EXPERIMENTAL EVALUATION OF INTRAVASCULAR FLOW METER AND MINIATURE RFID ANTENNAS USING MEMS PRESSURE SENSORS AND FLEXIBLE PCB TECHNIQUES

A THESIS IN
Electrical Engineering

Presented to the Faculty of the University
of Missouri-Kansas City in partial fulfillment of
the requirements for the degree

MASTER OF SCIENCE

by
ERIK JOSEPH TIMPSON P.E.

B.S.E.E., University of Missouri-Rolla, 2004

Kansas City, Missouri
2011

EXPERIMENTAL EVALUATION OF INTRAVASCULAR FLOW METER AND MINIATURE RFID ANTENNAS USING MEMS PRESSURE SENSORS AND FLEXIBLE PCB TECHNIQUES

Erik Joseph Timpson, Candidate for the Master of Science Degree

University of Missouri-Kansas City, 2011

ABSTRACT

A testbed for flow meters was built and validated. The testbed targeted intravascular pressure and flow monitoring applications. The testbed consists of a data acquisition system and two MEMS pressure sensors (SM5108). For simplicity the evaluation of the testbed was performed using air flow. Air was pumped across a simulated blockage and pressure was built up to intravascular pressure ranges (35 to 230 mmHg). The base fractional flow rate (FFR) equation was used to calculate flow, hinting that the SM5108 pressure sensors can become part of an implantable flow meter. In addition, small and flexible RFID coil antennas were designed, built and tested. Measurements show that enough power can be collected by the coil antennas to power a microchip.

The faculty listed below, appointed by the Dean of the School of Computing and Engineering have examined a thesis titled "Experimental Evaluation of Intravascular Flow Meter and Miniature RFID Antennas Using MEMS Pressure Sensors and Flexible PCD Techniques," presented by Erik J. Timpson, candidate for the Master of Science degree, and hereby certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Walter D. Leon-Salas, Ph. D., Committee Chair
Department of Computer Science and Electrical Engineering

Reza Derakhshani, Ph. D.
Department of Computer Science and Electrical Engineering

Deb Chatterjee, Ph. D.
Department of Computer Science and Electrical Engineering

CONTENTS

ILLUSTRATIONS

ACKNOWLEDGEMENTS

I want to start by thanking God for everything, especially for my wife Angela who is an un-measureable source of joy in my life. She has given me more love than I thought possible and gave birth to our son Emery during the course of my Masters studies. These two things combined with all her other little deeds, smiles and laughs move me to treasure every moment we have together. I truly thank her for sacrificing so many moments we could have had together so that I may write this thesis.

This thesis would also not be possible without Karl and Pam, my parents. I had no idea how much they loved me until I looked into the eyes of my son. No doubt it was this love that carried them through the long nights of watching Emery awaiting my arrival. They worked just as hard as I did to get this thesis done.

Like them, I can honestly say seeing Emery go through his first year of life has been the most rewarding experience I have ever had - bar none. Therefore, it is also appropriate that I thank my greatest inspiration – Emery Gold Timpson.

I would like to thank my Advisor, Dr. Leon, for providing such a rich opportunity to help people. He is the type of professor I want to be like. He is energetic, genuinely intrigued and excited about research opportunities and works tirelessly to see new paths are explored daily. He teaches socratically with patience and a gentle hand always willing to draw a picture in order to open a mind. I'm grateful I had the opportunity to learn from him and look to the discoveries we have yet to make together.

CHAPTER 1

INTRODUCTION

1.1 Cardiovascular Diseases

In 2007 heart disease was the leading cause of death in the United States of America [1]. Death rates reached an age adjusted average of 747 annual deaths per 100,000 as seen in Figure 1.1. These deaths are not just tragic in and of themselves but also create significant economic demand.

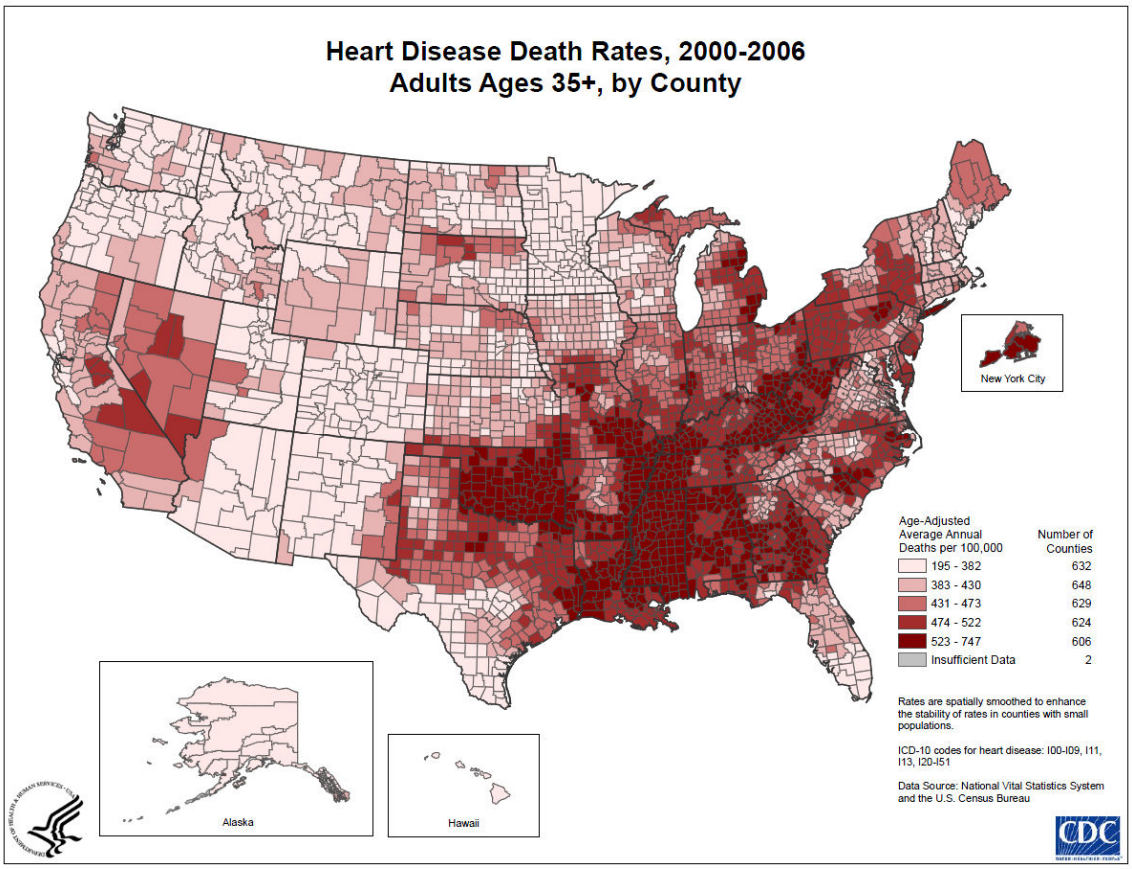


Figure 1.1: Heart disease death rates, 2000-2006 [2]

The cost of heart disease to the nation's economy, as adjusted by a review of the 1957-1958 data, totals $4.1 billion [3]. In 2010, the total costs of cardiovascular diseases in the United States were estimated to be $444 billion [4]. Take a moment and consider what it will be in 2060. The serious problem of heart disease has deservingly been given a tremendous amount of investigation. The investigation has lead to many preventative measures, diagnosis techniques and treatment options.

1.2 Typical Treatment Options

Treatment options fall into three categories: (1) non-invasive, (2) minimally-invasive, and (3) surgical procedures [5]. Non-invasive measures include medication, diet and exercise. Minimally-invasive includes interventional cardiology. Surgical procedures include coronary artery-bypass surgery. While none of these three provide a cure, each one does eliminate or alleviate the symptoms for varying amounts of time. In other words, the disease and its causes are still present after treatment and require the patient to modify his or her lifestyle to prevent the disease from progressing and the symptoms from recurring [6].

1.3 Problems Associated with Treatment Options

When considering interventional cardiology and the fact that life style changes are not easy, the rates of restenosis are not staggering. Stenosis is defined as the abnormal narrowing of a passage in the body, and used as a medial term for a blockage in an artery. In layman's terms, restenosis is when the artery becomes blocked again. Restenosis can be defined as a reduction in the circumference of the lumen of an artery or blood vessel of 50% or more. Restenosis has a high incidence rate (25-50%) in patients who had undergone balloon angioplasty, with the majority of patients needing further angioplasty within 6

months [7]. Prevention of restenosis can be done in two stages. The first stage is medication typically focused on anti-platelet drugs (lessens blood clotting). The second stage of prevention is to use a drug-eluting stent. There has been some success with these new stents in reducing the occurrence of restenosis, with clinical studies showing an incidence rate of 5% or lower [7].

Even with the hope that restenosis is a thing of the past the problem of measuring a blockage, the restricted flow, and the arterial pressures still involve a hospital stay.

1.4 Creative Proposal to Solve the Treatment Problems

One of the most creative solutions would be to implant a stent that has the ability to self-diagnose restenosis. In a survey of stent designs, the authors suggest Figure 1.2 to be the structure of classifications of stent design features [8]. The ability to self-diagnose restenosis should now be added to the ADDITIONS section of the pyramid.

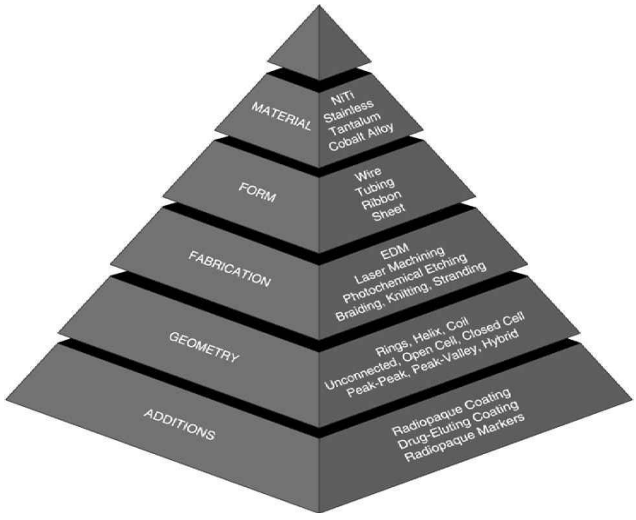


Figure 1.2: Stent design pyramid [8]

The wireless electronic self-diagnose chip would monitor for restenosis and communicate with the doctor at regular checkups. The monitoring could be done with MEMS pressure sensors. The wireless communication could leverage Radio Frequency Identification (RFID) technology. The self-diagnosing chip would allow the physician to only have to read on their computer the restenosis status rather than having to do the current complex and time consuming methods of restenosis diagnosis. The ability to do all this would easily prove useful.

1.5 Previous Work Supporting the Proposal

The most relevant paper to this thesis was written by several engineers at the University of Michigan, Ann Arbor titled "A Wireless Microsensor for Monitoring Flow and Pressure in a Blood Vessel Utilizing a Dual-Inductor Antenna Stent and Two Pressure Sensors." That paper shows progress in the direction of a wireless implantable system for sensing flow and pressure inside a blood vessel [10]. The paper shows the use of the system but makes no reference to the uncertainties of their measurements and moreover does not set reasonable expectations for a system designer to meet. In that paper the pressure sensor that is used is capacitive which has noticeable differences from available resistive MEMS pressure sensors.

Another relevant paper to this field of study was written by several engineers at the University of California. A fully implantable wireless pressure sensor system was developed to monitor bladder pressures in vivo [11]. This paper uses the same pressure sensor in this thesis but in a different way (bladder not blood pressure). The authors claim resolution of .02 psi (about 1 mmHg), however the entire sensing system has a sizeable package (Figure 1.3).

This fully implantable package is obviously too large for intravascular purposes; inasmuch, a wireless RFID technology would be more reasonable to consider for diagnosing restenosis.

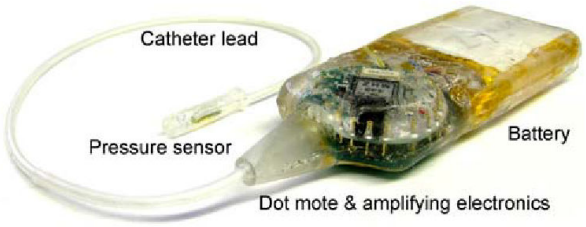


Figure 1.3: The implantable device after being fully packaged [10]

A more reasonable size of implantable device was proposed by Oklahoma State University. Though not published, the Oklahoma State VLSI department is working on a prototype neural RFID IC. The overall dimensions of this IC are 5mm x 14mm x 1mm. Though this is more reasonable, a finial size of less than this, say 1mm x 4mm x .5 mm is necessary to not be a blockage itself.

1.6 Plan of Development

There is one very clear problem that comes after considering the proposed wireless microelectronic chip and sensor and previous work. Will this device have the accuracies suitable for a physician's diagnosis? This is the primary focus of this thesis; a question that needs to be answered before building a stent that has the ability to self diagnose restenosis. The process of building a stent with said ability is vast and varied. A secondary question is can a miniature RFID collect enough power for a microchip. By leveraging current capabilities one could divide the process of building a wireless electrical self diagnosing chip into three phases. (1) Build a test bed and measuring device, of reasonable size, that has the

capability to produce and measure intravascular pressures with the desired accuracies. (2) Consolidate and miniaturize the test bed and device produced in phase one. (3) Make the phase two device biocompatible and begin the long road to human implementation.

1.7 Contributions

Phase one in the plan of development has begun. A brief description can be found in an IEEE Sensors conference paper [11]. This thesis is meant to be a more thorough and complete documentation of the effort. The contributions can be segmented into 5 parts: (1) research MEMS sensors; (2) research flow measuring techniques; (3) development a data acquisition board (4) calibration and experimentation; and (5) uncertainty analysis.

Many types of pressure sensors exist. Navigating them for a particular application proves challenging. The decision to utilize available MEMS pressure sensors is logical based on their small size and time savings in comparison to an exhaustive search of all available pressure sensors. That said some searching shows that the sensors investigated are very competitive to others in a comparable size.

Flow measuring techniques were researched which lead to the decision to use Fractional Flow Rate (FFR). Dr. Steve Marso, a practicing physician at St. Luke's hospital clearly outlined many options for measuring flow through arteries. Dr. Marso suggested FFR due to it being widely used and documented. Fortunately pressure readings directly relate to FFR lending FFR useful in terms of the sensors chosen.

The output of the sensors needs to be amplified, converted to digital format, stored and displayed. Figure 1.4 shows the sensor front-end and a microcontroller to acquire pressure readings from two MEMS pressure sensors.

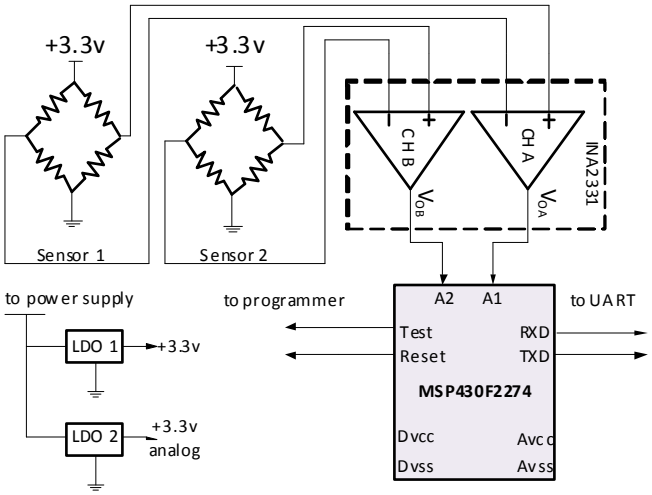


Figure 1.4: Schematic of pressure data acquisition system [11]

The microcontroller sends the pressure data through its UART to a PC. At the PC a Matlab Graphical User Interface (GUI) program receives, filters and displays the pressure data.

Developing the testing environment was a challenge in and of its self. To begin with, the wire-bonded MEMS pressure sensors were fragile. Being fragile meant that great caution is necessary to insure no damage was done. A repeatable and measurable intravascular pressure and flow environment was created and tested. The calibration of this environment was also performed each time before taking measurement.

Last, a complete uncertainty analysis was performed to insure the measurement were accurate. The Guide to expressing Uncertainty in Measurement (GUM) method was used. The final uncertainty obtained was compared to the physician's expectation.

CHAPTER 2

BACKGROUND

2.1 Introduction

This chapter focuses on introducing the reader to pressure sensors, flow measurements, medical instrumentation, Fractional Flow Rate (FFR), physicians expectations, and the uncertainty target of a FFR measuring device. Introduction to all of these areas is critical in understanding the logic behind a FFR measuring device. Each topic serves a different purpose that will be identified below.

2.2 Pressure Sensors

There are many types of pressure sensors. Pressure sensors come in all shapes sizes and use various methods for measuring pressure. There are three ways to refer to pressure measurements: gage, absolute and differential. Gage examples include tire pressure, blood pressure, and pressure of air tanks. Absolute examples are vacuum pressures, atmospheric pressures, and altimeter pressures. Differential pressure measurements are typically for flow or level in pressurized vessels.

The difference between Gage and Absolute is simple. Gage is the pressure relative to the current atmospheric pressure (no matter what atmospheric pressure is). Absolute is what the pressure is from a true vacuum. To illustrate, blood pressure is typically 120 mmHg over 80 mmHg (gage) or 196 mmHg over 156 mmHg (absolute). The conversion is to add atmospheric pressure (typically 76 mmHg).

Differential pressure measurements need not consider atmospheric pressure since an offset to both values would be subtracted out. Here one must consider why there is a difference. Tube architecture could cause a difference as in a differential pressure flow meter. A restriction could cause a difference as in Bernoulli's principle. Once one knows why there is a difference one can determine what the magnitude of that difference means (whether it be flow or level of a pressurized vessel).

MEMS pressure sensors currently dominate the market for greater-than-atmospheric-pressure sensors [12]. A MEMS pressure transfers a pressure change into mechanical motion. This motion can then convert into a resistance or capacitance change or even through a piezoelectric and directly into an electrical signal. Figure 2.1 shows the applied pressure resulting in a diaphragm strain. This strain is then converted into a resistance difference. The SM5108 has a linier relationship between pressure and resistance.

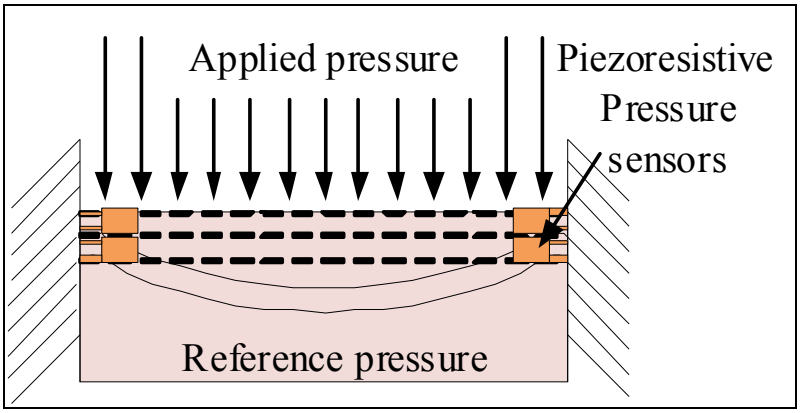


Figure 2.1: Diagram of a MEMS pressure working.

When the diaphragm is round with a clamped circular plate with small deflections the equation to model the deflection (w), radial distance from the center of the diaphragm (r), diaphragm area (a), applied pressure (P) and flexible rigidity (D) is given in 2.0.

$$w(r) = \frac{Pa^4}{64D}\left[1 - \left(\frac{r}{a}\right)^2\right]^2 \qquad (2.0)$$

Interfacing with a MEMS device is a challenging endeavor especially since the sensors used are not packaged to keep their low profile. The pressure sensors have five aluminum pads that must be wire bonded. There are two materials for wire bonding, gold and aluminum. There are two main types of wire bonding, ball and wedge. As it turned out the pressure sensors would only allow a gold or aluminum ball bond when mounted with lock tight to a stiff ceramic (Figure 2.2).

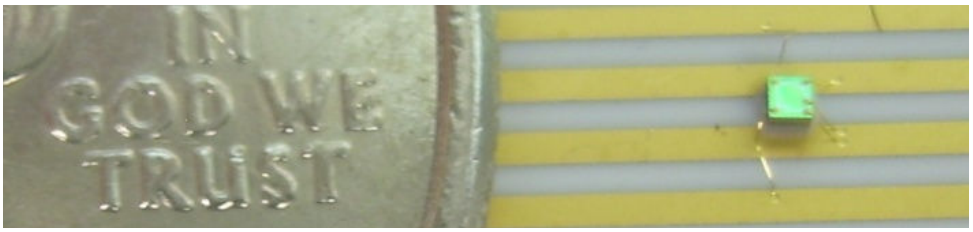


Figure 2.2: Wire bonded pressure sensor next to U.S. dime.

2.3 Flow Measurement and Medical Instrumentation

One of the primary measurements the physician would like to acquire from a patient is that of the concentration of O_2 and other nutrients [13]. Because this is difficult to measure, doctors may choose the second class measurement of blood flow [13]. Measuring blood flow is not easy due to its invasive nature. There are many ways to measure flow. The two most common ways are coronary angiography and catheter FFR.

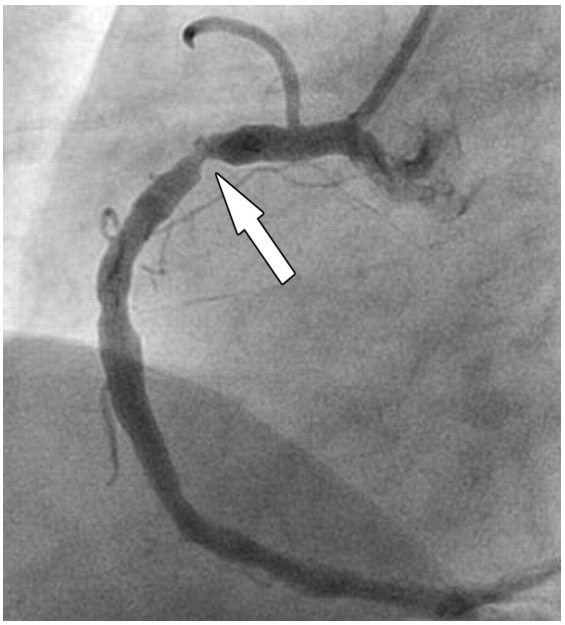


Figure 2.3: Example of coronary angiography [14]

Coronary Angiography is a test that uses dye and special X-rays to show the inside of your coronary arteries [15]. Coronary Angiography is the gold standard for diagnosing blockages. Figure 2.3 shows a clear picture of a blockage in 56-year old man with chest pain [14]. In order to get the die in the blood a procedure called cardiac catheterization is used. A long, thin, flexible tube called a catheter is put into a blood vessel. The tube is then threaded into your coronary arteries and the die is injected [15].

A second way of determining flow is also using cardiac catheterization but this time instead of just injecting die from afar, that catheter goes directly into the blockage and pressure is measured proximally and distally. When both these pressures are known one can calculate Fractional Flow Rate (FFR) with equation 2.1.

$$FFR = \frac{P2}{P1} \tag{2.1}$$

P2 is the distal pressure and P1 is the proximal pressure. When there is no blockage the pressures are equal and FFR = 1. When a blockage begins, FFR moves down to 0.9 and then at 0.8 some doctors will choose some sort of intervention. When the blockage is below .8 most doctors will act to increase flow.

2.4 Current FFR Measurement Apparatuses

There are two primary manufactures of FFR measuring equipment: St Jude Medical and Volcano [16]. Both use sensors at the end of a long catheter (Figure 2.4 and 2.5). Both have a computer interface that a doctor would use to determine the severity of a blockage. Most importantly, as previously mentioned, both require invasive surgery whenever a doctor would like to take an FFR measurement.

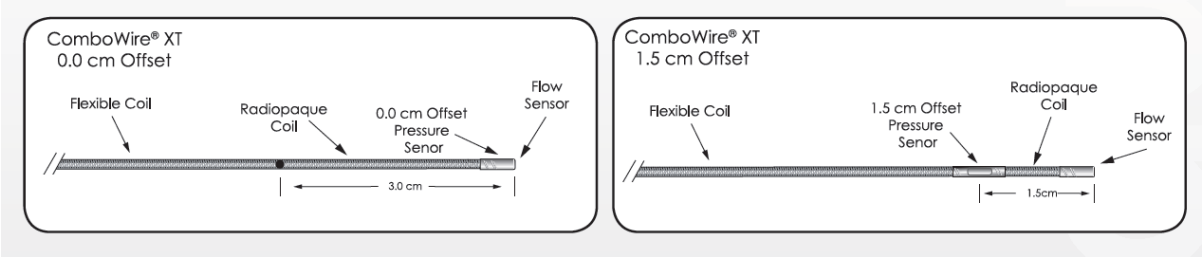


Figure 2.4: Example of Volcano sensor tip [17]

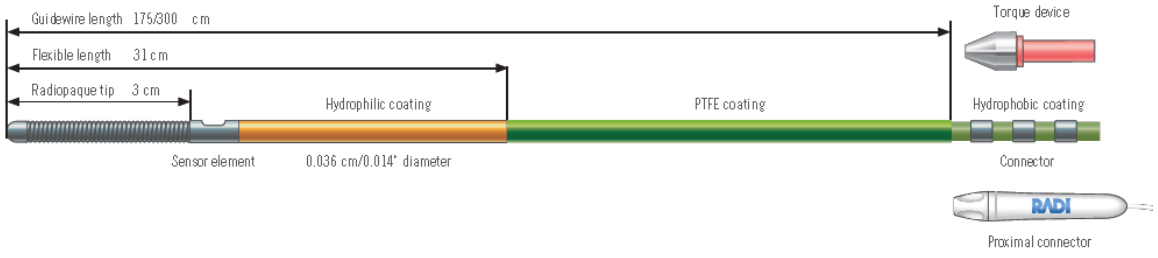


Figure 2.5: Example of St Jude Medical sensor tip [18]

The process of using said devices is simple in nature but requires great skill. First, a catheter is inserted into the femoral or radial arteries using a guide wire. Second, the catheter is guided to the blockage (Figure 2.6). Third, the pressure is measured both proximally and distally. Last FFR is calculated per equation 2.1. This calculation is usually done by the computer and displayed for a doctor to see (Figure 2.7).

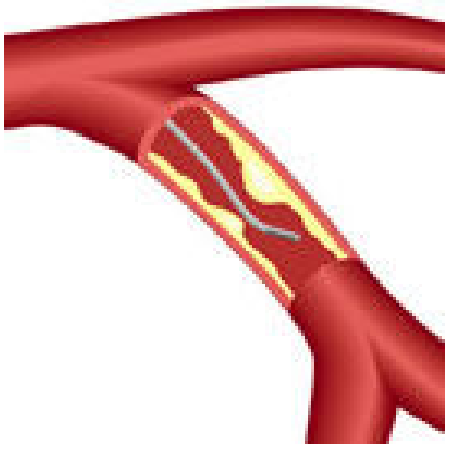


Figure 2.6: Catheter in place measuring FFR [16]

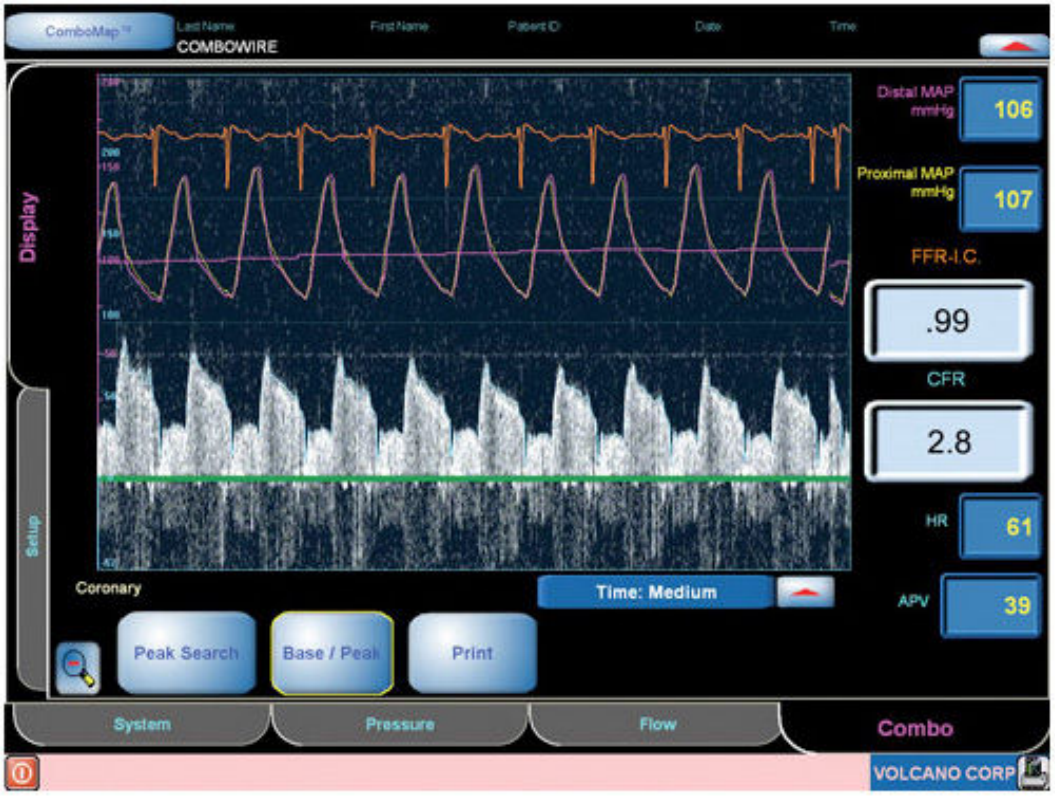


Figure 2.7: Software screen of Volcano's FFR Device [17]

2.5 Physician's Expectations - The Uncertainty Target of FFR

Medical Instrumentation has a very wide verity of measurement uncertainties. Physicians obviously want the best measurement possible when life is on the line. Physicians also want to be as least invasive as possible; however, the less invasive the less accurate. Here in lies the notorious engineering balance. An interview with a leading cardio-vascular specialist, Dr Steve Marso, hinted that an acceptable uncertainty target for FFR is 10 percent.

CHAPTER 3

THE DESIGNED AND IDENTIFIED SOLUTION

3.1 Introduction

To restate the primary focus of this thesis from Chapter 1, we must build a device and testbed and then investigate if the device has the accuracies suitable for a physician's diagnosis. In the process of designing anything one must find the engineering balance. The engineering balance is found when there is a balance between the boundary conditions. For this example there are 3 boundaries: power, size and accuracy. The design must be low power it cannot be powered on batteries and must be powered inductively. The design must be small so that it is able to fit in a stent without being a blockage itself. The design must meet or exceeded the accuracies that a physician needs.

The primary focus of the research is to find a system that would meet the required accuracy. This system needs a data acquisition interface, and it needs to be calibrated. The experimental setup around the solution needs to simulate a blockage, and the heart. The secondary focus of this research is to design, build and test a miniature RFID coil antenna that is capable of collecting enough power for a microchip. In the upcoming sections a more detailed view of the antenna and applicable results will be discussed.

3.2 Data Acquisition

Many types of data acquisition exist. They range in price and accuracy. When considering the balance of price and accuracy, if one happens to be skilled in microcontroller programming, a microcontroller is an obvious choice. The final system evaluated is based on

a microcontroller and a Matlab Graphical User Interface. The microcontroller added a significant power savings over the high accuracy bench top data acquisition system used initially, i.e. Agilent 3458 DMM (Figure 3.1)

Figure 3.1: (Top) The MEMS sensor in a closed pressure chamber being read by an Agilent 3458 DMM. (bottom) Pressure chamber opened to expose the MEMS sensor

Initially high accuracy bench top equipment was used to evaluate the MEMs pressure sensors. Pressure was accurate to .04% and resistance was accurate to .05%. This system showed the linearity of the pressure sensor. The R^2 value was one. The MEMs sensor also showed no Hysteresis. Figure 3.2 has the data taken from this system. As mentioned earlier, the first system was too power hungry and needless to say huge.

Increading		Decreasing	
PSIA	Resistance	PSIA	Resistance
14.572	94.731	20.12	132.328
15.176	98.861	19.02	124.899
15.885	103.622	18.105	118.741
17.02	111.367	17.118	112.026
18.012	118.067	16.018	104.617
19.065	125.201	15.118	98.555
20.122	132.348	14.542	94.594

Figure 3.2: A table of measured values of the system in Figure 3.1

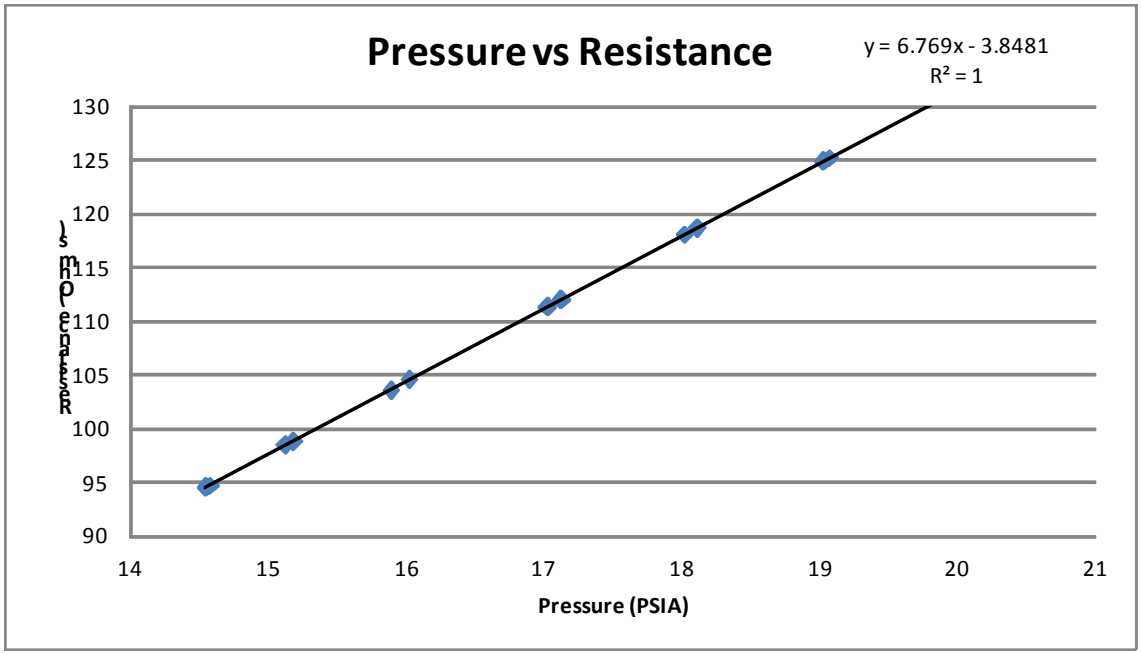


Figure 3.3: The values graphed with a trendline, equation, and calculated R^2 value.

In the end it is obvious the system will need to be on a single chip. The midpoint between the bench top described above and a single chip is shown in Figure 3.4. This circuit board was made to show that commercially available microelectronics can be used to make accurate measurements with the MEMS sensors used above, as shown on Figure 1.4.

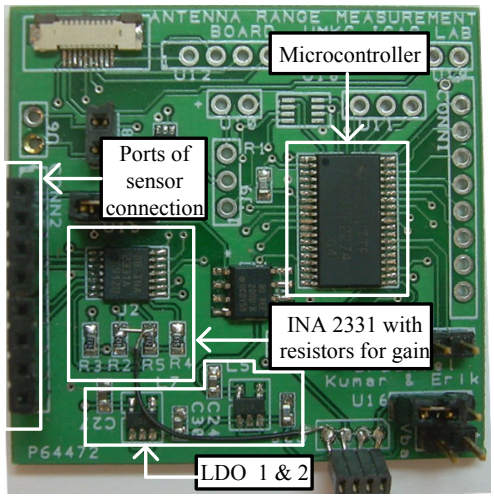


Figure 3.4: The finial printed circuit board based off figure 1.4 system block diagram.

The pressure sensor contains four piezo resistors connected in a bridge configuration. Each bridge has four wires that need to be connected i.e. Vdd, Gnd, V_{diff}+ and V_{diff}-. Thus with two sensors, eight wires leave the board where it is labeled "ports of sensor connection." The differential voltage from the sensors is feed into the INA 2331 instrumentation amplifier. The model of the amplifier is shown in Figure 3.5. The gain of said amplifier is given by equation 3.1.

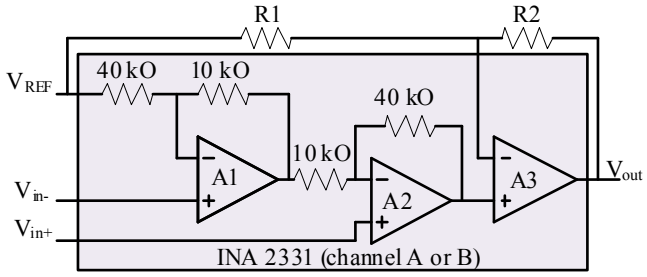


Figure 3.5: Schematic of INA2331 Instrumentation Amplifier.

$$Gain = 5 + 5 \times \left(R2 / R1 \right) \qquad (3.1)$$

After amplification the signal is converted from analog to digital by the microcontroller. After conversion the digital value is sent through the UART to a computer that has a Matlab GUI program running. The GUI is shown in Figure 3.6. One can clearly see the communication control box that connects to the appropriate communications port. The Data control box contains the buttons to begin capturing the data (Start), stop capturing the data (Stop), save the data to an excel file (Save), and clear the data (Clear). With that functionality the data acquisition system is considered complete.

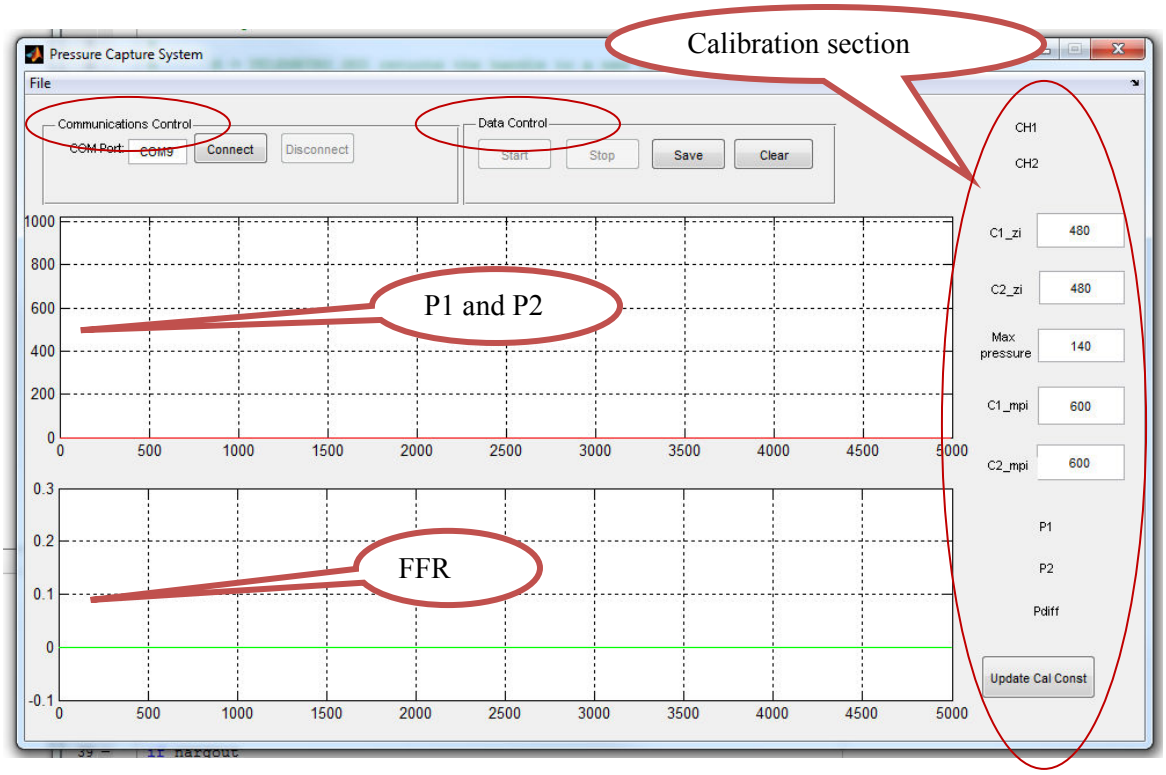


Figure 3.6: Screen shot of the Matlab Pressure Capture System GUI

3.3 Calibration

Calibration is defined as the transformation of the 12-bit integer value sent by the microcontroller into a pressure value. The calibration process requires only two steps. Step one, the two pressure sensors are left at atmospheric pressure and the integers from Chanel 1 and Chanel 2 are read and recorded (from the GUI). These values will be Zero Integer one and two (ZI1 and ZI2). The second step is to bring both pressure sensors to the same known maximum pressure as seen in Figure 3.7. These values will be the Maximum Integer one and two (MI1 and MI2). The maximum pressure value is recorded as the last of the five calibration coefficients (MP).

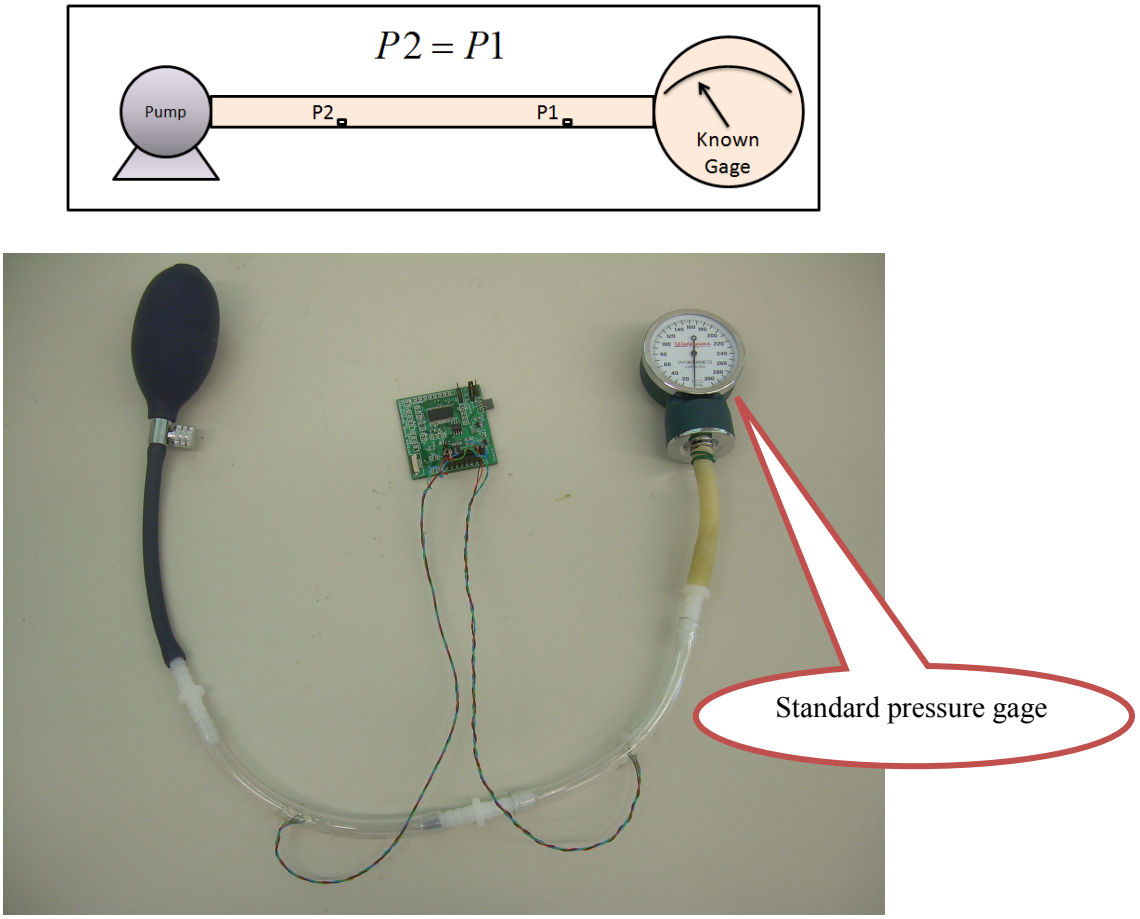


Figure 3.7: Showing the two pressure sensors at the same pressure.

With all five calibration coefficients and assuming a linear pressure to integer relationship, one can convert from an integer representation to pressure using equation 3.2. This conversion could be done in the microcontroller but it is done just as easy in Matlab.

$$P = (I - ZI) * \left[MP \Big/ (MI - ZI) \right] \tag{3.2}$$

The process of calibration is displayed on the right side of the Matlab GUI. The integer values are displayed at the top right hand side. After one knows the values one can place them in the correct boxes top to bottom (ZI1 – MP2). Then when the Update calibration constants (Update Cal Const) button is pressed, the GUI will start to display to correct pressures for channel one and channel two and their difference (above the Update Cal Const button).

3.4 Simulating Blockages

Now that we have two working MEMS pressure gages we can insert them into quarter inch tubing and place a valve between them to simulate a blockage. This can be visualized with Figure 3.8

There must be a back pressure for the experiment to work. Experimentally the back pressure was achieved with a latex tube and clamp. If one refers back to equation (1.1), FFR can now be easily calculated as the blockage increases over time.

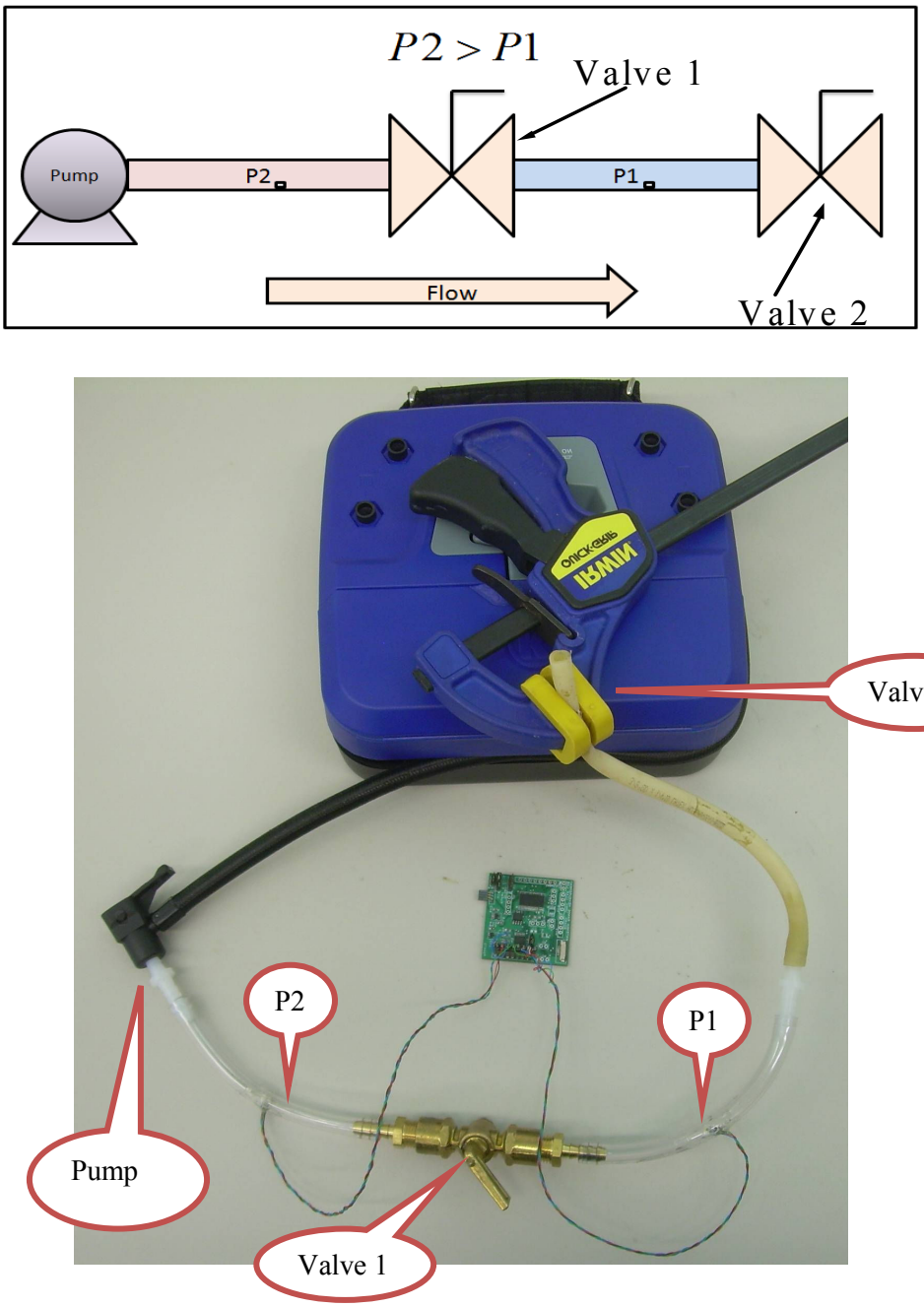


Figure 3.8: Showing the experiment, Valve 1 simulated blockage and Valve 2 creates the back pressure.

3.5 Simulating the Heart

Just before Valve 2, one can also insert an artificial heart. An artificial heart varies pressure approximately 60 beats per minute. Blood pressure is usually given in two numbers systolic and diastolic. These values can vary wildly in people. Figure 3.9 has some good approximations of maximum and minimum. By changing the pressure from maximum to minimum in at comparable time as the heart, would show an even closer example of intravascular pressures.

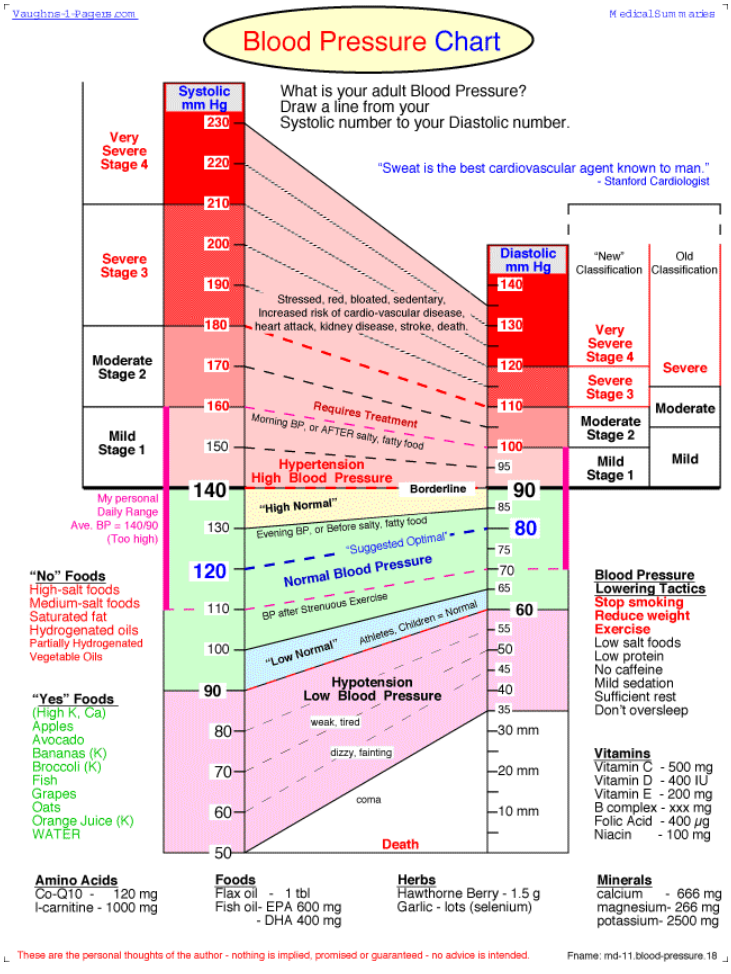


Figure 3.9: Blood pressure ranges [20]

Figure 3.10 shows ECG, arterial pressure (Pa), right arterial pressure (Pra), pulmonary artery pressure (Ppa), LV area (LVA), and airway pressure (Paw) recorded over time following apnea in a patient after cardiac bypass surgery during the closed chest condition [21]. If one could mimic this (Pa) curve that would be the closest to an intravascular environment. Figure 3.10 graphically represents pressure changes as compared to the heart beat.

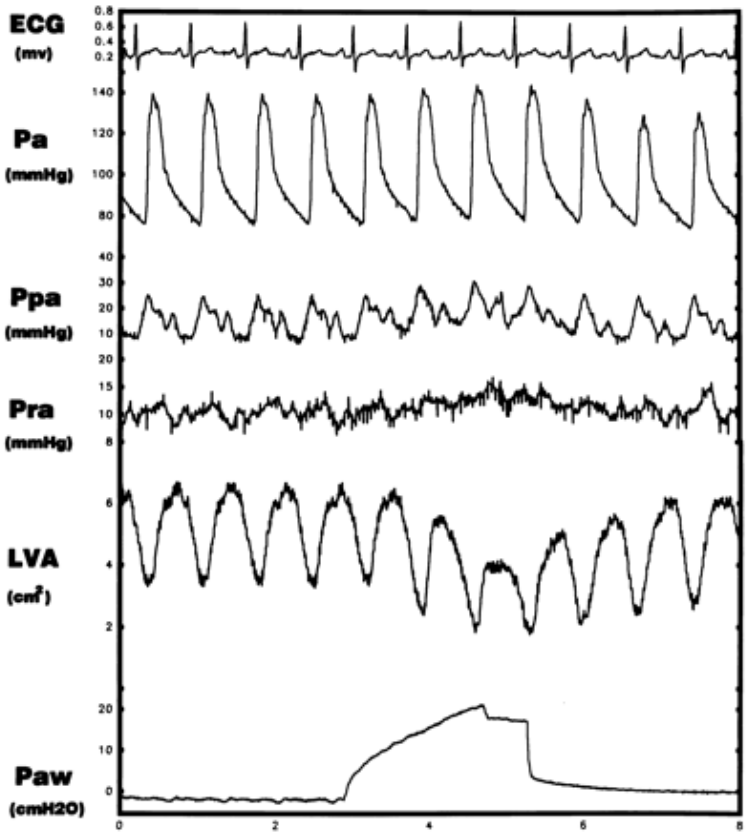


Figure 3.10: Blood pressure vs time and compared to other parameters [20]

3.6 Antenna Design and Manufacture

In order to power the final chip, a coil antenna will be used. A coil antenna tuned to 13.56 MHz (ISO15693 RFID standard) collects electromagnetic energy in the near field of a transmitting antenna. Initial variations of these miniature RFID antennas were designed and fabricated. Figure 3.11 shows what the coils look like before being cut out with a laser. Figure 3.12 shows the reverse side where the chip and cap and sensors will be soldered or bonded. Two types of coils were manufactured – square and rectangle. Square is an good choice as it maximizes the loop area. A rectangle would fit better inside a stent. Flexible PCB materials enabled the coil to bend around the stent to minimize impact of flow. For development purposes some of the coils have 10 contact flexible cables stemming off to interface with the board in Figure 3.4.

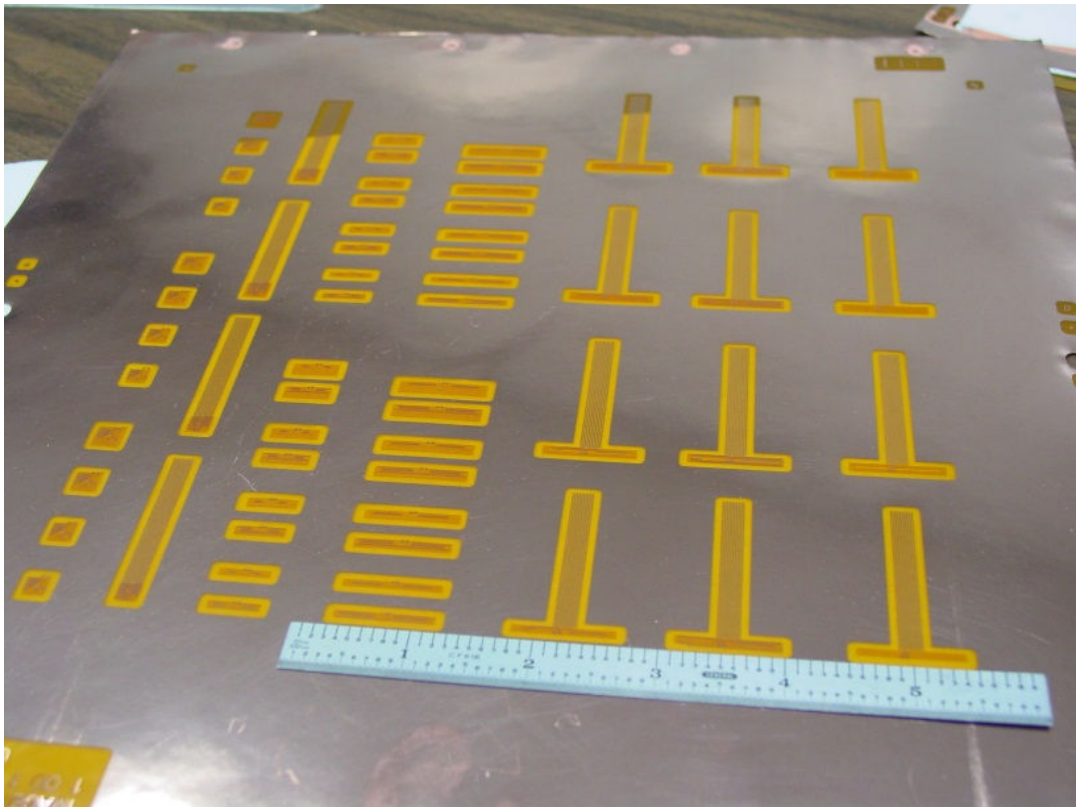


Figure 3.11: Picture of antenna before laser cutting

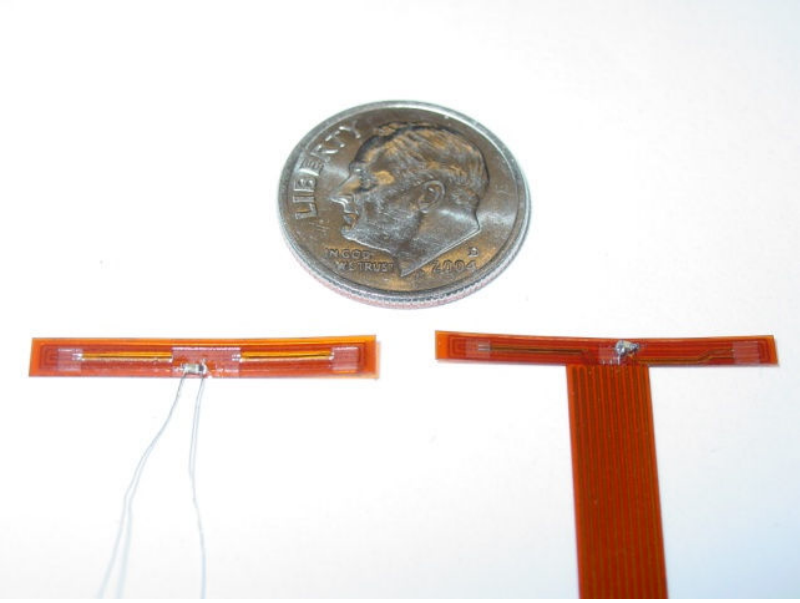


Figure 3.12: Picture of circuit side next to U.S. dime.

In order for the coil antenna to work it must be tuned. The antennas were tuned using three different techniques: Impedance Analyzer, Network Analyzer, and brute force. All three gave similar capacitance values to counteract the reactance of the coil antenna.

3.7 Results and Discussion

The first experiment was using an air pump to provide continual flow and turning the valve twice (moderate blockage and sever blockage). A severe blockage is one that results in an FFR of below 0.8. In Figure 3.13 one can see the blockage becoming worse in two steps. This blockage ends at a point where intervention would be necessary.

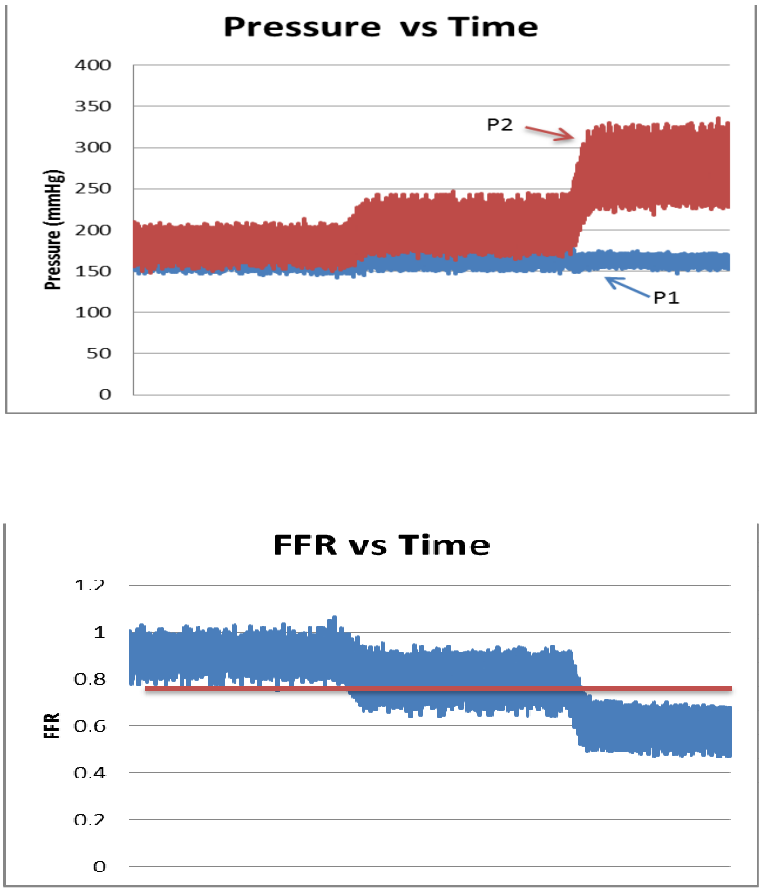


Figure 3.13: (Top) Pressure as blockage increases (Bottom) FFR as blockage increases with continual flow. Red line denotes medical intervention is necessary.

The second experiment was using a pump and a heart simulator to provide variable flow by turning the valve twice (moderate blockage and severe blockage). A sever blockage is one that results in an FFR of below 0.8. In Figure 3.13 and 3.14 one can see the blockage becoming worse in two steps. The last third of the graph show the point where intervention would be necessary.

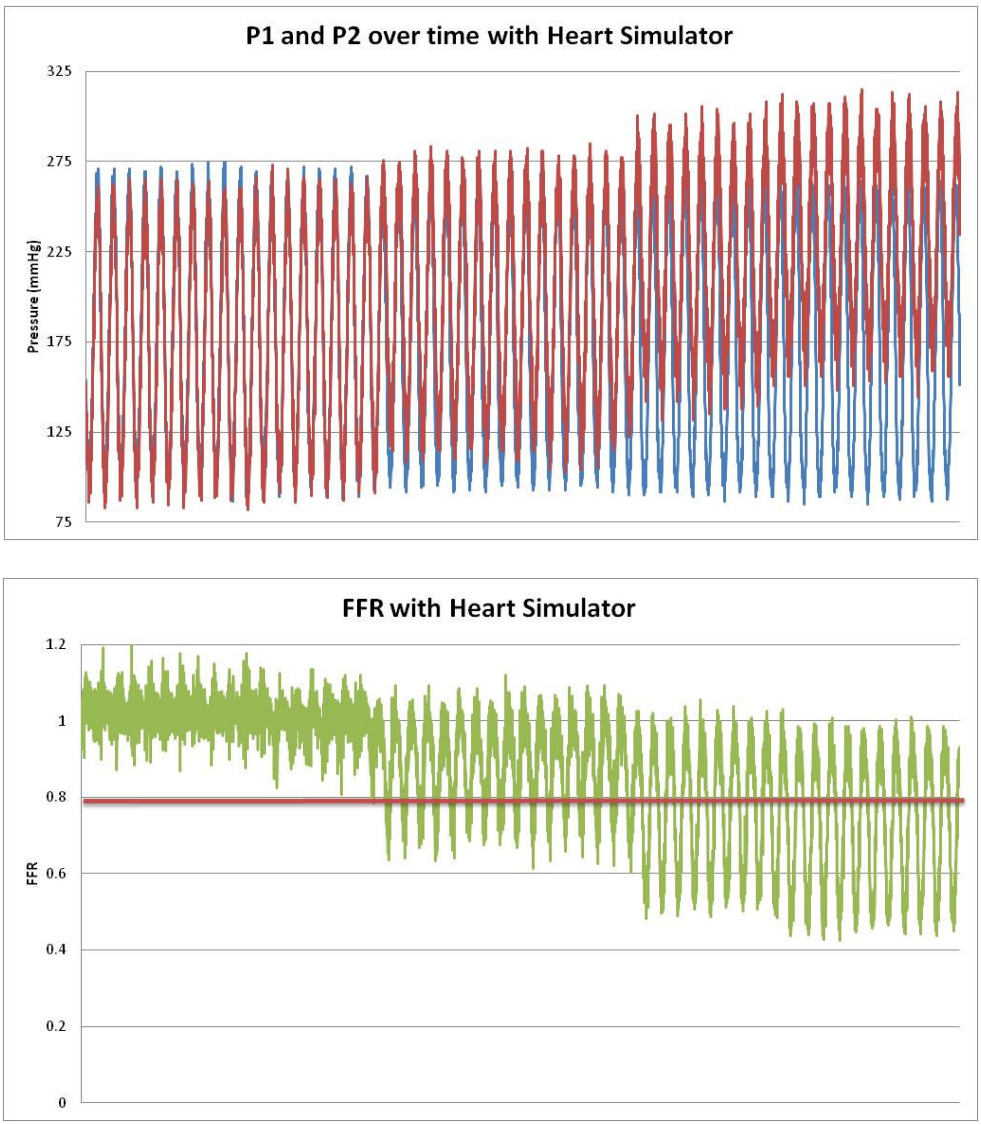


Figure 3.14: (Top) Pressure with increasing blockage and Heart Simulator (Bottom) FFR with Heart simulator

The third experiment was using a commercial RIFD reader antenna, a signal generator and power meter to determine how much power one can collated with the new miniture antenna. The first step in this process is to build connectorized versions of the antenna. To insure the antenna remained toned 4 different 0402 capacitance values were used for the same coil.

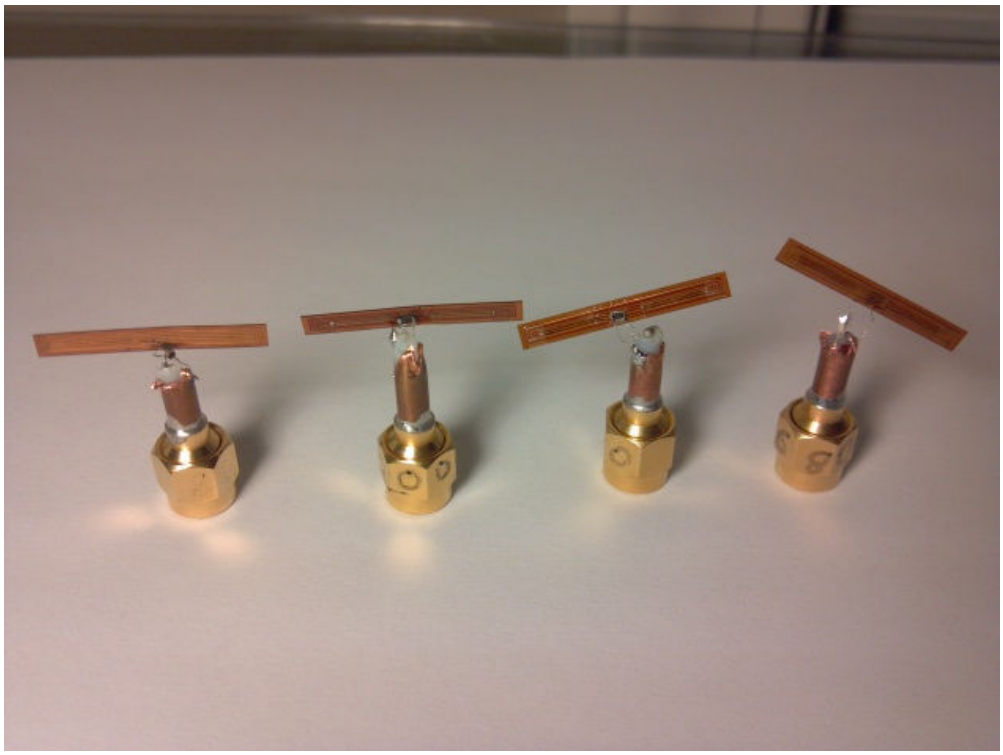


Figure 3.15: Connectorized versions of the miniture RFID antennas

The second step in this experiment is to broadcast 13.56 MHz with an Agilent E8257C. The power was maximized at 25 dBm into a commercial RFID antenna. The last step in the experiment is to measure the power received at different distances from the reader. The power was measured on a Agilent E4419B Power meter and a E4412A sensor.

After determining what the collected power was through air a beaker of water was placed between the tranmitting anteanna and the test anteanna. In Figure 3.16 one can see the entire measurement setup including the beaker of water. The results were tabulated in Figure 3.17

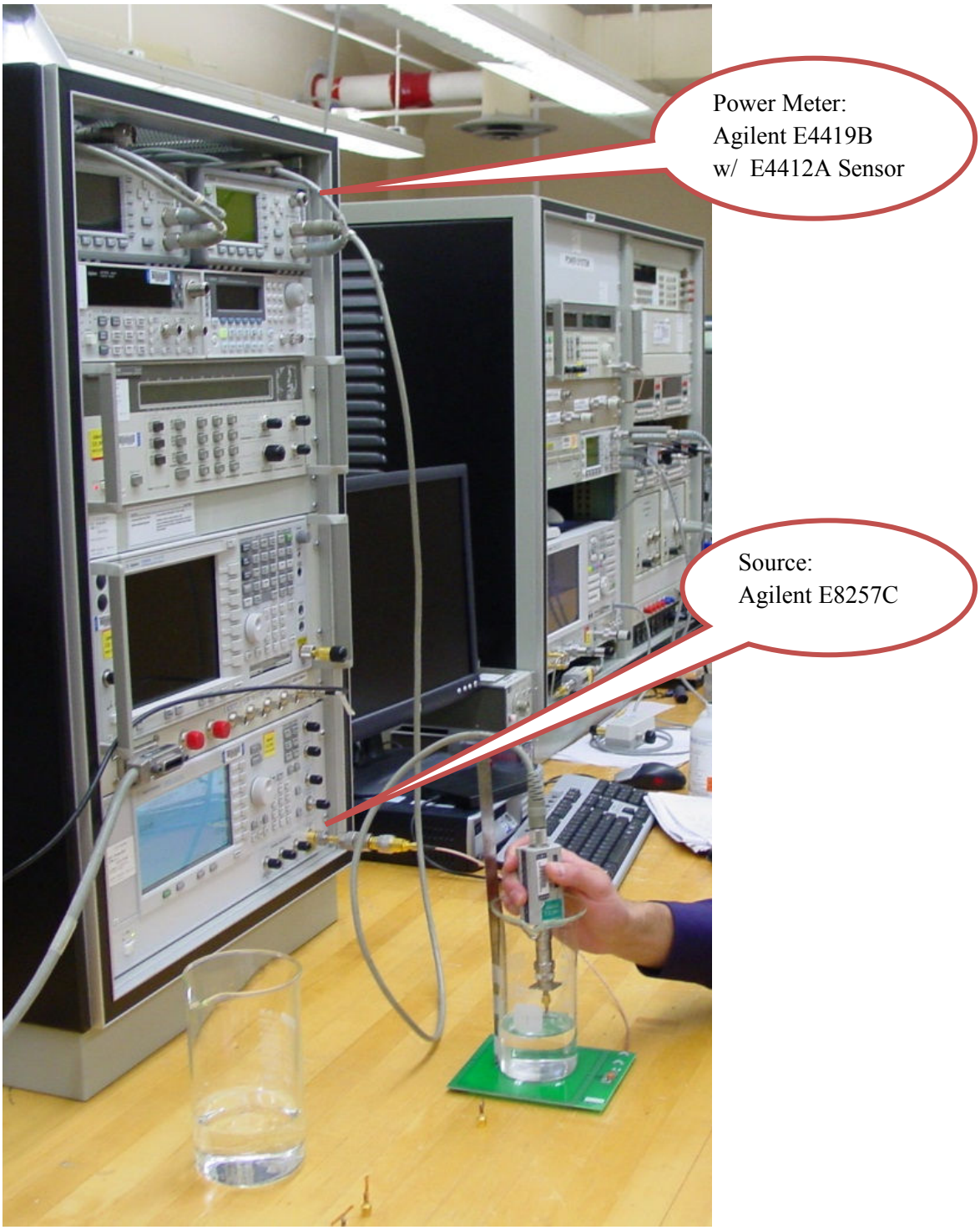


Figure 3.16: Antenna coupling tests (picture of source and power meter)

13.56Mhz at 25dBm through Air

	680 pF	820 pF	1000 pF	2200 pF
3 in	-25.2	-24.6	-24.4	-28.5
2 in	-20.7	-19.4	-19.1	-23.7
1 in	-16.3	-14.8	-14.5	-19.4
0 in	-14.4	-12.6	-12.4	-17.7

13.56Mhz at 20dBm through Air

	680 pF	820 pF	1000 pF	2200 pF
3 in	-28.4	-27.3	-27.2	-32.1
2 in	-23.5	-22.2	-22.6	-27
1 in	-19.5	-18	-18.3	-23
0 in	-17.6	-16.1	-16.4	-20.9

13.56Mhz at 20dBm through Water

	680 pF	820 pF	1000 pF	2200 pF
4 in	-34.2	-32.2	-32.1	-36.2
3 in	-29.1	-27.8	-27.1	-31.8
2 in	-25.1	-23.7	-22.5	-26.6
1 in	-20	-18.6	-18.4	-22.6

Figure 3.17: Tables of power at different distances with different capacitors

Based in the data collected above, a microchip could be powered 4 inches into the body (assueming the 20% of the body that is not water does not have a tremendious effect). The link loss is 52.1 dBm and so if one is brodcasting at 36 dBm than a microchip would receive 24 microwatts of power. Assuming it is a low power chip with a resistance of 10kOhm than it would have a Vdd of about half a volt.

The last evaluation is the power spectrical density of the MEMS Pressure sensors. This is seen in three stages. The first stage is when there is not stimuli. The second

stages is with the pump as stimuli. The last stage is seen with the pump and heart simulator as stimuli.

The measurement system sampling rate was 300 Hz. The nyquest cutoff is then 150 Hz. That said, there is sufficient bandwidth for the application selected. This can be seen in last picture of Figure 3.18. In this picture one can see the heart simulator frequency of about 3 Hz and other high-frequency components that are the product of mechanical vibrations in the pump.

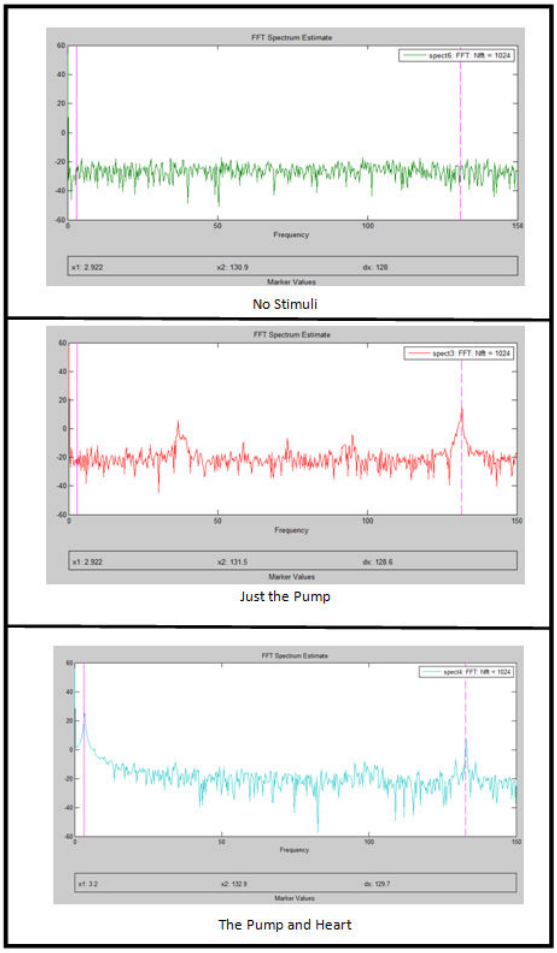


Figure 3.18: The three stages of power spectral density

CHAPTER 4

UNCERTAINTY ANALYSIS

4.1 Introduction

In chapter 3 results were presented for pressure and FFR but uncertainties were not discussed. This Chapter will utilize the Guide to the Expression of Uncertainty in Measurement (GUM method) [22] to calculate the total uncertainty of the FFR measurement. Though this is obviously not a critical measurement at the time, having good traceability is a best practice. At the risk of over simplification, the guide recommends classifying the uncertainties, combining and then expanding the total uncertainty.

Uncertainties can be classified into two categories Type A and Type B. Type A uncertainties are those which are evaluated by statistical methods. Type B uncertainties are those which are evaluated by other means (e.g. manufacture specifications, data sheets, judgment, etc.). The process of combining and expanding uncertainties will be covered in Section 4.4. Last the uncertainty of the FFR measurements acquired in chapter 3 will be shown in tabular form.

4.1 Type A Uncertainties

When considering the pressure measurements one must consider the system in which they came. The system has sensor resolutions and sensitivity in and of itself. Next there is analog signal noise. There is the quantization errors associated with the analog to digital conversion. Last there is the error of the standard used to calibrate the system.

For simplification purposes we can start by lumping the first three together and then only if we want to find dominant source can we delineate further. To determine the said

lump of the first three sources a simple experiment is conducted. The sensor is left measuring pressure at an unchanging pressure for a long amount of time. This experiment resulted Figures 4.1 through 4.4 (*Please Note: C1 is P1 and C2 is P2*).

Variable	n	Mean	StDev	Median
C1	4999	526.95	1.64	527
C2	4999	588.04	2.15	588

Figure 4.1: Descriptive statistics

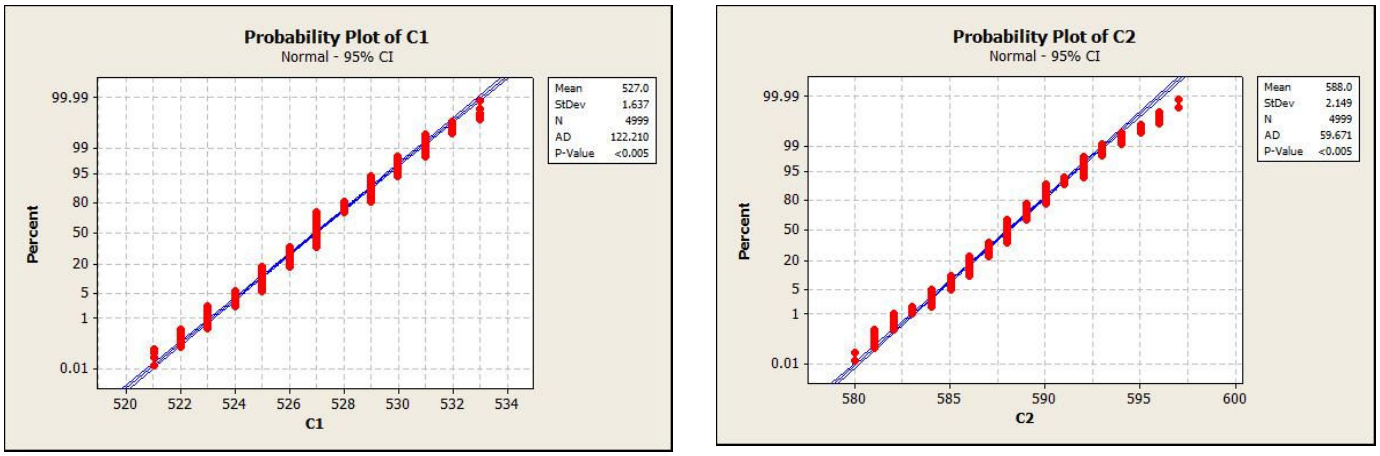


Figure 4.2 Normality tests

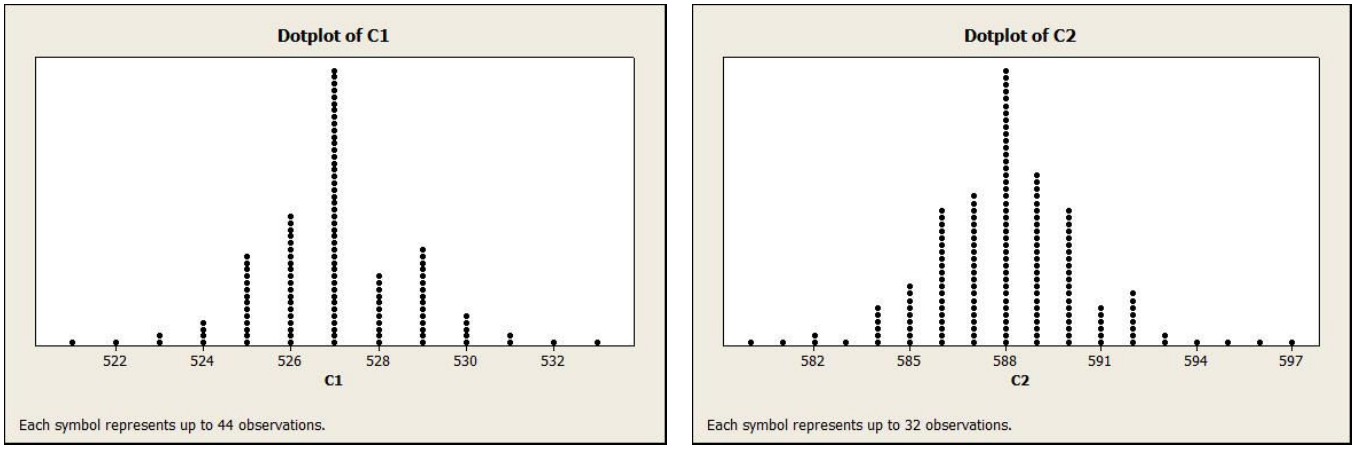


Figure 4.3: Dot Plots

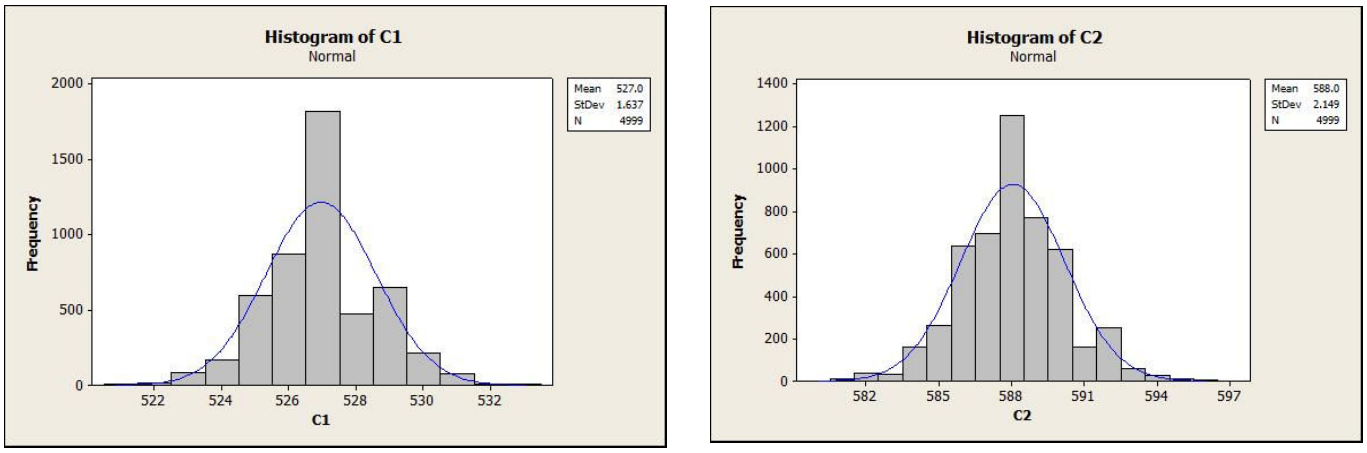


Figure 4.4: Histograms with Distribution Fit

The thorough statistical analysis above includes general descriptive statistics, a normality test, dot plots and histograms with a normal distribution fit on top. The general descriptive statistics show the standard deviation that will be used in the next section. Before one can blindly use a standard deviation as the uncertainty a normality test should be done. This mathematical test would show the distribution is normal if the p value is greater than 0.05.

Our P-values for both pressures are less than 0.005 meaning the distribution is not normal. Because of this, one should look at a dot plot or histogram to see if the distribution even looks normal. After investigation of both, one can see our data is dominated by quantization noise and so is too sharp to be normal. Two mathematical inequalities can still work in our favor as well; Chebyshev's and Vysochanskii-Petunin. Both say that even through the distribution is not normal, if the distribution is continuous and in the case of Vysochanskii-Petunin, unimodal, then two standard deviations is sufficient in capturing the error.

Once the error in the integer is known one can translate said error into pressure error by using the calibration coefficients and the Pressure Equation 3.2. Here the same process of understanding error propagation can be done as it was in equations 4.2 – 4.4 but now with a system equation of 3.2. Using the statistical analysis for the basis of the zero integers works well. However, one cannot use this for the maximum pressure integers as there is more error associated with determining theses quantities. To determine the error of these terms one can look at historical data.

Date	mpi1	mpi2
7/18/2011	652	710
7/14/2011	649	707
7/13/2011	646	707
7/5/2011	647	705
6/30/2011	623	685
MEAN	643.4	702.8
STDEV	10.4038	9.04212364
%	1.62%	1.29%
% (K=2)	3.23%	2.57%

Figure 4.5: Errors associated with maximum pressure calibration factors

4.3 Type B Uncertainties

The first Type B uncertainty used in the calibration procedure outlined in section 3.3 is the uncertainty of the standard pressure gage used to calibrate the MEMs pressure sensors. This uncertainty is assumed to be 2 % based on the following calibration results documentation.

The standard pressure gage used in section 3.3 will be called the Test Instrument (TI). The TI was compared to a NIST traceable standard pressure gage with an uncertainty of .04% (better than 4 times that of which it is calibrating). When comparing the two gages the following results were obtained (Figure 4.1). In short, the known gage used in section 3.3 is 2 % accurate – our first Type B accuracy.

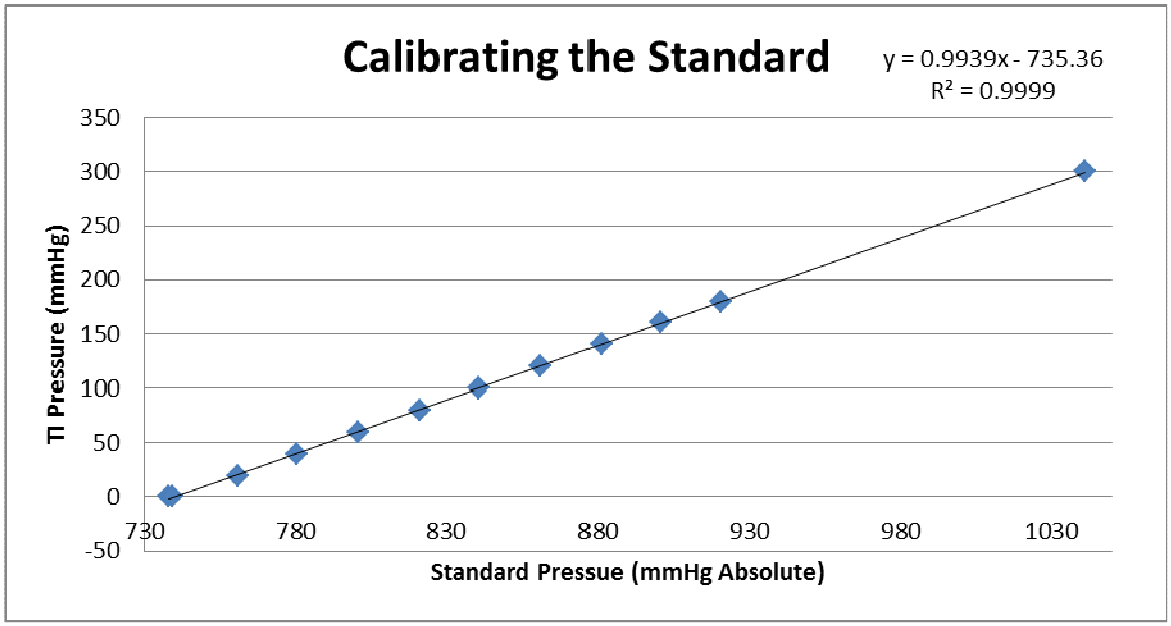


Figure 4.6: Comparison of the TI (our standard) with another Absolute pressure standard

Standard mmHg	TI_ mmHg	Calk_mmHg	Error (abs)	Error (%)
738.170	0	735	-2.810	-0.4%
760.750	20	755	-5.512	-0.7%
780.249	40	775	-5.133	-0.7%
800.834	60	795	-5.84	-0.7%
821.315	80	815	-6.443	-0.8%
840.710	100	835	-5.96	-0.7%
861.089	120	855	-6.461	-0.8%
881.001	140	875	-6.495	-0.7%
900.448	160	894	-6.064	-0.7%
920.671	180	914	-6.409	-0.7%
1041.176	300	1034	-7.646	-0.7%
739.182	0	735	-3.822	-0.5%
Resolution is 2 mmHg			**Average Error**	**Average (K=1)**
In persentage =	1.25%		-5.72	-0.7%
				Average (K=2)
RSS of Resolution Error and Abs Error (K=2) = 2.0%				-1.36%

Figure 4.7: Tabular form showing the combining of uncertainties.

Our second Type B uncertainty comes from the mathematical evaluation of the system equation expressing FFR. Equation 2.1 is the system equation. The GUM method mathematical equation for evaluating the propagation of error of equation 2.1 is given in (4.1).

$$dFFR = \sqrt{\left(\frac{\partial FFR}{\partial P1} \times dP1\right)^2 + \left(\frac{\partial FFR}{\partial P2} \times dP2\right)^2} \tag{4.1}$$

Simple steps (equations 4.2 and 4.3) show the transformation into equation 4.4. The steps can be described as evaluating the change in FFR due to a change in P1 (4.2) and evaluating the change in FFR due to a change in P2 (4.3). Equation 4.4 expresses the

uncertainty of FFR in terms of quantities that we either know or can find out through statistical evaluation.

$$\frac{\partial FFR}{\partial P1} = -\frac{P2}{P1^2} \tag{4.2}$$

$$\frac{\partial FFR}{\partial P2} = \frac{1}{P1} \tag{4.3}$$

$$dFFR = \sqrt{\left(-\frac{P2}{P1^2} \times dP1\right)^2 + \left(\frac{1}{P1} \times dP2\right)^2} \tag{4.4}$$

4.4 Combining and Expanding Uncertainties

Once the errors are classified we can begin to combine them into one expression of uncertainty for FFR. First all errors will be tabulated and classified. Second, the errors will be Root Sum Squared (RSS) for the two Pressure Errors in the system equation 4.4. Third, a sample calculation using example pressures and equation 4.4 will be used to determine a system error. Fourth, the system error will be expanded to a coverage factor of 2 sigma.

Error Source	Type	Quantity
Pressure standard	B	2%
ZI1	A	0.62%
ZI2	A	0.73%
MPI1	A	3.23%
MPI2	A	2.57%
P1	B	3.85%
P2	B	3.34%

Figure 4.7: Errors tabulated

P1	P2	dP1	dP2	dFFR (K=1)	FFR	Percent
110	120	4.23	4.01	0.047	0.917	5.10%

Figure 4.8: Example calculation using equation 4.3

FFR with a coverage factor of 2 sigma =	10.2%

Figure 4.9: Error expanded

To conclude, the expanded system error with a coverage factor of 2 sigma is 10.2 present. This means that we know the FFR value to ±10.2%. Even though the dot plots are dominated by quantization uncertainty making them non normal moving to a 16 bit ADC would only make the cure pass a normality test and not improve the sigma. Thus, the 12 bit ADC is sufficient and to improve the uncertainties one would be better off reducing the sources of noise in the circuit.

CHAPTER 5

FUTURE WORK

5.1 Introduction

It will not be long before implantable sensors are strewn about human bodies. In fact according to e-Science news as of January 21, 2009 Fraunhofer researchers are currently conducting clinical trials of an implantable pressure sensor [22]. As it was shown in this thesis, the accuracies necessary can be obtained for a flow meter. What remains to be found are other applications and the necessary accuracies for those applications. Specifically with regard to this thesis the next steps toward miniaturization can be discussed. By far the largest obstacle that must be obtained is biocompatibility.

5.2 Other Applications

Other applications for this work start with the same parameter, pressure, and with only a few minor changes can provide other uses. Imagine the ability to measure eye pressure in glaucoma patients and based off the pressure reading have a small implantable pump regulate eye pressure. Imagine being able to measure cranial blood pressure trends in people who suffer from migraines and be able to warn them to take medicine before the migraine even starts.

Applications of this work will ultimately end with many more parameters than pressure. Implanted sensors of all types could interface with the outside world like cell phones and thermostats. The sensors could call for help or regulate the room temperature. ECG, EKG, core temperature of people working in extreme climates, inertial sensors (accelerometers, magnetometers, gyros) in athletes providing real time feedback on

performance. An artificial pancreas will not only give connivance to the diabetic but save lives. All of these just name a few of the limitless application on the horizon.

5.3 Next Steps for Miniaturization

The next steps of this work in the way to miniaturization can be defined well in terms of VLSI. A system designer could start at the top and work down until the entire circuit board, wireless communication circuit, and sensor could be on one silicon chip about 1 mm by 2 mm. A good first step is to pick a technology that would lend to the low power, high accuracy, and MEMS sensor compatibility.

Another step that could be useful is standardizing an implantable wireless power source (RFID antenna with a Zener diode and an LDO) capable of powering a low current 3 volt chip. This standard platform could be used by the hundreds of electrical engineers that need to enter this new field.

5.4 Considering Biocompatibility

The human body is a great place to find spectacular electrochemistry. After all, any exposed electrode over one volt in water is characterized by electrolysis. This biocompatibility problem has been addressed through magnificent packaging and creative design techniques. The pace maker is the best example of this. The electrodes in a pace maker are 2mm square gold and they allow cell growth on them. The pacemaker measure the impedance change caused by these cells and changes the voltage to get the desired wave form. But when dealing with sensors, one must be more sensitive to the interface between the sensor and the body. Specifically with regard to the sensor used in this thesis, if cells build up on the diaphragm, the system would have to be recalibrated because of the errors

resulting from the buildup. This can be done but while in the design stages why not propose material research.

When considering the cutting edge of most engineering disciples it is often up to the material scientist to drive the march forward. Think about Leonardo da Vinci planning to build a helicopter. He had the plans right but did not have the materials to succeed. Today materials are being researched at an alarming rate. Still we need more biocompatible materials.

The material research that would be ideal could be in two forms: (1) Better packaging and (2) a biocompatible fabrication process. The latter is much more ideal due the sheer size of fabrication knowledge that could be levered.

CHAPTER 6

SUMMARY AND CONCLUDING REMARKS

The first chapter pointed at cardiovascular disease as the primary driver for the research. The chapter proposed that a sensor with the ability of self diagnosis is a creative solution to the diagnosis of restenosis. The main question this thesis answers is - Will this device have the accuracies suitable for a physician's diagnosis? First a proof of concept device must be built. Second, an experimental test bed must be built. Last a system uncertainty was calculated.

The background necessary to make sense of this work falls into three large categories. (1) Pressure sensors and flow meters. (2) Current medical equipment and its measurement. (3) Physician's expectations. The pressure sensor used in this thesis is a resistive bridge that changes linearly in resistance with a linear pressure change. Current medical equipment is made by two vendors that utilize a wired catheter. The measurement they make is functional flow rate (FFR) and it is defined as the distal pressure divided by the proximal pressure. Physicians expect accuracy of 10%.

The MEMS sensors were wire bonded. A differential signal amplification and data acquisition circuit board was made to connect to the sensors and talk to a computer. A Matlab GUI was made to interface with the data acquisition board and update calibration coefficients. A test bed was made that simulated different blockages and the heart. A calibration procedure was created. All of this work resulted in the ability to measure FFR. Our experiments indicate that the pressure sensors employed at a sampling frequency of 300 Hz have a bandwidth of 150 Hz suggesting that MEMS sensors could provide high-frequency information that can be exploited for improved diagnosis.

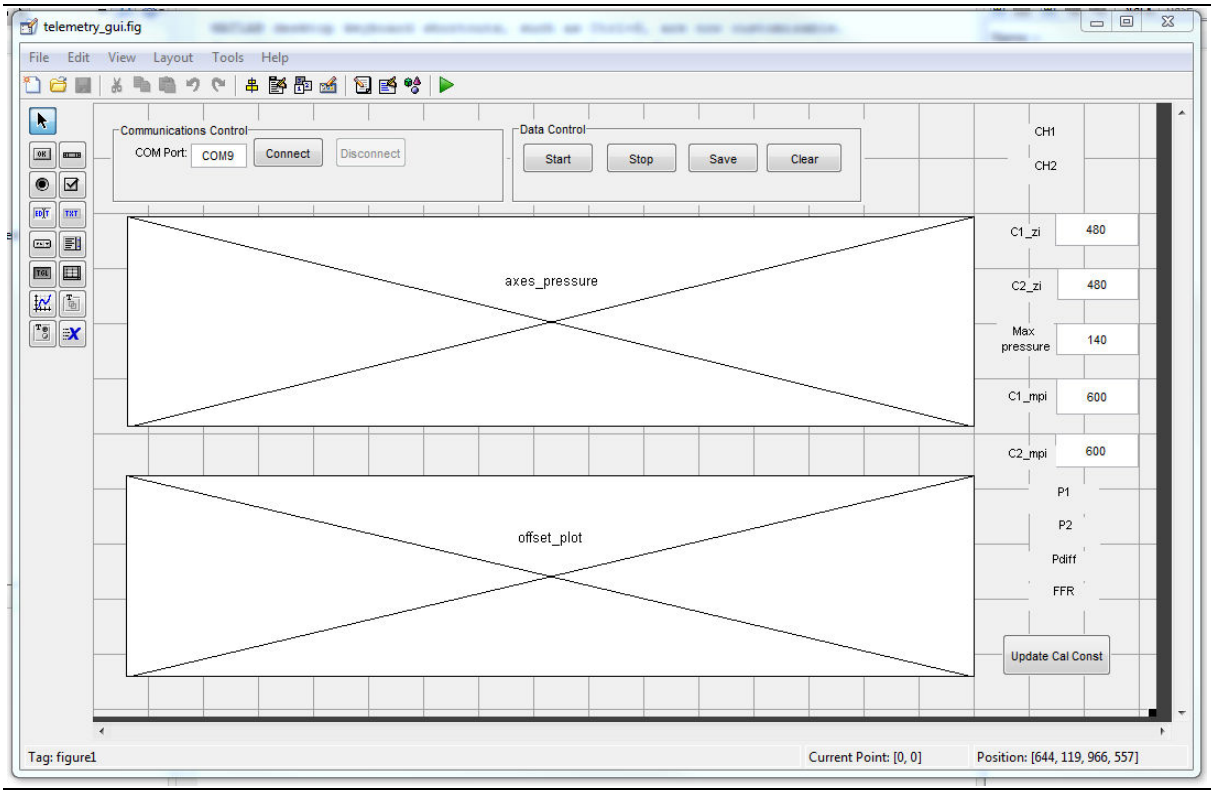
After taking the measurements one is left with the fundamental question of how good are they. The Guide to the expression of Uncertainty Measurements was used to analyze, document and arrive at the conclusion of uncertainty for this system. The uncertainty of this system is 10.2%.

The 12 bit accuracy of the ADC used was sufficient. Noise could be reduced further to improve the FFR measurement accuracy. This reduction could be done by strictly insuring the geometric centroid of the out current is that of the in current. The miniature RFID coil antennas, with the right broadcasting antenna, are capable of collecting 24 microwatts of power through 4 inches of water.

Overall, this thesis (1) researched MEMS sensors (2) researched flow measuring techniques (3) development an amplifying data acquisition board (4) built and designed calibration and experimentation (5) characterized the uncertainty of the proof of concept that was built. The characteristic uncertainty of the proof of concept system matched with the physician's expectations.

APPENDIX

MATLAB GUI CODE




```
function varargout = telemetry_gui(varargin)
% TELEMETRY_GUI M-file for telemetry_gui.fig
%      TELEMETRY_GUI, by itself, creates a new TELEMETRY_GUI or raises the
existing
%      singleton*.
%
%      H = TELEMETRY_GUI returns the handle to a new TELEMETRY_GUI or the
handle to
%      the existing singleton*.
%
%      TELEMETRY_GUI('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in TELEMETRY_GUI.M with the given input
arguments.
%
%      TELEMETRY_GUI('Property','Value',...) creates a new TELEMETRY_GUI
or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before telemetry_gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
```

```
%      stop.  All inputs are passed to telemetry_gui_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help telemetry_gui

% Last Modified by GUIDE v2.5 29-Jun-2011 20:03:39

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @telemetry_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @telemetry_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

%==========================================================================
%========================== Global variables ==============================

%global bytes_available;
%global buffer;
%global sample_ptr;
%global Vref;
%global max_samples;
%global serial_obj;
%global t;

%bytes_available = 0;
%sample_ptr      = 1;
%buffer          = zeros(max_samples, 1);
%Vref            = 3.3;
%max_samples     = 2000;
%serial_obj = serial('COM13', 'BaudRate', 115200, 'InputBufferSize', 500);
%t = timer('StartDelay',1, 'Period', 1, 'ExecutionMode','fixedRate');
%set(t, 'TimerFcn', {'timer_callback_read_serial', serial_obj, Vref});
%set(t, 'TimerFcn', {'timer_callback'});
```

```
%==========================================================================

% --- Executes just before telemetry_gui is made visible.
function telemetry_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to telemetry_gui (see VARARGIN)

global sample_ptr;
global buffer;
global data_ptr;
global buffer_size;
global data_buffer_size;
global CH1;
global CH2;
global delta;
global time_index;
global pressure_difference;
global filtered_diff;
global filt1;
global A1;
global B1;
global zi1;
global zi2;
global mp;
global mpi1;
global mpi2;
global FFR;



load 'C:\Users\public\Documents\Masters Project\Matlab\filter1' filt1;
%load 'C:\Users\kh9cb\Desktop\Pressure_capture_system\Matlab\filter2'
filt2;
A1 = filt1.tf.den;
B1 = filt1.tf.num;

buffer_size             = 2000;             %make this 1.5x the length
between SYNC sequences
data_buffer_size        = 5000;
handles.bytes_available = 0;
sample_ptr              = 1;
data_ptr                = 1;
buffer                  = zeros(1, buffer_size);
CH1                     = zeros(1, data_buffer_size);
CH2                     = zeros(1, data_buffer_size);
pressure_difference     = zeros(1, data_buffer_size);
FFR                     = zeros(1, data_buffer_size);
delta                   = 74;
time_index              = linspace(0, data_buffer_size-1,
data_buffer_size);
handles.Max             = 1024;
```

```
COMPort                 = get(handles.edit_COM_port, 'String');
handles.serial_obj      = serial(COMPort, 'BaudRate', 9600,
'InputBufferSize', 2000);
handles.t               = timer('StartDelay',1, 'Period', 0.5,
'ExecutionMode','fixedRate');
set(handles.t, 'TimerFcn', {'timer_callback', hObject, handles});

set(handles.button_start, 'Enable', 'off');
set(handles.button_stop, 'Enable', 'off');

plot(handles.axes_pressure, time_index, CH1, 'blue', time_index, CH2,
'red');
axis(handles.axes_pressure, [0 data_buffer_size 0 handles.Max]);
grid(handles.axes_pressure);

%pressure_difference = ((CH2-CH1)-delta);

plot(handles.offset_plot,pressure_difference, 'green');
axis(handles.offset_plot, [0 data_buffer_size -0.1 0.3]);
grid(handles.offset_plot);

zi1 = str2num(get(handles.edit_zi1, 'String'));
zi2 = str2num(get(handles.edit_zi2, 'String'));
mp = str2num(get(handles.edit_mp, 'String'));
mpi1 = str2num(get(handles.edit_mpi1, 'String'));
mpi2 = str2num(get(handles.edit_mpi2, 'String'));


%plot(handles.axes_pressure, CH2, 'red');

% Choose default command line output for telemetry_gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using telemetry_gui.

% UIWAIT makes telemetry_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = telemetry_gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%axes(handles.axes_data);
cla;

popup_sel_index = get(handles.popupmenu1, 'Value');
switch popup_sel_index
    case 1
        plot(rand(5));
    case 2
        plot(sin(1:0.01:25.99));
    case 3
        bar(1:.5:10);
    case 4
        plot(membrane);
    case 5
        surf(peaks);
end


% --------------------------------------------------------------------
function FileMenu_Callback(hObject, eventdata, handles)
% hObject    handle to FileMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function OpenMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to OpenMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
file = uigetfile('*.fig');
if ~isequal(file, 0)
    open(file);
end

% --------------------------------------------------------------------
function PrintMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to PrintMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
printdlg(handles.figure1)

% --------------------------------------------------------------------
function CloseMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to CloseMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
```

```
                    ['Close ' get(handles.figure1,'Name') '...'],...
                    'Yes','No','Yes');
if strcmp(selection,'No')
    return;
end

delete(handles.figure1)


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end

set(hObject, 'String', {'plot(rand(5))', 'plot(sin(1:0.01:25))',
'bar(1:.5:10)', 'plot(membrane)', 'surf(peaks)'});


% --- Executes on button press in button_start.
function button_start_Callback(hObject, eventdata, handles)
% hObject    handle to button_start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%global serial_obj;

if handles.serial_obj.Status == 'open'
   start(handles.t);
   set(handles.button_stop, 'Enable', 'on');
   set(handles.button_start, 'Enable', 'off');
   guidata(hObject, handles);
end
```

```
% --- Executes on button press in button_stop.
function button_stop_Callback(hObject, eventdata, handles)
% hObject    handle to button_stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
stop(handles.t);
set(handles.button_stop, 'Enable', 'off');
set(handles.button_start, 'Enable', 'on');
guidata(hObject, handles);

% --- Executes on button press in button_save.
function button_save_Callback(hObject, eventdata, handles)
% hObject    handle to button_save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global CH1;
global CH2;
global data_buffer_size;
global pressure_difference;
global filtered_diff;
global delta;
global zi1;
global zi2;
global mp;
global mpi1;
global mpi2;
global P1;
global P2;

delta  = 74;
%pressure_difference = 3.3*((CH2-CH1)-delta)/1024;

file = uiputfile('*.xls');
if ~isequal(file, 0)
    fid = fopen(file, 'wt');

    fprintf(fid, 'zi1\t zi2\t mp\t mpi1\t mpi2\n');
    fprintf(fid, '%d\t %d\t %d\t %d\t %d\n', zi1, zi2, mp, mpi1, mpi2);

    fprintf(fid, 'CH1\t CH2\t pressure_difference\t filtered_diff\t P1\t
P2\n');
    for ii=1:data_buffer_size
      fprintf(fid, '%d\t %d\t %2.3f\t %2.3f\t %2.3f\t %2.3f\n', CH1(ii),
CH2(ii), P1(ii), P2(ii),pressure_difference(ii), filtered_diff(ii));
    end
    fclose(fid);
    save 'C:\Users\Public\Documents\Masters Project\Matlab\pressure_data'
CH1 CH2 pressure_difference filtered_diff;
end


% --- Executes on button press in button_connect.
function button_connect_Callback(hObject, eventdata, handles)
% hObject    handle to button_connect (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%global t;
    com_str                 = get(handles.edit_COM_port, 'String');
    handles.serial_obj.Port = com_str;
    fopen(handles.serial_obj);

    if handles.serial_obj.Status ~= 'open'
       set(handles.text_status, 'String', 'Failed to open serial port');
       set(handles.button_disconnect, 'Enable', 'off');
       set(handles.button_connect, 'Enable', 'on');
       set(handles.button_start, 'Enable', 'off');
       set(handles.button_stop, 'Enable', 'off');
    else
       set(handles.text_status, 'String', 'COM port opened successfully');
       set(handles.button_disconnect, 'Enable', 'on');
       set(handles.button_connect, 'Enable', 'off');
       set(handles.button_start, 'Enable', 'on');
       set(handles.button_stop, 'Enable', 'off');

    end
    guidata(hObject, handles);



function edit_COM_port_Callback(hObject, eventdata, handles)
% hObject    handle to edit_COM_port (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_COM_port as text
%        str2double(get(hObject,'String')) returns contents of
edit_COM_port as a double


% --- Executes during object creation, after setting all properties.
function edit_COM_port_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_COM_port (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in button_disconnect.
function button_disconnect_Callback(hObject, eventdata, handles)
% hObject    handle to button_disconnect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
stop(handles.t);
fclose(handles.serial_obj);
set(handles.text_status, 'String', 'COM port closed');
set(handles.button_disconnect, 'Enable', 'off');
set(handles.button_connect, 'Enable', 'on');
set(handles.button_start, 'Enable', 'off');
set(handles.button_stop, 'Enable', 'off');
guidata(hObject, handles);


% --- Executes on button press in buttonClear.
function buttonClear_Callback(hObject, eventdata, handles)
% hObject    handle to buttonClear (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global CH1;
global CH2;
global delta;
global data_buffer_size;
global sample_ptr;
global data_ptr;
global buffer_size
global buffer
global pressure_difference;
global FFR;

sample_ptr = 1;
data_ptr   = 1;
CH1                  = zeros(1, data_buffer_size);
CH2                  = zeros(1, data_buffer_size);
pressure_difference  = zeros(1, data_buffer_size);
FFR                  = zeros(1, data_buffer_size);
delta  = 74;
buffer = zeros(1, buffer_size);

plot(handles.axes_pressure, CH1, 'blue');
axis(handles.axes_pressure, [0 data_buffer_size 0 handles.Max]);
plot(handles.axes_pressure, CH2, 'red');
axis(handles.axes_pressure, [0 data_buffer_size 0 handles.Max]);

%pressure_difference = ((CH2-CH1)-delta);
plot(handles.offset_plot, pressure_difference, 'green');
axis(handles.offset_plot, [0 data_buffer_size -0.1 0.3]);

fread(handles.serial_obj, handles.serial_obj.BytesAvailable, 'uint8');



% --- Executes on button press in pushbutton_update.
function pushbutton_update_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_update (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
global zi1;
global zi2;
global mp;
global mpi1;
global mpi2;

zi1 = str2num(get(handles.edit_zi1, 'String'));
zi2 = str2num(get(handles.edit_zi2, 'String'));
mp = str2num(get(handles.edit_mp, 'String'));
mpi1 = str2num(get(handles.edit_mpi1, 'String'));
mpi2 = str2num(get(handles.edit_mpi2, 'String'));



function edit_zi1_Callback(hObject, eventdata, handles)
% hObject    handle to edit_zi1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_zi1 as text
%        str2double(get(hObject,'String')) returns contents of edit_zi1 as
a double


% --- Executes during object creation, after setting all properties.
function edit_zi1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_zi1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_zi2_Callback(hObject, eventdata, handles)
% hObject    handle to edit_zi2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_zi2 as text
%        str2double(get(hObject,'String')) returns contents of edit_zi2 as
a double


% --- Executes during object creation, after setting all properties.
function edit_zi2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_zi2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_mp_Callback(hObject, eventdata, handles)
% hObject    handle to edit_mp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_mp as text
%        str2double(get(hObject,'String')) returns contents of edit_mp as
a double


% --- Executes during object creation, after setting all properties.
function edit_mp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_mp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_mpi1_Callback(hObject, eventdata, handles)
% hObject    handle to edit_mpi1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_mpi1 as text
%        str2double(get(hObject,'String')) returns contents of edit_mpi1
as a double


% --- Executes during object creation, after setting all properties.
function edit_mpi1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_mpi1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_mpi2_Callback(hObject, eventdata, handles)
% hObject    handle to edit_mpi2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_mpi2 as text
%        str2double(get(hObject,'String')) returns contents of edit_mpi2
as a double


% --- Executes during object creation, after setting all properties.
function edit_mpi2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_mpi2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

MATLAB TIMER CALL BACK CODE

```
function timer_callback(obj, event, hObject, handles)

  global sample_ptr;
  global buffer;
  global data_ptr;
  global CH1;
  global CH2;
  global delta;
  global buffer_size;
  global data_buffer_size;
  global time_index;
  global pressure_difference;
  global filtered_diff;
  global A1;
  global B1;
  global time_index;
  global zi1;
```

```
  global zi2;
  global mp;
  global mpi1;
  global mpi2;
  global P1;
  global P2;
  global FFR;

  bytes_available = handles.serial_obj.BytesAvailable;
  len             = buffer_size;

%  if bytes_available > 0
%    if sample_ptr + bytes_available >= len              %takes care of
the buffer boundaries
%        buffer(sample_ptr:len) = fread(handles.serial_obj, len-
sample_ptr+1, 'uint8');
%        if bytes_available - (len-sample_ptr+1) > 0     %number of bytes
left in the serial object memory
%           buffer(1: bytes_available - (len-sample_ptr+1) ) =
fread(handles.serial_obj,  bytes_available - (len-sample_ptr+1), 'uint8');
%           sample_ptr = bytes_available - (len-sample_ptr+1)+1;
%        else
%           sample_ptr = 1;
%        end
%    else
%        buffer(sample_ptr:sample_ptr + bytes_available-1) =
fread(handles.serial_obj, bytes_available, 'uint8');
%        sample_ptr = sample_ptr + bytes_available;
%    end

  if bytes_available > 0

    % ========== RX buffer management ==========
    x = bytes_available - (len - sample_ptr);

    if x <= 0
      buffer(sample_ptr:sample_ptr + bytes_available-1) =
fread(handles.serial_obj, bytes_available, 'uint8');
      sample_ptr = sample_ptr + bytes_available;
    else
      buffer(1:sample_ptr-x)     = buffer(x+1:sample_ptr);          %shift
buffer
      buffer(sample_ptr+1-x:len) = fread(handles.serial_obj,
bytes_available, 'uint8');
      sample_ptr                 = sample_ptr - x + bytes_available;
    end

    %========= Finds and processes a packet ==========
    sync_pos = strfind(buffer, 'PAK');

    for nn=1:length(sync_pos)-1

%       for sample_index=1:4
```

```
%           CH1(data_ptr)  = buffer( sync_pos(nn) + 3 + (sample_index-1)*2
)*256 + buffer( sync_pos(nn) + 4 + (sample_index-1)*2 );
%           CH2(data_ptr)  = buffer( sync_pos(nn) + 5 + (sample_index-1)*2
)*256 + buffer( sync_pos(nn) + 6 + (sample_index-1)*2 );
           CH1(data_ptr)  = buffer( sync_pos(nn) + 3 )*256 + buffer(
sync_pos(nn) + 4 );
           CH2(data_ptr)  = buffer( sync_pos(nn) + 5 )*256 + buffer(
sync_pos(nn) + 6 );

           data_ptr = data_ptr + 1;
           if data_ptr >= data_buffer_size
              data_ptr = 1;
           end
 %      end
       buffer(sync_pos(nn): sync_pos(nn) + 2) = 0;                    %
deletes packet header so it wont be read again
    end

    plot(handles.axes_pressure, time_index, CH1, 'blue', time_index, CH2,
'red');
    %plot(handles.axes_pressure, CH2, 'red');
    axis(handles.axes_pressure, [0 data_buffer_size 0 handles.Max]);
    grid(handles.axes_pressure);

    delta = 74;
    %pressure_difference = 3.3*((CH2-CH1)-delta)/1024; <-old old
    %pressure_difference = ((CH2-CH1)-delta);   <-old

    P1 = (CH1-zi1)*(mp/(mpi1-zi1));

    P2 = (CH2-zi2)*(mp/(mpi2-zi2));

    pressure_difference = P2-P1;

    FFR = P1./P2;

    filtered_diff = filter(B1, A1, pressure_difference);
    plot(handles.offset_plot, time_index, pressure_difference, 'green',
time_index, filtered_diff, 'magenta');
    axis(handles.offset_plot, [0 data_buffer_size -50 50]);
    grid(handles.offset_plot);

    if data_ptr>5
      set(handles.text_ch1, 'String', strcat('CH1=', num2str(  mean(
CH1(data_ptr-6:data_ptr-1)), 3   )));
      set(handles.text_ch2, 'String', strcat('CH2=', num2str(  mean(
CH2(data_ptr-6:data_ptr-1)), 3   )));

      set(handles.text_p1, 'String', strcat('P1=', num2str(  mean(
P1(data_ptr-6:data_ptr-1)), 3   )));
      set(handles.text_p2, 'String', strcat('P2=', num2str(  mean(
P2(data_ptr-6:data_ptr-1)), 3   )));
```

```
      set(handles.text_pdiff, 'String', strcat('PDIFF=', num2str(  mean(
pressure_difference(data_ptr-6:data_ptr-1)), 3   )));
      set(handles.text_FFR, 'String', strcat('FFR=', num2str(  mean(
FFR(data_ptr-6:data_ptr-1)), 3   )));
    end

    guidata(hObject, handles);
  end
```

MATLAB FILTER CODE

```
function Hd = efilt
%EFILT Returns a discrete-time filter object.

%
% M-File generated by MATLAB(R) 7.9 and the Signal Processing Toolbox
6.12.
%
% Generated on: 21-Jun-2011 18:46:40
%

% Equiripple Lowpass filter designed using the FIRPM function.

% All frequency values are in Hz.
Fs = 100;  % Sampling Frequency

Fpass = 5;               % Passband Frequency
Fstop = 7;               % Stopband Frequency
Dpass = 0.057501127785;  % Passband Ripple
Dstop = 0.0001;          % Stopband Attenuation
dens  = 20;              % Density Factor

% Calculate the order from the parameters using FIRPMORD.
[N, Fo, Ao, W] = firpmord([Fpass, Fstop]/(Fs/2), [1 0], [Dpass, Dstop]);

% Calculate the coefficients using the FIRPM function.
b  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(b);

% [EOF]
```

MICROCONTROLLER CODE

```
// This code can be to be used with matlab interface or with Hyper Terminal with minor changes
#include "msp430x22x4.h"
#include "Receiver_Serial.h"

void MCU_Init(void);          // Microcontroller Initialization
```

```
void ATD(void);
void ATD_ascii(void);

const char splash[] = {"\r\n**ICAS LAB, ADC & UART Unit**\r\n"};
const char errors[] = {"\r\n Please press s to start A2D conv and display data\r\n"};
const char error[]  = {"\r\n Please press 'i' or 'a' to start A2D conv and display data\r\n"};
const char integer[]= {"\r\n Data in Integer format\r\n"};
const char ascii[]  = {"\r\n Data in ASCII format\r\n"};

enum TState {Idle, Tgo} State;

unsigned int digital_data[7];

int main( void )
{
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer to prevent time out reset
  MCU_Init();

  TXString((char*)splash, sizeof splash);
  digital_data[0] = 'P'; digital_data[1] = 'A'; digital_data[2] = 'K';
  P2OUT = 0x01; // Makes P2.0 pin high
  P3OUT = 0x01; // Makes P3.0 pin high
  P1OUT = 0x04; // Makes P1.2 pin high
  State = Idle;
 // while(1);

  for(;;){
      ATD_ascii(); //Start A2D module
  }
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(LPM3_bits);         // Exit LPM3
}

// Initializes MCU peripherals...
void MCU_Init(void)
{
  BCSCTL1 = CALBC1_1MHZ;                    // Set DCO
  DCOCTL = CALDCO_1MHZ;                      // 16Mhz at a max Vcc of 3v

  // USCI_A0 is programmed to work in UART mode for serial comm. with PC
  // Initialize all the registers befor you reset the UCSWRST
```

```
  P3SEL |= 0x30;                        // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL0 = 0;                         // no parity, LSB first, 8 bits, 1 stop, async.
  UCA0CTL1 |= UCSSEL_2;                       // SMCLK
  UCA0BR0 = 104; // 65;                    // 9600 => BR0= 65, BR1= 3 from 8 Mhz
  UCA0BR1 = 0;  // 3;                     // 9600 => BR0= 131, BR1= 6 from 16 Mhz
  UCA0MCTL = UCBRS2 + UCBRS0;                 // Modulation UCBRSx = 5
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;                      // Enable USCI_A0 RX interrupt

  // Configure ADC

  ADC10AE0 |= 0x06;                       // P2.1 (A1), P2.2 (A2) ADC option select
  ADC10AE1 |= 0xF0;                       // p4.3,4,5,6 (A12 - A15) ADC option select
  // Configure I/O pins

  P1DIR = 0xFF;                        // p1.1 //all P1 pins as output pins
  P2DIR = 0x01;                         // P2.0  as output
  P3DIR = 0x01;                         // P3.0 as output

  __enable_interrupt();
}

// Analog to Digital converter settings inside the microcontroller
void ATD (void){
  ADC10CTL1 = INCH_1 + ADC10DIV_3 + ADC10SSEL_3; // A1, P2.1, DC10 Clock Divider Select 4,
SMCLK
  ADC10CTL0 = ADC10SHT_3 + ADC10ON + ADC10IE;    // 16xADC10CLKs, ADC10 On/Enable, ADC10
Interrupt Enable
  ADC10CTL0 |= ENC + ADC10SC;                 // Sampling and conversion start
  while(ADC10CTL1 & ADC10BUSY);
//  digital_data[0]=ADC10MEM;
  TXIntValue(ADC10MEM);

  ADC10CTL0 &= ~ENC;

  ADC10CTL1 = INCH_2 + ADC10DIV_3 + ADC10SSEL_3; // A2, P2.2, DC10 Clock Divider Select 4,
SMCLK
  ADC10CTL0 = ADC10SHT_3 + ADC10ON + ADC10IE;    // 16xADC10CLKs, ADC10 On/Enable, ADC10
Interrupt Enable
  ADC10CTL0 |= ENC + ADC10SC;                 // Sampling and conversion start
  while(ADC10CTL1 & ADC10BUSY);
//  digital_data[1]=ADC10MEM;
  TXIntValue(ADC10MEM);

  ADC10CTL0 &= ~ENC;
  ADC10CTL0 &= ~ADC10ON;
}
```

```
void ATD_ascii (void){

  ADC10CTL1 = INCH_1 + ADC10DIV_3 + ADC10SSEL_3; // A1, P2.1, DC10 Clock Divider Select 4,
SMCLK
  ADC10CTL0 = ADC10SHT_3 + ADC10ON + ADC10IE;    // 16xADC10CLKs, ADC10 On/Enable, ADC10
Interrupt Enable
  ADC10CTL0 |= ENC + ADC10SC;                 // Sampling and conversion start
  while(ADC10CTL1 & ADC10BUSY);
  digital_data[3] = (ADC10MEM>>8)%256;          //MSB
  digital_data[4] = (ADC10MEM%256);            //LSB
  P1OUT = P1OUT ^ 0x04;

  ADC10CTL0 &= ~ENC;

  ADC10CTL1 = INCH_2 + ADC10DIV_3 + ADC10SSEL_3; // A2, P2.2, DC10 Clock Divider Select 4,
SMCLK
  ADC10CTL0 = ADC10SHT_3 + ADC10ON + ADC10IE;    // 16xADC10CLKs, ADC10 On/Enable, ADC10
Interrupt Enable
  ADC10CTL0 |= ENC + ADC10SC;                 // Sampling and conversion start
  while(ADC10CTL1 & ADC10BUSY);
  digital_data[5] = (ADC10MEM>>8)%256;          //MSB
  digital_data[6] = (ADC10MEM%256);            //LSB
  P1OUT = P1OUT ^ 0x04;

  ADC10CTL0 &= ~ENC;
  ADC10CTL0 &= ~ADC10ON;

  int i;
  for(i=0;i<7;i++){
    UCA0TXBUF = digital_data[i];
    while (!(IFG2&UCA0TXIFG));
  }

}

/*// Echo back RXed character, confirm TX buffer is ready first
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{

  if(IFG2&UCA0RXIFG){
    if (UCA0RXBUF == 'i'){
       TXString ((char*)integer, sizeof integer);
       ATD();
     }else
       if (UCA0RXBUF == 'a'){
```

```
        TXString ((char*)ascii, sizeof ascii);
          ATD_ascii();
       }else
        TXString ((char*)error, sizeof error);
  }
}*/

#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
  if(IFG2&UCA0RXIFG){
    if(UCA0RXBUF == 'a'){
      if (State == Tgo)
        State = Idle;
       else
         State = Tgo;
    }
  }
}
```

REFERENCES

[1] J. Xu, K.D. Kochanek, S.L. Murphy, and B. Tejada-Vera, "Deaths: Final data for 2007," *National Vital Statistics Report*, vol. 58, no. 19, Hyattsville, MD: National Center for Health Statistics, 2010.

[2] Centers for Disease Control and Prevention (CDC) / National Center for Health Statistics (2011, Sep. 6). Leading Causes of Death [Online]. Available http://www.cdc.gov/nchs/fastats/lcod.htm

[3] J. S. Felton and R. Cole, "The high cost of heart disease," *Circulation*, vol. 27, pp. 957-962, 1963. doi: 10.1161/01.CIR.27.5.957

[4] Centers for Disease Control and Prevention (CDC) / National Center for Chronic Disease Prevention and Health Promotion (2010, July 21) Heart Disease and Stroke: The Nation's Leasing Killers [Online] Available: http://www.cdc.gov/chronicdisease/resources/publications/AAG/dhdsp.htm

[5] Banner Health (2011, Oct. 23) Heart Treatment Options, Webmaster Banner Health [Online] Available: http://www.bannerhealth.com/Services/Heart+Care/Treatment+Options/Heart+Treatment+Options.htm

[6] Surgical Associates of Texas, P.A. (2000, April) Coronary Artery Disease, [Online] Available: http://www.texheartsurgeons.com/cad.htm

[7] H. Hamid and J. Coltart, " 'Miracle stents' – a future without restenosis," *MJM McGill Journal of Medicine*, vol. 10, no. 2, pp. 105-111, 2007.

[8] D. Stoeckel *et al.* "A survey of stent design," Min. Invas. Ther. & Allied Technol., vol. 11, no. 2, pp. 137-147, 2002.

[9] K. Takahata *et al.* "A Wireless Microsensor for Monitoring Flow and Pressure in a Blood Vessel Utilizing a Dual-Inductor Antenna Stent and Two Pressure Sensors," in the 17th IEEE International Conference on MEMS, 2004 © IEEE, doi: 10.1109/MEMS.2004.1290561

[10] R. Tan *et al.* "Development of a fully implantable wireless pressure monitoring system," in the Biomed. Microdevices, 2008 © Springer Science + Buisness Media, LLC doi: 10.1007/s10544-008-9232-1

[11] K. Satyamurthy *et al.* "Experimental Evaluation of an Intravascular Differential Pressure Flow Meter using MEMS Pressure Sensors," presented at the IEEE Sensors Conference, Limerick Ireland 2011.

[12] A.A. Barlian *et al.* "Review: Semiconductor Piezoresistance for Microsystems," *Proceedings of the IEEE*, vol.97, no.3, pp.513-552, March 2009 doi: 10.1109/JPROC.2009.2013612

[13] J.G. Webster, "Blood Pressure and Sound," in *Medical Instrumentation Application and Design*, 4th ed. Hoboken: John Wiley & Sons, Inc., 2010, ch.7, sec. 7.1, pp. 295

[14] Radiological Society of North America (2009, May 01) Dual-Source vers 64-Section CT Coronary Angiography at Lower Heart Rates: Comparison of Accuracy and Radiation Dose, Webmaster Radiology [Online] Available: http://radiology.rsna.org/content/253/1/56/F9.expansion.html doi: 10.1148/radiol.2531090065

[15] National Heart and Blood Institute (2011, Sep. 6) What Is Coronary Angiography?, Webmaster National Institutes of Health [Online] Available: http://www.nhlbi.nih.gov/health/health-topics/topics/ca/

[16] B. Cohen, (2009, Nov. 25) Fractional Flow Reserve (FFR) Gains in Updated ACC/AHA/SCAI Guidelines, Webmaster Angioplasty.org [Online] Available: http://www.ptca.org/news/2009/1125_FFR.html

[17] ComboWire XT Pressure and Flow Wire, Volcano data sheet 600448-001/001, Rancho Cordova, CA

[18] PressureWire Certus, St Jude Medical Data Sheet Item 100018911, St. Paul, MN

[19] Volcano corporation (2011, Oct. 20) ComboMap Pressure and Flow System, Webmaster Site by second round, INC [Online] Available: http://www.volcanocorp.com/products/combomap-pressure-flow-system.php

[20] Vaughn Aubuchon (2011, Oct. 13) Normal Blood Pressure Chart webpage, Webmaster Vaughn Aubuchon [Online] Available: http://www.vaughns-1-pagers.com/medicine/blood-pressure.htm

[21] American College of Chest Physicians (2011, Nov. 10) Determinants of Aortic Pressure Variation During Positive-Pressure Ventilation in Man [Online] Available: http://chestjournal.chestpubs.org/content/116/1/176.full Vol. 116 no. 1 pp. 176-186 doi:10.1378/chest.116.1.176

[22] e! Science News (2011, Nov. 10) Sensor in artery measures blood pressure, Webmaster Health & Medicine [Online] Available: http://esciencenews.com/articles/2009/01/21/sensor.artery.measures.blood.pressure

VITA

Erik Joseph Timpson was born in Salt Lake City, Utah. His childhood was in Utah, Idaho, Wyoming, Michigan and Nevada before settling in Missouri. After graduating High school in 2000 he was torn between two scholarship offers. One was to pursue film and theater and the other Engineering. He was moved by the logic that one could go into Engineering and do theater on the side, but one could not go into theater and do engineering on the side; and so he chose. He received an Associate's degree in Engineering at Longview Community College and a Bachelors Degree in Electrical Engineering magna cum laude with Honors and minors in Math, Physics and Biology from University of Missouri-Rolla under the Sprint Minority Engineering Scholarship.

At Longview, Erik was a supplemental instruction leader for Calculus I and Engineering Physics I. He was the Engineering Club President. He was in the Phi Theta Kappa Honor Society and a Longview Community College Ambassador. He received "Outstanding Student leader" and "Mathematics and Science Student of the Year" Awards for his time at Longview.

Erik continued to be a polymath at Rolla. He was a peer instructor for Engineering Physics II and was the first pear instructor for Computer Engineering 111. He was the vice president of Eta Kappa Nu and a Kappa Mu Epsilon member. As a member of IEEE he took first in a sectional paper contest and contended regionally. Erik took third in an Undergraduate Research Symposium as the only student with two concurrent projects. Erik was vice president of the Society of Hispanic Engineers (SHPE). Erik was also the Chief Engineer of the Solar House Team. Though the list of his contributions is long he will be

remember most for playing the character of Mordred in Camelot the Musical for the University.

Professionally Erik has continued to work diligently. He worked for Sprint, the telecommunications company from 2001 to 2008. At Sprint he had a variety of engineering responsibilities, the most notable was writing a mechanical durability specification for mobile devices. In 2008 he left Sprint to work for Honeywell Federal Manufacturing and Technology (FM&T). At Honeywell he is a Metrology Engineer responsible for all the electrical standards used by the facility. In 2009 he became a Professional Engineer. His crowning achievement at Honeywell was being selected to participate in the Technical Fellow Program. In this program he will be compensated for studying in the Ph.D. program at the University of Missouri-Columbia.