = ReadI2C();
                IdleI2C();
            AckI2C();
                IdleI2C();
            VtMED = ReadI2C();
                IdleI2C();
            AckI2C();
                IdleI2C();
            VtLOW = ReadI2C();
                IdleI2C();
            NotAckI2C();
                IdleI2C();
            StopI2C();
        }
        else if(StatusReg == 0x02)
        {
            StartI2C();
                IdleI2C();
            WriteI2C(0x90);         // AD7745 Address + WRITE
                IdleI2C();
            WriteI2C(0x01);         // Sets ADDRESS POINTER REGISTER of
AD7745 to CAP DATA REGISTER 1
                IdleI2C();
```

```
                RestartI2C();
                      IdleI2C();
                WriteI2C(0x91);         // AD7745 Address + READ
                      IdleI2C();
                CapHIGH = ReadI2C();
                      IdleI2C();
                AckI2C();
                      IdleI2C();
                CapMED = ReadI2C();
                      IdleI2C();
                AckI2C();
                      IdleI2C();
                CapLOW = ReadI2C();
                      IdleI2C();
                NotAckI2C();
                      IdleI2C();
                StopI2C();
            }
        }
}
```

B.2. Visual Basic Application Code (Login.vb)

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

    Private Sub Login_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    End Sub

    Private Sub ButtonLogin_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonLogin.Click
        If TextBoxUsername.Text = "ece480.team6" And TextBoxPassword.Text =
"rainsensor" Then
            Main.Show()
            Me.Hide()
        End If
    End Sub

    Private Sub ButtonCancel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonCancel.Click
        Me.Close()
    End Sub
End Class
```

B.3. Visual Basic Application Code (Main.vb)

```
Public Class Main
    Inherits System.Windows.Forms.Form
    Dim setRs232 As New Rs232
    Dim serial_in As String
    Dim x As Integer
    Dim a, b As Integer
    Dim HexVal As String
    Dim start As DateTime
    Dim time As TimeSpan
    Dim BaseCapacitance As Decimal
```

```
    Declare Sub Sleep Lib "kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)

    Private Sub Main_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        With setRs232
            .Port = 3
            .BaudRate = 2400
            .DataBit = 8
            .StopBit = Rs232.DataStopBit.StopBit_1
            .Parity = Rs232.DataParity.Parity_None
            .Timeout = 10000
        End With
        setRs232.Open()
        ButtonReset_Click(sender, e)
        'Timer1.Enabled = True
        a = 0
        b = 0
    End Sub

    Private Sub ButtonStart_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonStart.Click
        ChartCapGraph.Series("Series1").Points.Clear()
        ChartCapGraph.Legend.Enabled = False
        ChartCapGraph.ChartAreas("Default").AxisX.Title = "Time (s)"
        ChartCapGraph.ChartAreas("Default").AxisY.Title = "Capacitance (fF)"
        Dim Z As Integer
        Dim Max As Decimal
        Dim Min As Decimal
        Dim Capacitance As Decimal
        Dim Temperature As Decimal

        start = DateTime.Now
        For Z = 1 To (Val(TextBoxTimer.Text) * 11.111111)
            setRs232.Write("g")

            setRs232.Read(1)
            serial_in = setRs232.InputStreamString
            x = Asc(CChar(serial_in))
            If (x < 16) Then
                HexVal = "0" + Hex$(x)
            Else
                HexVal = Hex$(x)
            End If
            TextBoxStatus.Text = "0x" + HexVal

            setRs232.Read(1)
            serial_in = setRs232.InputStreamString
            x = Asc(CChar(serial_in))
            If (x < 16) Then
                HexVal = "0" + Hex$(x)
            Else
                HexVal = Hex$(x)
            End If

            setRs232.Read(1)
            serial_in = setRs232.InputStreamString
            x = Asc(CChar(serial_in))
            If (x < 16) Then
                HexVal = HexVal + "0" + Hex$(x)
```

```
Else
    HexVal = HexVal + Hex$(x)
End If

setRs232.Read(1)
serial_in = setRs232.InputStreamString
x = Asc(CChar(serial_in))
If (x < 16) Then
    HexVal = HexVal + "0" + Hex$(x)
Else
    HexVal = HexVal + Hex$(x)
End If
TextBoxHex.Text = "0x" + HexVal

Capacitance = CDec("&H" + HexVal) * (0.0000004882812791) - 4.096

setRs232.Read(1)
serial_in = setRs232.InputStreamString
x = Asc(CChar(serial_in))
If (x < 16) Then
    HexVal = "0" + Hex$(x)
Else
    HexVal = Hex$(x)
End If

setRs232.Read(1)
serial_in = setRs232.InputStreamString
x = Asc(CChar(serial_in))
If (x < 16) Then
    HexVal = HexVal + "0" + Hex$(x)
Else
    HexVal = HexVal + Hex$(x)
End If

setRs232.Read(1)
serial_in = setRs232.InputStreamString
x = Asc(CChar(serial_in))
If (x < 16) Then
    HexVal = HexVal + "0" + Hex$(x)
Else
    HexVal = HexVal + Hex$(x)
End If
TextBoxHex2.Text = "0x" + HexVal

Temperature = CDec("&H" + HexVal) / 2048 - 4096
Temperature = Math.Round(Temperature, 2)
TextBoxTemperature.Text = Temperature.ToString + " (C)"

Capacitance = Math.Round(Capacitance, 5)
If (Z = 1) Then
    Max = Capacitance
    Min = Capacitance
End If
If (Capacitance > Max) Then
    Max = Capacitance
End If
If (Capacitance < Min) Then
    Min = Capacitance
End If
```

```
            TextBoxCapacitance.Text = (Capacitance * 1000).ToString + " (fF)"
            time = DateTime.Now.Subtract(start)
            ChartCapGraph.Series("Series1").Points.AddXY(time.TotalSeconds,
Capacitance * 1000)
            TextBoxCapacitance.Refresh()
            TextBoxHex.Refresh()
            TextBoxStatus.Refresh()
            TextBoxHex2.Refresh()
            TextBoxTemperature.Refresh()

            If ((Math.Abs(Capacitance - BaseCapacitance) * 1000) <= 4) Then
                TextBoxWhat.Text = "None"
                Timer1.Enabled = False
                PictureBoxWiper.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\1.jpg"
                PictureBoxWiper.Refresh()
                PictureBoxMist.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxRain.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxDownpour.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxLeaf.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxFinger.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxError.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
            ElseIf ((Math.Abs(Capacitance - BaseCapacitance) * 1000) > 4) And
((Math.Abs(Capacitance - BaseCapacitance) * 1000) <= 15) Then
                TextBoxWhat.Text = "Mist"
                Timer1.Enabled = True
                Timer1.Interval = 60
                PictureBoxMist.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\a.jpg"
                PictureBoxRain.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxDownpour.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxLeaf.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxFinger.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxError.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
            ElseIf ((Math.Abs(Capacitance - BaseCapacitance) * 1000) > 15) And
((Math.Abs(Capacitance - BaseCapacitance) * 1000) <= 200) Then
                TextBoxWhat.Text = "Rain"
                Timer1.Enabled = True
                Timer1.Interval = 30
                PictureBoxMist.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\a.jpg"
                PictureBoxRain.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\a.jpg"
                PictureBoxDownpour.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxLeaf.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxFinger.ImageLocation = "C:\Documents and
```

```
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxError.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
            ElseIf ((Math.Abs(Capacitance - BaseCapacitance) * 1000) > 200) And
((Math.Abs(Capacitance - BaseCapacitance) * 1000) <= 600) Then
                TextBoxWhat.Text = "Downpour"
                Timer1.Enabled = True
                Timer1.Interval = 1
                PictureBoxMist.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\a.jpg"
                PictureBoxRain.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\a.jpg"
                PictureBoxDownpour.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\b.jpg"
                PictureBoxLeaf.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxFinger.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxError.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
            ElseIf ((Math.Abs(Capacitance - BaseCapacitance) * 1000) > 600) And
((Math.Abs(Capacitance - BaseCapacitance) * 1000) <= 1200) Then
                TextBoxWhat.Text = "Leaf"

                Timer1.Enabled = True
                Timer1.Interval = 1
                PictureBoxMist.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\a.jpg"
                PictureBoxRain.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\a.jpg"
                PictureBoxDownpour.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\b.jpg"
                PictureBoxLeaf.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\b.jpg"
                PictureBoxFinger.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
                PictureBoxError.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
            ElseIf ((Math.Abs(Capacitance - BaseCapacitance) * 1000) > 1200) Then
                TextBoxWhat.Text = "Finger/Hand"
                Timer1.Enabled = False
                PictureBoxWiper.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\1.jpg"
                PictureBoxWiper.Refresh()
                PictureBoxMist.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\a.jpg"
                PictureBoxRain.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\a.jpg"
                PictureBoxDownpour.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\b.jpg"
                PictureBoxLeaf.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\b.jpg"
                PictureBoxFinger.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\c.jpg"
                PictureBoxError.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\d.jpg"
            Else
                TextBoxWhat.Text = "Error"
                PictureBoxMist.ImageLocation = "C:\Documents and
```

```
Settings\admin\Desktop\Final\App\App\Pics\a.jpg"
                PictureBoxRain.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\a.jpg"
                PictureBoxDownpour.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\b.jpg"
                PictureBoxLeaf.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\b.jpg"
                PictureBoxFinger.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\c.jpg"
                PictureBoxError.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\c.jpg"
            End If

            TextBoxDelta.Text = Math.Round((Math.Abs(Capacitance - BaseCapacitance)
* 1000), 5).ToString() + " (fF)"
            TextBoxWiperSpeed.Text = Timer1.Interval
            TextBoxWiperSpeed.Refresh()
            TextBoxWhat.Refresh()
            TextBoxDelta.Refresh()
            PictureBoxMist.Refresh()
            PictureBoxRain.Refresh()
            PictureBoxDownpour.Refresh()
            PictureBoxLeaf.Refresh()
            PictureBoxFinger.Refresh()
            PictureBoxError.Refresh()
            Sleep(1)
        Next

        ChartCapGraph.ChartAreas("Default").AxisY.Minimum = Min * 1000
        ChartCapGraph.ChartAreas("Default").AxisY.Maximum = Max * 1000
        ChartCapGraph.ChartAreas("Default").AxisX.Minimum = 0
        ChartCapGraph.ChartAreas("Default").AxisX.Maximum = time.TotalSeconds


    End Sub

    Private Sub ButtonReset_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonReset.Click

        setRs232.Write("i") 'Used to initialize the C-D Converter

        TextBoxCapacitance.Clear()
        TextBoxHex.Clear()
        TextBoxStatus.Clear()
        setRs232.Read(1)
        serial_in = setRs232.InputStreamString
        x = Asc(CChar(serial_in))
        If (x < 16) Then
            HexVal = "0" + Hex$(x)
        Else
            HexVal = Hex$(x)
        End If

        setRs232.Read(1)
        serial_in = setRs232.InputStreamString
        x = Asc(CChar(serial_in))
        If (x < 16) Then
            HexVal = HexVal + "0" + Hex$(x)
        Else
```

```
            HexVal = HexVal + Hex$(x)
        End If

        setRs232.Read(1)
        serial_in = setRs232.InputStreamString
        x = Asc(CChar(serial_in))
        If (x < 16) Then
            HexVal = HexVal + "0" + Hex$(x)
        Else
            HexVal = HexVal + Hex$(x)
        End If

        BaseCapacitance = CDec("&H" + HexVal) * (0.0000004882812791) - 4.096
        Dim TempInt As Integer
        TempInt = Math.Round(BaseCapacitance * 1000, 5)
        TextBoxBaseCapacitance.Text = (TempInt).ToString + " (fF)"
        TextBoxBaseCapacitance.Refresh()
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        If b = 0 Then a = a + 1
        If b = 1 Then a = a - 1
        If a = 37 Then b = 1
        If a = 1 Then b = 0
        PictureBoxWiper.ImageLocation = "C:\Documents and
Settings\admin\Desktop\Final\App\App\Pics\" + a.ToString + ".jpg"
        PictureBoxWiper.Refresh()
    End Sub
End Class
```

B.3. Visual Basic Application Code (CRs232.vb)

```
Imports System.Runtime.InteropServices
Imports System.Text
Imports System.Threading
Imports System.ComponentModel
Imports System.IO



#Region "RS232"
Public Class Rs232 : Implements IDisposable
   '=================================================
   '
   '  Module                        :     Rs232
   '  Description       :     Class for handling RS232 comunication with VB.Net
   '  Created                       :     10/08/2001 - 8:45:25
   '  Author                        :     Corrado Cavalli (corrado@mvps.org)
    'WebSite                        :     www.codeworks.it/net/index.htm
   '
   '  Notes                         :
   '-------------------------------------------------------------------------------
---------------
      '
                        *     Revisions *
      '
```

```
      '     02/12/2000       First internal alpha version built on framework beta1
      '
   '  1st Public release Beta2 (10/08/2001)
   '
   '  Rev.1 (28.02.2002)
   '  1.    Added ResetDev, SetBreak and ClearBreak to the EscapeCommFunction
constants
   '  2.    Added the overloaded Open routine.
   '  3.    Added the modem status routines, properties and enum.
   '  4.    If a read times out, it now returns a EndOfStreamException (instead of
a simple Exception).
   '  5.Compiled with VS.Net final

   '  Rev.2 (01.03.2002)
   '  Added Async support
   '
   '  Rev.3 (07.04.2002)
   '  Minor bugs fixed
   '
   '  Rev.3 (05/05/2002)
   '  Fixed BuildCommmDCB problem
   '
   '  Rev.4 (24/05/2002)
   '  Fixed problem with ASCII Encoding truncating 8th bit
   '
   '  Rev.5 (27/05/2002)
   '  Added IDisposable / Finalize implementation
   '
   '  Rev.6 (14/03/2003)
   '  Fixed problem on DCB fields Initialization
   '
   '  Rev.7 (26/03/2003)
   '  Added XON/XOFF support
   '
   '  Rev.8 (12/07/2003)
      '  Added support to COM port number greater than 4
      '
      '     Rev.9 (15/07/2003)
      '     Added CommEvent to detect incoming chars/events
      '     Updated both Tx/Rx method from Non-Ovelapped to Overlapped mode
      '     Removed unused Async methods and other stuff.
      '
      '     Rev.10 (21/07/2003)
    '    Fixed incorrect character handling when using EnableEvents()
   '
    '   Rev.11 (12/08/2003)
    '   Fixed some bugs signaled by users
   '
      '     Rev.12 (01/09/2003)
      '     Removed AutoReset of internal buffers and added PurgeBuffer() method
      '
      '     Rev.13 (02/09/2003)
      '     Removed GetLastErrorUse in favour of Win32Exception()
      '
      '     Rev.14 (14/09/2003)
      '     Added IsPortAvailable() function
      '  Revised some API declaration
    '  Fixed problem with Win98/Me OS
   '
```

```
 '   Rev.15 (24/09/2003)
 '       Fixed bug introduced on Rev.14
'
 '   Rev.16 (12/10/2003)
 ' Added SetBreak/ClearBreak() methods
'
 '   Rev.17 (02/11/2003)
 '   Fixed field on COMMCONFIG
   '
   '   Rev.18 (03/03/2004)
   '   Fixed bug: Testing mhRS for <>0 is not correct
   '
   '   Rev.19 (08/04/2004)
   '   Fixed bug: Fixed bug on DTR property
   '
   '   Rev.20 (12/07/2004)
   '   CommEvent is no more raised on a secondary thread
 ' pEventsWatcher now uses a background thread
   '
   '   Rev.21 (24/10/2004)
   '   EscapeCommFunction declaration fixed
   '     Pariti enum fixed to Parity
   '
   '     Rev. 22 (05/03/2005)
 ' Fixed memory leak problem causing program closing
 ' without any message on some systems.
 ' Thanks to Ralf Gedrat for testing this scenario
 '
 '   Rev.23 (05/04/2005)
 '   Fixed bug DisableEvents not working bug
 '
 '   Rev.24 (20/04/2005)
 '   Fixed memory leak on Read method
 '   Added InBufferCount property
 '   IsPortAvailable method is now shared
 '   Thanks to Jean-Pierre ZANIER for the feedback

 '=====================================================
'// Class Members
   Private mhRS As IntPtr = New IntPtr(0)     '// Handle to Com Port

Private miPort As Integer = 1   '//  Default is COM1
Private miTimeout As Int32 = 70   '// Timeout in ms
Private miBaudRate As Int32 = 9600
Private meParity As DataParity = 0
Private meStopBit As DataStopBit = 0
Private miDataBit As Int32 = 8
Private miBufferSize As Int32 = 512   '// Buffers size default to 512 bytes
Private mabtRxBuf As Byte()   '//  Receive buffer
Private meMode As Mode  '//  Class working mode
   Private moThreadTx As Thread
   Private moThreadRx As Thread
   Private moEvents As Thread
Private miTmpBytes2Read As Int32
Private meMask As EventMasks
Private mbDisposed As Boolean
   Private mbUseXonXoff As Boolean
   Private mbEnableEvents As Boolean
   Private miBufThreshold As Int32 = 1
```

```
      Private muOvlE As OVERLAPPED
      Private muOvlW As OVERLAPPED
      Private muOvlR As OVERLAPPED
      Private mHE As GCHandle
      Private mHR As GCHandle
      Private mHW As GCHandle
      '-------------------------------------------------------------------------------
------------

#Region "Enums"
      '// Parity Data
      Public Enum DataParity
            Parity_None = 0
        Parity_Odd
            Parity_Even
            Parity_Mark
      End Enum
      '// StopBit Data
      Public Enum DataStopBit
            StopBit_1 = 1
            StopBit_2
      End Enum
      <Flags()> Public Enum PurgeBuffers
            RXAbort = &H2
            RXClear = &H8
            TxAbort = &H1
            TxClear = &H4
      End Enum
      Private Enum Lines
            SetRts = 3
            ClearRts = 4
            SetDtr = 5
            ClearDtr = 6
            ResetDev = 7            '    // Reset device if possible
            SetBreak = 8            '    // Set the device break line.
            ClearBreak = 9          '    // Clear the device break line.
      End Enum
      '// Modem Status
      <Flags()> Public Enum ModemStatusBits
            ClearToSendOn = &H10
            DataSetReadyOn = &H20
            RingIndicatorOn = &H40
            CarrierDetect = &H80
      End Enum
      '// Working mode
      Public Enum Mode
            NonOverlapped
            Overlapped
      End Enum
      '// Comm Masks
      <Flags()> Public Enum EventMasks
            RxChar = &H1
            RXFlag = &H2
            TxBufferEmpty = &H4
            ClearToSend = &H8
            DataSetReady = &H10
            CarrierDetect = &H20
            Break = &H40
            StatusError = &H80
```

```
            Ring = &H100
        End Enum

#End Region
#Region "Structures"
        <StructLayout(LayoutKind.Sequential, Pack:=1)> Private Structure DCB
            Public DCBlength As Int32
            Public BaudRate As Int32
            Public Bits1 As Int32
            Public wReserved As Int16
            Public XonLim As Int16
            Public XoffLim As Int16
            Public ByteSize As Byte
            Public Parity As Byte
            Public StopBits As Byte
            Public XonChar As Char
            Public XoffChar As Char
            Public ErrorChar As Char
            Public EofChar As Char
            Public EvtChar As Char
            Public wReserved2 As Int16
        End Structure
        <StructLayout(LayoutKind.Sequential, Pack:=1)> Private Structure COMMTIMEOUTS
            Public ReadIntervalTimeout As Int32
            Public ReadTotalTimeoutMultiplier As Int32
            Public ReadTotalTimeoutConstant As Int32
            Public WriteTotalTimeoutMultiplier As Int32
            Public WriteTotalTimeoutConstant As Int32
        End Structure
    <StructLayout(LayoutKind.Sequential, Pack:=8)> Private Structure COMMCONFIG
        Public dwSize As Int32
        Public wVersion As Int16
        Public wReserved As Int16
        Public dcbx As DCB
        Public dwProviderSubType As Int32
        Public dwProviderOffset As Int32
        Public dwProviderSize As Int32
        Public wcProviderData As Int16
    End Structure
    <StructLayout(LayoutKind.Sequential, Pack:=1)> Public Structure OVERLAPPED
        Public Internal As Int32
        Public InternalHigh As Int32
        Public Offset As Int32
        Public OffsetHigh As Int32
        Public hEvent As IntPtr
    End Structure
    <StructLayout(LayoutKind.Sequential, Pack:=1)> Private Structure COMSTAT
        Dim fBitFields As Int32
        Dim cbInQue As Int32
        Dim cbOutQue As Int32
    End Structure

#End Region
#Region "Constants"
        Private Const PURGE_RXABORT As Integer = &H2
        Private Const PURGE_RXCLEAR As Integer = &H8
        Private Const PURGE_TXABORT As Integer = &H1
        Private Const PURGE_TXCLEAR As Integer = &H4
        Private Const GENERIC_READ As Integer = &H80000000
```

```
      Private Const GENERIC_WRITE As Integer = &H40000000
      Private Const OPEN_EXISTING As Integer = 3
      Private Const INVALID_HANDLE_VALUE As Integer = -1
      Private Const IO_BUFFER_SIZE As Integer = 1024
      Private Const FILE_FLAG_OVERLAPPED As Int32 = &H40000000
      Private Const ERROR_IO_PENDING As Int32 = 997
      Private Const WAIT_OBJECT_0 As Int32 = 0
      Private Const ERROR_IO_INCOMPLETE As Int32 = 996
      Private Const WAIT_TIMEOUT As Int32 = &H102&
      Private Const INFINITE As Int32 = &HFFFFFFFF


#End Region
#Region "Win32API"
      '// Win32 API
   <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
SetCommState(ByVal hCommDev As IntPtr, ByRef lpDCB As DCB) As Int32
   End Function
   <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
GetCommState(ByVal hCommDev As IntPtr, ByRef lpDCB As DCB) As Int32
   End Function
   <DllImport("kernel32.dll", SetlastError:=True, CharSet:=CharSet.Auto)> Private
Shared Function BuildCommDCB(ByVal lpDef As String, ByRef lpDCB As DCB) As Int32
   End Function
   <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
SetupComm(ByVal hFile As IntPtr, ByVal dwInQueue As Int32, ByVal dwOutQueue As
Int32) As Int32
   End Function
   <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
SetCommTimeouts(ByVal hFile As IntPtr, ByRef lpCommTimeouts As COMMTIMEOUTS) As
Int32
   End Function
   <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
GetCommTimeouts(ByVal hFile As IntPtr, ByRef lpCommTimeouts As COMMTIMEOUTS) As
Int32
   End Function
   <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
ClearCommError(ByVal hFile As IntPtr, ByRef lpErrors As Int32, ByRef lpComStat As
COMSTAT) As Int32
   End Function
   <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
PurgeComm(ByVal hFile As IntPtr, ByVal dwFlags As Int32) As Int32
   End Function
    <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
EscapeCommFunction(ByVal hFile As IntPtr, ByVal ifunc As Int32) As Boolean
    End Function
    <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
WaitCommEvent(ByVal hFile As IntPtr, ByRef Mask As EventMasks, ByRef lpOverlap As
OVERLAPPED) As Int32
    End Function
    <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
WriteFile(ByVal hFile As IntPtr, ByVal Buffer As Byte(), ByVal
nNumberOfBytesToWrite As Integer, ByRef lpNumberOfBytesWritten As Integer, ByRef
lpOverlapped As OVERLAPPED) As Integer
    End Function
    <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
ReadFile(ByVal hFile As IntPtr, <Out()> ByVal Buffer As Byte(), ByVal
nNumberOfBytesToRead As Integer, ByRef lpNumberOfBytesRead As Integer, ByRef
lpOverlapped As OVERLAPPED) As Integer
```

```
    End Function
    <DllImport("kernel32.dll", SetlastError:=True, CharSet:=CharSet.Auto)> Private
Shared Function CreateFile(ByVal lpFileName As String, ByVal dwDesiredAccess As
Integer, ByVal dwShareMode As Integer, ByVal lpSecurityAttributes As Integer, ByVal
dwCreationDisposition As Integer, ByVal dwFlagsAndAttributes As Integer, ByVal
hTemplateFile As Integer) As IntPtr
    End Function
    <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
CloseHandle(ByVal hObject As IntPtr) As Boolean
    End Function
    <DllImport("kernel32.dll", SetlastError:=True)> Public Shared Function
GetCommModemStatus(ByVal hFile As IntPtr, ByRef lpModemStatus As Int32) As Boolean
    End Function
    <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
SetEvent(ByVal hEvent As IntPtr) As Boolean
    End Function
    <DllImport("kernel32.dll", SetlastError:=True, CharSet:=CharSet.Auto)> Private
Shared Function CreateEvent(ByVal lpEventAttributes As IntPtr, ByVal bManualReset
As Int32, ByVal bInitialState As Int32, ByVal lpName As String) As IntPtr
    End Function
    <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
WaitForSingleObject(ByVal hHandle As IntPtr, ByVal dwMilliseconds As Int32) As
Int32
    End Function
    <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
GetOverlappedResult(ByVal hFile As IntPtr, ByRef lpOverlapped As OVERLAPPED, ByRef
lpNumberOfBytesTransferred As Int32, ByVal bWait As Int32) As Int32
    End Function
    <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
SetCommMask(ByVal hFile As IntPtr, ByVal lpEvtMask As Int32) As Int32
    End Function
    <DllImport("kernel32.dll", SetlastError:=True, CharSet:=CharSet.Auto)> Private
Shared Function GetDefaultCommConfig(ByVal lpszName As String, ByRef lpCC As
COMMCONFIG, ByRef lpdwSize As Integer) As Boolean
    End Function
    <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
SetCommBreak(ByVal hFile As IntPtr) As Boolean
    End Function
    <DllImport("kernel32.dll", SetlastError:=True)> Private Shared Function
ClearCommBreak(ByVal hFile As IntPtr) As Boolean
    End Function


#End Region
#Region "Events"
      Public Event CommEvent As CommEventHandler
#End Region
#Region "Delegates"
      Public Delegate Sub CommEventHandler(ByVal source As Rs232, ByVal Mask As
EventMasks)
#End Region

      Public Property Port() As Integer
            '================================================
            '
            '           Description :           Comunication Port
            '           Created                 :           21/09/2001 - 11:25:49
            '
            '
```

```
      *Parameters Info*
            '
            '           Notes                   :
            '=================================================
            Get
                  Return miPort
            End Get
            Set(ByVal Value As Integer)
                  miPort = Value
            End Set
      End Property
      Public Sub PurgeBuffer(ByVal Mode As PurgeBuffers)
            '=================================================
            '
      ©2003 ALSTOM FIR S.p.A All rights reserved
            '
            '     Description :     Purge Communication Buffer
            '     Created                 :     01/09/03 - 10:37:39
            '     Author                  :     Corrado Cavalli
            '
            '                               *Parameters Info*
            '
            '     Notes                   :     This method will clear any
character into buffer, use TxAbort/RxAbort
            '                                           to terminate any pending
overlapped Tx/Rx operation.
            '=================================================
            If (mhRS.ToInt32 > 0) Then PurgeComm(mhRS, Mode)
      End Sub
      Public Overridable Property Timeout() As Integer
            '=================================================
            '
            '           Description:            Comunication timeout in seconds
            '           Created                 :           21/09/2001 - 11:26:50
            '
            '
      *Parameters Info*
            '
            '           Notes                   :
            '=================================================
            Get
                  Return miTimeout
            End Get
            Set(ByVal Value As Integer)
                  miTimeout = CInt(IIf(Value = 0, 500, Value))
                  '// If Port is open updates it on the fly
                  pSetTimeout()
            End Set
      End Property
      Public Property Parity() As DataParity
            '=================================================
            '
            '           Description :           Comunication parity
            '           Created                 :           21/09/2001 - 11:27:15
            '
            '
      *Parameters Info*
            '
            '           Notes                   :
```

```
        '=================================================
        Get
              Return meParity
        End Get
        Set(ByVal Value As DataParity)
              meParity = Value
        End Set
End Property
Public Property StopBit() As DataStopBit
        '=================================================
        '
        '           Description:            Comunication StopBit
        '           Created                 :           21/09/2001 - 11:27:37
        '
        '
*Parameters Info*
        '
        '           Notes                   :
        '=================================================
        Get
              Return meStopBit
        End Get
        Set(ByVal Value As DataStopBit)
              meStopBit = Value
        End Set
End Property
Public Property BaudRate() As Integer
        '=================================================
        '
        '           Description:            Comunication BaudRate
        '           Created                 :           21/09/2001 - 11:28:00
        '
        '
*Parameters Info*
        '
        '           Notes                   :
        '=================================================
        Get
              Return miBaudRate
        End Get
        Set(ByVal Value As Integer)
              miBaudRate = Value
        End Set
End Property
Public Property DataBit() As Integer
        '=================================================
        '
        '           Description :           Comunication DataBit
        '           Created                 :           21/09/2001 - 11:28:20
        '
        '
*Parameters Info*
        '
        '           Notes                   :
        '=================================================
        Get
              Return miDataBit
        End Get
        Set(ByVal Value As Integer)
```

```
                  miDataBit = Value
            End Set
      End Property
      Public Property BufferSize() As Integer
            '=================================================
            '
            '           Description :           Receive Buffer size
            '           Created                 :           21/09/2001 - 11:33:05
            '
            '
      *Parameters Info*
            '
            '           Notes                   :
            '=================================================
            Get
                  Return miBufferSize
            End Get
            Set(ByVal Value As Integer)
                  miBufferSize = Value
            End Set
      End Property
      Public Overloads Sub Open()
            '=================================================
            '
            '           Description :           Initializes and Opens comunication
port
            '           Created                 :           21/09/2001 - 11:33:40
            '
            '
      *Parameters Info*
            '
            '           Notes                   :
            '=================================================
            '// Get Dcb block,Update with current data
            Dim uDcb As DCB, iRc As Int32
            '// Set working mode
            meMode = Mode.Overlapped
            Dim iMode As Int32 = Convert.ToInt32(IIf(meMode = Mode.Overlapped,
FILE_FLAG_OVERLAPPED, 0))
            '// Initializes Com Port
            If miPort > 0 Then
                  Try
                        '// Creates a COM Port stream handle
                        mhRS = CreateFile("\\.\COM" & miPort.ToString, GENERIC_READ
Or GENERIC_WRITE, 0, 0, OPEN_EXISTING, iMode, 0)
                        If (mhRS.ToInt32 > 0) Then
                              '// Clear all comunication errors
                              Dim lpErrCode As Int32
                              iRc = ClearCommError(mhRS, lpErrCode, New COMSTAT)
                              '// Clears I/O buffers
                              iRc = PurgeComm(mhRS, PurgeBuffers.RXClear Or
PurgeBuffers.TxClear)
                              '// Gets COM Settings
                              iRc = GetCommState(mhRS, uDcb)
                              '// Updates COM Settings
                              Dim sParity As String = "NOEM"
                              sParity = sParity.Substring(meParity, 1)
                              '// Set DCB State
                              Dim sDCBState As String = String.Format("baud={0}
```

```
parity={1} data={2} stop={3}", miBaudRate, sParity, miDataBit, CInt(meStopBit))
                              iRc = BuildCommDCB(sDCBState, uDcb)
                              uDcb.Parity = CByte(meParity)
                              '// Set Xon/Xoff State
                              If mbUseXonXoff Then
                                    uDcb.Bits1 = 768
                              Else
                                    uDcb.Bits1 = 0
                              End If
                              iRc = SetCommState(mhRS, uDcb)
                              If iRc = 0 Then
                                    Dim sErrTxt As String = New
Win32Exception().Message
                                    Throw New CIOChannelException("Unable to set
COM state " & sErrTxt)
                              End If
                              '// Setup Buffers (Rx,Tx)
                              iRc = SetupComm(mhRS, miBufferSize, miBufferSize)
                              '// Set Timeouts
                              pSetTimeout()
                              '//Enables events if required
                              If mbEnableEvents Then Me.EnableEvents()
                        Else
                              '// Raise Initialization problems
                              Dim sErrTxt As String = New Win32Exception().Message
                              Throw New CIOChannelException("Unable to open COM" +
miPort.ToString + ControlChars.CrLf + sErrTxt)
                        End If
                  Catch Ex As Exception
                        '// Generica error
                        Throw New CIOChannelException(Ex.Message, Ex)
                  End Try
            Else
                  '// Port not defined, cannot open
                  Throw New ApplicationException("COM Port not defined,use Port
property to set it before invoking InitPort")
            End If
      End Sub
      Public Overloads Sub Open(ByVal Port As Integer, ByVal BaudRate As Integer,
ByVal DataBit As Integer, ByVal Parity As DataParity, ByVal StopBit As DataStopBit,
ByVal BufferSize As Integer)
            '=================================================
            '
            '           Description:            Opens comunication port (Overloaded
method)
            '           Created          :          21/09/2001 - 11:33:40
            '
            '
      *Parameters Info*
            '
            '           Notes            :
            '=================================================
            Me.Port = Port
            Me.BaudRate = BaudRate
            Me.DataBit = DataBit
            Me.Parity = Parity
            Me.StopBit = StopBit
            Me.BufferSize = BufferSize
            Open()
```

```
	End Sub
	Public Sub Close()
		'====================================================
		'
		'		Description:		Close comunication channel
		'		Created			:		21/09/2001 - 11:38:00
		'
		'
	*Parameters Info*
		'
		'		Notes			:
		'====================================================
		If mhRS.ToInt32 > 0 Then
			If mbEnableEvents = True Then
				Me.DisableEvents()
			End If
			Dim ret As Boolean = CloseHandle(mhRS)
			If Not ret Then Throw New Win32Exception
			mhRS = New IntPtr(0)
		End If
	End Sub
	ReadOnly Property IsOpen() As Boolean
		'====================================================
		'
		'		Description:		Returns Port Status
		'		Created			:		21/09/2001 - 11:38:51
		'
		'
	*Parameters Info*
		'
		'		Notes			:
		'====================================================
		Get
			Return CBool(mhRS.ToInt32 > 0)
		End Get
	End Property
	Public Overloads Sub Write(ByVal Buffer As Byte())
		'====================================================
		'
		'		Description:		Transmit a stream
		'		Created			:		21/09/2001 - 11:39:51
		'
		'
	*Parameters Info*
		'		Buffer				:	Array of Byte() to write
		'		Notes			:
		'====================================================
		Dim iRc, iBytesWritten As Integer, hOvl As GCHandle
		'------------------------------------------------------------------
		muOvlW = New Overlapped
		If mhRS.ToInt32 <= 0 Then
			Throw New ApplicationException("Please initialize and open port
before using this method")
		Else
			'// Creates Event
			Try
			hOvl = GCHandle.Alloc(muOvlW, GCHandleType.Pinned)
				muOvlW.hEvent = CreateEvent(Nothing, 1, 0, Nothing)
				If muOvlW.hEvent.ToInt32 = 0 Then Throw New
```

```
ApplicationException("Error creating event for overlapped writing")
                        '// Clears IO buffers and sends data
                        iRc = WriteFile(mhRS, Buffer, Buffer.Length, 0, muOvlW)
                        If iRc = 0 Then
                              If Marshal.GetLastWin32Error <> ERROR_IO_PENDING Then
                                    Throw New ApplicationException("Write command
error")
                              Else
                                    '// Check Tx results
                                    If GetOverlappedResult(mhRS, muOvlW,
iBytesWritten, 1) = 0 Then
                                          Throw New ApplicationException("Write
pending error")
                                    Else
                                          '// All bytes sent?
                                          If iBytesWritten <> Buffer.Length Then
Throw New ApplicationException("Write Error - Bytes Written " &
iBytesWritten.ToString & " of " & Buffer.Length.ToString)
                                    End If
                              End If
                        End If
                    Finally
                        '//Closes handle
                        CloseHandle(muOvlW.hEvent)
                  If (hOvl.IsAllocated = True) Then hOvl.Free()
                    End Try
              End If
        End Sub
        Public Overloads Sub Write(ByVal Buffer As String)
              '=================================================
              '
              '     Description :     Writes a string to RS232
              '     Created               :     04/02/2002 - 8:46:42
              '
              '                               *Parameters Info*
              '
              '     Notes                 :  24/05/2002 Fixed problem with ASCII
Encoding
              '=================================================
              Dim oEncoder As New System.Text.ASCIIEncoding
              Dim oEnc As Encoding = oEncoder.GetEncoding(1252)
              '-------------------------------------------------------------
              Dim aByte() As Byte = oEnc.GetBytes(Buffer)
              Me.Write(aByte)
        End Sub
        Public Function Read(ByVal Bytes2Read As Integer) As Integer
              '=================================================
              '
              '           Description:          Read Bytes from Port
              '           Created               :          21/09/2001 - 11:41:17
              '
              '
        *Parameters Info*
              '           Bytes2Read       :          Bytes to read from port
              '           Returns                         :          Number of
readed chars
              '
              '           Notes                 :
              '=================================================
```

```
            Dim iReadChars, iRc As Integer, bReading As Boolean, hOvl As GCHandle
            '---------------------------------------------------------------
            '// If Bytes2Read not specified uses Buffersize
            If Bytes2Read = 0 Then Bytes2Read = miBufferSize
            muOvlR = New Overlapped
            If mhRS.ToInt32 <= 0 Then
                  Throw New ApplicationException("Please initialize and open port
before using this method")
            Else
                  '// Get bytes from port
                  Try
                hOvl = GCHandle.Alloc(muOvlR, GCHandleType.Pinned)
                        muOvlR.hEvent = CreateEvent(Nothing, 1, 0, Nothing)
                        If muOvlR.hEvent.ToInt32 = 0 Then Throw New
ApplicationException("Error creating event for overlapped reading")
                        '// Clears IO buffers and reads data
                        ReDim mabtRxBuf(Bytes2Read - 1)
                        iRc = ReadFile(mhRS, mabtRxBuf, Bytes2Read, iReadChars,
muOvlR)
                        If iRc = 0 Then
                              If Marshal.GetLastWin32Error() <> ERROR_IO_PENDING
Then
                                    Throw New ApplicationException("Read pending
error")
                              Else
                                    '// Wait for characters
                                    iRc = WaitForSingleObject(muOvlR.hEvent,
miTimeout)
                                    Select Case iRc
                                          Case WAIT_OBJECT_0
                                                '// Some data received...
                                                If GetOverlappedResult(mhRS,
muOvlR, iReadChars, 0) = 0 Then
                                                      Throw New
ApplicationException("Read pending error.")
                                                Else
                                                      Return iReadChars
                                                End If
                                          Case WAIT_TIMEOUT
                                                Throw New IOTimeoutException("Read
Timeout.")
                                          Case Else
                                                Throw New
ApplicationException("General read error.")
                                    End Select
                              End If
                        Else
                              Return (iReadChars)
                        End If
                  Finally
                        '//Closes handle
                        CloseHandle(muOvlR.hEvent)
                        If (hOvl.IsAllocated) Then hOvl.Free()
                  End Try
            End If
      End Function
      Overridable ReadOnly Property InputStream() As Byte()
            '===============================================
            '
```

```
            '           Description:            Returns received data   as Byte()
            '           Created                 :           21/09/2001 - 11:45:06
            '
            '
      *Parameters Info*
            '
            '           Notes                   :
            '=================================================
      Get
         Return mabtRxBuf
      End Get
   End Property
      Overridable ReadOnly Property InputStreamString() As String
            '=================================================
            '
            '     Description :     Return a string containing received data
            '     Created                 :     04/02/2002 - 8:49:55
            '
            '                                   *Parameters Info*
            '
            '     Notes                   :
            '=================================================
            Get
         Dim oEncoder As New System.Text.ASCIIEncoding
         Dim oEnc As Encoding = oEncoder.GetEncoding(1252)
               '---------------------------------------------------------------
               If Not Me.InputStream Is Nothing Then Return
oEnc.GetString(Me.InputStream)
            End Get
      End Property
      Public Sub ClearInputBuffer()
            '=================================================
            '
            '           Description:            Clears Input buffer
            '           Created                 :           21/09/2001 - 11:45:34
            '
            '
      *Parameters Info*
            '
            '           Notes                   :  Gets all character until end of
buffer
            '=================================================
            If mhRS.ToInt32 > 0 Then
                  PurgeComm(mhRS, PURGE_RXCLEAR)
            End If
      End Sub
      Public WriteOnly Property Rts() As Boolean
            '=================================================
            '
            '           Description:            Set/Resets RTS Line
            '           Created                 :           21/09/2001 - 11:45:34
            '
            '
      *Parameters Info*
            '
            '           Notes                   :
            '=================================================
            Set(ByVal Value As Boolean)
                  If mhRS.ToInt32 > 0 Then
```

```
                        If Value Then
                              EscapeCommFunction(mhRS, Lines.SetRts)
                        Else
                              EscapeCommFunction(mhRS, Lines.ClearRts)
                        End If
                  End If
            End Set
      End Property
      Public WriteOnly Property Dtr() As Boolean
            '====================================================
            '
            '           Description:            Set/Resets DTR Line
            '           Created                 :           21/09/2001 - 11:45:34
            '
            '
      *Parameters Info*
            '
            '           Notes                   :
            '====================================================
            Set(ByVal Value As Boolean)
                  If mhRS.ToInt32 > 0 Then
                        If Value Then
                              EscapeCommFunction(mhRS, Lines.SetDtr)
                        Else
                              EscapeCommFunction(mhRS, Lines.ClearDtr)
                        End If
                  End If
            End Set
      End Property
      Public ReadOnly Property ModemStatus() As ModemStatusBits
            '====================================================
            '
            '     Description :     Gets Modem status
            '     Created                 :     28/02/2002 - 8:58:04
            '
            '                                   *Parameters Info*
            '
            '     Notes                   :
            '====================================================
            Get
                  If mhRS.ToInt32 <= 0 Then
                        Throw New ApplicationException("Please initialize and open
port before using this method")
                  Else
                        '// Retrieve modem status
                        Dim lpModemStatus As Int32
                        If Not GetCommModemStatus(mhRS, lpModemStatus) Then
                              Throw New ApplicationException("Unable to get modem
status")
                        Else
                              Return CType(lpModemStatus, ModemStatusBits)
                        End If
                  End If
            End Get
      End Property
      Public Function CheckLineStatus(ByVal Line As ModemStatusBits) As Boolean
            '====================================================
            '
            '     Description :     Check status of a Modem Line
```

```
            '     Created                 :     28/02/2002 - 10:25:17
            '
            '                                   *Parameters Info*
            '
            '     Notes                   :
            '=================================================
            Return Convert.ToBoolean(ModemStatus And Line)
      End Function
      Public Property UseXonXoff() As Boolean
            '=================================================
            '
            '     Description :     Set XON/XOFF mode
            '     Created                 :     26/05/2003 - 21:16:18
            '
            '                                   *Parameters Info*
            '
            '     Notes                   :
            '=================================================
            Get
                  Return mbUseXonXoff
            End Get
            Set(ByVal Value As Boolean)
                  mbUseXonXoff = Value
            End Set
      End Property
      Public Sub EnableEvents()
            '=================================================
            '
            '     Description :     Enables monitoring of incoming events
            '     Created                 :     15/07/2003 - 12:00:56
            '
            '                                   *Parameters Info*
            '
            '     Notes                   :
            '=================================================
            If mhRS.ToInt32 <= 0 Then
                  Throw New ApplicationException("Please initialize and open port
before using this method")
            Else
                  If moEvents Is Nothing Then
                        mbEnableEvents = True
                        moEvents = New Thread(AddressOf pEventsWatcher)
                        moEvents.IsBackground = True
                        moEvents.Start()
                  End If
            End If
      End Sub
      Public Sub DisableEvents()
            '=================================================
            '
            '     Description :     Disables monitoring of incoming events
            '     Created                 :     15/07/2003 - 12:00:56
            '
            '                                   *Parameters Info*
            '
            '     Notes                   :
            '=================================================
            If mbEnableEvents = True Then
                  SyncLock Me
```

```
                        mbEnableEvents = False                      '// This should
kill the thread
                    End SyncLock
                    '// Let WaitCommEvent exit...
                    If muOvlE.hEvent.ToInt32 <> 0 Then SetEvent(muOvlE.hEvent)
                    moEvents = Nothing
                End If
            End Sub
            Public Property RxBufferThreshold() As Int32
                '================================================
                '
            ©2003 www.codeworks.it All rights reserved
                '
                '     Description :     Numer of characters into input buffer
                '     Created                 :     16/07/03 - 9:00:57
                '     Author                  :     Corrado Cavalli
                '
                '                                   *Parameters Info*
                '
                '     Notes                   :
                '================================================
                Get
                    Return miBufThreshold
                End Get
                Set(ByVal Value As Int32)
                    miBufThreshold = Value
                End Set
            End Property
        Public Shared Function IsPortAvailable(ByVal portNumber As Int32) As Boolean
            '================================================
            '                                                                    ©2003
www.codeworks.it All rights reserved
            '
            '   Description :     Returns true if a specific port number is supported
by the system
            '   Created                 :     14/09/03 - 17:00:57
            '   Author                  :     Corrado Cavalli
            '
            '                                 *Parameters Info*
            '   portNumber  :     port number to check
            '
            '   Notes                   :
            '================================================
            If portNumber <= 0 Then
                Return False
            Else
                Dim cfg As COMMCONFIG
                Dim cfgsize As Int32 = Marshal.SizeOf(cfg)
                cfg.dwSize = cfgsize
                Dim ret As Boolean = GetDefaultCommConfig("COM" + portNumber.ToString,
cfg, cfgsize)
                Return ret
            End If
        End Function
        Public Sub SetBreak()
            '================================================
            '                                                                    ©2003
www.codeworks.it All rights reserved
            '
```

```
        '   Description :     Set COM in break modem
        '   Created                 :     12/10/03 - 10:00:57
        '   Author                  :     Corrado Cavalli
        '
        '                             *Parameters Info*
        '
        '
        '   Notes                   :
        '=================================================
        If mhRS.ToInt32 > 0 Then
            If SetCommBreak(mhRS) = False Then Throw New Win32Exception
        End If
    End Sub
    Public Sub ClearBreak()
        '=================================================
        '                                                                                ©2003
www.codeworks.it All rights reserved
        '
        '   Description :     Clear COM break mode
        '   Created                 :     12/10/03 - 10:02:57
        '   Author                  :     Corrado Cavalli
        '
        '                             *Parameters Info*
        '
        '
        '   Notes                   :
        '=================================================
        If mhRS.ToInt32 > 0 Then
            If ClearCommBreak(mhRS) = False Then Throw New Win32Exception
        End If

    End Sub
    Public ReadOnly Property InBufferCount() As Int32
        '=================================================
        '                                                                                ©2003
www.codeworks.it All rights reserved
        '
        '   Description :     Returns the number of bytes inside Rx buffer
        '   Created                 :     20/04/05 - 10:02:57
        '   Author                  :     Corrado Cavalli/Jean-Pierre ZANIER
        '
        '
        '=================================================
        Get
            Dim comStat As COMSTAT
            Dim lpErrCode As Int32
            Dim iRc As Int32
            comStat.cbInQue = 0
            If mhRS.ToInt32 > 0 Then
                iRc = ClearCommError(mhRS, lpErrCode, comStat)
                Return comStat.cbInQue
            End If
            Return 0
        End Get
    End Property


#Region "Finalize"
    Protected Overrides Sub Finalize()
```

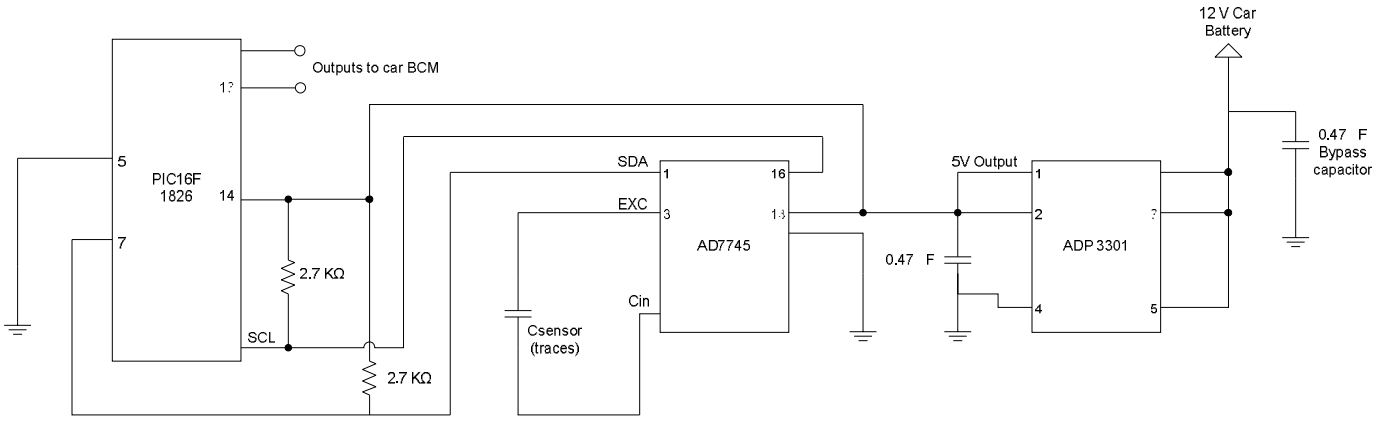
```
        '=====================================================
        '
        '   Description :     Closes COM port if object is garbage collected and
still owns
        '                       COM port reosurces
        '
        '   Created                 :     27/05/2002 - 19:05:56
        '
        '                                 *Parameters Info*
        '
        '   Notes                   :
        '=====================================================
        Try
            If Not mbDisposed Then
                If mbEnableEvents Then Me.DisableEvents()
                Close()
            End If
        Finally
            MyBase.Finalize()
        End Try
    End Sub
#End Region

#Region "Private Routines"
    Private Sub pSetTimeout()
        '=====================================================
        '
        '         Description:           Set comunication timeouts
        '         Created                :           21/09/2001 - 11:46:40
        '
        '
      *Parameters Info*
        '
        '         Notes                  :
        '=====================================================
        Dim uCtm As COMMTIMEOUTS
        '// Set ComTimeout
        If mhRS.ToInt32 <= 0 Then
            Exit Sub
        Else
            '// Changes setup on the fly
            With uCtm
                .ReadIntervalTimeout = 0
                .ReadTotalTimeoutMultiplier = 0
                .ReadTotalTimeoutConstant = miTimeout
                .WriteTotalTimeoutMultiplier = 10
                .WriteTotalTimeoutConstant = 100
            End With
            SetCommTimeouts(mhRS, uCtm)
        End If
    End Sub
    Private Sub pDispose() Implements IDisposable.Dispose
        '=====================================================
        '
        '   Description :     Handles correct class disposing Write
        '   Created                 :     27/05/2002 - 19:03:06
        '
        '                                 *Parameters Info*
        '
```

```
        '   Notes                   :
        '=================================================
        If (Not mbDisposed AndAlso (mhRS.ToInt32 > 0)) Then
            '// Closes Com Port releasing resources
            Try
                Me.Close()
            Finally
                mbDisposed = True
                '// Suppress unnecessary Finalize overhead
                GC.SuppressFinalize(Me)
            End Try
        End If


    End Sub
    Private Sub pEventsWatcher()
        '=================================================
        '                                                                                ©2003
www.codeworks.it All rights reserved
        '
        '   Description :     Watches for all events raising events when they
arrive to the port
        '   Created                 :     15/07/03 - 11:45:13
        '   Author                  :     Corrado Cavalli
        '
        '                                 *Parameters Info*
        '
        '   Notes                   :
        '=================================================
        '// Events to watch
        Dim lMask As EventMasks = EventMasks.Break Or EventMasks.CarrierDetect Or
EventMasks.ClearToSend Or _
        EventMasks.DataSetReady Or EventMasks.Ring Or EventMasks.RxChar Or
EventMasks.RXFlag Or _
        EventMasks.StatusError
        Dim lRetMask As EventMasks, iBytesRead, iTotBytes, iErrMask As Int32, iRc
As Int32, aBuf As New ArrayList
        Dim uComStat As COMSTAT
        '-----------------------------------
        '// Creates Event
        muOvlE = New Overlapped
        Dim hOvlE As GCHandle = GCHandle.Alloc(muOvlE, GCHandleType.Pinned)
        muOvlE.hEvent = CreateEvent(Nothing, 1, 0, Nothing)
        If muOvlE.hEvent.ToInt32 = 0 Then Throw New ApplicationException("Error
creating event for overlapped reading")
        '// Set mask
        SetCommMask(mhRS, lMask)
        '// Looks for RxChar
        While mbEnableEvents = True
            WaitCommEvent(mhRS, lMask, muOvlE)
            Select Case WaitForSingleObject(muOvlE.hEvent, INFINITE)
                Case WAIT_OBJECT_0
                    '// Event (or abort) detected
                    If mbEnableEvents = False Then Exit While
                    If (lMask And EventMasks.RxChar) > 0 Then
                        '// Read incoming data
                        ClearCommError(mhRS, iErrMask, uComStat)
                        If iErrMask = 0 Then
                            Dim ovl As New Overlapped
```

```
                            Dim hOvl As GCHandle = GCHandle.Alloc(ovl,
GCHandleType.Pinned)
                            ReDim mabtRxBuf(uComStat.cbInQue - 1)
                            If ReadFile(mhRS, mabtRxBuf, uComStat.cbInQue,
iBytesRead, ovl) > 0 Then
                                If iBytesRead > 0 Then
                                    '// Some bytes read, fills temporary buffer
                                    If iTotBytes < miBufThreshold Then
                                        aBuf.AddRange(mabtRxBuf)
                                        iTotBytes += iBytesRead
                                    End If
                                    '// Threshold reached?, raises event
                                    If iTotBytes >= miBufThreshold Then
                                        '//Copies temp buffer into Rx buffer
                                        ReDim mabtRxBuf(iTotBytes - 1)
                                        aBuf.CopyTo(mabtRxBuf)
                                        '// Raises event
                                        Try
                                            Me.OnCommEventReceived(Me, lMask)
                                        Finally
                                            iTotBytes = 0
                                            aBuf.Clear()
                                        End Try
                                    End If
                                End If
                            End If
                            If (hOvl.IsAllocated) Then hOvl.Free()
                        End If
                    Else
                        '// Simply raises OnCommEventHandler event
                        Me.OnCommEventReceived(Me, lMask)
                    End If
                Case Else
                    Dim sErr As String = New Win32Exception().Message
                    Throw New ApplicationException(sErr)
            End Select
        End While
        '// Release Event Handle
        CloseHandle(muOvlE.hEvent)
        muOvlE.hEvent = IntPtr.Zero
        If (hOvlE.IsAllocated) Then hOvlE.Free()
        muOvlE = Nothing
    End Sub



#End Region

#Region "Protected Routines"
    Protected Sub OnCommEventReceived(ByVal source As Rs232, ByVal mask As
EventMasks)
        '================================================
        '                                                                    ©2003
www.codeworks.it All rights reserved
        '
        '   Description :     Raises CommEvent
        '   Created                 :     15/07/03 - 15:09:50
        '   Author                  :     Corrado Cavalli
        '
```

```
        '                                   *Parameters Info*
        '
        '   Notes                   :
        '=================================================
        Dim del As CommEventHandler = Me.CommEventEvent
        If (Not del Is Nothing) Then
            Dim SafeInvoker As ISynchronizeInvoke
            Try
                SafeInvoker = DirectCast(del.Target, ISynchronizeInvoke)
            Catch
            End Try
            If (Not SafeInvoker Is Nothing) Then
                SafeInvoker.Invoke(del, New Object() {source, mask})
            Else
                del.Invoke(source, mask)
            End If
        End If
    End Sub
#End Region

End Class
#End Region

#Region "Exceptions"
Public Class CIOChannelException : Inherits ApplicationException
      '=================================================
      '
      '           Module                  :           CChannellException
      '           Description:            Customized Channell Exception
      '           Created                 :           17/10/2001 - 10:32:37
      '
      '           Notes                   :           This exception is raised when
NACK error found
      '=================================================
      Sub New(ByVal Message As String)
            MyBase.New(Message)
      End Sub
      Sub New(ByVal Message As String, ByVal InnerException As Exception)
            MyBase.New(Message, InnerException)
      End Sub
End Class
Public Class IOTimeoutException : Inherits CIOChannelException
      '=================================================
      '
      '     Description :     Timeout customized exception
      '     Created                 :     28/02/2002 - 10:43:43
      '
      '                                   *Parameters Info*
      '
      '     Notes                   :
      '=================================================
      Sub New(ByVal Message As String)
            MyBase.New(Message)
      End Sub
      Sub New(ByVal Message As String, ByVal InnerException As Exception)
            MyBase.New(Message, InnerException)
      End Sub
End Class
#End Region
```

Appendix C. Schematic



Appendix D. PCB Layout

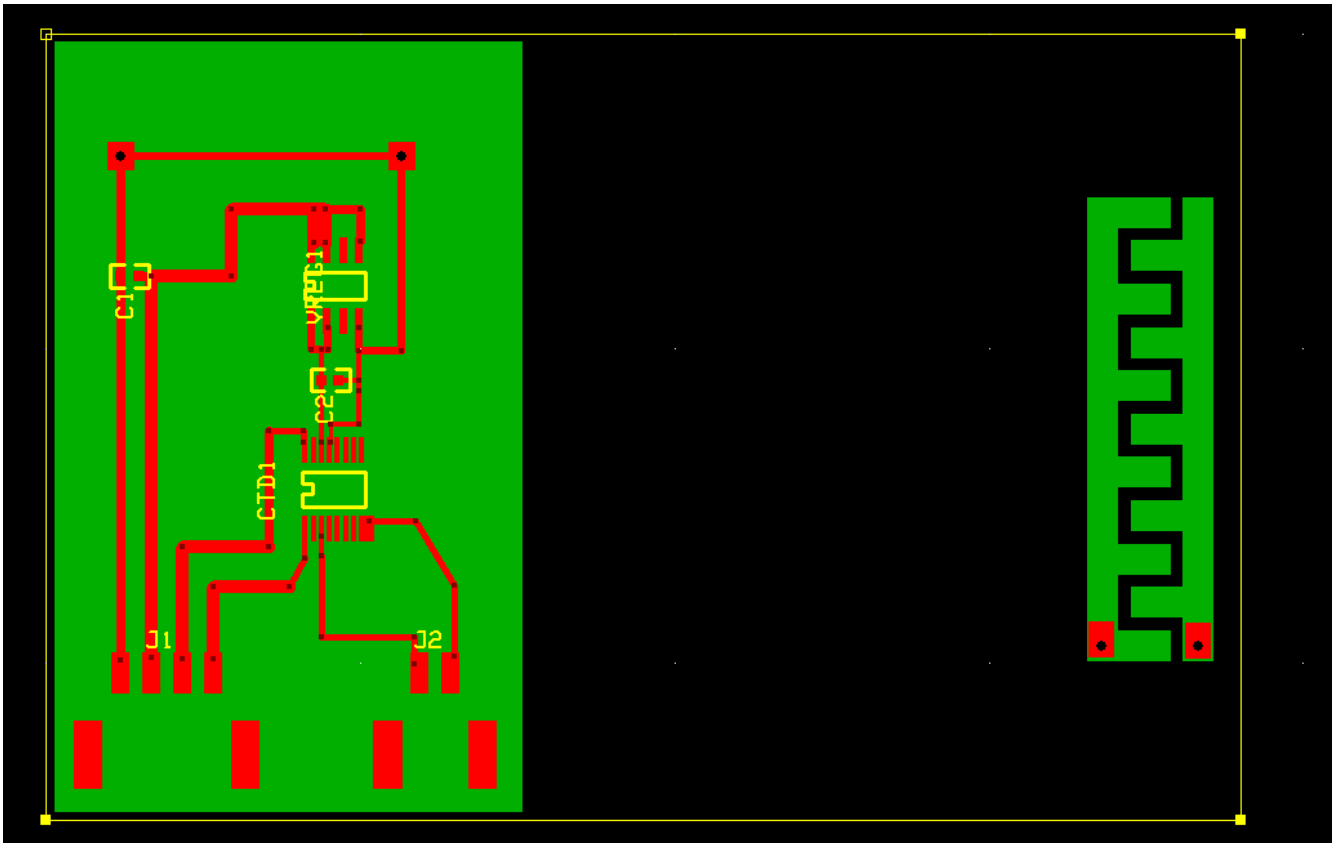


Figure 30: Display prototype unit with no PIC

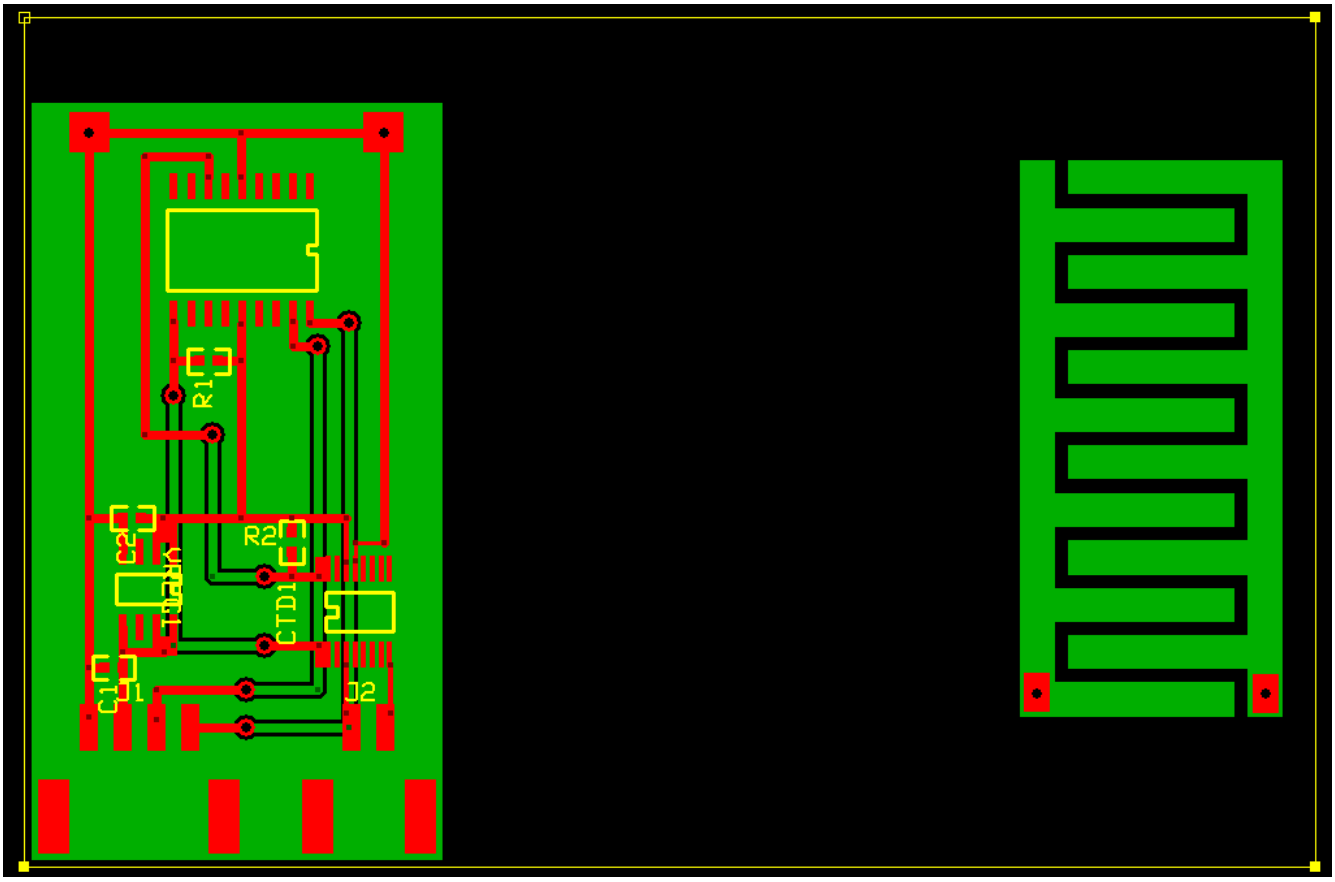


Figure 31: Production level prototype with PIC

Appendix E. Test Data

E.1. Mist Data

Case # 2: Mist			
Iteration	**Initial Cap**	**Final Cap**	**Change in Cap**
1	6	20	14
2	5	24	19
3	4	20	16
4	2	16	14
5	1	16	15
6	0	10	10
7	-8	3	11
8	-6	4	10
9	-9	1	10
10	-10	-2	8
11	-12	-1	11
12	-13	-2	11

13	-15	-5	10
14	-13	-4	9
15	-15	-4	11
16	-4	4.6	8.6
17	-4	5	9
18	-5	7	12
19	2	10	8
20	0	9	9
21	-1	10	11
22	-3	9	12
23	-5	3	8
24	-5	7	12
25	-6	6	12
26	-6	4	10
27	-7	6	13
28	-8	1	9
29	-9	3	12
30	-10	0	10
31	-12	0	12
32	-11	0	11
33	-12	1	13
34	-18	-4	14
35	-15	-5	10
36	-15	-1	14
37	-5	9	14
38	-11	0	11
39	-12	-2	10
40	10	25	15
41	10	18	8
42	2	18	16
43	3	15	12
44	3	12	9
45	0	12	12
46	-1	10	11
47	-2	13	15
48	-4	12	16
49	-7	2	9
50	-10	2	12

E2. Rain Data

Case # 4: Rain	
Iteration	**Delta**
1	38
2	53
3	109
4	125

5	17.9
6	54
7	99
8	30
9	30
10	5
11	20
12	6.57
13	36
14	54
15	75
16	27
17	91
18	51
19	33
20	89
21	5.8
22	68
23	45
24	52.35
25	138
26	146
27	124
28	25.24
29	24
30	80.4
31	35.4
32	21
33	5.7
34	40.4
35	55
36	138
37	25.8
38	87.523
39	118
40	56
41	22
42	74
43	78
44	102
45	39
46	40.4
47	90
48	47
49	74.56
50	89.93
51	166
52	104.1
53	74.58

54	68.04
55	158.78
56	109.1
57	70.01
58	86.609
59	85.99
60	153
61	101
62	46
63	33
64	47
65	27.47
66	89
67	78
68	23
69	73
70	45
71	54
72	31
73	36
74	107
75	148
76	90.5
77	35
78	156
79	32
80	58.6
81	79.56
82	40
83	107
84	71
85	22.32
86	28.2
87	111
88	20.7
89	103
90	71.5
91	43
92	102
93	108
94	105
95	53.5
96	75.5
97	73.5
98	31
99	108
100	127

E.3. Downpour Data

Case # 6: Downpour				
Iteration	**Initial Cap**	**Final Cap**	**Change in Cap**	**Delta**
1			0	210
2			0	232
3			0	253

E.4. Leaf Data

Case # 7: Leaf			
Iteration	**Initial Cap**	**Final Cap**	**Change in Cap**
1	67	932	865
2	49	1039	990
3	56	938	882
4	52	1036	984
5	46	930	884
6	26	736	710
7	38	938	900
8	49	997	948
9	42	938	896
10	52	810	758
11	64	1093	1029
12	61	1031	970
13	61	1028	967
14	61	1107	1046
15	61	855	794
16	60	865	805
17	61	969	908
18	61	915	854
19	58	1043	985
20	51	973	922
21	54	1042	988
22	50	942	892
23	43	958	915
24	46	971	925
25	49	1044	995
26	50	938	888
27	48	898	850
28	48	902	854
29	46	1069	1023
30	39	1109	1070
31	46	728	682
32	14	602	588
33	14	888	874

34	13	815	802
35	8	807	799
36	13	876	863
37	13	1152	1139
38	14	992	978
39	14	1028	1014
40	14	1112	1098
41	13	973	960
42	14	1143	1129
43	8	1077	1069
44	8	1162	1154
45	5	978	973
46	5	764	759
47	4	821	817
48	4	854	850
49	4	738	734
50	4	860	856

E.5. Finger / Hand Data

Case # 9: Finger/Hand			
Iteration	**Initial Cap**	**Final Cap**	**Change in Cap**
1	50	1430	1380
2	55	1350	1295
3	57	1359	1302
4	59	1370	1311
5	64	1387	1323
6	58	1353	1295
7	61	1394	1333
8	63	1301	1238
9	65	1394	1329
10	68	1402	1334
11	62	1348	1286
12	63	1424	1361
13	63.8	1372	1308.2
14	66	1375	1309
15	68	1330	1262
16	71	1395	1324
17	65	1440	1375
18	54	1335	1281
19	56	1358	1302
20	57	1400	1343
21	59	1388	1329
22	64	1358	1294
23	65	1343	1278
24	68	1350	1282
25	68	1358	1290

26	70	1370	1300
27	70	1330	1260
28	59	1350	1291
29	67	1360	1293
30	69	1390	1321
31	71	1384	1313
32	19	1406	1387
33	22	1436	1414
34	31	1460	1429
35	36	1276	1240
36	37	1528	1491
37	45	1317	1272
38	47	1348	1301
39	48	1325	1277
40	59	1295	1236
41	53	1306	1253
42	56	1358	1302
43	60	1360	1300
44	62	1348	1286
45	63	1603	1540
46	65	1356	1291
47	65	1557	1492
48	67	1480	1413
49	70	1307	1237
50	71	1377	1306

E6. Testing Pictures

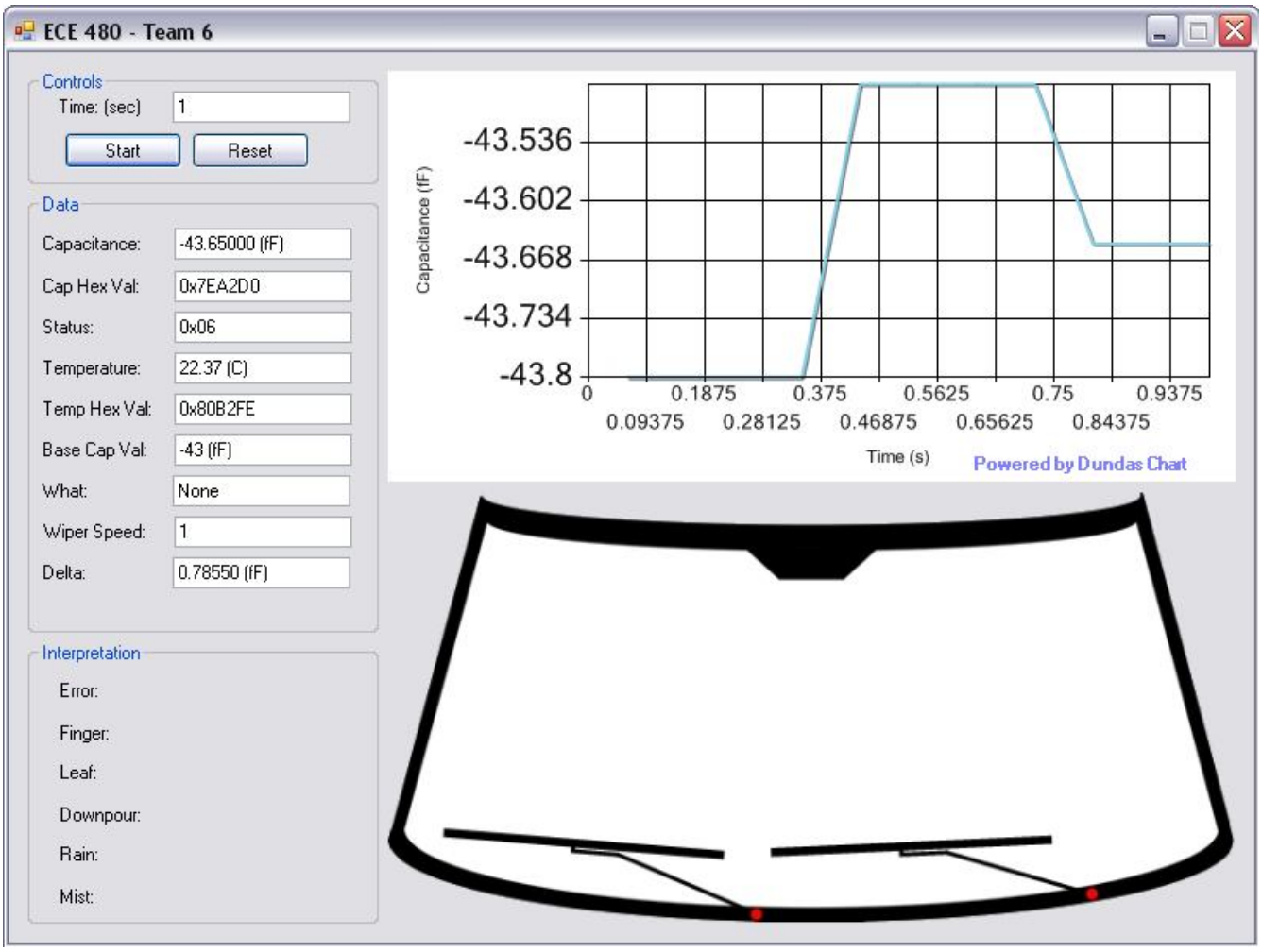


Figure 32: Startup display of Visual Basic program

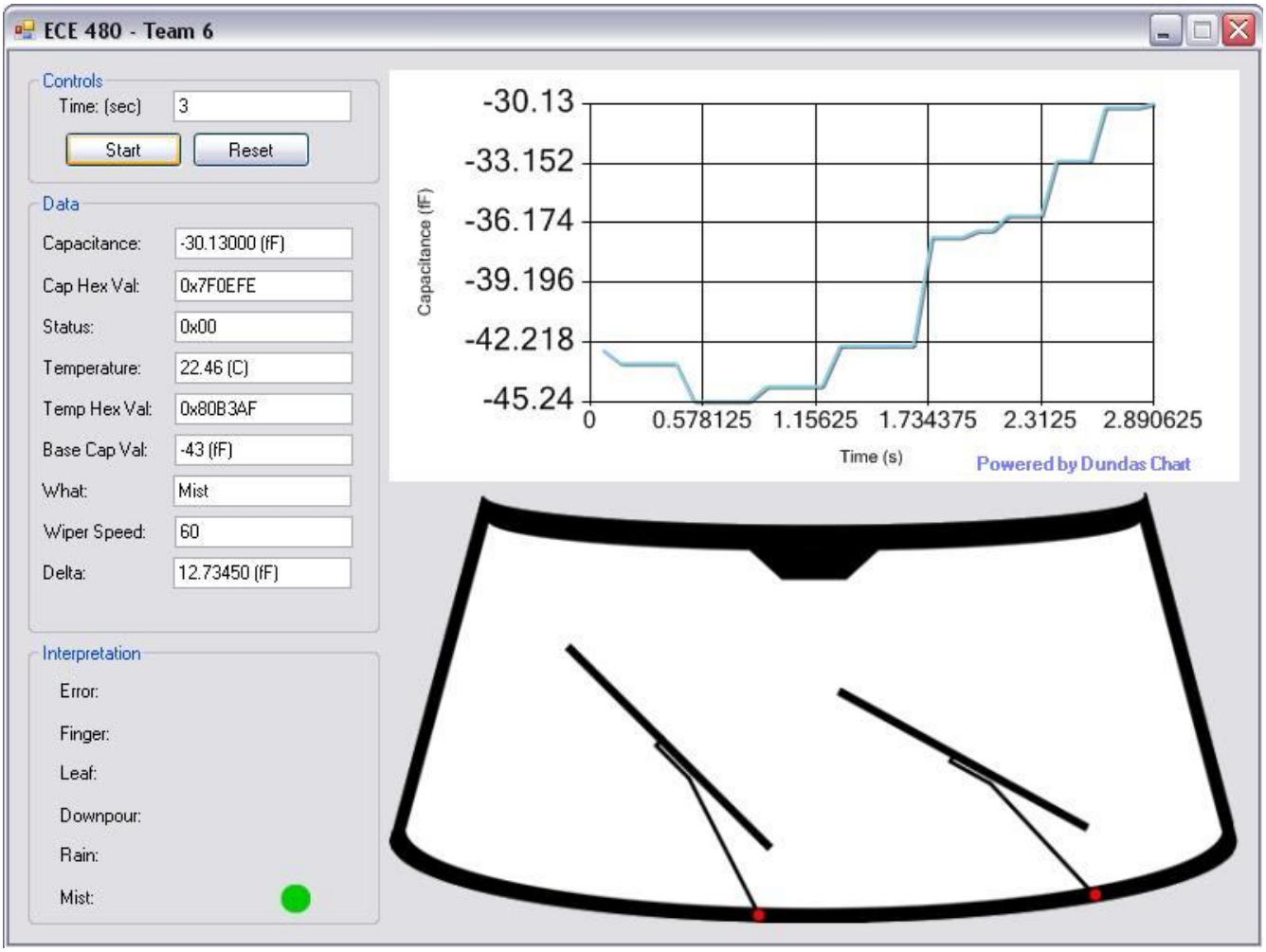


Figure 33: Test with Mist

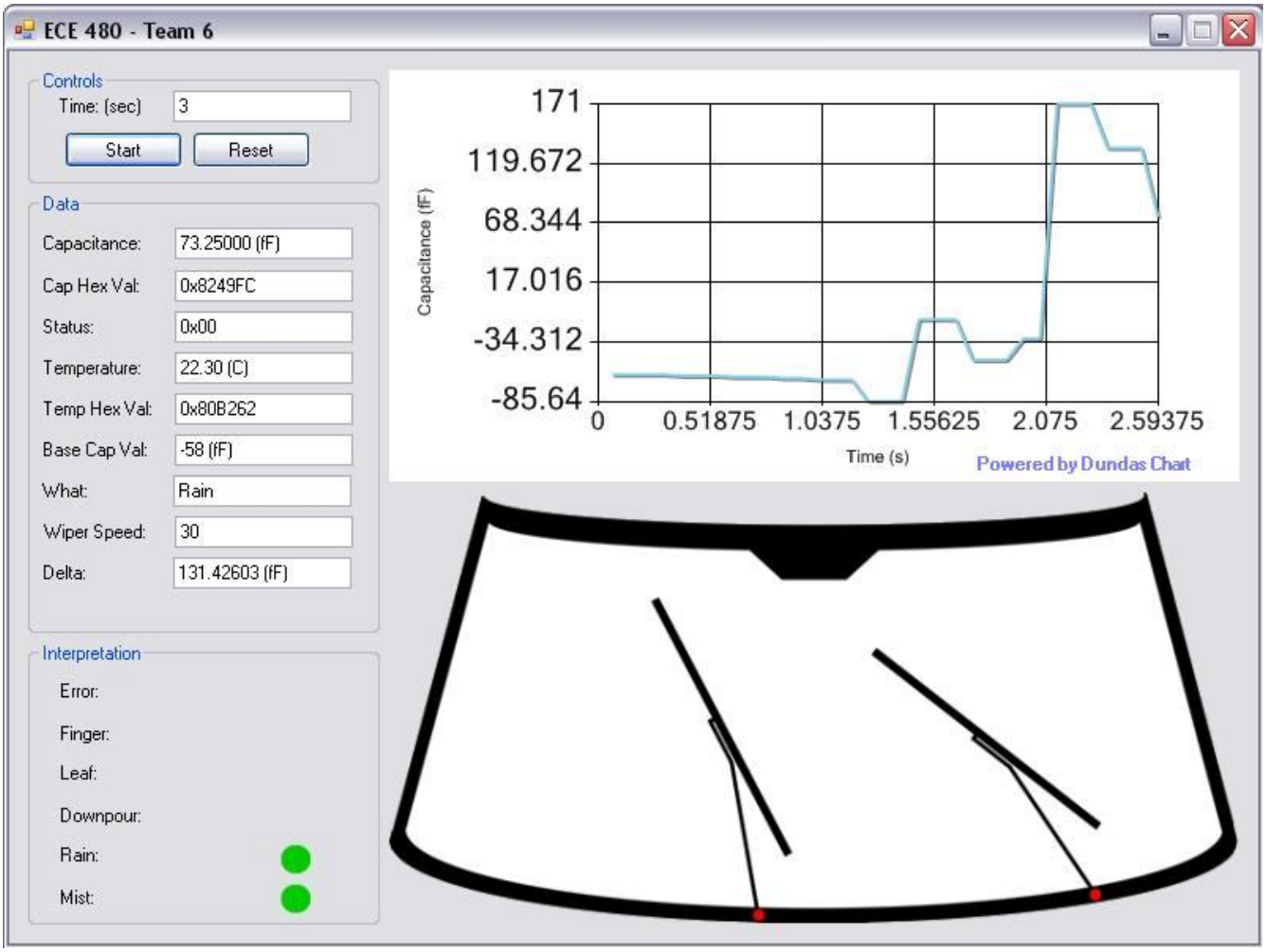


Figure 33: Test with Rain level

Figure 34: Test with Downpour

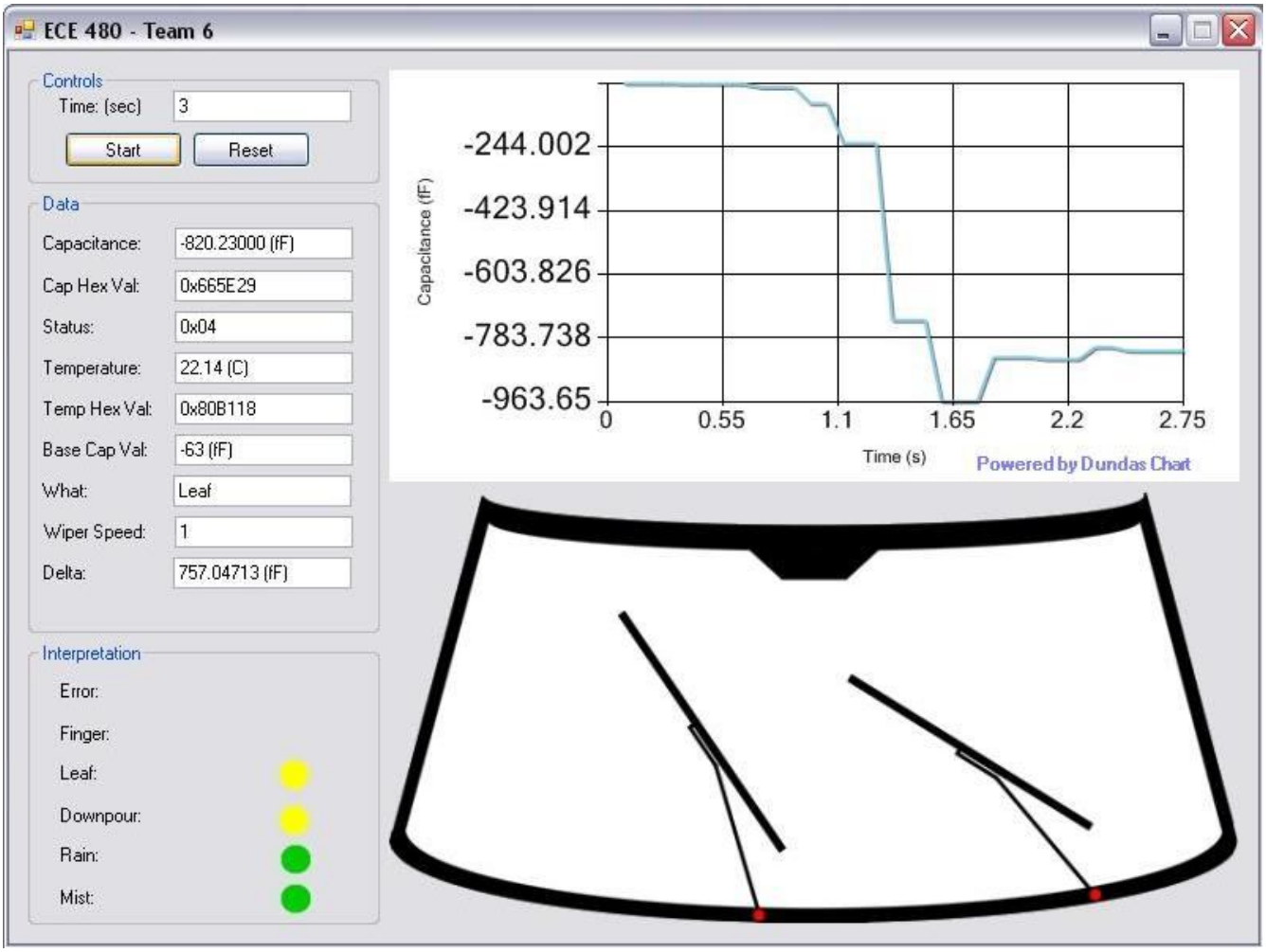


Figure 35: Test with Leaf

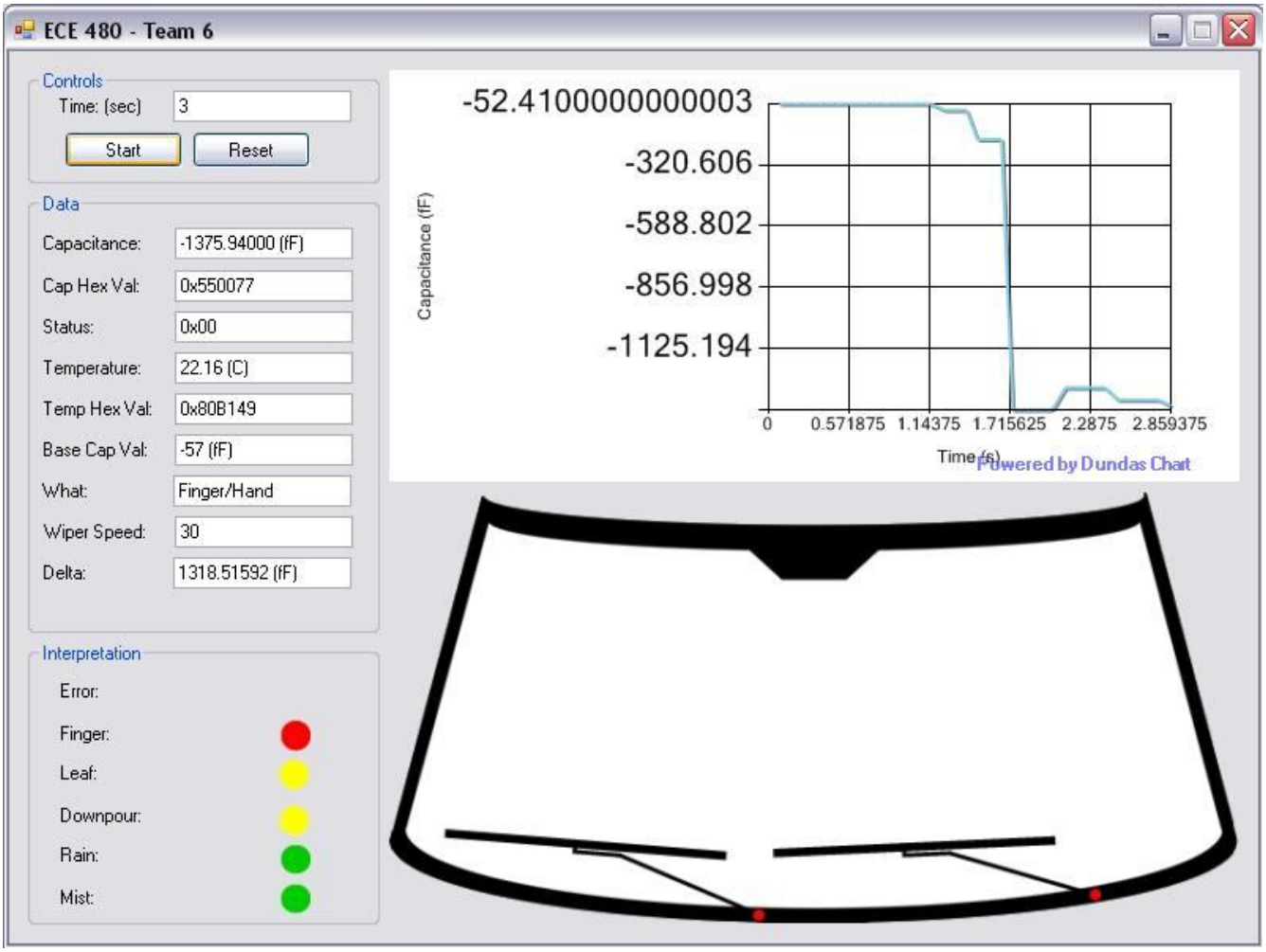


Figure 36: Test with hand

Appendix F. COMSOL Reference (Arslan Qaiser – Application Note)

COMSOL:

COMSOL Multiphysics is an interactive engineering and physics tool that performs equation based modeling in a visual interface. This software allows the modeling and simulation of any physical phenomena in a way that's easy to implement. It comes pre-installed with different model libraries that can be readily used. Some of the libraries include modules such as Chemical Engineering Modules, MEMS Modules, RF Modules and Structural Mechanics Module.

In this application note, only the Electrostatics part of MEMS module will be discussed. Specifically it

will be shown how to approach a 3D electrostatics problem by first creating a 2D geometry using the array tools and then extrude it to a 3D geometry, perform Mesh analysis and compute the capacitance using the Electrostatics application mode's port boundary conditions.

Implementation:

In order to design a capacitive sensor (also called a comb drive) in COMSOL, a series of steps need to be followed:

A. Model Navigator

1. Start by opening the **Model Navigator** after running COMSOL
2. Select the **MEMS Module** → **Electrostatics** → **Electrostatics** as shown in the following figure:

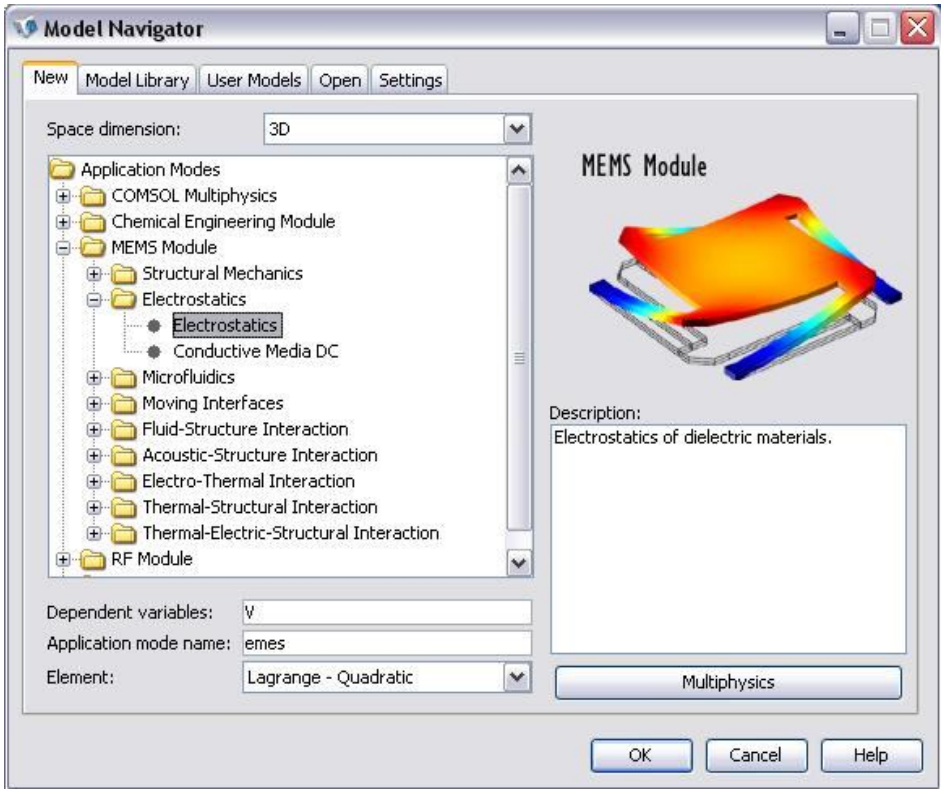


Figure 3

B. Geometry Modeling

1. To draw a 2D geometry, click on **Work-Plane Setting** under **Draw** and select z=0

2. Draw a rectangle by clicking on the **Rectangle/Square** icon in the draw toolbar. Once drawn, it is labeled as 'R1'.

3. Similarly, draw a series of rectangles by copy/paste and shift in the y-direction by $8x10^{-6}$ as shown:

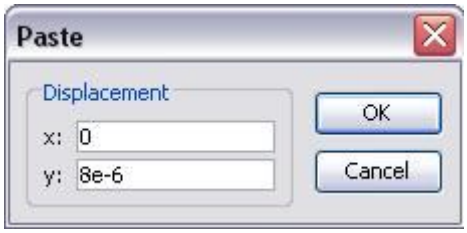


Figure 4

Note: The displacement can be different from $8x10^{-6}$ depending on the size of the sensor that is desired. For this application note, this size was chosen to make it easy for the reader to follow.

4. Select all the rectangles by using Ctrl+A then copy/paste again, this time adding both x-displacement and y-displacement to get the following: (x: $14x10^{-6}$, y: $4x10^{-6}$)

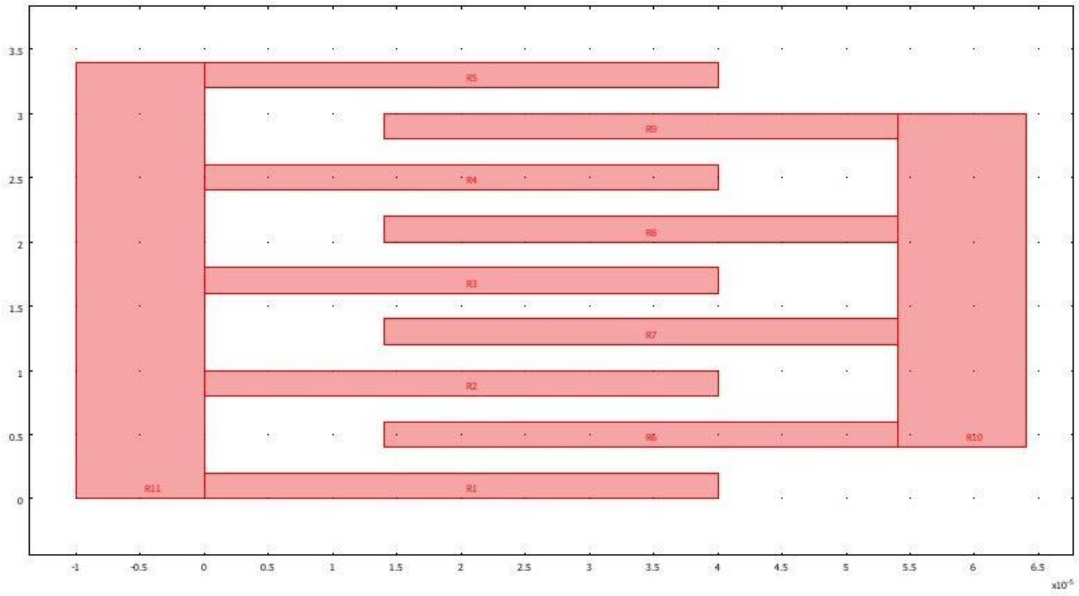


Figure 5

5. Use the **Union** and **Delete Interior Boundaries** button to merge all the rectangles (from R1-R11) and get to uniform 'comb like' structures as shown. These are called composite objects (CO2 and CO3 below)

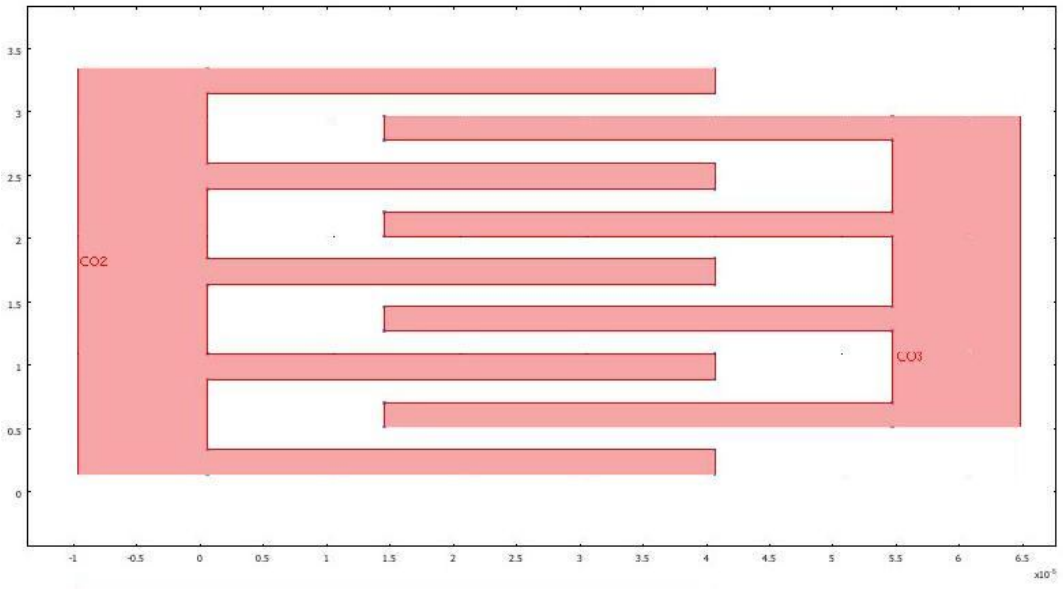


Figure 6

6. Draw a rectangle that covers the comb structure (labeled as R1). This will serve as the substrate and bounding air space for the analysis. This will be helpful in the 3D analysis when a top and bottom layer will be added to the comb structure.

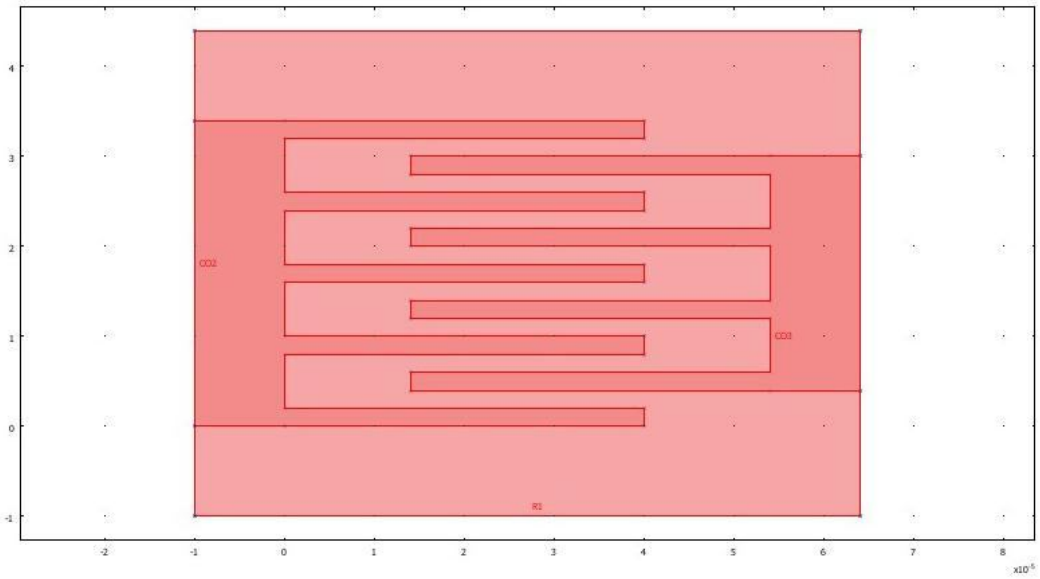


Figure 7

C. Extruding to 3D Geometry

1. The first step is to extrude the two comb drives to a 3D geometry. Select **Extrude** from the **Draw** menu and select the composite objects (CO2 and CO3).

2. Type in $2x10^{-6}$ in the distance field. The distance field corresponds to the height of the comb in the 3D geometry. This value can change based on the application of the capacitive sensor. The 3D comb drive can be seen in Figure 9 below.

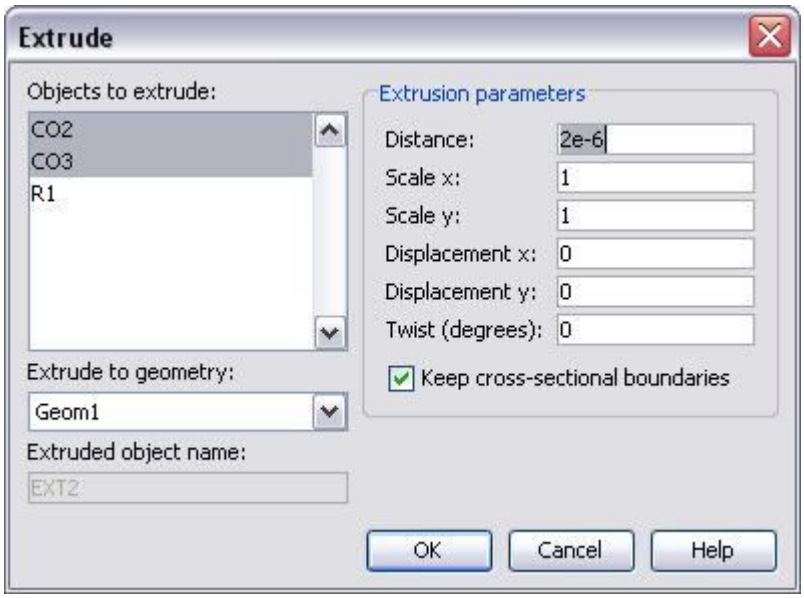


Figure 8

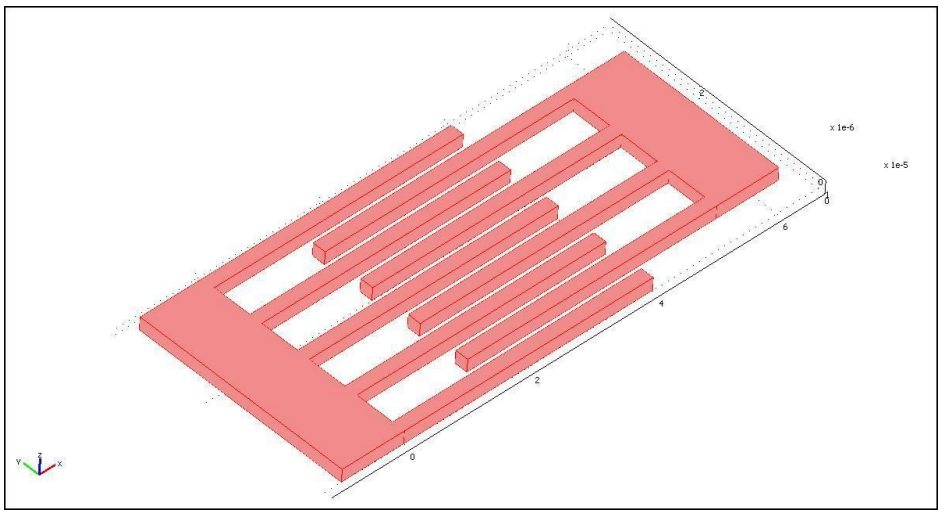


Figure 9

3. Next step is to include a layer of air on the top layer of the 3D comb. To do this, the rectangle R1 will be extruded in a similar manner as CO2 and CO3. But in this case, the distance is set to $12x10^{-6}$.

4. Once the top layer has been added, the bottom substrate layer can be extruded in the same way. To add the bottom layer, set the distance to $-10x10^{-6}$. The negative sign indicates that the bottom layer will be extruded in the –z-direction. The following figure shows the complete 3D comb geometry.

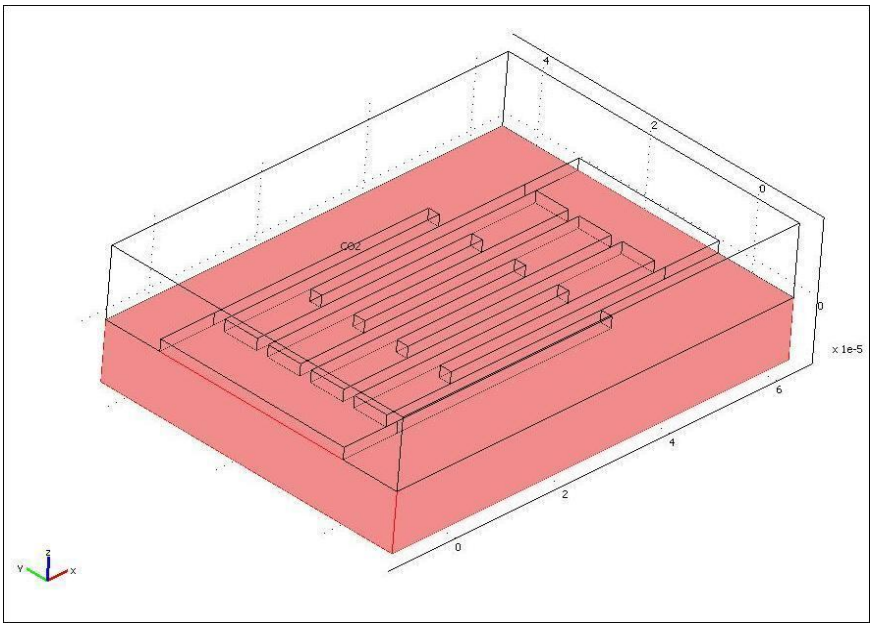


Figure 10

D. Physics Settings

1. *Subdomain Settings:*

The 3D comb geometry shown in Figure 10 consists of two sub-domains. First is the top layer of air and second is the bottom silicon substrate layer. Select **Subdomain** from the **Physics** menu.

Note: These layers can represent different materials. For example, the top layer could be glass or plastic and the bottom layer can be a substrate made of Teflon, FR4 instead of silicon. The way COMSOL identifies these different materials is through a property called 'Dielectric permittivity' (ϵ_r) of the material. A list of different Dielectric permittivity is given in Table 3 (Appendix).

In this case, the following values are used:

Settings	Subdomain 1 (Silicon)	Subdomain 2 (Air)
ϵ_r	11.9	1.0
ρ	0	0

Table 1

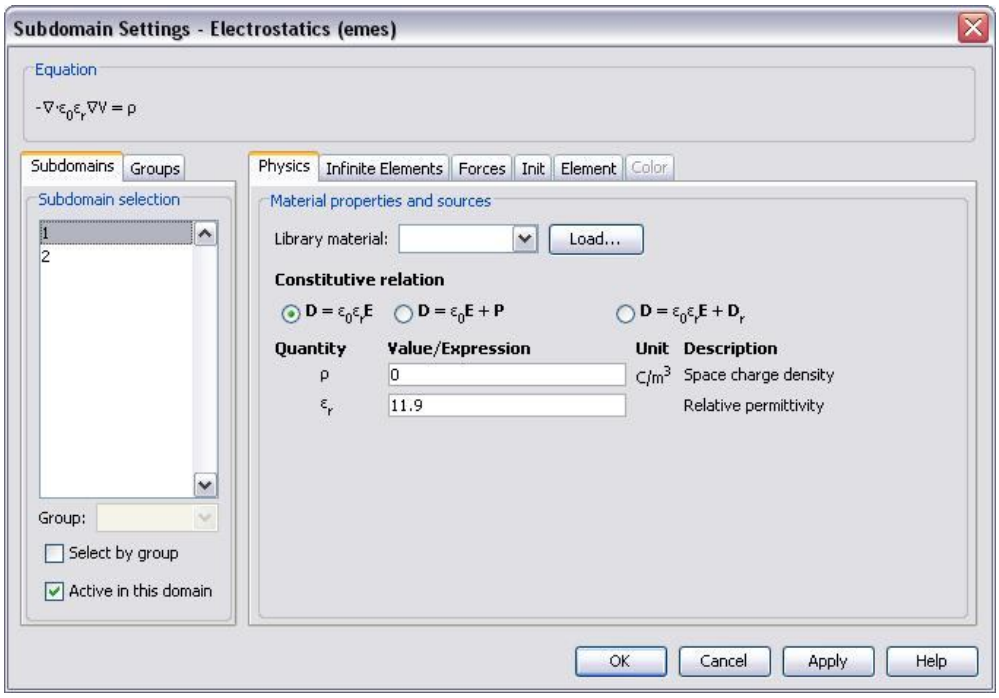


Figure 11

2. ***Boundary Conditions:***

   There are three boundary conditions that need to be defined. First is the ground, second is the port (voltage) and third is the symmetry or zero charge condition. To access boundary conditions, select **Boundary Settings** under the **Physics** menu.

Table 2

Settings	Boundary: 1-5, 7,12,13,52,53	Boundary: 8-11, 14-27, 42-46	Boundary: 28-41, 47-51
Boundary Condition	Symmetry/ Zero Charge	Ground	Port (voltage)

3. After applying the above boundary conditions, click on the **Port** tab and make sure the dialog box looks exactly like shown in figure 12.

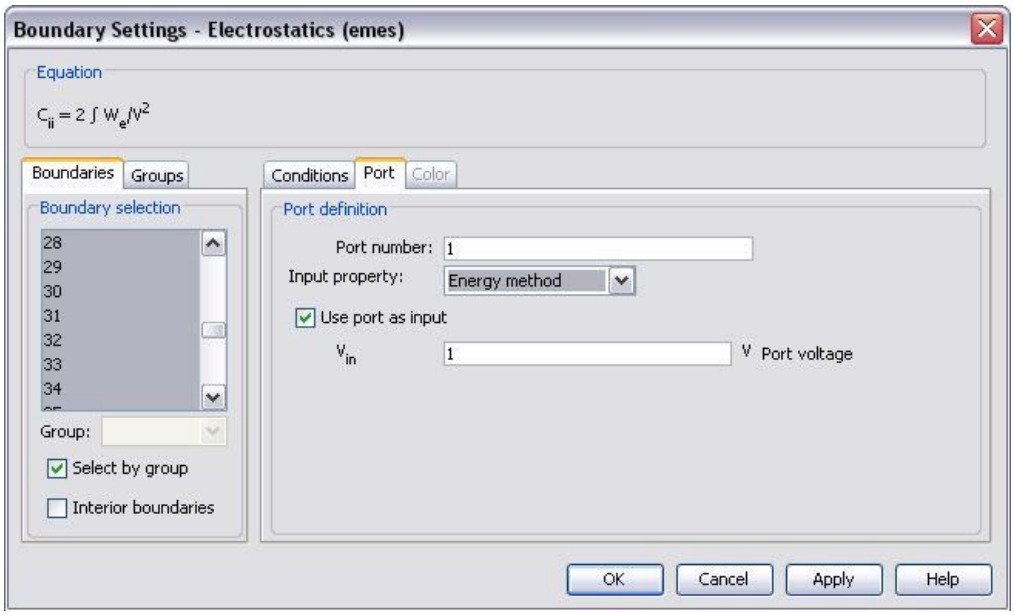


Figure 12

4. Make sure that the **Select by group** box is checked. Click on one set of Boundaries and make that it corresponds to the right comb. The following two figures show the comb geometries corresponding to different sets of Boundary conditions.

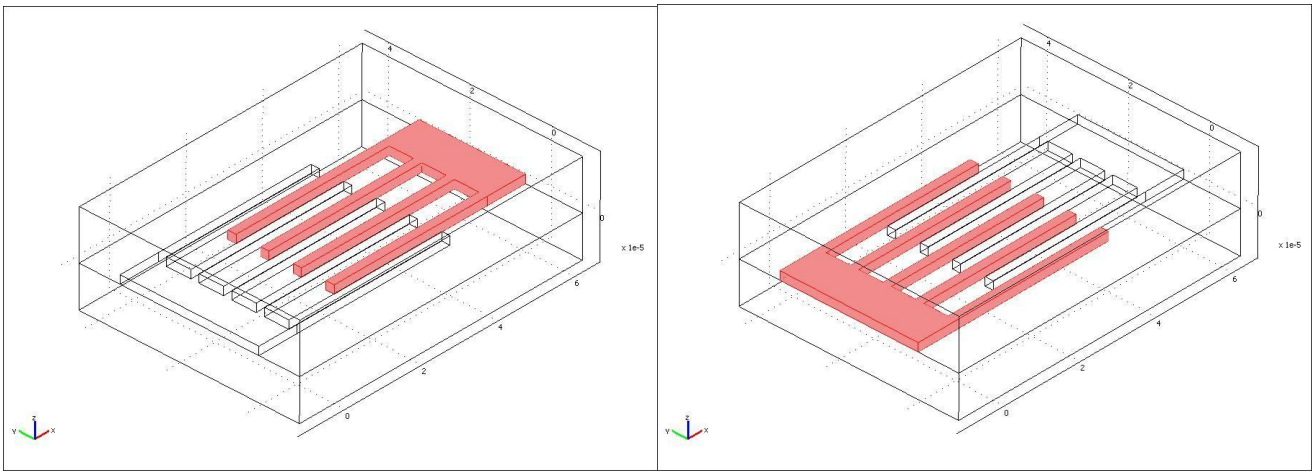


Figure 13: Left picture shows the comb that is grounded.

Right picture shows the comb that is a port.

E. Mesh Analysis

1. In order to do a Mesh Analysis, click on **Mesh → Initialize Mesh**.

2. The following Progress box appears:

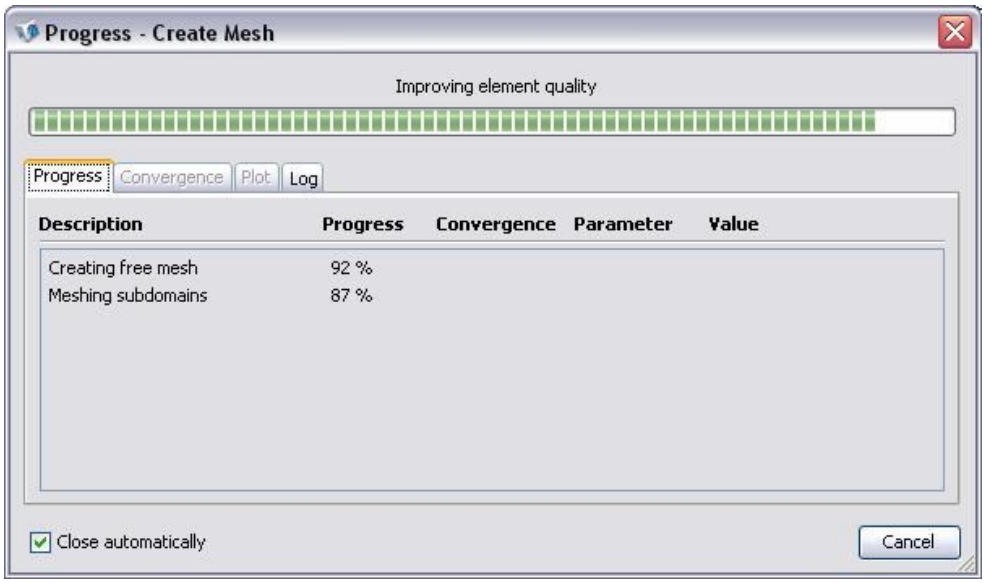


Figure 14

3. Once the Create Mesh Analysis is complete, the comb structure will look like the following figure:

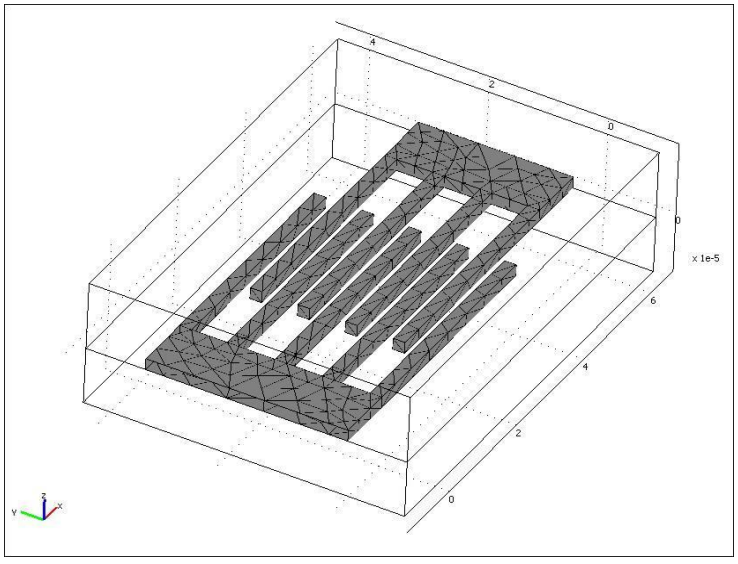


Figure 15

F. Solve Problem

1. In order to get a solution to the problem, click on **Solve → Solve Problem**

2. The following Progress box appears:

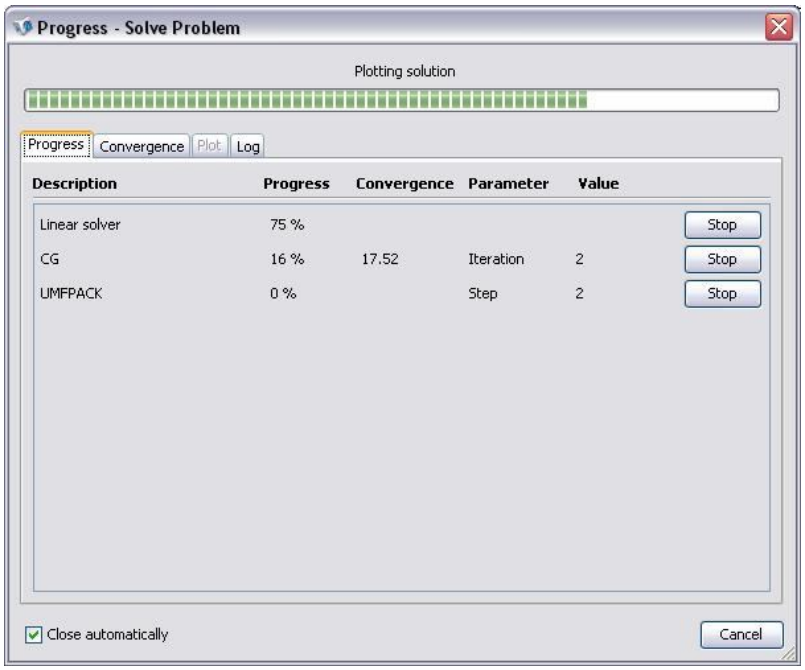


Figure 16

3. Once the Solve Problem Analysis is complete, the final capacitive sensor structure appears as follows:

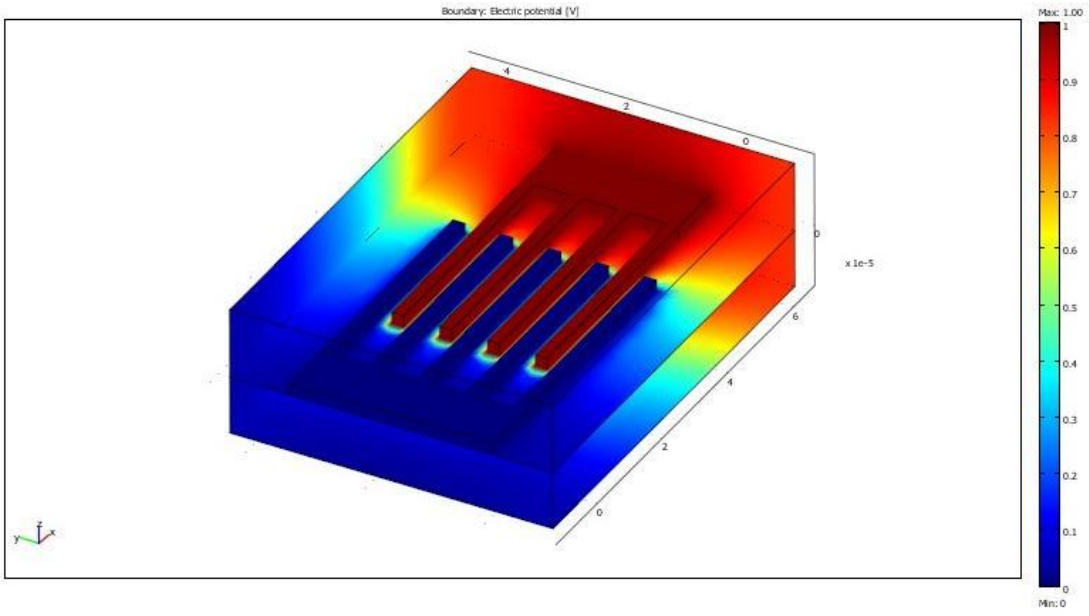
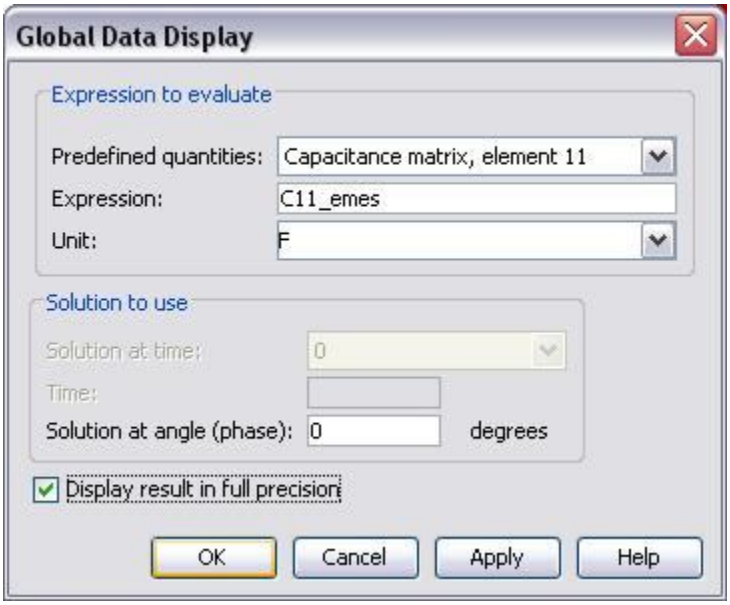


Figure 17: Red Color shows high voltage (1V) and blue color is low voltage (ground)

G. Compute Capacitance

1. In order to calculate the capacitance of the capacitive sensor, select **Postprocessing → Data Display → Global**. The following Global Data Display box appears:




```
Number of degrees of freedom solved for: 11708
Solution time: 0.594 s
Value: 2.1938850480665124E-14 [F], Expression: C11_emes, Phase: 0 degrees
```

Figure 18

2. *Figure 18 shows the final results:*

Capacitance = 2.19388 x 10^{-14} F = 0.0219388 pF

Material	Dielectric constant	Material	Dielectric constant
Air	1.0	Nylon	3.4-22.4
Amber	2.6-2.7	Paper (dry)	1.5-3.0
Asbestos Fiber	3.1-4.8	Paper (coated)	2.5-4.0
Epoxy Resin	3.4-3.7	Paraffin (solid)	2.0-3.0
Ethyl Alcohol	6.5-25	Plexiglas	2.6-3.5

(absolute)			
Fiber	5.0	Polystyrene	2.4-3.0
Formica	3.6-6.0	Quartz	5.0
Glass (electrical)	3.8-14.5	Quartz (fused)	3.78
Glass (Pyrex)	4.6-5.0	Rubber (hard)	2.0-4.0
Glass (window)	7.6	Styrofoam	1.03
Silicone (glass) (molding)	3.2-4.7	Teflon	2.1
Silicone (glass)	3.7-4.3	Titanium Dioxide	100
Soil (dry)	4.4	Vaseline	2.16
Mica (electrical)	4.0-9.0	Water (distilled)	34-78

Table 3: Dielectric Constant Table

Appendix G. References

Lee, Mark. Cypress Semiconductor Corp. "The Art of Capacitive Touch Sensing."*PlanetAnalog.com*. 06 Mar. 2006. Web. 18 Feb. 2010. <http://www.planetanalog.com/features/showArticle.jhtml?articleID=181401898>.

Brychta, Michael. "Measure Capacitive Sensors With A Sigma-Delta Modulator."*Electronicdesign.com*. 28 Apr. 2005. Web. 15 Feb. 2010. <http://electronicdesign.com/article/analog-and-mixed-signal/measure-capacitive-sensors-with-a-sigma-delta-modu.aspx>.

Fairchild Semiconductor. "MC78XX/LM78XX/MC78XXA 3-Terminal 1A Positive Voltage Regulator." *Datasheetanalog.org*. Fairchild Semiconductor, 2001. Web. 13 Mar. 2010. <http://www.datasheetcatalog.org/datasheet/fairchild/LM7805.pdf>.

Microchip Technology Inc. "28/40/44-Pin Enhanced Flash Microcontrollers with 10-Bit A/D and NanoWatt Technology." *Microchip.com*. Microchip Technology Inc., 2004. Web. 24 Feb. 2010. <http://ww1.microchip.com/downloads/en/DeviceDoc/39631a.pdf>.

Erlich Industrial Development, Corp. "Voltage Regulator." *Eidusa.com*. Erlich Industrial Development, Corp. Web. 13 Mar. 2010. <http://www.eidusa.com/Electronics_Voltage_Regulator.htm>.