WIRELESS SENSOR SYSTEMS FOR *IN-VIVO* APPLICATIONS

WIRELESS SENSOR SYSTEMS FOR *IN-VIVO* APPLICATIONS

By

Tsunghuan (Kurt) Huang

Bachelor of Engineering

McMaster University

Hamilton, Ontario, Canada

May 2005

A THESIS

SUBMITTED TO THE SCHOOL OF GRADUATE STUDIES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER'S OF APPLIED SCIENCE

McMaster University

Hamilton, Ontario, Canada

MASTER OF APPLIED SCIENCE (2008) McMaster University
(Electrical and Computer Engineering) Hamilton, Ontario

TITLE: WIRELESS SENSOR SYSTEMS FOR *IN-VIVO* APPLICATIONS

AUTHOR: Tsunghuan (Kurt) Huang, B.Eng. (McMaster University)

SUPERVISOR: Professor M. J. Deen

NUMBER OF PAGES: 135

ABSTRACT

Rapid developments in microelectronics technology have allowed for phenomenal achievements in biomedical engineering. In the past few decades, an enormous amount of researches were done in the field of medical implantable microelectronic systems. The prevalence of research in this particular field have led to the design of novel systems for *in-vivo* applications, for example, using microelectronic systems to replace catheterization in clinical studies of urinary incontinence.

In this thesis research, we study two types of wireless modules towards our goal of wireless systems for in-vivo applications. The first system, a 2.4 GHz wireless pressure sensor system, is designed as a pressure sensing module to operate as a part of a pill imaging device published in [32]. This pressure module samples pressure data and passes them to the pill imaging capsule via a serial-port-interface (SPI). The 2.4 GHz wireless pressure system has an overall system dimension of 75.0 × 20.5 × 17.5 mm^3 with a current consumption of 5 mA when operating from a 3 V supply. The pressure sensitivity of this system is observed as 1.14 cmH_2O/LSB (least significant bit). The second system, 125 kHz RFID (radio-frequency identification) dual sensor system, is designed to explore the possibility of powering the device and transmitting data using the RFID technology. The 125 kHz RFID dual sensor system has an overall system dimension of 30.0 × 15.0 × 15.0 mm^3 with a current consumption of 1.5 mA while operating at 3 V. The pressure sensitivity of this system is observed as 2.93 cmH_2O/LSB and the temperature sensitivity is observed as 0.069 ^{o}C/LSB. And, the detections of rapid pressure changes in both systems are successful.

The work performed in this thesis research has provided a cost-effective method of designing medical implantable systems using off-the-shelf components as compared to full-custom designs. In this research, it is also observed that power consumption is a major issue in medical implantable systems. Finally, the possibility of transmitting data and powering such systems using RFID technology has been verified.

ACKNOWLEDGEMENTS

First of all, I would like to express my sincerest thanks to my supervisor, Professor M. Jamal Deen, for his kind guidance throughout the course of this thesis and for giving me the chance to pursue an academic advancement in my life. His regular supports and technical insights have been most helpful and have enabled me to practically overcome the impossible.

Next, I would also like to thank Dr. O. Marinov for his all-around support through out the course of this thesis and for being not only a senior researcher but also a true friend during challenging times of my work.

I am also very grateful to everyone in McMaster University's Microelectronics Research Laboratory for providing an excellent working environment and for offering help whenever possible. In particular, I would like to thank my colleagues Waleed Shinwari, Munir El-Desouki, Moussa Kfouri, Darius P. Palubiak, Hamed Mazhab-Jafari, Saman Asgaran, and Farseem Mohammedy for their support.

Last, but not the least, I would like to express my deepest appreciation to my dear family, for being a constant source of support throughout my life. And to my wife, Jing Wan, I thank you for always being supportive and considerate.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

In the past few decades, the rapid developments in microelectronics technologies have led to significant achievements in designing and manufacturing novel and important sensing and stimuli biomedical engineering systems. A particular subfield of these biomedical engineering systems, implantable microelectronic systems, has emerged and shown fast development and widespread applications. From the world's first transistorized, battery-powered, wearable pacemaker invented in 1957 [1] to the recent availability of a capsule imaging system, PillCam SB [2], the interests of microelectronic devices for medical use have been growing very fast while concurrently taking advantage of semiconductor technology advancements to improve sensitivity and functionality of biomedical systems. Investigations of such devices have been carried out over the past few decades and a large amount of implants has been proposed and designed to provide medical treatments, to assist in clinical studies, and to improve patient care. However, two important constraints of such devices, power consumption and size limitation, have been the major challenges of these types of systems. Thus, the objectives of this work are to design wireless sensor systems for *in-vivo* applications and to obtain further insights into the field of biomedical implantable devices.

In the following section, an overview of microelectronic device for medical use will be first presented. Then, applications of such devices are briefly introduced. Motivation of this thesis in the field of urinary incontinence studies is given afterwards. Lastly, this chapter is concluded by describing the organization of this thesis.

1.1 Microelectronic Device for Medical Use Overview

A microelectronic device for medical use can be categorized as a type of medical instrumentation system. Generally speaking, this type of system should consist of the following functional components, a primary sensing unit, a variable conversion element, a signal processing element, data storage, power source, data transmission element, a

output display, control and feedback element, etc. A generalized medical instrumentation system is shown is Figure 1.1.

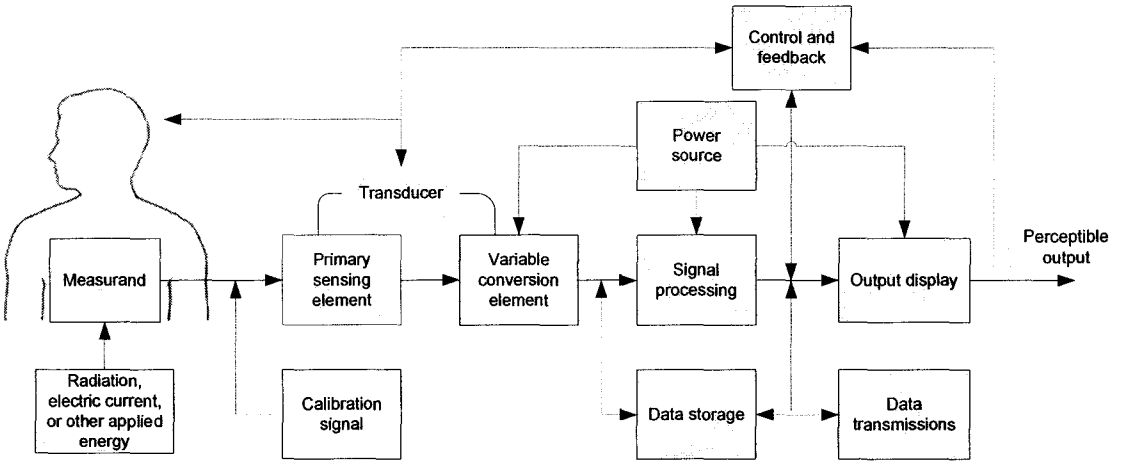


Figure 1.1: Block Diagram of a Generalized Medical Instrumentation System [3].

Measurand, is a quantity that the designated system measures. The measurand in a human may be internal such as blood pressure, gastrointestinal temperature or bladder pressure, or on the body surface such as bioelectric potentials, or radiation emanating from the body. A primary sensing element and its variable conversion element are called a transducer. A transducer converts one form of energy to another. For example, a pressure transducer converts changes in pressure to changes in voltages. A typical primary sensing element for a pressure transducer is a diaphragm. The output of a transducer sensing unit is usually a voltage. This analog output, usually, cannot be directly transmitted wirelessly. Hence, a signal processing unit is required to convert this output to a digital format, which is then processed before transmission. For example, a digital-signal-processing technique, such as exponential averaging can be applied during this stage in the system to compensate the undesirable characteristics of a transducer. Other elements, such as data storage, can temporarily store measured results for signal processing use, or together with a control-and-feedback element, these data could be used as reference for the transducer. The data transmission element could provide wireless link of the system to a host which can be linked to a personal computer for more extensive data monitoring [3].

In the design of a microelectronic device for medical use, the above components should be carefully considered. Each element should be carefully selected to perform according to its intended use and redundant elements should not be considered so as to simplify the design. Also and very important, compatibilities between elements and functions of each component should be thoroughly studied before the integration of a system starts. In the next section, applications of microelectronic devices for medical use are studied to gain further insights in this field of biomedical sensor systems.

1.2 Microelectronic Device Medical Applications

General applications of microelectronic devices in medical field can be roughly classified into two main categories, stimulation-purpose and sensor-based biological data measurement-purpose. For stimulation purposes, such devices are used to generate electrical signals to stimulate living tissues, such as muscular and nerve tissues, either continuously, periodically or only when required. For sensor-based biological data measurement purposes, these devices can be used to sense or measure a wide range of data. For example, biological data such as blood pressure, bladder pressure, gastrointestinal pressure, blood pH, gastric pH, or cardiac output, can be measured and used to provide valuable information in medical and/or clinical studies.

Three popular microelectronic-based systems in medical use are cardiac pacemakers, cochlear implants, and capsule imaging devices. The basic design concepts of these devices will be studied in the following sections.

1.2.1 Cardiac Pacemaker

A cardiac pacemaker unit monitors the electrical impulses in the heart and, when required, generates an electrical pulse with appropriate intensity to the myocardium of the heart to stimulate the heart to operate at a desired rate. Thus, a cardiac pacemaker's function is making the heart beat in a normal rhythm. A general block diagram of a cardiac pacemaker is shown in Figure 1.2.

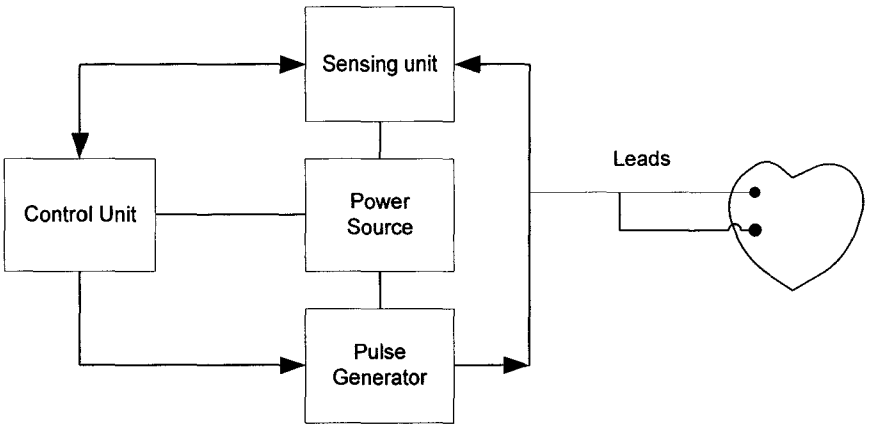


Figure 1.2: Block Diagram of Cardiac Pacemaker [4].

As shown in Figure 1.2, a cardiac pacemaker consists of the following components, a control unit, a sensing unit, a pulse generator, a power source and leads. Generally, the pacemaker is connected to the heart through one to three leads, which are attached directly to the heart's chamber. Leads are acting as primary sensing elements which convert heart beat to a myocardial potential. A sensing unit acts as the variable conversion element. A control unit represents the signal processing element in this system. The pulse generator is acting as the control and feedback element to provide stimulation to the heart as required.

Heat beats are converted to myocardial potential and passed to the sensing unit through the leads. The sensing unit amplifies, filters, and converts this potential to a digital format and sends it the control unit. The control unit, or signal processing element, determines the appropriate time to send a signal to control the pulse generator to generate a voltage pulse. The voltage pulse then creates an electric field of certain intensity to stimulate the myocardium and create a wave of action potentials.

To summarize, a cardiac pacemaker measures myocardial potential and determine when to stimulate the myocardium. Therefore, cardiac pacemakers have both stimulation and sensor-based measurement functions and no external parts are required to maintain their operation. Also, the majority of pacemaker designs are based on digital signal processing (DSP) electronics. Furthermore, wireless data transmission is not required in this type of application. Next, the design of cochlear implants will be discussed.

1.2.2 Cochlear Implants

A cochlear implant is an electronic system that restores hearing in people with profound hearing loss. It bypasses the inner ear and provides information to the hearing centres through direct stimulation of the hearing nerve. The receiver and stimulator parts of the device are surgically implanted into inner ear. The microphone, speech processing and transmitter units are worn externally. The cochlear implant stimulates the auditory or hearing nerve directly, bypassing the damaged part of the inner ear or cochlea. An illustration of a cochlear implant device is shown in Figure 1.3. The components shown in Figure 1.3 are a) microphone; b) behind-the-ear speech processor; c) body-worn speech processor; d) transmitting aerial; e) receiver-stimulator; f) electrode bundle; g) cochlea; h) auditory or cochlear nerve [5].

The operation of a cochlear implant is now briefly described. On the outside, the microphone picks up sounds and sends it to the speech processor. The speech processor takes the acoustic signal and converts it to digital information to be transmitted wirelessly. This information is sent to the transmitter and is passed on through the skin to the implanted part of the device. Then, the receiver implanted under the skin decodes this information and transfers it to the electrode array. The electrode array distributes this information and generates stimulations to the nerve endings in the cochlea to produce nerve impulses. Finally, these nerve impulses are sent to the brain where they can be interpreted as sound [6].

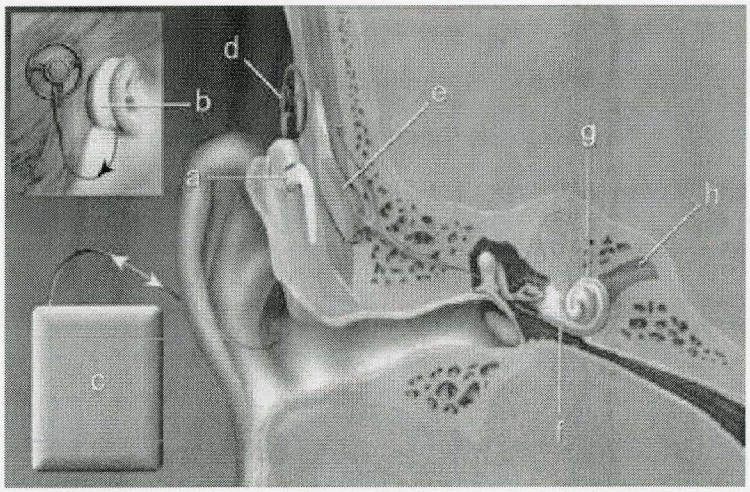


Figure 1.3: Illustration of Ear with Cochlear Implant [5]

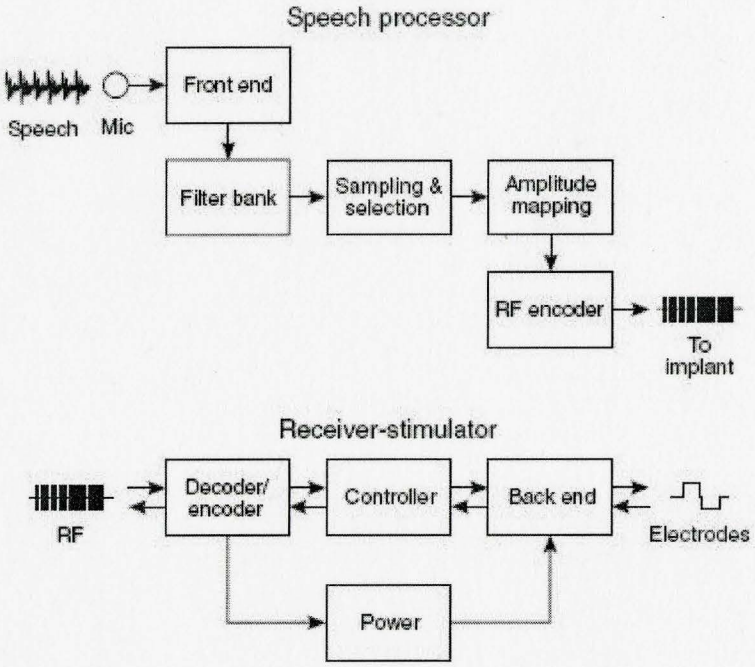


Figure 1.4: Block Diagram of a Cochlear Implant [5].

A generalized block diagram of a cochlear implant is shown in Figure 1.4. The external part of a cochlear implant, the speech processor consists of a microphone to capture acoustic signals, a front-end to optimize the signal, a filter bank to select desired

frequencies, a sampling and selection stage to process the signal, an amplitude mapping stage to match the processed signal to human domain, and an RF encoder to encode the signal for transmission to the implanted part of the device, the receiver-stimulator. The receiver-stimulator part of the device consists of a decoder/encoder stage which extracts power and signals from the inductive link, a controller that controls the distribution of stimulation signals and a back-end stage to facilitate the stimulations to the electrodes.

In this type of device, the speech processor and receiver-stimulator together act as the central signal processing stage. The processed signals are then sent to the receiver-stimulator which acts as a control and feedback element in the system. In this case, the primary sensing element, a microphone, measures acoustic signals from outside of the body. Then elements of speech processor, such as front-end, filter bank, and sampling and selection stage, represent the variable conversion element.

In summary, a cochlear implant converts acoustic signals to stimulations on the cochlea nerve endings and no measurements of biological data is required. Therefore, a cochlear implant only possesses cochlea stimulation function and is composed of both external and internal parts. Again, it is observed that the main part of this device design is based upon a DSP chip. Furthermore, an inductive link is used to create the communication between the RF encoder of the external device and the decoder of the implanted device. Through this inductive link, power is being supplied to the implanted device at a particular frequency and the data signal is amplitude-modulated at the same frequency.

1.2.3 Capsule Imaging Systems

Since the invention of a capsule imaging system in 2001, capsule endoscopy has been applied for the diagnosis of some types of gastrointestinal diseases. In this section, the study of a capsule imaging device will be presented to understand its design.

The basic function of a capsule imaging device is to capture images of a patient's gastrointestinal (GI) tract and wirelessly transmit these images to a wearable data recorder

positioned on a patient's belt. A functional block diagram of such device is shown in Figure 1.5. The device is primarily composed of the following components, as shown in Figure 1.5: 1) a lens; 2) light-emitting-diodes (LEDs); 3) an imaging sensor; 4) batteries; 5) a RF transceiver, and 6) an antenna.

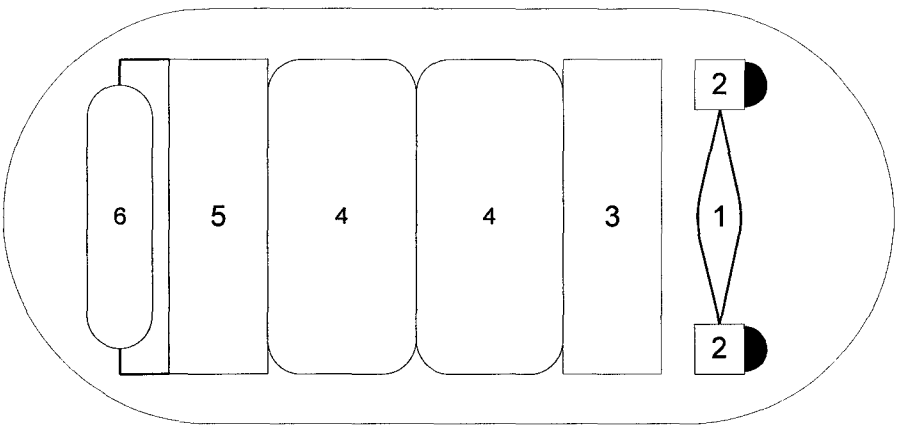


Figure 1.5: Structure of a Pill Imaging Device: 1) a lens; 2) LEDs; 3) an imaging sensor; 4) batteries; 5) a RF transceiver, and 6) an antenna.

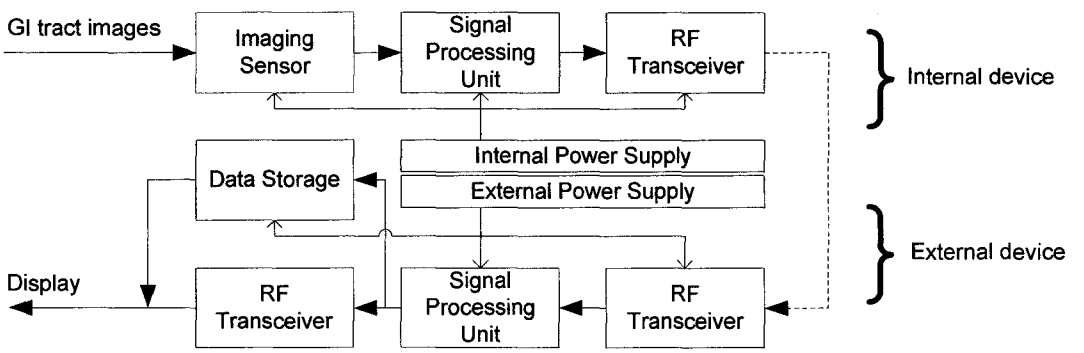


Figure 1.6: Block Diagram of a Pill Imaging Device.

In Figure 1.6, the external part of the system primarily consists of a signal processing unit, data storage, a power supply and one or two RF transceivers. The internal part of the system is composed of an imaging sensor, a signal processing unit, a power supply and an RF transceiver. The basic operation of such system is briefly described. First, the

imaging sensor measures the light intensity using its array of photodiodes and converts these signals to a readable image format. This image is then processed and compressed by the signal processing unit and is encoded for transmission. Later, this encoded message is transmitted by the RF transceiver to the external device. The external receiver decodes the message and passes it onto the signal processing unit for image storage or image recovery. Finally, stored images or real-time images are sent to a display unit for diagnosis [7].

In conclusion, a capsule-based imaging system has only the sensor-based measurement function which captures visual information of a patient's gastrointestinal tract. The system is composed of internal and external devices. It is observed that image compression process and the transmission rate are two important factors in the design of such system. Also, minimum size requirement and power consumption are also other key factors in the design.

1.3 Thesis Motivation

Wireless Microelectronic Device for Urinary Incontinence Studies

This section of Chapter 1 contains a discussion of the fundamentals of urinary incontinence. The types of urinary incontinence and methods of diagnosis are presented to provide an understanding of the challenges and drawbacks posed by ordinary clinical methods for examining and monitoring patients, in both short-term and long-term studies. In later part of this chapter, a proposed wireless microelectronic system and method for detecting episodes of urine leakage are presented. Detecting and recording such episodes can provide useful information for further clinical studies and researches, and remedies.

1.3.1 Urinary Incontinence Overview

Urinary incontinence is a common and often embarrassing problem. The severity of it ranges from occasionally leaking urine when coughing and sneezing to having sudden, unpredictable episodes of strong urinary urgency. Sometimes, the urgency may be too strong to control. The Canadian Incontinence Foundation estimates that there are

over 3.3 million Canadian suffering from all types of urinary incontinence [8]. Although urinary incontinence is prevalent, it is not a normal part of aging. In women, urinary incontinence could be an inevitable consequence of childbirth or changes after menopause. It is a medical condition with many possible causes, some relatively simple and self-limited (e.g. urinary tract infection, obstruction of the bladder outlet, etc.) and others more complex (e.g. interstitial cystitis, prostate enlargement, bladder cancer, etc.).

Urinary incontinence has been widely studied for the past few decades, but methods for detection of urinary incontinence in the long-term ambulatory monitoring (LTAM) studies are still unclear. Even though ordinary pad tests can assist in detecting the episodes of urinary incontinence during LTAM studies, the episodes of urinary incontinence cannot be recorded. Therefore, to detect and record such episodes seem to be necessary in LTAM studies. Detection and record of urinary incontinence episodes in LTAM studies could provide valuable information for clinical research. This information could be used in control devices such as bladder control actuators, imaging capsules, or to provide more information for medical diagnosis.

1.3.2 Types of Urinary Incontinence

Stress Urinary Incontinence

Stress urinary incontinence represents the loss of urine when pressure has been put on the bladder by coughing, sneezing, laughing, exercising or lifting heavy things. It has nothing to do with psychological stress. Stress urinary incontinence occurs when the sphincter muscle at the bladder is weakened. The problem is especially noticeable when the volume of urine approaches to the bladder's limit. Stress incontinence is also one of the most common types of incontinence affecting women. Physical alterations resulting from pregnancy, childbirth and menopause can all cause stress incontinence. In men, removal of the prostate gland can also lead to this type of incontinence [9].

Urge Urinary Incontinence

Urge urinary incontinence is when a person feels a sudden and intense urge to urinate followed by an involuntary loss of urine. This type of urinary incontinence may be caused by a urinary tract infection or by anything that irritates the bladder. It can also be caused by bowel problems or damages to the nervous system associated with multiple sclerosis, Parkinson's disease, Alzheimer's disease, stroke or injury. In urge urinary incontinence, the bladder is said to be overactive. In this case, the bladder can be contracting even when the bladder is not full [9].

Overflow Urinary Incontinence

Overflow urinary incontinence is an inability to empty the bladder which leads to overflow. With overflow urinary incontinence, a patient may feel as if he or she never completely empties his or her bladder. This type of urinary incontinence is common in people with a damaged bladder or blocked urethra and in men with prostate gland problems. Nerve damage from diabetes also can lead to overflow urinary incontinence. Some medications can also cause or increase the risk of developing overflow urinary incontinence [9].

Total Incontinence

Total urinary incontinence is described as the complete absence of control, either continuous leakage or periodic uncontrolled emptying of the bladder's contents in [8].

1.3.3 Specialized Diagnosis Methods of Urinary Incontinence

Postvoid Residual Measurement

A postvoid residual measurement is done as follows: First, the patient is asked to empty his/her bladder content. This allows for the measurement of the volume of a bladder to be performed. Then, a catheter is inserted into the bladder to measure the

amount of residual urine. The releasing function of a bladder can be assessed by the amount of residual urine left in the bladder.

Urodynamic Testing

This test measures the bladder's pressure when it's empty and during the filling process. A catheter insertion is required to fill the bladder with water or to release the water while a pressure transducer as a part of the catheter measures and records the pressure. A normal bladder should compensate for the increase in volume by relaxing the surrounding muscle. In Figure 1.7, the cystometric curve of a normal bladder is shown.

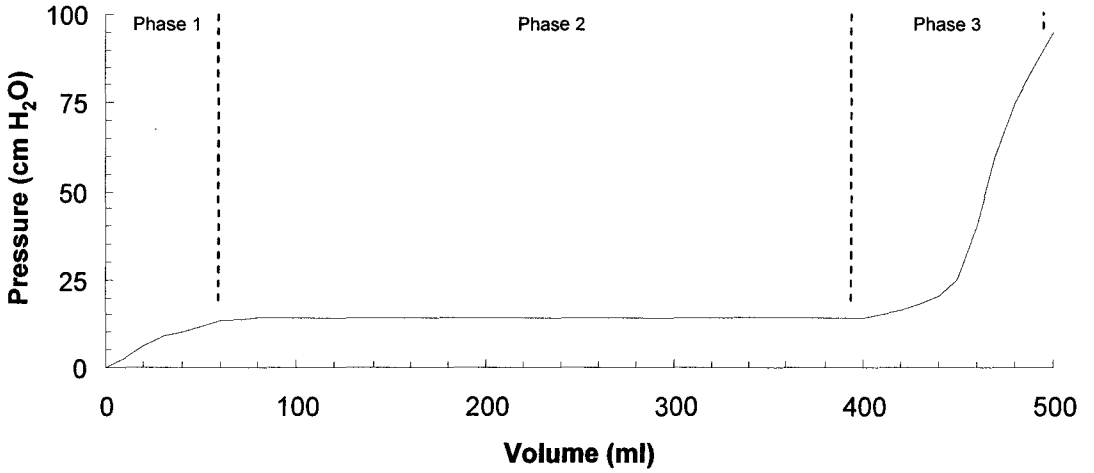


Figure 1.7: Cystometric Curve of a Normal Bladder [10].

A normal cystometric curve is divided into three phases: 1) phase one: an initial phase of pressure elevation to reach a resting bladder pressure of 2-8 cmH_2O (determined by the viscoelastic properties of the bladder wall), 2) phase two: a second tonic phase with minimal pressure increase during accommodation of a large volume of fluid, and 3) phase three: a final evacuation phase during which intravesical pressure increases to empty the bladder of fluid. During the second phase of tonic bladder filling, bladder pressure should not raise more than 10 cmH_2O. Moreover, Figure 1.7 is obtained by performing a cystometry on a patient's bladder in an urodynamic evaluation. A cystometry involves the instillation of a filling media into the bladder with simultaneous measurement of intravesical pressure as the bladder fills, producing a curve of intravesical pressure plotted

against volume. In the cystometry, the rate of bladder filling is assumed to be a medium-fill rate (10-100 ml/min) [10]. Then, by assuming a medium bladder filling rate of 50 ml/min, a pressure-time graph of a normal bladder is shown in Figure 1.8.

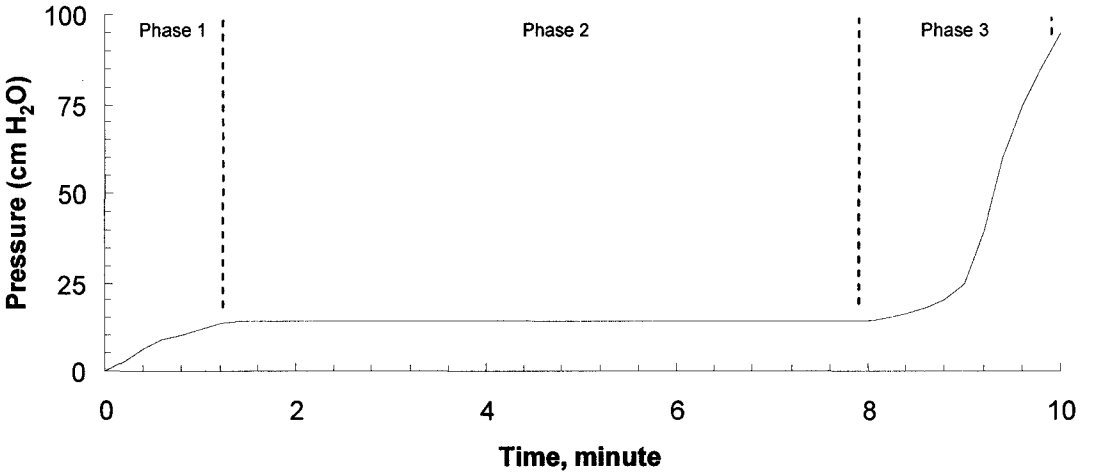
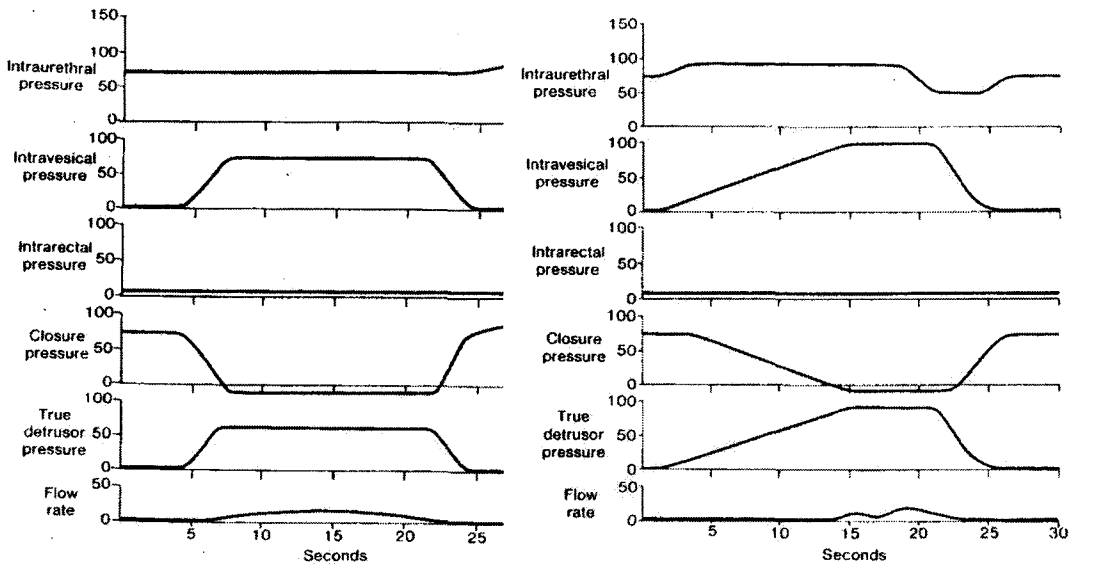


Figure 1.8: Pressure-Time Curve of a Normal Bladder [10].



(a) (b)

Figure 1.9: Abnormal Voiding Patterns: (a) Abnormally High Intravesical Pressure; (b) Detrusor Sphincter Dyssynergia [10].

In Figure 1.8, it can be seen that in the initial phase of bladder filling, the bladder pressure is increased by about 15 cmH_2O over a duration of 60 seconds (0.25 cmH_2O/second). In

the final evacuation phase, the bladder pressure is increased by 85 cmH_2O over a duration of about 1 minute. The rate of pressure increasing during the last phase is approximately 1.42 cmH_2O/second. In addition, examples of abnormal voiding patterns are shown in Figure 1.9. In Figure 1.9(a), an abnormal pressure increasing rate is about 35 cmH_2O/second, and in Figure 1.9(b), an abnormal pressure increasing rate is about 7.7 cmH_2O/second. Therefore, it is observed that the rate of pressure increase is less then 2 cmH_2O/second in a normal bladder.

Cystogram

In a cystogram, injecting a fluid containing a special dye through a catheter into a bladder is a required procedure. A series of X-ray images are taken during the micturition process to assist the diagnosis of problems in a patient's bladder tract.

In summary, three diagnosis methods described above all require the insertion of a catheter into a patient's bladder. The insertion of a catheter is not only painful, but it could also lead to infections. The author in [11] clearly states that the duration of catheterization is the most important risk factor for the development of catheter-associated bacteriuria, a disease caused by bacteria infection in the bladder. The author also points out that there is a 10% to 30% chance that a catheterized patient would develop bacteriuria during short-term catheterization. For long-term catheterization, most patients will develop bacteriuria.

Therefore, the needs to detect and record urinary incontinence episodes and to develop possible solutions to reduce bacteriuria infection of catheterized patients have stimulated us to design a device which can be utilized in the future clinical studies of urinary incontinence. The proposed method of urine leakage episodes and design criteria of our device are presented next.

1.3.4 Proposed UI Episodes Detection Method and Design Criteria

In the studies of [10], it is observed that uninhibited contractions or attempted inhibitions of a bladder can lead to episodes of urine leakage. Urine leakage caused by uninhibited contraptions and/or attempted inhibition of a bladder can be seen in Figure 1.10 and Figure 1.11.

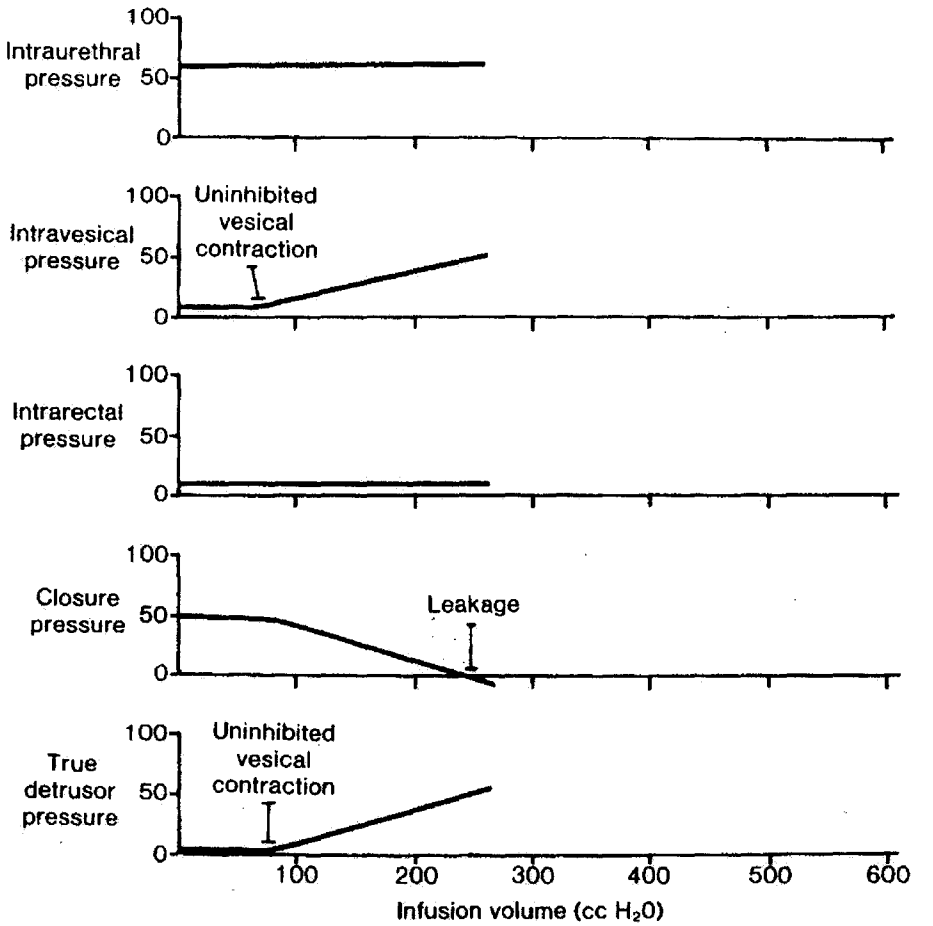


Figure 1.10: Cystometrogram of a Hypertonic Bladder [10].

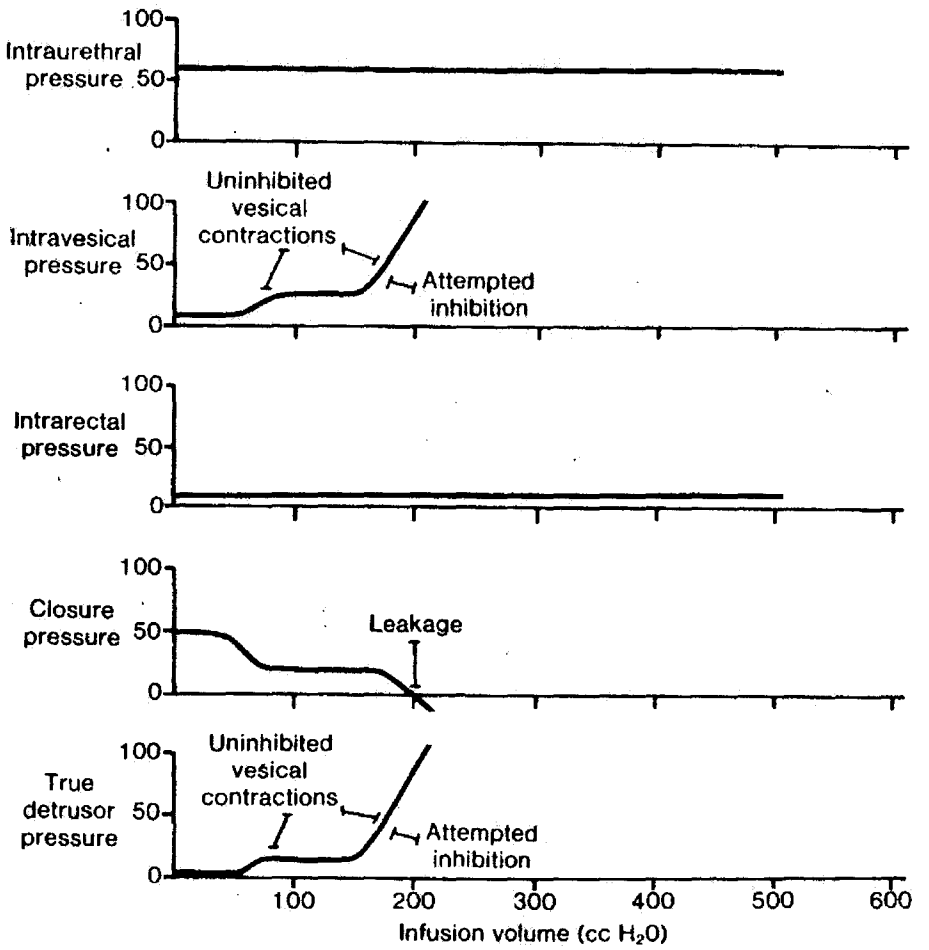


Figure 1.11: Another Cystometrogram of a Hypertonic Bladder [10].

It is also stated in [10] that uninhibited contractions occur in bladders where rapidly increasing pressure can happen and attempted inhibition can occur in a hypertonic bladder with gradual pressure increase at low bladder volumes. Therefore, detection of episodes of urine leakage can be done by monitoring bladder pressure for events where pressure changes rapidly (faster than 2 cmH_2O/second) or rises above normal level (about 15 cmH_2O) during the second phase of bladder filling. Also in practice, a patient should be asked to empty his/her bladder before the start of our proposed detection method to ensure different phases of the bladder filling can be captured.

Moreover, the design and evaluation of a wireless microelectronic device is required if efforts to replace the catheterization procedure are to be successful. This will also minimize the risks of catheterization while providing the functionalities of monitoring bladder pressure and detecting leakage episodes.

In addition, it is stated in [44] that a patient might have upper urinary tract deterioration if the leakage point pressure is greater than 40 cmH_2O. In [45], authors have concluded that the leakage point pressure measurements are useful in stress and/or urge incontinence patients. Therefore, the design for a urinary leakage detection system should meet the following four criteria.

- Functions in bladder of pressure range 1 to 150 cmH_2O.
- Monitors the acquired pressure for leakage episodes by detecting events where rapid changes (2 cmH_2O/ second) in pressure occur.
- Transmit acquired data for diagnosis.
- Have a minimal overall dimension.
- Have a pressure sampling rate of at least 10 samples per second [3].
- If a pH transducer is used, it should have a sampling rate of at least 2 samples per second [3].
- If a temperature transducer is used, it should have a sampling rate of at least 1 sample every 10 seconds [3].

1.4 Contribution

In this thesis, the design procedure, sensor calibration and real-time measurement results for two wireless sensor systems for detection of leakage episodes in patients with urinary incontinence are described and discussed. Both systems are built with off-the-shelf components to provide a cost-effective solution to the clinical and medical researches in urinary incontinence studies. The systems presented in this thesis can also be used in where in-vivo monitoring of biological

pressure and/or temperature is necessary. In addition, other sensors such as pH sensor can also be substituted to provide different information.

1.5 Thesis Organization

In Chapter 2 of this thesis, a comprehensive study of several microelectronic implants is presented. Systems targeted for sensor-based measurement purposes have been selected to provide basic understanding of them. For each system, various design aspects are studied. At the end of the chapter, a summary of these systems and the general specifications of system design are presented. Chapter 3 presents the implementation the first wireless sensor system. A pressure sensor is used to detect rapid changes of pressure. Chapter 4 describes the implementation of the second wireless sensor system in which a different method of wireless communication was used to study benefits of various communication schemes. Next, experimental results of pressure and temperature experiments are described in Chapter 5. Then, in Chapter 6, comparisons of system performances of both systems are presented. And finally, Chapter 7 summarizes the work performed in the previous chapters and possible improvements of the work are also given in this chapter.

CHAPTER 2: LITERATURE REVIEW

In this chapter, detailed discussions of several implantable microelectronic systems are presented. In particular, systems targeting for sensor-based biological data measurement have been selected to provide a sound understanding of system implementation in this field. For each system, a brief overview of its application and functionality will first be discussed. Moreover, aspects such as the design of implantable part and the external part of the system, communication method, and the system's overall performance are discussed.

This investigation has two primary objectives. The first objective is to acquire a sound understanding of the design of such implantable systems. The second objective is to utilize the information studied and to implement a system using the most commonly available components (e.g. off-the-shelf components). In the following sections, several different systems which are relevant to the design of our desired system are studied. At the end of this chapter, a summary of systems studied is provided.

2.1 Cases of Implantable Microelectronic Systems

2.1.1 A Microcontroller-Based Implantable Telemetry System for Sympathetic Nerve Activity and ECG Measurement [12]

A passive telemetry system was proposed in [12]. This system meets the need for continuous recording of ECG signals and sympathetic nerve activity for cardiovascular neural control research with conscious animal preparations. The proposed system is composed of a personal computer, a backpack receiver, and an implantable telemeter. The primary function of the personal computer is to receive and display ECG and nerve signals obtained by the telemeter. The backpack receiver is to provide power to the implantable telemetry through an inductive link and to transfer measured ECG and nerve signals through an FM transmitter to the personal computer. The primary function of

implantable telemeter is to acquire ECG and nerve signals and then processed them for transmission. An illustration of the system is shown in Figure 2.1.

Implantable Part

As seen in Figure 2.1, the implantable part of this system consists of an ECG amplifier, a nerve signal amplifier, a microcontroller, a power receiving circuit and an RLC passive series resonant circuit. Firstly, amplifiers are used to provide gain to the ECG and nerve signals. Then a microcontroller is used to receive the gain-controlled signal, which was provided by the backpack through the power receiving circuit with on-off-keying modulated wave to control the gain of these amplifiers. The microcontroller also samples the amplified ECG and nerve signal using its internal analog-to-digital converters. Lastly, the sampled signals are being processed by the microcontroller and a 21-bit serial digital data is produced to control the oscillation of the RLC resonant circuit.

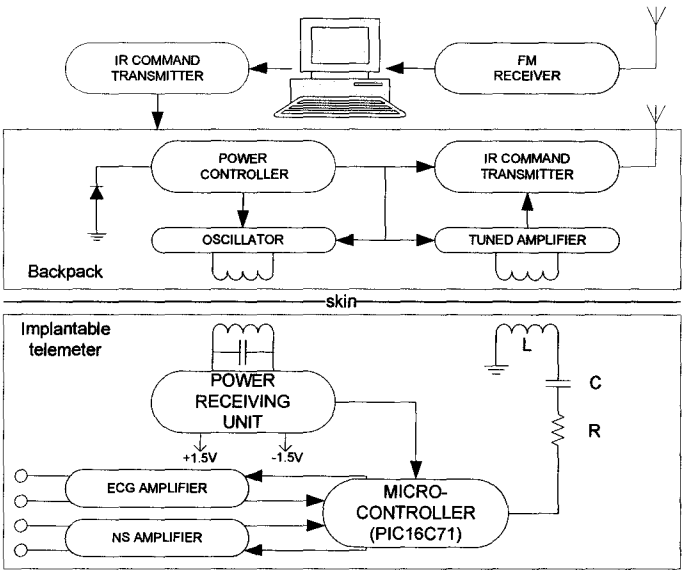


Figure 2.1: Microcontroller-based Implantable Telemetry System [12].

External Part

The external part of the system, the backpack shown in Figure 2.1, consists of a power controller, an oscillator, a tuned-amplifier, and a FM transmitter. The backpack can

be switched on by the infrared transmitter on a personal computer. The oscillator controlled by the power controller generates a 9 V peak-to-peak (VPP) sinusoidal wave at 200 kHz to power the implantable telemeter through the inductive link. The tuned-amplifier picks up oscillations generated by the 21-bit serial data and shapes it for transmission. Finally, the FM transmitter transmits this data to a personal computer for display and storage.

Communication Method

Between the telemeter and the backpack, the gain control signal is done through a 144 bps on-off-keying of the 9 VPP sinusoidal wave at 200 kHz and the 21-bit serial data from the telemeter is sent to the passive RLC resonant circuit, which has a resonant frequency of 8 MHz. This signal is then received by a tuned amplifier in the backpack and transmitted to the personal computer through an FM transmitter.

Overall Performance

The implantable telemeter is built on a 40 × 35 mm^2 printed circuit board. It has a total volume of 15 cc and a weight of 46 g. The data transmission range is up to 8 cm. Power consumption of the implantable telemeter is not specified and is not a major concern since the power is provided wirelessly by the backpack.

2.1.2 Design of Miniaturized Telemetry Module for Bi-Directional Wireless Endoscopy [13]

Only one-way communication, transmission from the imaging capsule to a receiver outside of body, is established in the capsule imaging system proposed in [7]. There is currently no easy way of controlling the imaging capsule. Therefore in [13], the authors have proposed a telemetry module to provide functions such as real-time camera control and illumination. The proposed system consists of an imaging capsule, an external controller, and a NTSC television. The primary function of this system is to obtain in-

body video and transmit it wirelessly to a NTSC television. A conceptional diagram of this system is shown in Figure 2.2.

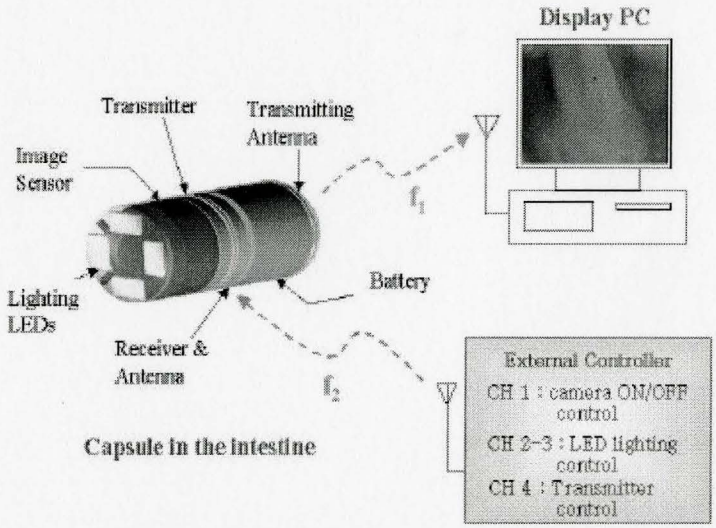


Figure 2.2: Conceptional Diagram of Bi-directional Wireless Endoscopy System [13].

Implantable Part

The proposed imaging capsule has the following components, a transmitter with antenna, a receiver with antenna, 4 LEDs, and an image sensor. The image sensor is used to capture images from inside the body. These images are transmitted through the transmitter. In addition, the output of the image sensor is NTSC compatible, which means it can be directly interface with any NTSC television. The receiver is used to receive control signals from outside the body and to provide these signals to the image sensor and the LEDs. Lastly, the LEDs are used to provide illumination for the image sensor.

External Part

The external part of the system consists of a NTSC television and a transmitter. The transmitter is to provide control signals for the LEDs and the image sensor to the imaging capsule. The television is to receive images and to display them directly.

Communication Method

The captured images are being amplitude-modulated at 315 MHz. This frequency is also used by a commercial television channel. Therefore, these images can be displayed directly on a certain channel of a NTSC television. On the other hand, the control signal is modulated using on-off-keying on a carrier frequency of 433 MHz. The receiver of the imaging capsule decodes this signal and provides controls to the CMOS image sensor and the LEDs.

Overall Performance

The completed imaging capsule is 11 mm in diameter and 7 mm in length. The external control-signal transmitter has a dimension of 20 cm × 30 cm × 15 cm. According to the specification of OV7910, the image sensor, it requires a minimum of 200 mW of power. That is, while operating at 5 V, it consumes an electrical current of 40 mA.

2.1.3 An Implantable Telemetry Platform System for In Vivo Monitoring of Physiological Parameters [14]

In [14], the authors are developing a minimally invasive monitoring system which can reduce patient's pain and discomfort. In addition, the authors in this paper also identify that most of the existing telemetry systems do not take advantage of microcontroller-based architecture and digital codification of the transmitted signal. For these reasons, they proposed a microcontroller-based multi-channel telemetry system to provide a versatile, implantable and reprogrammable telemetry module with the ability to interface with different types of sensors. The proposed system is composed of an implantable unit, a receiving unit and a graphical-user-interface (GUI) implemented on a personal computer.

Implantable Part

The block diagram of the implantable unit is shown in Figure 2.3. The implantable unit consists of a pressure sensor, a microcontroller, an antenna and a battery. The core component of this unit is a microcontroller, rfPIC12F675F. The microcontroller offers a 10-bit ADC with a multiplexer to select from different input channels. It also has a built-in transmission module. The microcontroller is operating at a speed of 4 MHz. The

battery is connected to the implantable unit via a magnetic switch. Therefore, acquisition of the implant can be turned on by applying a magnetic field near the device. Finally, a pressure sensor, LL-3-072-15, from Kulite Semiconductor is used as the primary sensing element.

External Part

An off-the-shelf receiver is used as the external receiver of the system. A serial level adapter is used to convert the output of the receiver to RS232 standard, which subsequently allows direct connection of the receiver to a personal computer. Moreover, a GUI is developed using LabView 7 Express software from National Instruments.

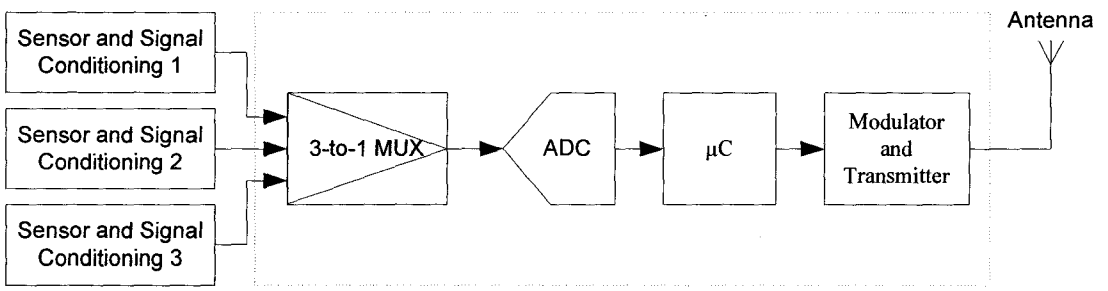


Figure 2.3: Block Diagram of Implantable Unit [14].

Communication Method

The frequency of the transmission module in the microcontroller is set externally by an oscillating crystal at 433.92 MHz. The sampled data are encoded by amplitude-shift-keying (ASK) modulation. Between the receiver and the personal computer, the serial standard EIA232C, with 2400 bps has been used. The transmitted message contains two identification bytes to ensure proper transmission.

Overall Performance

The implantable unit has a packaged size of 19 mm in diameter with a height of 27 mm. The battery used in the unit is a 3-V CR1025 lithium ion cell, with a capacity of

30 mAh at room temperature. The authors also stated that the unit has a maximum operation lifetime of 2 weeks and a worst-case lifetime of 56 hours. This estimation is made without taking the power consumption of pressure sensor into consideration. According to the specification of the pressure sensor used in this paper [14], the input resistance of such sensor has a minimum value of 1000 Ω. With an input resistance of 1000 Ω and when the device is operating at 3 V, this pressure sensor requires an electrical current of 3 mA. Therefore, when the pressure sensor's power consumption is taken into account, the operation life time of the implantable unit will be greatly reduced.

2.1.4 A High Level Language Implementation of a General Purpose Telemetry System for Biomedical Application [15]

Lack of long term data of the structural integrity of artificial implants, and the need of real-time monitoring of fracture fixation plates and intramedullary rods, have led authors in [15] to discuss their design of a telemetry system that provides long term and real-time acquisition of these data. Thus, the design of a monolithic microminiature implantable data acquisition and telemetry system for use with a variety of sensors is proposed in [15]. The proposed system consists of two parts, a biological implant and a remote monitoring unit. The primary function of such system is to acquire information from inside a patient with artificial implants and to transmit the information to the remote monitoring unit at pre-programmed times. In addition, the system should be able to act as a host for generic biological sensors while having the ability to exploit the full range of these devices. The primary function of the remote monitoring unit is to receive, process, and store the acquired information from the implant. In addition, the remote monitoring unit should have the ability to communication with a host workstation and to receive programming instructions from the host for the implant. Lastly, the initial application of this system is to measure orthopaedic implant stress using strain gage sensors.

Implantable Part

A schematic illustration of the biological implant is shown in Figure 2.4. The implant is composed of a digital controller (a very-large-scale integration (VLSI) die), two sensors, a battery and an antenna. The VLSI die, acting as the central controlling unit of the implant, is composed by the following partitions, control logic, communication interface, sample & timing interface, and lastly the memory controller. The control logic is based on the architecture of a microcontroller, which has a set of instruction for various operations. The communication interface is made of a transmission verification logic, which determines the validity of each incoming transmission from the host. The sample and timing interface is basically used to control the analog-to-digital converters and its associated multiplexers. The memory controller has the ability to manipulate available memories in the digital controller.

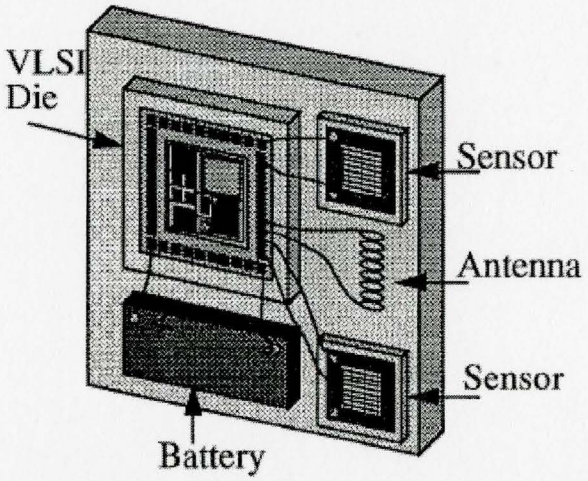


Figure 2.4: Illustration of Implant Environment [15].

External Part and Communication Method

No information about external part of the system was given [15]. There is also no information regarding the specific communication frequency, data modulation method, or the design of the antenna.

Overall Performance

Unfortunately, the authors only focus on the design of the digital controller part of the implantable device. No overall performance of the system, as a whole, was given in the paper [15]. However, the final design dimension of the digital controller is reported to be 2.5 mm × 2.5 mm and it is fabricated using Hewlett Packard 0.8 μm process.

2.1.5 Battery-less Wireless Communication System [38, 39, 40, 41]

Due to the requirement of very small size in an endoscopic capsule, the authors in [38, 39, 40, and 41] have designed a small-size healthcare device with wireless communication capabilities through a human body. The proposed system consists of two parts, a capsule device and an external antenna to power and to receive data from the capsule. The application of this system is to measure the pH level in the stomach. A schematic of this system is shown in Figure 2.5.

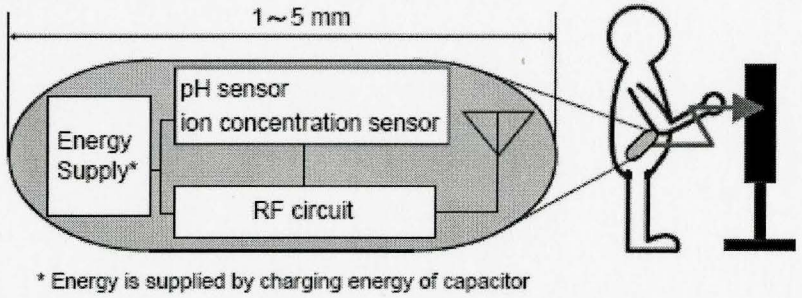


Figure 2.5: Schematic View of *In-vivo* Wireless Communication System [38].

The primary function of the capsule is to measure pH level in the stomach and to transmit the data through human body through an inductive link with pulse interval modulation (PIM).

Implantable Part

A block diagram of the wireless communication system in [38] is shown in Figure 2.6. The capsule part of the proposed system consists of 3 capacitors, a PIM circuit, a

cascaded voltage booster circuit, and a control circuit. The cascaded voltage booster circuit is used to boost the coil voltage in order to charge the capacitors. The PIM circuit acts as the wireless transmitter in this system. The control circuit is controlling the PIM circuit to generate pulses with different intervals.

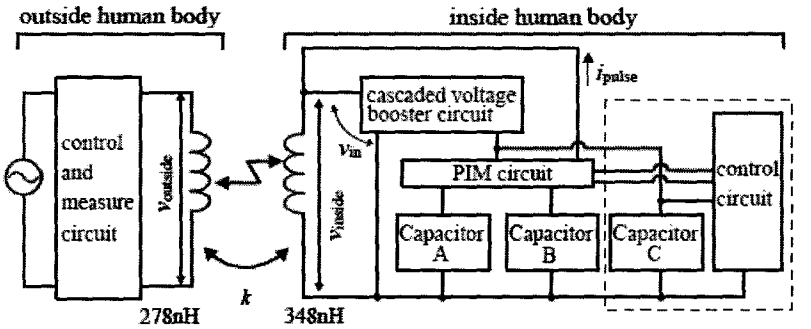


Figure 2.6: Block Diagram of the Wireless Communication System [38].

External Part and Communication Method

The communication method used in the system is the pulse interval modulation (PIM) at 13.56 MHz. In PIM, intervals between generated pulses are representing measured data in the system.

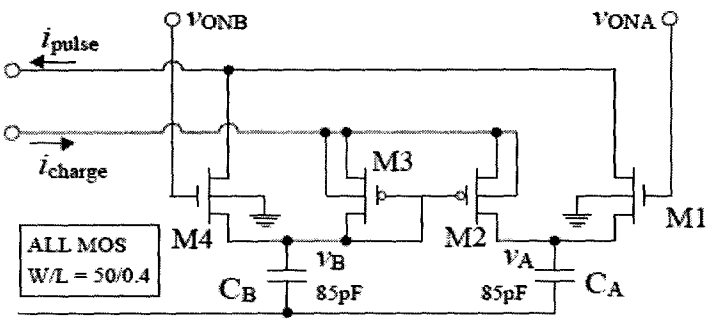


Figure 2.7: PIM Circuit [38].

The PIM circuit of the system is shown in Figure 2.7. In the charge phase, the transistors, M1 and M4 are off and i_{charge} is used to charge the capacitors. When v_B is greater than the threshold voltage of M2 and M3, the capacitor charging is stopped. In the transmission

phase, M1 and M4 are turned on. Then, two pulses are generated by C_B and C_A. Lastly, no information on the external part of the system is given in the paper.

Overall Performance

In [38], the authors focus on the design of the transmitter of the system. No information on the control circuit is given. The system is custom designed using 0.35 μm Si CMOS process and the core size of the system is 1.8 mm^2.

2.1.6 SmartPill GI Monitoring System [42, 43]

This is a commercially developed system to monitor pH, pressure, and temperature from within human's gastrointestinal (GI) tract. The main application of this system is to provide gastric emptying time, combined small and large bowel transit time, total transit time, pressure contraction patterns from the antrum and duodenum and motility indices. A picture of this system is shown in Figure 2.8.

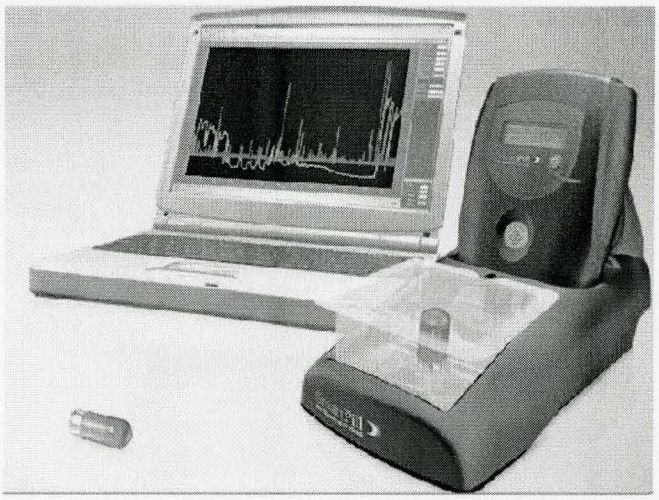


Figure 2.8: SmartPill GI Monitoring System [42].

Implantable Part

"The SmartPill pH.p capsule is a single-use, ingestible capsule that measures pressure, pH and temperature from within the entire GI tract and wirelessly transmits that

information to a Data Receiver worn by the patient." [42] The capsule has a pressure sampling rate of 2 times per second, a pH sampling rate of 1 time every 5 seconds, and a temperature sampling rate of 1 time every 20 seconds [43]. The capsule has dimensions of 26 x 13 mm^2 and a transmission distance of 5 feet [43].

External Part and Communication Method

A data receiver [42] is used to receive data from the capsule. It can also communicate with a PC via USB in real-time or store a complete record of data for further analysis. No further information about this commercially developed system is given.

Overall Performance

Table 2.1: Data Collection Parameters of SmartPill pH.p Capsule [43].

Component	Range	Accuracy	Calibration
pH Sensor	1 to 9 pH unit	+/- .5 pH units	single point @ pH 6
Pressure Sensor	0 to 350 mmHg	0-99 mmHg +/- 5 mmHg 100-350 mmHg +/- 10% of applied pressure	Automatic
Temperature Sensor	20 to 40 Deg C	+/- 1 Deg C	N/A

A table of sensors' parameters are shown in Table 2.1. Additional operational specifications are listed below [43],

1. Capsule Size: 26 x 13 mm
2. Battery Life:
   - pH.p Capsule: 5+ days
   - Data Receiver: 5+ days (at full charge)
3. Transmission Interval: 1x / 20 sec (0-24 hours)
   1x / 40 sec (24 hours to end of life)
4. Transmission Distance: 5 feet (BMI dependent)

2.2 Chapter Summary and General Specifications for Design

In this chapter, six implantable microelectronic systems related to the design of wireless microelectronic implantable device have been studied. The most important information about these systems' implantable and external parts has been presented. It is observed that a microcontroller-based system offers great flexibility in the design of such systems and an off-the-shelf transceiver could be used to simplify the overall design. In addition, powering implantable units with an inductive link seems to be a promising technique to reduce overall system size.

In summary, general specifications for design of wireless sensor system is listed below.

General specifications,

1. The primary sensing element for design should be a pressure sensor with adequate sensitivity.
2. A microcontroller should be used as the variable conversion and signal processing unit.
3. The data transmission function should be provided using a transmitter.
4. A graphical user interface (GUI) software should be implemented to act as the output display unit in the design.

CHAPTER 3: IMPLEMENTATION OF 2.4 GHz WIRELESS PRESSURE SENSOR SYSTEM

The goals of this 2.4 GHz wireless pressure sensor system are to design a small pressure sensor system that can monitor bladder pressure wirelessly and indicate rapid changes in bladder pressure. This design is also to provide insights into systems that can collect, process, and transmit data wirelessly. By monitoring rapid changes in bladder pressure, the episodes of urine leakage can then be detected. This signal can be further used to control other related devices to aid in clinical studies and researches. Another objective is to build this prototype using off-the-shelf components.

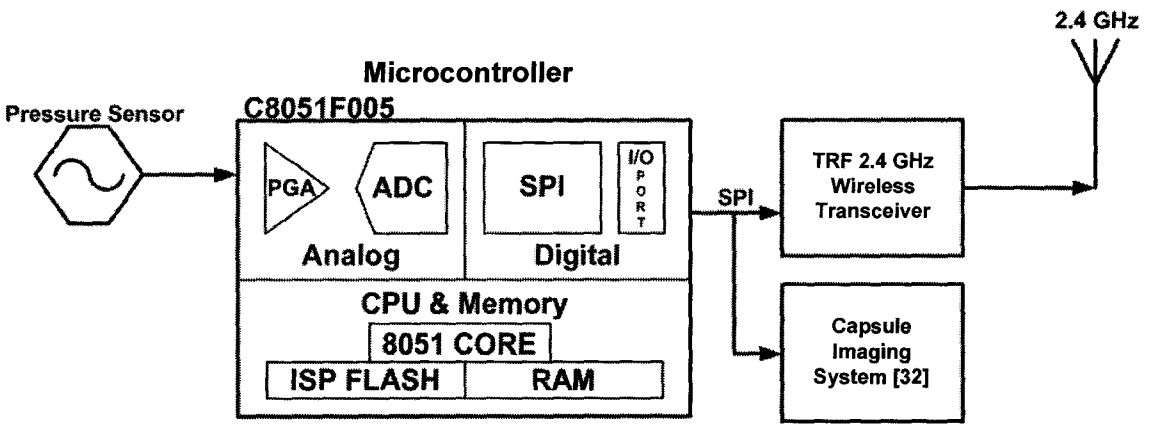


Figure 3.1: Block Diagram of 2.4 GHz Wireless Pressure Sensor System [16].

The system block diagram is shown in Figure 3.1. The system consists of a pressure sensor, a microcontroller, a wireless transceiver, and an antenna. This system is built completely with off-the-shelf components. It was designed to sample data using a microcontroller's analog-to-digital converter. Then, sampled data is processed with simple digital signal processing algorithms. Lastly, the data is passed onto a receiving station using a wireless transmitter. In addition, this device is designed to function as a pressure module which can be used by a previously developed wireless imaging system in

our research group [32]. Communication between this module and the imaging system is done via a serial-port-interface (SPI).

This chapter presents a detailed description of the design of the 2.4 GHz wireless pressure sensor system. Frequency band description, possible variants, electrical design, and implementation of the system are presented. At the end of this chapter, technical specifications of the system extracted from the design are summarized.

3.1 2.4 GHz ISM Band

The Federal Communication Commission (FCC) has defined 2450 MHz ± 50 MHz as one of the industrial, scientific, and medical (ISM) radio bands. This frequency band is chosen for this prototype because they are given over to experimental radio service [33]. The maximum exposure limits of human body for microwave (2.4 GHz) are defined by Health Canada as the following. The rms (root-mean-square) value of the electric field strength should not exceed 137 V/m, and the rms value of magnetic field strength should not exceed 0.364 A/m. In addition, the SAR (Specific Absorption Ratio) limit over the whole body mass should not exceed 0.4 W/kg, and the SAR for trunk, averaged over any one gram of tissue should not exceed 8 W/kg [46].

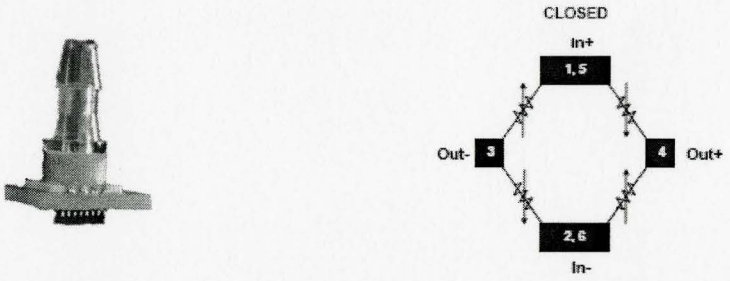
3.2 Possible Variants – Components Selection

To achieve stated design specification, a careful selection of the components is required. In this section, detailed descriptions of selected components are presented to provide a sound understanding of why these specific components are chosen.

Pressure Sensor

To obtain accurate vital signals is critical for medical uses and researches. For intravesical pressure examinations, the required pressure range is from 0 to 150 cm H_2O, which is approximately 0 to 2 PSI (pounds per square inch). It is important to have a

suitable pressure sensor that can provide good sensitivity and linear output. Therefore, an absolute pressure sensor from Merit Sensor Systems was chosen. The pressure sensor can be seen in Figure 3.2(a), and its configuration is shown in Figure 3.2(b). The pressure sensor is in a typical Wheatstone bridge configuration, in which, changes in pressure modifies the resistances' values. Therefore, changes in resistances of a Wheatstone bridge lead to changes in its output voltage.



(a) Pressure Sensor (b) Configuration

Figure 3.2: Pressure Sensor: (a) Pressure Sensor; (b) Configuration. [17]

This pressure sensor has a pressure range of 0 to 15 PSI, a sensitivity of 80 $\mu V/cmH_2O$, an offset of ±30 mV, and a power consumption of 0.83 mA when operating at 3 V. To fulfill the requirement of pressure detection range of 0 to 150 cm H_2O, a sufficient gain stage of analog-to-digital converters (ADCs) in a microcontroller for detecting an analog output range of 0 to 12 mV is required. In order to have good pressure sensitivity, the ADC of a microcontroller has to have an adequate resolution. Then, the selection of a microcontroller for this system is basically set by these constraints. In the next section, specifications of the selected microcontroller are presented.

Microcontroller

The C8051F005 microcontroller from Silicon Laboratories is chosen because it meets the design requirements. This microcontroller, as seen in Figure 3.3, has an internal clock, which eliminates the need for an external oscillator. It has a configurable operating frequency from 2 to 16 MHz. It also provides a 12-bit ADC with a variable gain preamplifier stage of 0.5 to 16 times gain with a 2.43 V internal voltage reference. This

microcontroller has 4 differential ADC input pairs, which can support up to 4 sensors. It has a programmable memory of 2304 bytes and a 32 kbytes FLASH memory for data storage and manipulation. Moreover, it offers JTAG In-System-Debugging and SPI features, which can be easily used to program the device and to communicate to other peripherals. It has a power consumption of 12.5 mA operating at 25 MHz and a device dimension of 12.0 × 12.0 × 1.2 mm^3. This microcontroller is configured to operate at 2 MHz and the current consumption with this setup is about 3 mA including ADC circuitry consumption.

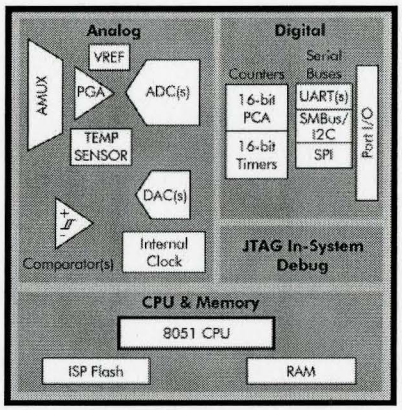


Figure 3.3: Mixed-signal C8051F005 Microcontroller (Silicon Laboratories) [16].

With a 16X gain preamplifier stage, the analog pressure sensor output is amplified from 0~12 mV to 0~192 mV. Using the internal 2.43 V voltage reference and 12-bit ADC resolution with bipolar differential input channels, the ADC's electrical resolution can be calculated as, (2.43 V)/(2^{11}-1 LSB) = 1.18 mV/LSB. Therefore, optimal pressure sensitivity for this system is calculated to be 0.99 cmH_2O/LSB.

Wireless Transceiver

The Laipac Tech TRF2.4G wireless transceiver was used in this system for its compatibility with the capsule imaging system described in [32]. Its transmitting frequency range is in the 2.4-2.524GHz ISM Band, using Gaussian Frequency-Shift-

Keying (GFSK) modulation. This transceiver can support 250 kbps and up to 1Mbps data transmission rate. It has a sensitivity of $-90dBm$ and a transmission current consumption of 10.5mA peak at $-5dBm$ output power. It has a dimension of 20.5 mm × 36.5 mm × 2.4 mm with a built-in antenna, as seen in Figure 3.4.

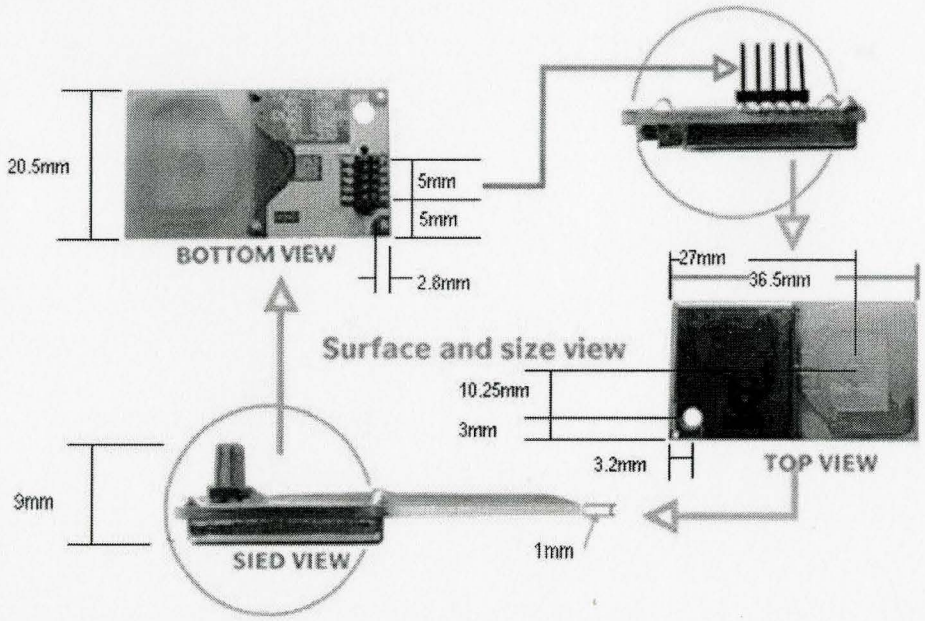


Figure 3.4: Various Views of Laipac Tech TRF2.4G Wireless Transceiver [18].

3.3 Electrical Design – Circuit Schematics

After the components were selected, the design of the system was carried out in Protel 2004. The system schematic was made by creating interconnections between components. As seen in Figure 3.5, the design of this system consists of a microcontroller (C8051F005), a pressure sensor (HEADER_PRESSURE), a wireless transceiver (5PIN_DUAL_ROW_1.25mmPITCH), a battery (HEADER_PWR), and two voltage regulators (TPS79030). There are 12 external components, e.g., capacitors, resistors, and light-emitting-diode. And the 6-pin header (H5) provides SPI communication to the capsule imaging device [32].

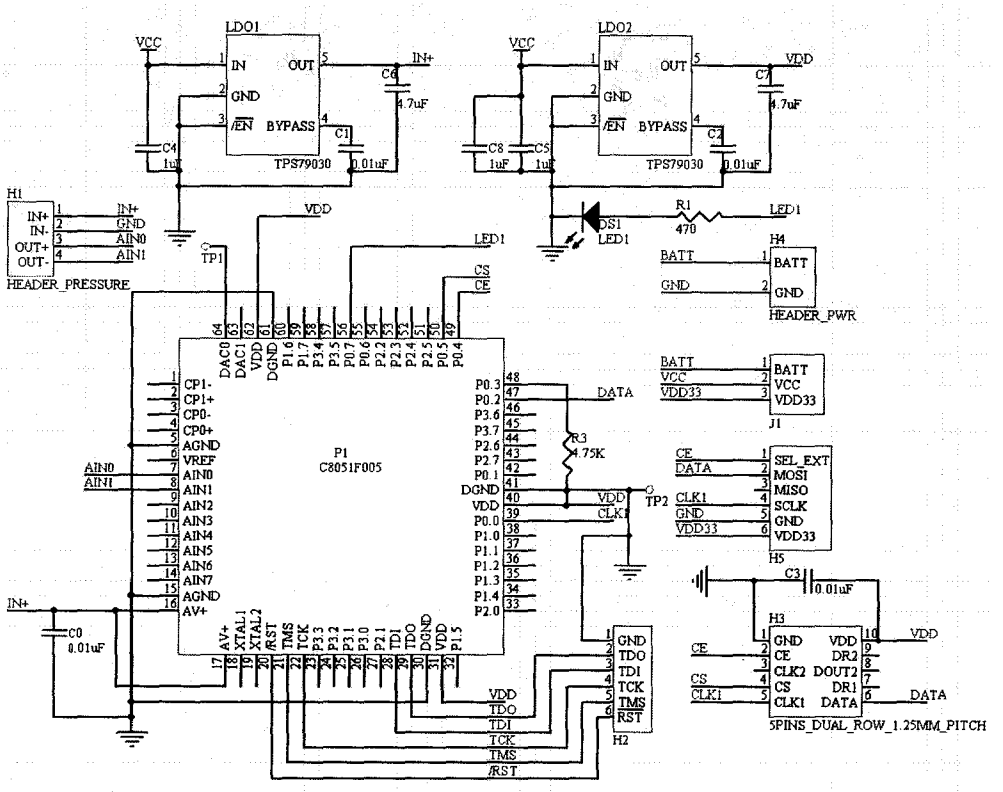
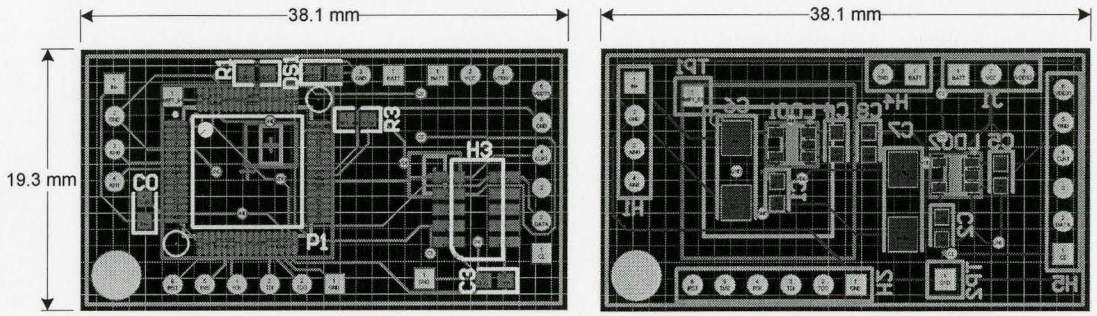


Figure 3.5: Schematic of 2.4 GHz Pressure Sensor System

3.4 Implementation – Hardware, Firmware and Software

3.4.1 Hardware – PCB Layout

After the system schematic was entered in Protel 2004, the PCB footprint of each component can be imported to Protel 2004's PCB project. Then, the locations of components are planned out before the interconnections between components are made carefully. In Figure 3.6 (a), top side of the PCB layout is shown. Microcontroller and headers are placed on the top side of the PCB. The bottom side of the PCB are mainly used for the voltage regulators and their external components, as seen in Figure 3.6 (b).



(a) Top view (b) Bottom view

Figure 3.6: Protel Layout of Prototype PCB; (a) Top view; (b) Bottom view.

3.4.2 Firmware

Firmware of the prototype is written in assembly language. The programming environment used for this prototype is the Silicon Laboratories Integrated Development Environment. Keli Software A51 assembler was used to compile and upload the program to the C8051F005 microcontroller. The flowchart of the firmware is shown in Figure 3.7. The firmware of this system can be found in Appendix A.1.

At the start of this firmware, input/output (I/O) ports, timers, SPI and ADC are initialized. The ADC is initialized to operate at 250 kHz and each conversion takes about 64 μs. In the main loop of the firmware, the pressure sensor output is sampled by the ADC to 12-bits value and stored in the microcontroller. A long averaging routine in the firmware prepares the average of 4000 ADC sampled pressure data and stores it for transmission. Therefore, each long-averaged result is an average of pressure data of 256 ms. Next, an exponential averaging of factor 16 is applied on this averaged data to create a time delay for about 20 seconds. An example of a transition of long averaged values from 0 to 1 V and the exponential averaging results is shown in Figure 3.8. It is clear in Figure 3.8 that by taking the difference between long averaging results and exponential averaging results, an episode of rapid change in pressure sensor output can be detected. This difference is calculated and stored after both averaging routines are finished.

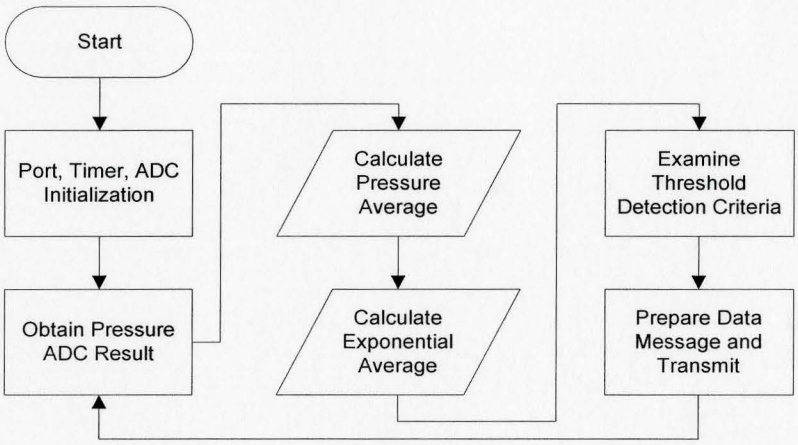


Figure 3.7: Flowchart of Prototype Firmware

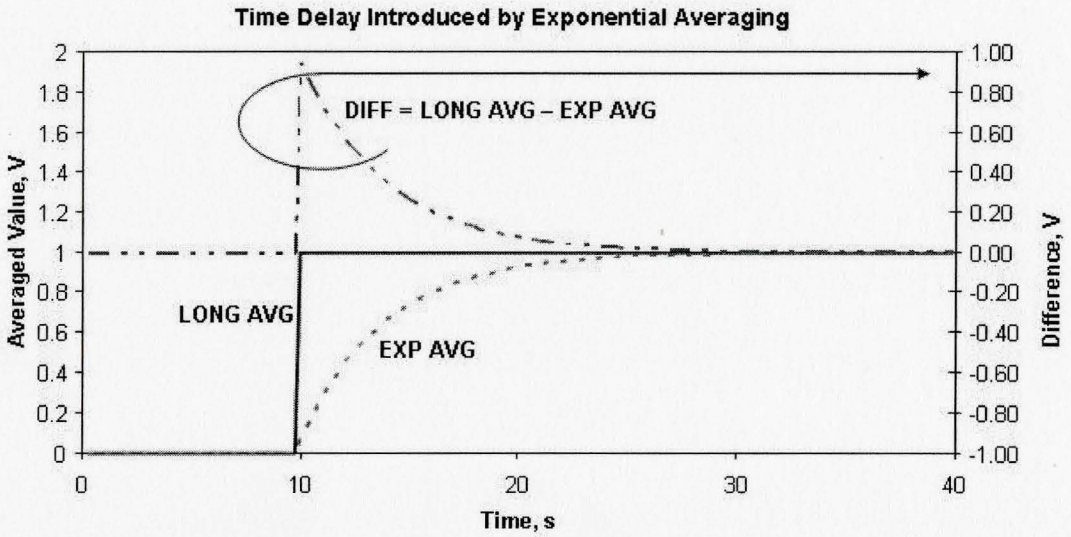


Figure 3.8: Time Delay Introduced by Exponential Averaging

And finally, a message with the following format is prepared and transmitted via SPI to the capsule imaging system [32].

Message Format of 2.4 GHz System:

8bit Counter		Long AVG, 2 Bytes		EXP AVG, 3 Bytes				AUTO REF VALUE, 2 Bytes		DIFF, 2 Bytes	
00h	XXh	XXh	XXh	00h	XXh	XXh	XXh	XXh	XXh	XXh	XXh

3.4.3 Software

The transmitted messages are received by receiver module used in [32] and stored on a PC. Then, MATLAB scripts are used to extract and save these data as Microsoft Excel format. Finally, the data is processed and analyzed. MATLAB scripts used in this work are provided in Appendix A.2.

3.5 Technical Specifications from Design

After the components have been selected and the electrical design completed, according to the manual specifications of each component, the 2.4 GHz pressure sensor system should have parameters shown in Table 3.1.

Table 3.1: Technical Specification for 2.4 GHz Pressure Sensor System

	2.4 GHz Pressure Sensor System	Units
Pressure Sensitivity	0.99	cmH_2O/LSB
Current Consumption	4	mA

CHAPTER 4: IMPLEMENTATION OF 125 kHz RFID DUAL SENSOR SYSTEM

In the previous chapter, it is concluded that the overall system size is large due to the size of the wireless transmitter and the power consumption of the transmitter dominates that of the entire system. In the design of this second system, the radio frequency identification (RFID) technology is used as the transmission and powering mechanism and a temperature sensor is added to implement a multi-sensor system and possibly to enhance the detection of urinary incontinence.

In [26], it is observed that when there is urine leakage, the amount of urine leaked can lead to decreased pelvic temperature. Therefore, by comparing bladder temperature and pelvic temperature, a difference in temperature could be used to detect episodes of urine leakage. In addition, it is also shown in [27] and [28] that urinary bladder can be used as a core site for body temperature measurement in the operating room or in the intensive care unit. Therefore, the choice of adding a temperature sensor to this system is beneficial.

With the main transmission method changed, the new system block diagram can be seen in Figure 4.1. Again, only off-the-shelf components are used to implement this system. First, this system consists of two main components: a transponder system and a reader system. The transponder system consists of two sensors, a microcontroller, a RFID transponder interface chip and an antenna. The reader system consists of a RFID base station chip, a microcontroller and an antenna. The firmware of the transponder system is made to sample both sensors' outputs with the controller's internal 10-bit ADCs, to process the results and convert them into a message, and to transmit the message by controlling the transponder chip. The base station's firmware is to receive the message, check for its validity, and send it to a PC via RS232 communication protocols. Lastly, a graphical user interface (GUI) was made using Microsoft Visual Basic Studio 2005 to display sensor results on screen of a PC in real time.

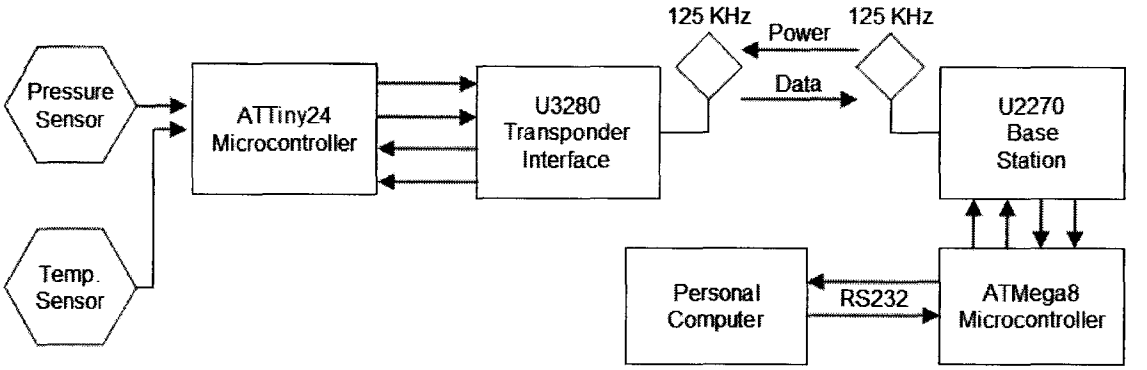


Figure 4.1: Block Diagram of 125 kHz RFID Dual Sensor System

This chapter presents a detailed description of the design of the 125 kHz RFID dual sensor system. Frequency band description, possible variants, electrical design, and implementation of the system are presented. At the end of this chapter, technical specifications of the system extracted from our design are presented.

4.1 125 kHz RFID Technology

The RFID technology is one in which radio waves are used to automatically identify people or objects. An RFID system typically consists of a transponder which is made up of a microchip with an antenna, and a reader with an antenna. The reader sends out electromagnetic waves and the transponder antenna is tuned to receive these waves at a selected frequency. A passive RFID transponder draws power from the field created by the reader and uses it to power the microchip's circuits. The microchip then modulates the waves that the transponder sends back to the reader, which converts the new waves into digital data.

RFID systems use many different frequencies, but generally, the most common frequencies are low-frequency (125 kHz), high-frequency (13.56 MHz) and ultra-high-frequency or UHF (860-960 MHz) [19]. Microwaves (2.45 GHz) are also used in some applications. Different frequencies have different characteristics that make them more suitable for different applications. Low-frequency transponders use less power and are

better at penetrating non-metallic substances. They are also ideal for scanning objects with high-water content, but their range is limited to less than 0.33 metre. High-frequency transponders work better on objects made of metal and can work around goods with high-water content. They have a maximum reading range of about 1 metre. UHF frequencies typically offer better range and can transfer data faster than low- and high-frequencies. But they use more power and are less likely to pass through materials [19].

For our purpose, the low-frequency version of RFID is used since human body is mostly made of water. This was chosen because the goal of the project is to seek low power consumption whenever possible. Moreover, the required transmission data rate of our system is low as compared to other wireless systems.

In addition, the maximum exposure limits of human body for low-frequency RFID are defined by Health Canada as the following. The rms (root-mean-square) value of the electric field strength should not exceed 600 V/m, and the rms value of magnetic field strength should not exceed 4.9 A/m. Also, the SAR (Specific Absorption Ratio) limit over the whole body mass should not exceed 0.4 W/kg, and the SAR for trunk, averaged over any one gram of tissue should not exceed 8 W/kg [46].

4.2 Possible Variants – Components Selection

In this section, detailed descriptions of selected components are presented to provide a sound understanding of why these specific components are chosen.

Pressure Sensor

The same pressure sensor from the previous prototype is used. See chapter 3, section 3.3.1 for details. But in this system, to minimize the power consumption of this pressure sensor, a 10 kΩ resistor is connected in series with the pressure sensor. By doing this, the excitation voltage across the pressure sensor is decreased to 0.716 V. The sensitivity of the pressure sensor can be calculated as about 20 $\mu V/cmH_2O$ and the

voltage output is 0 to 3 mV. And the current consumption of the sensor is reduced to 0.2 mA.

Transponder Microcontroller

A low power AVR 8-bit Microcontroller, ATtiny24, is used as the heart of the transponder system. The block diagram of this microcontroller is shown in Figure 4.2. The microcontroller is configured to operate with the internal oscillator at 1 MHz at 2.7 V. It has a 10-bit ADC with programmable gain for 4 differential ADC channels with an internal voltage reference of 1.1 V. In-System-Programmable via SPI, low power idle, programmable brown-out detection circuit features are also suitable to achieve the goals of this project. Moreover, it has a dimension of only 4.00 × 4.00 × 0.75 mm^3. Its current consumption is 380 μA operating in active mode at 1 MHz at 1.8 V, and 100 nA operating in power-down mode, makes it a very favourable candidate to this project.

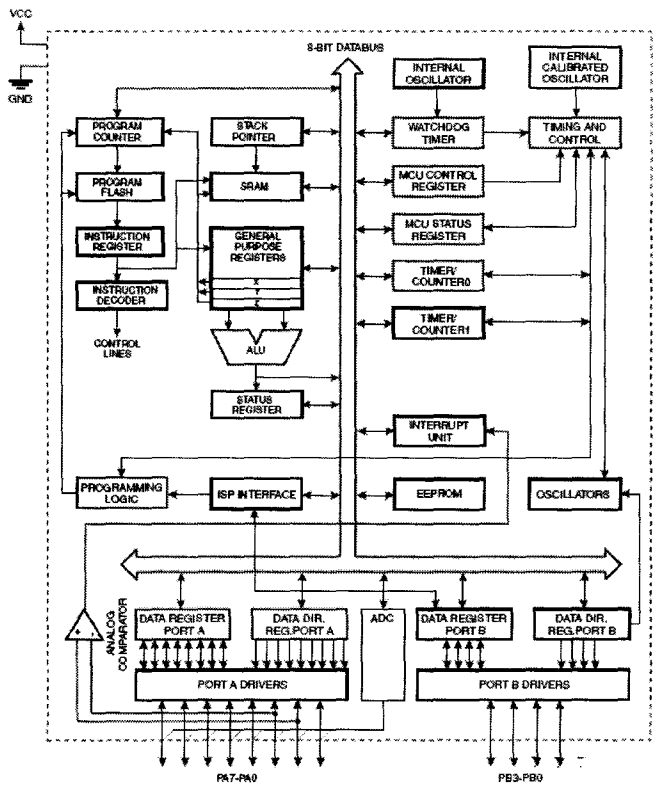


Figure 4.2: Block diagram of Atmel ATtiny24 Microcontroller [20].

With a 20X preamplifier stage gain, the pressure output is amplified from 0~3 mV to 0~60 mV. Using the 1.1 V internal voltage reference and 10-bit ADC, the electrical resolution of the ADC is calculated to be 1.08 mV/LSB. Therefore, for detecting 0 to 150 cmH_2O pressure range using the same pressure sensor from previous prototype, the optimal pressure sensitivity of 2.93 cmH_2O/LSB can be detected.

Temperature Sensor

A multi-gain analog temperature sensor, LM94022 is chosen. This temperature sensor has a maximum sensor gain of -13.6 mV/°C and a low power consumption of 5.4 μA. The transfer characteristic of this temperature sensor is shown in Figure 4.3.

By configuring the ADC to operate in differential mode with 1X gain stage and supplying the positive channel of the ADC with a fixed 2.7 V from a voltage regulator, as illustrated in Figure 4.4, a temperature detection range is made approximately from -11 to 70 °C. With 1.1 V internal voltage reference and a 10-bit ADC resolution from the microcontroller and a temperature transfer gain of -13.6 mV/°C, a temperature sensitivity of 0.086 °C/LSB can be achieved.

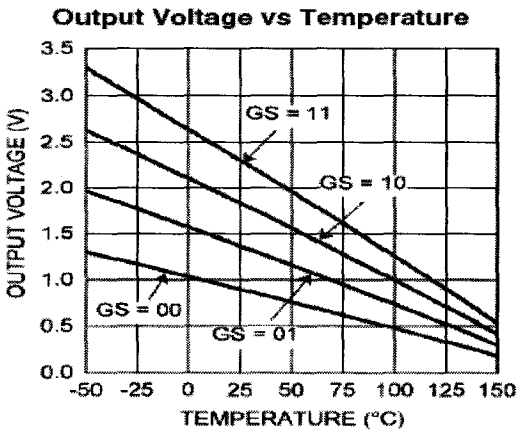


Figure 4.3: Transfer Characteristic of LM94022 [21].

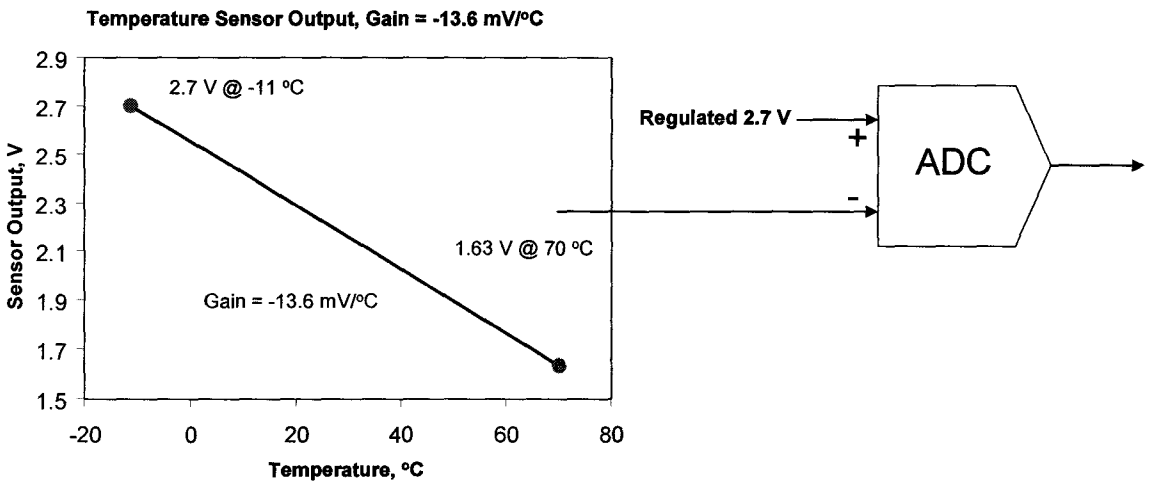


Figure 4.4: Illustration of Temperature Sensor Setup

RFID Transponder Chip

The transponder interface for microcontroller, U3280M, from Atmel is used in the transponder system as a communication interface. It can supply the microcontroller with power from an RF field via a LC-resonant circuit, which means the microcontroller can be powered wirelessly, without a battery. It also has a power management feature that controls the switching between field supply and batter power supply. The block diagram of this chip is shown in Figure 4.5. As seen in Figure 4.5, the damping stage of this transponder chip can be controlled by the serial interface or a modulating signal, MOD. The ability of controlling the damping stage using the serial interface was first tested. The result was not satisfactory, because no field damping was observed. Therefore, to control the damping stage, an external modulating signal was required.

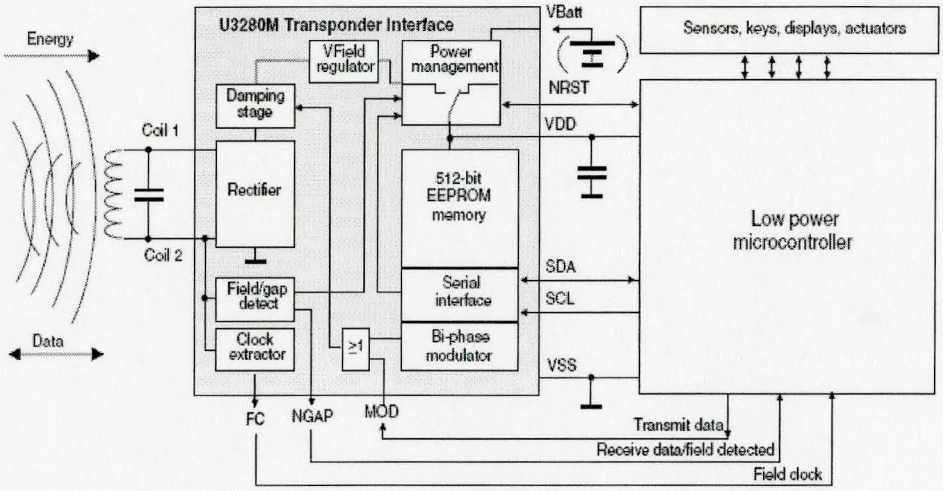


Figure 4.5: Block Diagram of U3280M Transponder Interface [22]

Moreover, the typical field supply voltage of this chip is 2.9 V, which also supports the selection of the ATtiny24 microcontroller. To fully utilize the field supply feature, an external buffer capacitor is required. According to the manual of this chip, the value of this capacitor was calculated to be 3.3 μF to support a 200 mV voltage ripple at the coil, a no field supply length of 500 μS and a 1 mA supply current. Assuming rate of data transfer is 2 kHz and the duration of each half period is 500 μS, the voltage on the coil is decreased when the MOD pin is pulled to high-state for half a period in the data transfer. In addition, the field detected signal, NGAP, can be used in the microcontroller to detect if a field is nearby. Lastly, an 11.0 × 13.5 cm^2 square-loop antenna with an inductance of 2.38 mH and a capacitor of 681 pF were used for the antenna circuit in the measurement setup.

Reader Chip

The read/write base station, U2270B, is chosen as the communication interface for the RFID reader system. It has an adjustable frequency range of 100 kHz to 150 kHz. It has an internal oscillator to provide required frequency and frequency adjustment can be made through an external resistor. It can also be controller by a microcontroller via a few

signals, namely mode-select (MS), carrier-frequency-enable (CFE), or standby. The block diagram of this chip is shown in Figure 4.6.

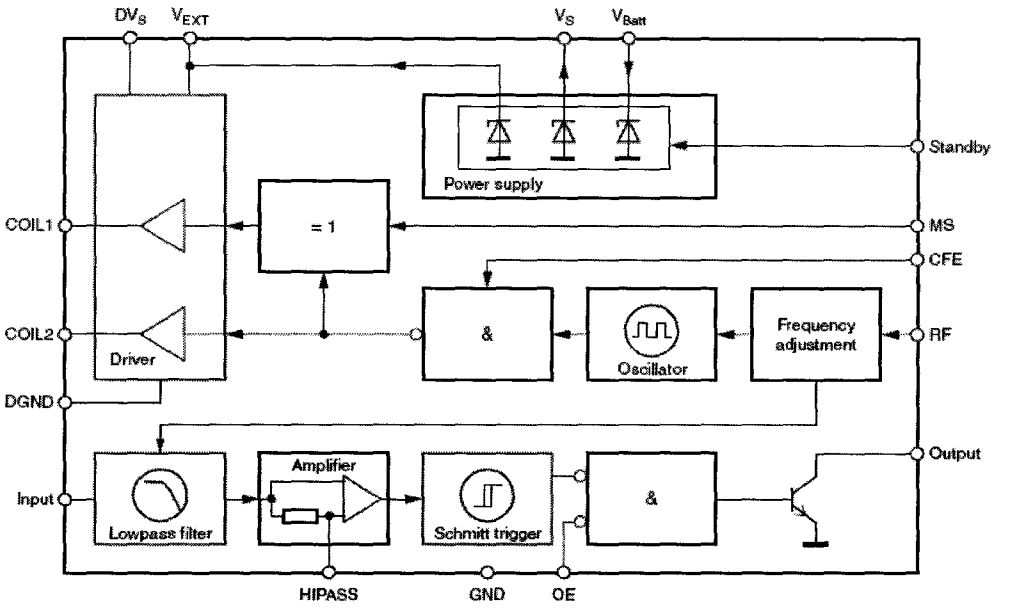


Figure 4.6: Block Diagram of U2270B Read/Write Base Station [23]

Moreover, an external amplitude demodulation (AMD) circuit connected between the antenna circuit and the input pin of the chip, shown in Figure 4.7, is required to demodulate the data signal from the carrier signal. The U2270B also has a series of internal circuitry to filter and remaining carrier signal and high-frequency disturbances after demodulation, such as a low-pass filter, a high-pass filter and an amplifier of a gain of 30. In addition, the U2270B chip also has an internal Schmitt trigger circuit to make the received data signal microcontroller-compatible. Lastly, a 14.0 × 16.0 cm^2 square-loop antenna with an inductance of 2.38 mH and a 681 pF capacitor were used to form the reader's antenna circuit in the measurement setup.

Reader Microcontroller

A generic microcontroller with regular operating speed of 8 to 16 MHz is needed to communicate with the U2270B read/write base station chip, to decode the received

data signal, and to generate readable messages to a personal computer. Therefore, a microcontroller with UART interface is preferred to communicate with a PC via RS232 communication standard. A generic microcontroller, Atmel ATmega8, was chosen simply because it offers the UART interface and it was compatible with the ATtiny24 development kit [24].

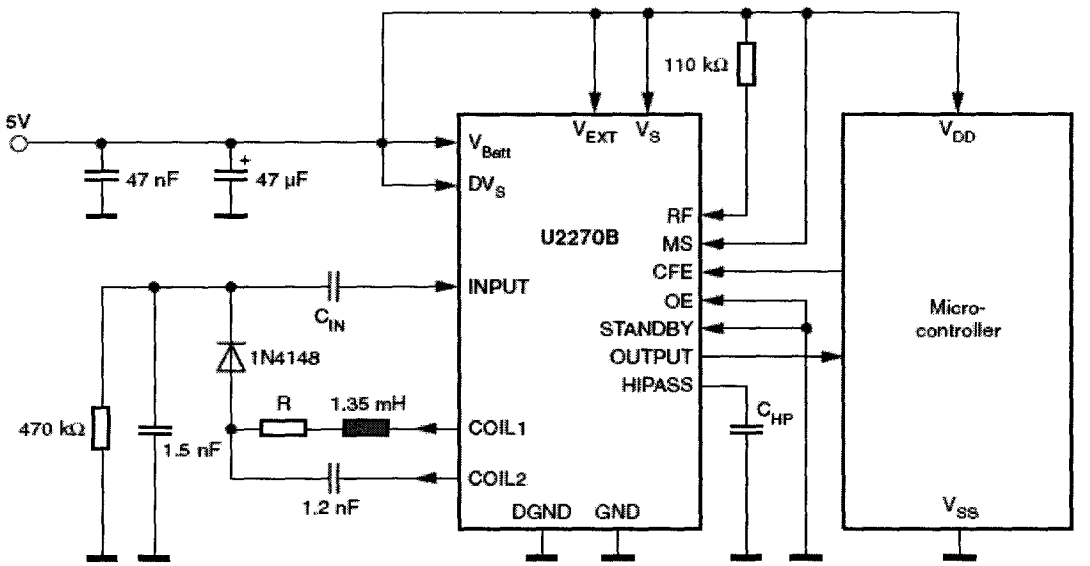


Figure 4.7: Diagram of Amplitude Demodulation Circuitry [23].

4.3 Electrical Design

Transponder Schematic

In Figure 4.8, final schematic of the transponder system is shown. The schematic is created using Protel 2004. The design of this system consists of a low-power microcontroller, ATtiny24, a transponder interface chip, U3280M, a voltage regulator, a battery, a temperature sensor, LM94022, an antenna and some external resistors and capacitors. The battery was added to the system mainly to compensate the amount of current consumption that the pressure sensor is consuming. The voltage regulator provides the pressure sensor switching on-and-off feature to the system. One of the gain

select signal, GS0, of the temperature sensor is pulled high. Hence, only two different gains, -10.9 mV/°C and -13.6 mV/°C, of the temperature sensor can be selected.

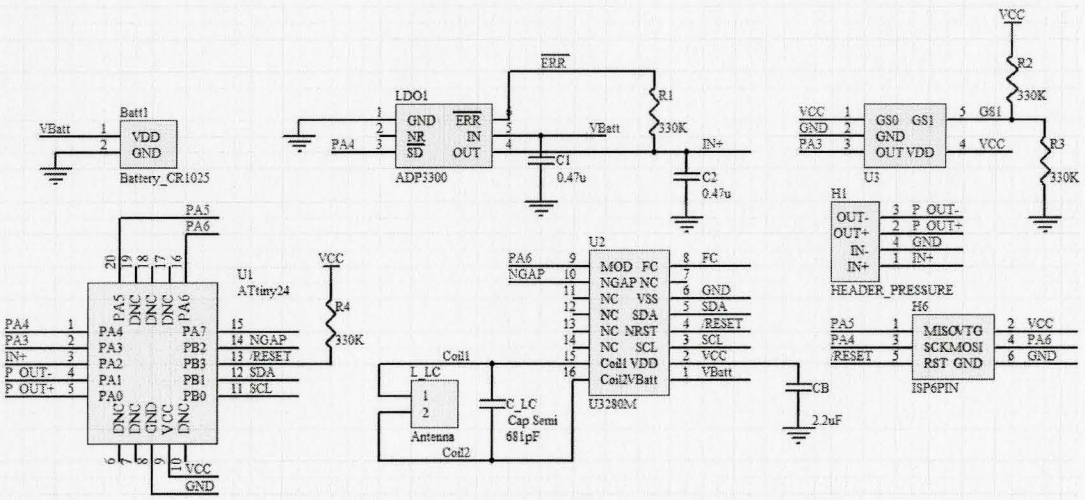


Figure 4.8: Schematic of Transponder System

Reader Schematic

The reader's schematics are shown in Figures 4.9 through 4.12. The main communication interface, U2270B Read/Write Base Station Chip, and its external components and shown in Figure 4.9. An adjustable resistor, Rf1, is placed to tune the frequency of the reader system. Other components - a resistor (R202), a capacitor (C202) and a diode (D201) - form a simple passive amplitude demodulation circuit to demodulate the data signal from the carrier frequency.

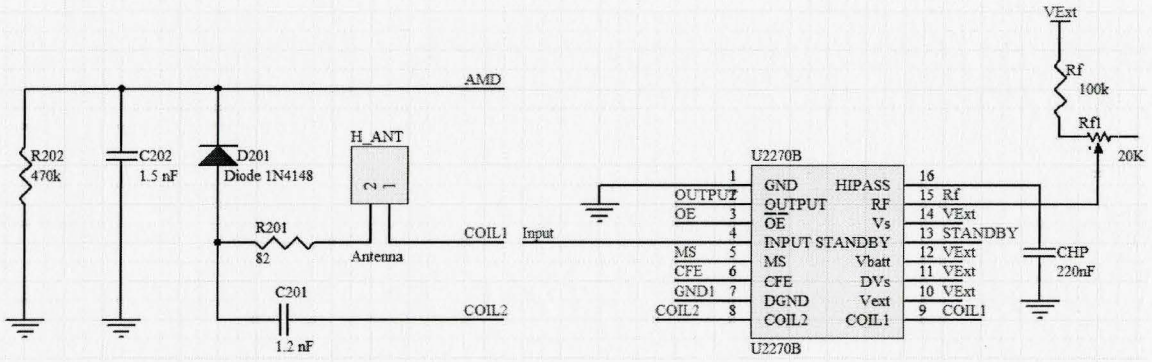


Figure 4.9: Schematic of Reader System 1 – U2270B and AMD Circuit

The microcontroller and the RS-232 line driver, MAX203E, are shown in Figure 4.10. The MAX203E chip is a generic RS-232 communication line driver acting as charge pump voltage converters [25]. It converts conventional logic levels, High-1, Low-0, to standard voltage levels for RS-232 communication. In which, -5 V stands for a high logic level and +5 V stands for a low logic level. It also offers ±15 V electrostatic discharge protection which protects the reader system from outside environment.

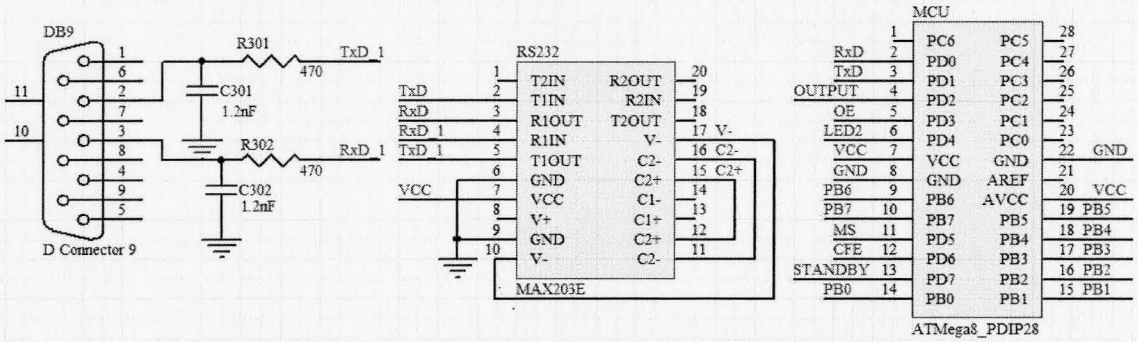


Figure 4.10: Schematic of Reader System 2 – ATmega8 and Max203E

Lastly, an external preamplifier stage, shown in Figure 4.11 was added to the system to extend the communication range. This preamplifier stage exists between the amplitude demodulation circuit and the input pin of the U2270B base station chip. In Figure 4.12, headers for power supplies, programming and LEDs are shown.

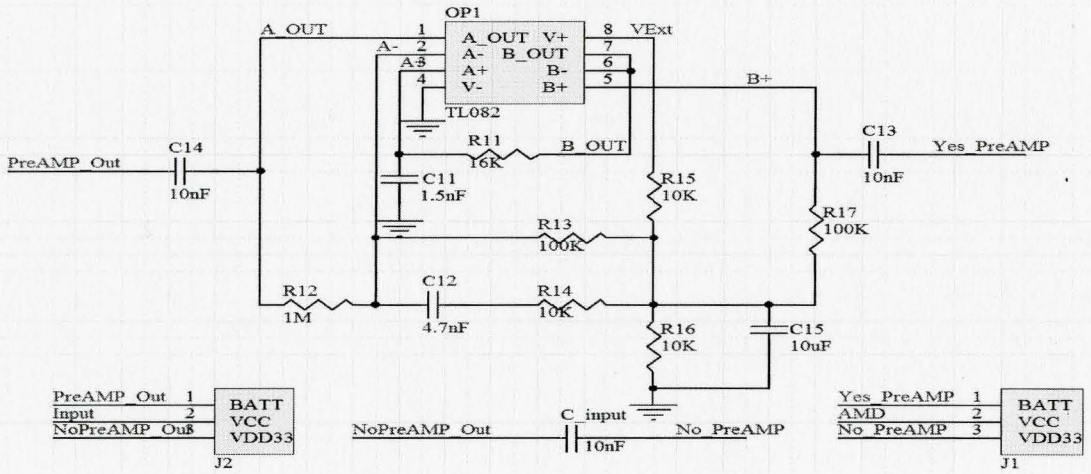


Figure 4.11: Schematic of Reader System 3 – Selectable Preamplifier Stage

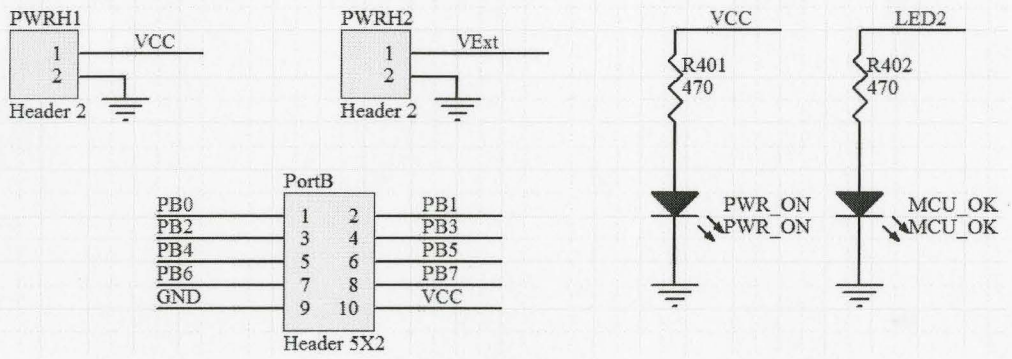


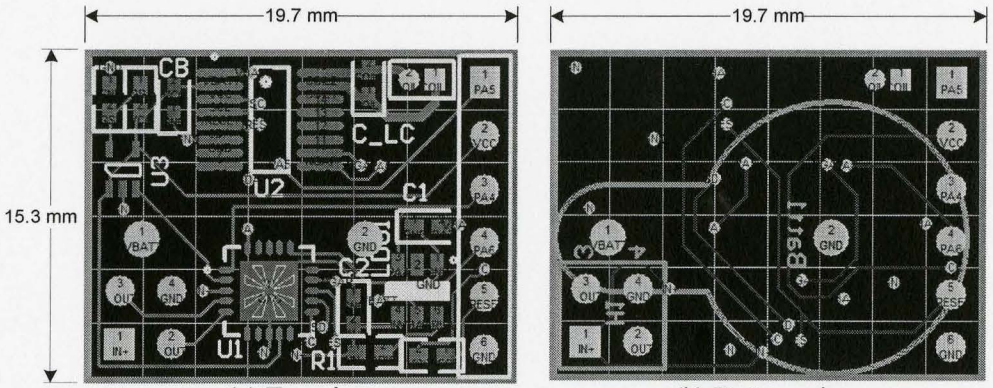
Figure 4.12: Schematic of Reader System 4 – Headers and LEDs

4.4 Implementation – Hardware, Firmware and Software

4.4.1 Hardware – PCB Layouts

Transponder PCB Layout

Figure 4.13 shows the PCB layouts of the transponder system. The microcontroller, ATtiny23, U3280M transponder interface, temperature sensor, pressure sensor header, voltage regulator, and other external components are all placed on the top side of the PCB and the bottom side is mainly used for a battery holder.



(a) Top view (b) Bottom view

Figure 4.13: PCB Layout of Transponder System; (a) Top view; (b) Bottom view

Reader PCB Layout

The PCB layout of the reader system is shown in Figure 4.14. The digital part of the system, such as microcontroller, RS-232 line driver and programming header, PortB, is placed on the left side of the board. The analog part of the system is placed on the right side of the board. By doing so, it is hoped to minimize the interference of the communication interface by the digital circuits. Two power supply headers are placed on the board. One is to supply the digital part of the system and the other is to supply the analog part.

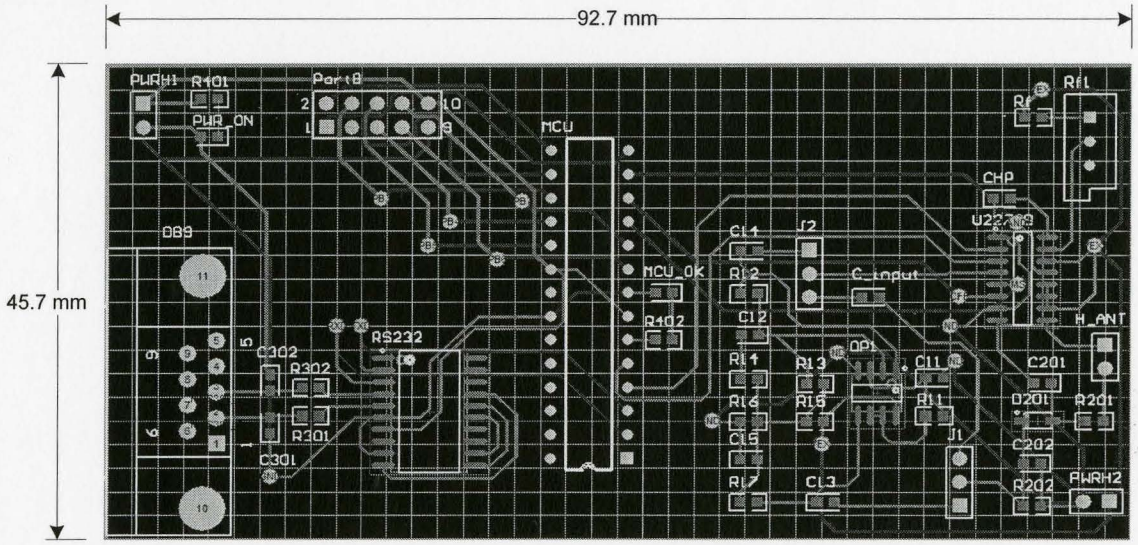


Figure 4.14: PCB Layout of Reader System

4.4.2 Firmware

Transponder Firmware

The firmware flowchart of the transponder system is shown in Figure 4.15. The system is currently running continuously and repetitively. This means that pressure and temperature data are being sampled, processed and transmitted at all times. In the transponder firmware, a simple moving average routine is used to minimize the offset effect of the pressure sensor. Both differential (2.7 V versus temperature sensor output)

ADC temperature result and single-ended (temperature sensor output versus voltage supply) ADC temperature result are being sampled to examine correctness of the functionality of temperature sensor. The format of the message to be transmitted is shown below.

Message Format of 125 kHz System:

8bit Preamble	8bit ID	8bit Counter	6LSB of TvsVCC + 2MSB Pressure High	8bit Pressure Low	4MSB of TvsVCC + 2MSB Temperature High	8bit Temperature Low	8bit Checksum
00000000	XXXXXXXX	XXXXXXXX	xxxxxxXX	XXXXXX	xxxx00XX	XXXXXX	XXXXXXXX

Each transmitted message contains 8 bytes of data. The first byte is the preamble byte. The second byte contains the identification (ID) of this transponder. The third byte contains value of a free counter, which is used to keep track of order of the transmitted data. The fourth to seventh bytes contain sampled pressure and temperature data. The last byte is the checksum which is the sum of third to seventh bytes. Lastly, these prepared messages are being transmitted at a data rate of 640 bps at 125 kHz. The firmware of the transponder is given in Appendix B.1.

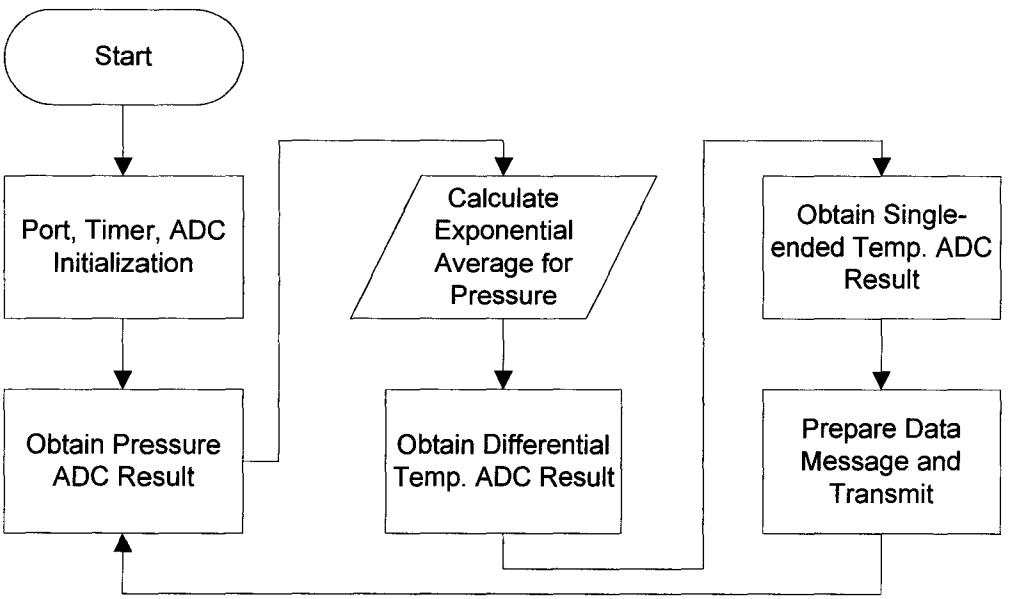


Figure 4.15: Flowchart of Transponder Firmware

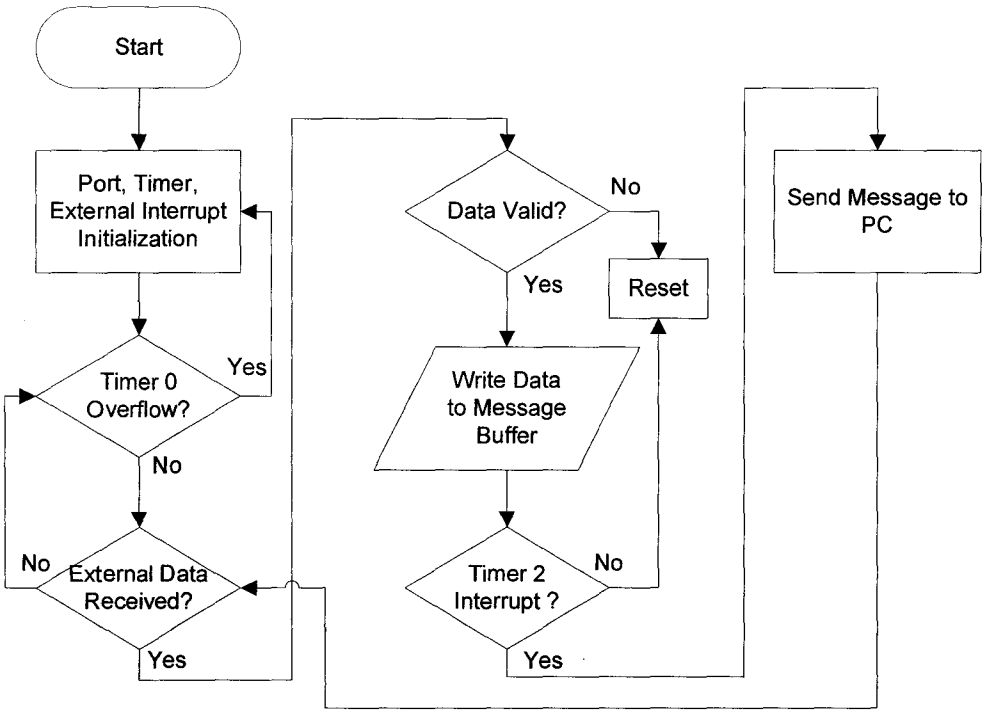


Figure 4.16: Flowchart of Reader Firmware

Reader Firmware

A flowchart of the reader's firmware is shown in Figure 4.16. The reader is set to generate 125 kHz electromagnetic wave at all times. Receiving data procedure is triggered by the presence of incoming data. After the presence of incoming data is detected by external interrupt, the message is then examined byte by byte.

The first byte of the message is the preamble byte which is used by the reader system to recover the clock rate of the incoming data. The second byte contains the identification (ID) of this transponder, which will be checked by the reader system for device matching purpose. The third byte contains value of a free counter, which is used to keep track of order of the incoming data. The fourth to seventh bytes contain sampled pressure and temperature data. The last byte is the checksum which is used to examine the validity of a received message. In other words, a message will only be received when it contains the

correct ID byte and a valid checksum. Then, after each message is successfully received, the firmware updates its outgoing message buffer and the new message will be sent to a PC when an interrupt is generated by a timer in the microcontroller. The reader's firmware can be found in Appendix B.2.

4.4.3 Software - Graphical User Interface

A screenshot of the graphical user interface, GUI, is shown in Figure 4.17. The main functions of this GUI are to decode the messages from the transponder, to display real-time data of sensors, and to save these data for further analysis. There is also a list of features which enable users to better interpret the real-time data.

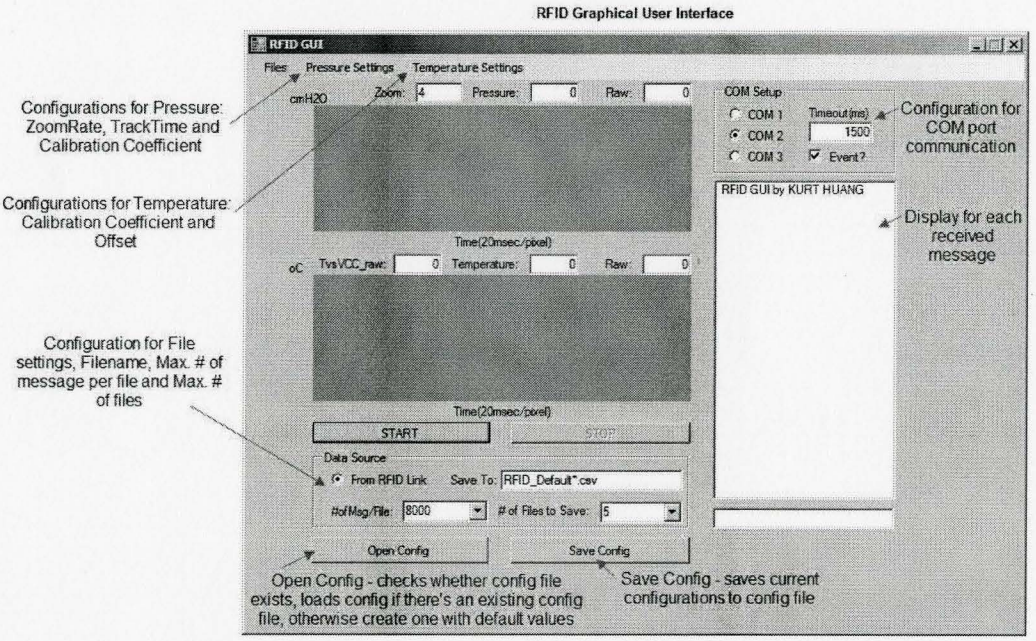


Figure 4.17: Screenshot of RFID Graphical User Interface

For the pressure part, *ZoomRate* in the *Auto-Zoom* feature is used to control the *zoom-in* and *zoom-out* function on the pressure display. *TrackTime* is used to control the zeroing time coefficient in the *Zeroing* feature. And *Pressure-Calibration-Coefficient* is used to calibrate the received pressure data against the actual height levels. For temperature display, *Temperature-Calibration-Coefficient* and *Offset* are used to calibrate

the received temperature data against the actual temperatures. And finally, the source codes of this GUI can be found in Appendix B.3.

4.5 Technical Specification from Design

From the manual specification of each component, the technical specification of this 125 kHz RFID dual sensor system is extracted and shown in Table 4.1

Table 4.1: Technical Specifications of 125 kHz RFID Dual Sensor System

	125 kHz RFID Dual Sensor System	Units
Pressure Sensitivity	2.93	cmH_2O/LSB
Temperature Sensitivity	0.086	°C/LSB
Current Consumption	1	mA

CHAPTER 5: EXPERIMENTAL RESULTS

5.1 Pressure Experiment

The purpose of this experiment is to determine the pressure sensitivity of each system and to verify its function in detecting rapid pressure change in each system. In this section, the pressure experiment setup is first presented. Next, the pressure data received at a PC is processed and presented.

Pressure Experiment Setup

The measurement setup is illustrated in Figure 5.1. Water tubes connect the pressure sensor on the system to a manometer and a water bottle. The pressure is set by moving the water bottle to different heights above the floor. The pressure data is then processed and arranged into a specific message by each system. Then, the messages are transmitted to a receiver by the on-board microcontroller periodically. Finally, the receiver passes on these messages to a PC, which then decodes the message, displays them on screen, and stores for further analysis.

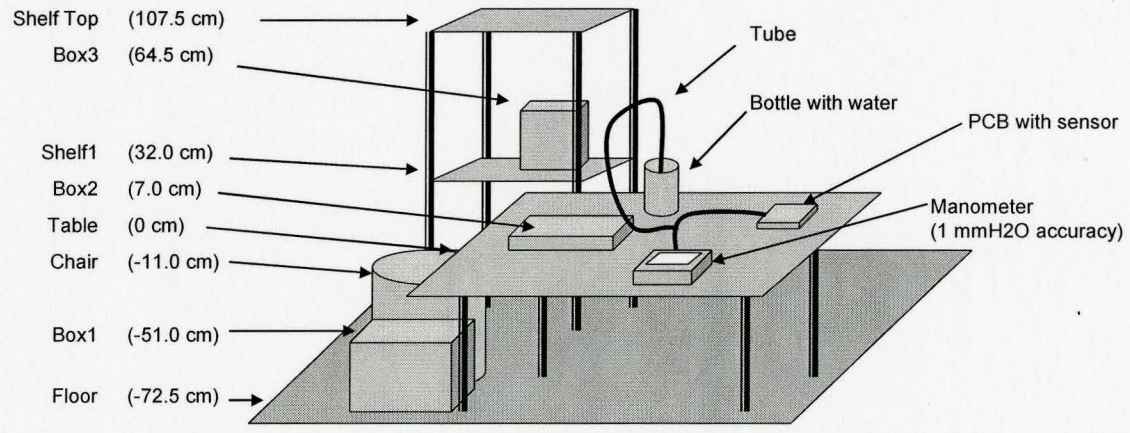


Figure 5.1: Illustration of Pressure Measurement Setup.

As for the manometer, it is a -5 to +5 PSI handheld pressure gauge, RK-68600-06, from Cole-Parmer Canada Inc. [34]. It has an accuracy of 1 mmH_2O and is used as the pressure reference in the measurement.

Pressure Measurement Results

First of all, Figure 5.2 shows the raw average pressure data, AVG, sampled by the 2.4 GHz wireless pressure system, received at a PC with respect to each reference pressure level. This verifies the sampling and transmission functions of the 2.4 GHz pressure sensor system. Then by doing a linear approximation on these data points in Microsoft Excel 2003, a linear function is obtained and is shown in the following equation.

$$Pressure = 1.14 \times Pressure\ Code\ (LSB) - 4.6,\ cmH_2O \qquad (5.1)$$

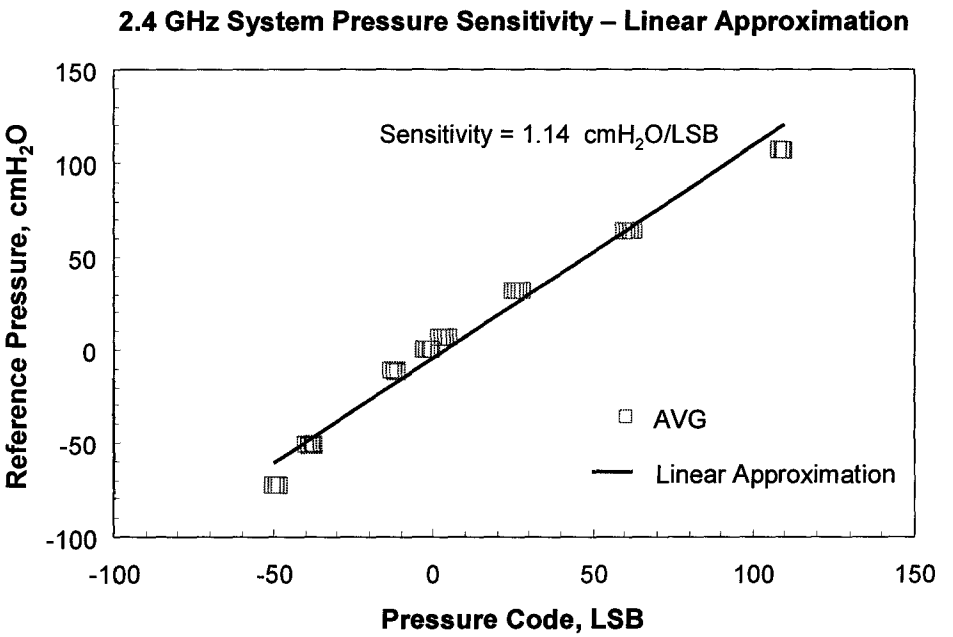


Figure 5.2: 2.4 GHz System – Pressure Sensitivity.

Therefore, the measured pressure sensitivity of this system is 1.14 cmH_2O/LSB and a pressure offset of 4.6 cmH_2O should be considered when converting the raw pressure codes to pressure (from LSB to cmH_2O).

Next, the same experiment is repeated with the 125 kHz RFID dual sensor system. The raw pressure data at each reference pressure level received at the PC is plotted in Figure 5.3. The linear approximation function of these data is shown below in equation 5.2.

$$Pressure = 2.92 \times Pressure\ Code\ (LSB) - 796,\ cmH_2O \qquad (5.2)$$

Hence, the measured pressure sensitivity of the 125 kHz RFID dual sensor system is 2.92 cmH_2O/LSB. And the pressure offset is -796 cmH_2O.

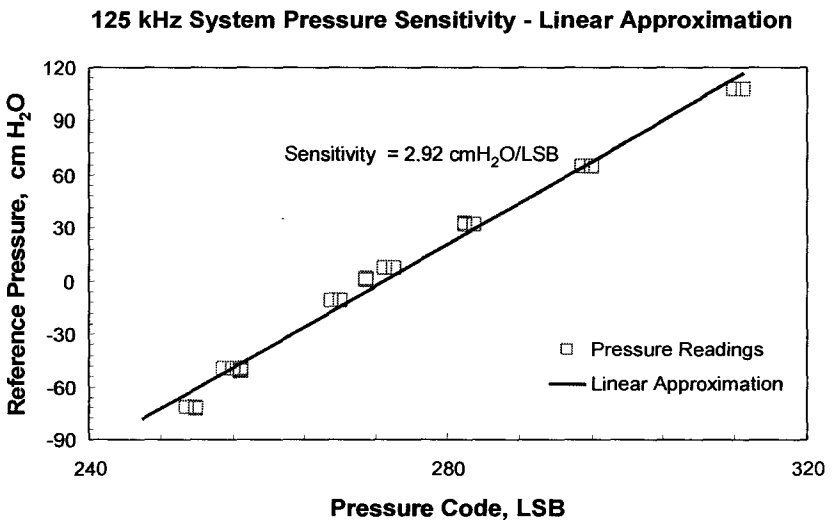


Figure 5.3: 125 kHz System – Pressure Sensitivity.

From Figure 5.2 and Figure 5.3, it can be observed that the pressure sensor has a concave/convex characteristic. This is due to the pressure sensor's nonlinear nature and the amplifier's non-linearity [35]. Fortunately, this non-linear characteristic of pressure sensor can be compensated using a quadratic model. Using quadratic approximation of the data points, the results are shown in Figure 5.4 and 5.5 for each system, respectively. The raw pressure code is converted to pressure using both linear and quadratic models in Figure 5.4 and Figure 5.5.

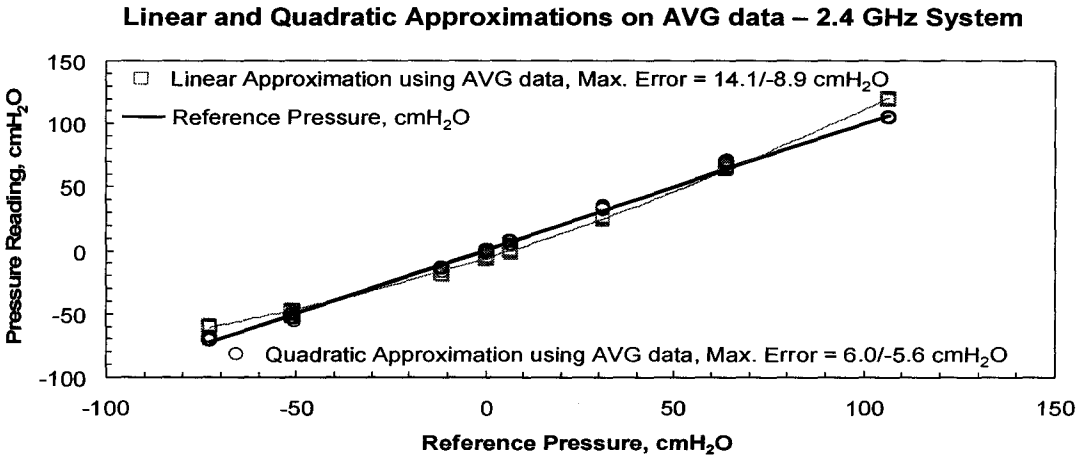


Figure 5.4: Quadratic Approximation of Pressure Data – 2.4 GHz System.

In Figure 5.4, it is shown that the quadratic model fits the data points better to the reference pressure. Using linear approximation of pressure data in the 2.4 GHz pressure sensor system, a maximum error of 14.1/-8.9 cmH_2O is observed. On the other hand, using the quadratic model, the maximum error is reduced to 6.0/-5.6 cmH_2O.

In Figure 5.5, both the linear and quadratic approximations of the pressure data from the 125 kHz RFID dual sensor system are shown. The maximum error introduced in the linear approximation is 11.5/-5.6 cmH_2O and it is 7.8/-4.5 cmH_2O in the quadratic model.

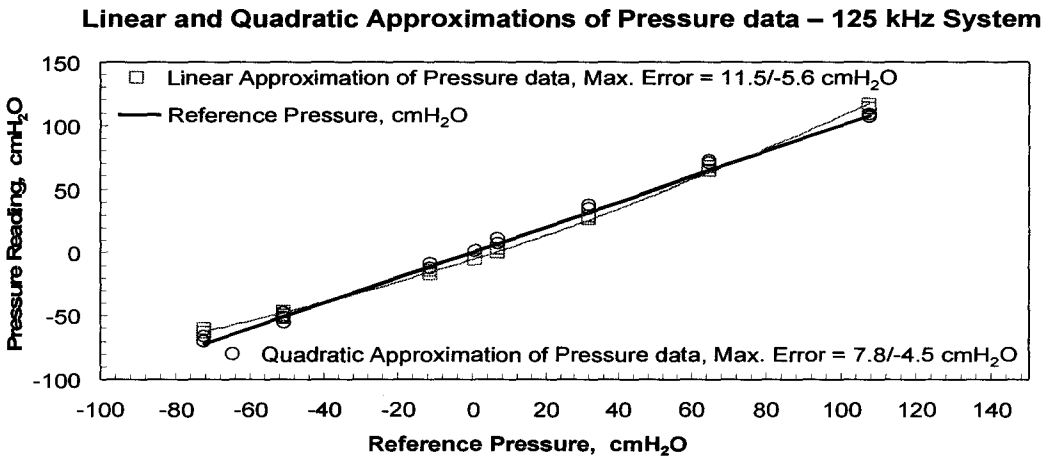


Figure 5.5: Quadratic Approximation of Pressure Data – 125 kHz System.

In Figure 5.6, it is shown that by using the difference between averaged and exponential averaged pressure data as a method to detect pressure changing episodes is working in the

2.4 GHz pressure sensor system. The spikes in the signal, Difference, clearly indicate episodes of pressure changes in Figure 5.6. This detection is made in the firmware of the system. On the other hand, for the 125 kHz RFID dual sensor system, the detection is made in the GUI. In Figure 5.7, the results of three different *tracktime* settings, 32, 256, and 10000 are presented. Also shown in Figure 5.7, by using different *tracktime* settings, different characteristics, namely, fast transients, medium transients, and real-time, of the pressure data can be targeted and displayed.

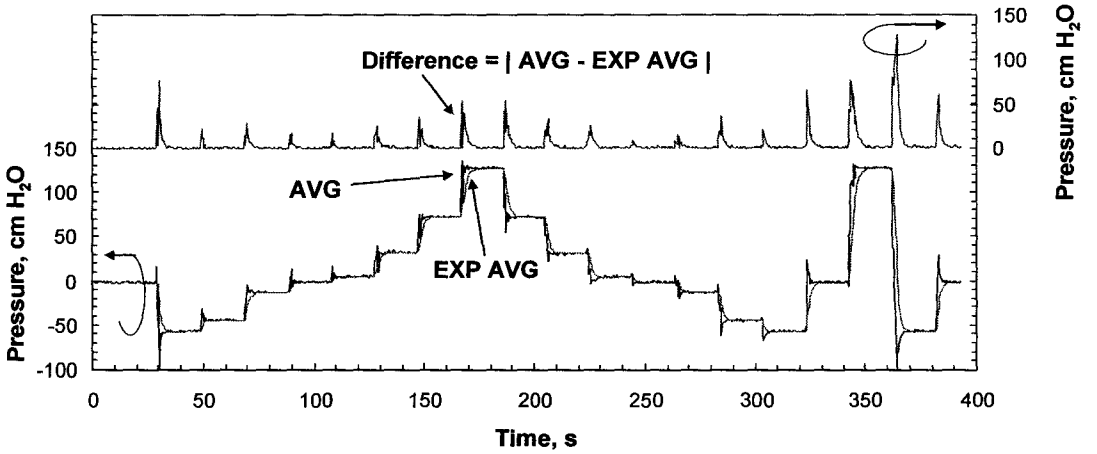
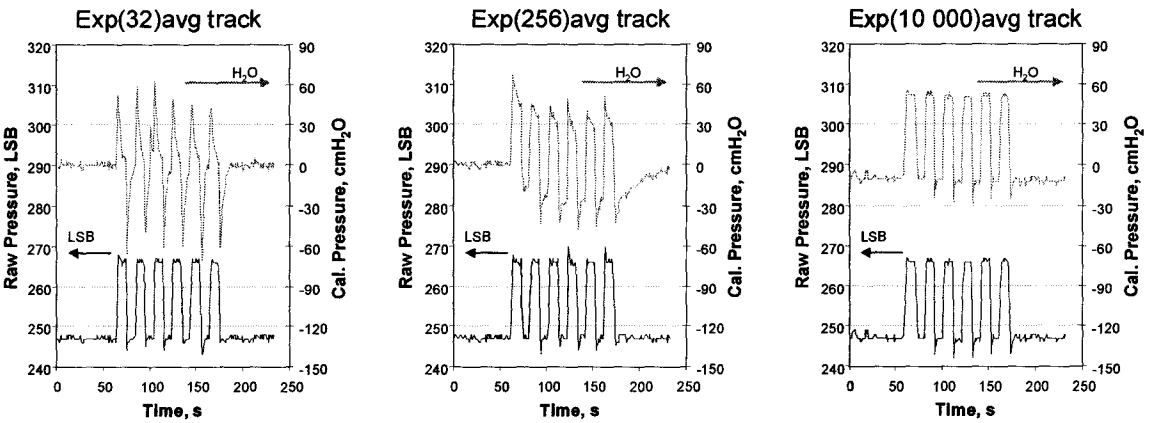


Figure 5.6: Detection of Rapid Pressure Change – 2.4 GHz System.



(a) *Tracktime* = 32 (b) *Tracktime* = 256 (c) *Tracktime* = 10000

Figure 5.7: Detection of Rapid Pressure Change – 125 kHz System.

5.2 Temperature Measurement

Temperature Measurement Setup

The temperature measurement setup is shown in Figure 5.8. The transponder is carefully placed inside an oven (Tenny Jr. from Tenny Engineering Inc.), with its antenna extended outside the oven. Temperatures from 15 to 50 °C with a 5 °C interval are set in the oven one after another while the reader records the data. Each temperature measurement lasts about 10-15 minutes, so a temperature stable environment is established and there is adequate data to analyze. Moreover, a digital thermometer, RK-91100-40, from Cole-Parmer Canada Inc., is used as the temperature reference [36]. This thermometer has an accuracy of 0.1 °C when operating above -150 °C and below 1000 °C.

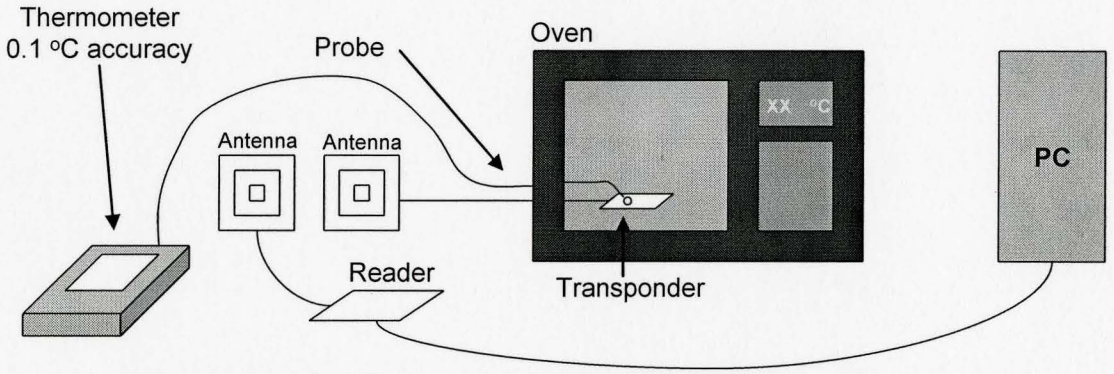


Figure 5.8: Illustration of Temperature Measurement Setup.

Temperature Measurement Results

The raw temperature data recorded in the PC is plotted against each reference temperature level recorded in Figure 5.9. By linearly approximating the temperature data against its reference temperature level, a linear function is obtained and shown in equation 5.3

$$Temperature = 0.069 \times Temperature\ Code\ (LSB) - 4.53,\ ^{\circ}C \qquad (5.3)$$

Therefore, the 125 kHz RFID dual sensor system has a temperature sensitivity of 0.069 °C/LSB.

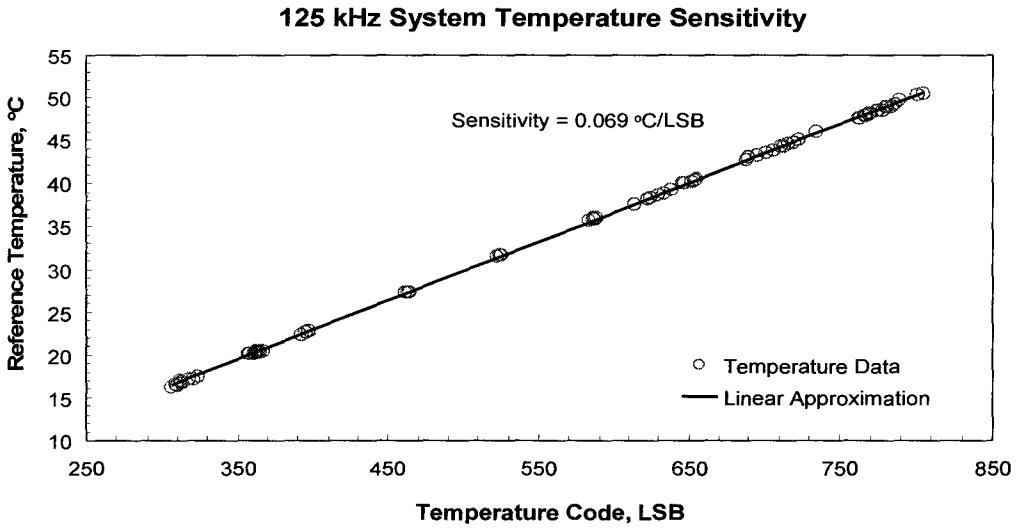


Figure 5.9: 125 kHz System - Temperature Sensitivity.

After obtaining the temperature sensitivity of the system, the received temperature data is converted, using the linear model, to temperature values and plotted against the reference temperature, shown in Figure 5.10.

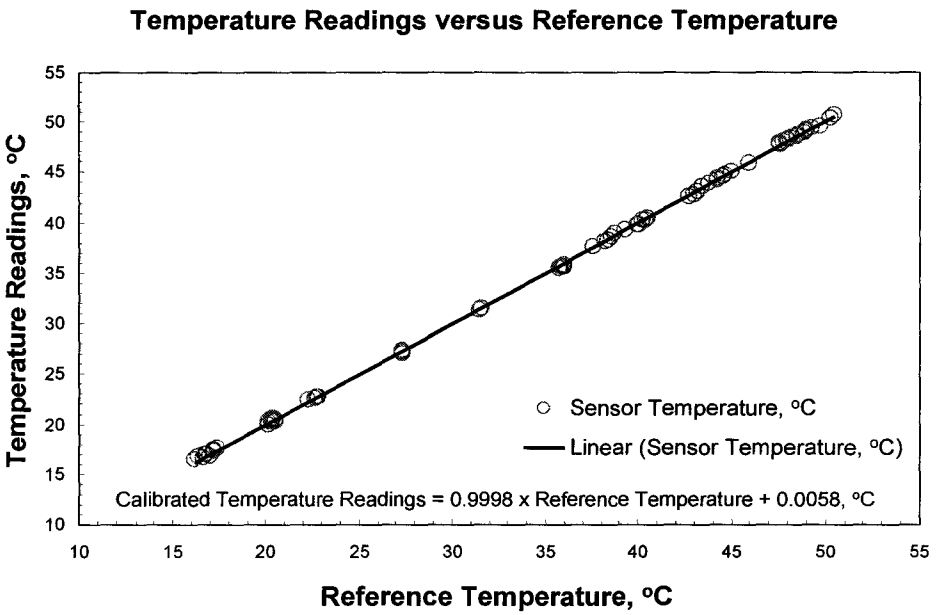


Figure 5.10: Calibrated Temperature versus Reference Temperature.

The temperature difference between measured temperature and reference temperature is shown in Figure 5.11. The standard deviation of the temperature difference is calculated as 0.150 °C.

From the temperature data, it is observed that temperature data has similar convex/concave characteristic as seen in the pressure data. Therefore, quadratic approximation is used to analyze the temperature data in the same manner. In Figure 5.12, temperature readings from quadratic model are plotted against the reference temperature. By comparing Figure 5.10 and Figure 5.12, it is observed that quadratic approximation does improve the temperature calibration slightly.

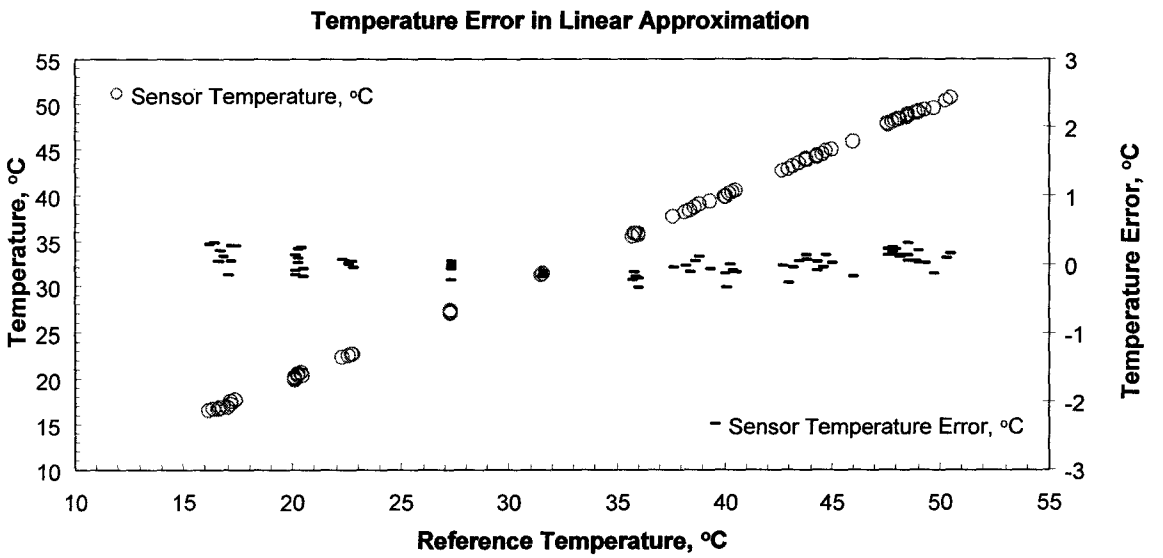


Figure 5.11: Temperature Error in Linear Approximation

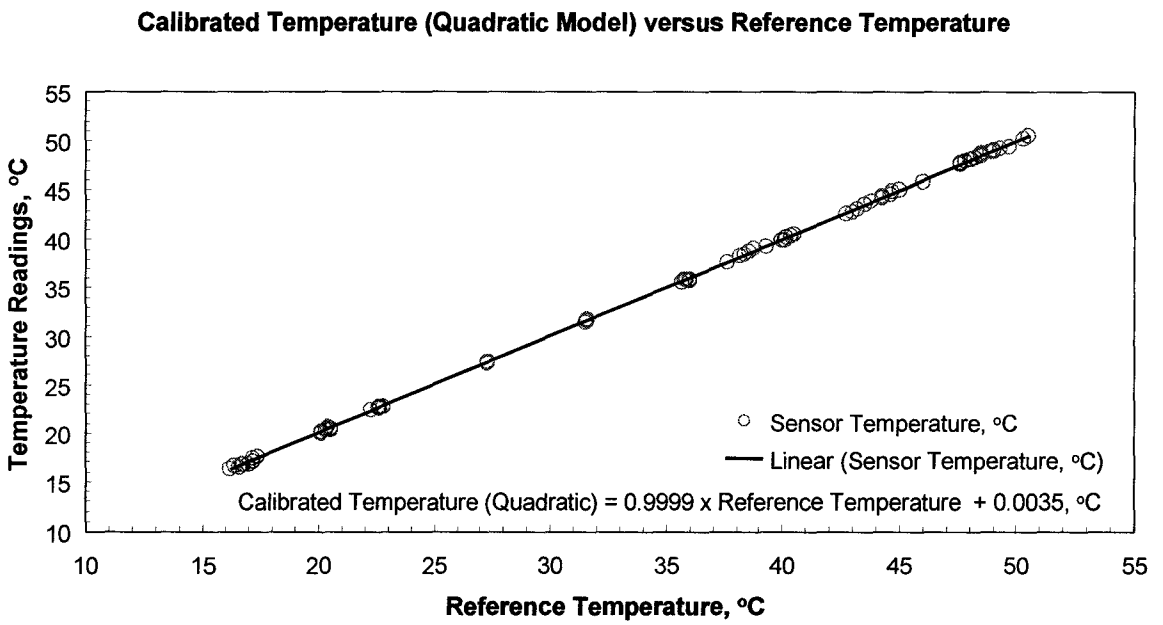


Figure 5.12: Calibrated Temperature (Quadratic) versus Reference Temperature.

Next, the temperature difference between calibrated temperature, using the quadratic model, and the reference temperature is shown in Figure 5.13. Now, the standard deviation of temperature error is calculated as 0.116 °C.

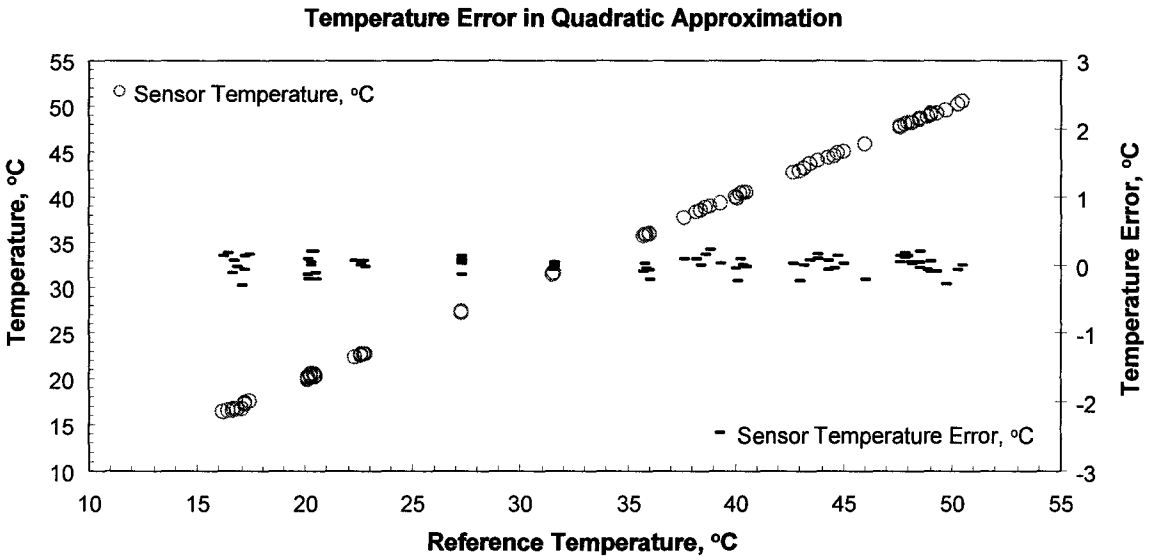


Figure 5.13: Temperature Error in Quadratic Approximation

5.3 Data Transfer, Storing and Consumption

Data Transfer and Storing

In the 2.4 GHz pressure sensor system, the data is being passed to the capsule imaging system [32] and transmitted together with its image data. The capsule imaging system has a transmission frequency of 2 frames-per-second (fps) and the pressure data is embedded in every frame of image. The transmission has a data loss of 10% on average. After the data is received in a custom-made receiver described in [32], it is then recorded and saved in binary files in a PC with a custom GUI used in [32] for further analysis.

In the 125 kHz dual sensor system, the prepared messages are being transmitted at a rate of 10 messages per second. At the receiver end, it is observed that the transmission has a 50% loss on average. Therefore, the actual rate of data transfer is 5 messages per second on average. Then, the received messages are transferred to a PC using RS232 standard. And finally, the data is decoded and saved into comma-separated-value format files.

Consumption

During the experiments, a power source is used to supply each system. It is observed that the 2.4 GHz pressure sensor system has a current consumption of approximately 5 mA (excluding the transmitter's consumption) while operating at 3 V. The current consumption of the transmitter is about 10 mA in transmitting mode. The 125 kHz dual sensor system has a current consumption of approximately 1.5 mA while operating at 3 V.

5.4 Chapter Summary

A table of measured parameters of two systems is shown in Table 5.1. It can be seen that there is trade-off between consumption and pressure sensitivity. Also, since the leakage detection mechanism of the 2.4 GHz pressure sensor system is made in the

firmware, the detection is fixed and the firmware more complex than it is in the 125 kHz dual sensor system. In the 125 kHz system, the detection mechanism is provided in the GUI. The firmware is simpler in this system, but the software in this case offers more flexibility. As mentioned previously, different characteristics of the pressure data can be chosen to display and evaluate.

Table 5.1: Measured Parameters of Both Systems

	2.4 GHz Pressure Sensor System	**125 kHz RFID Dual Sensor System**	**Units**
Pressure Sensitivity	1.14	2.92	cmH_2O/LSB
Detection Mechanism	Firmware	Software	
Temperature Sensitivity	N/A	0.069	°C/LSB
Data Rate	2	5	messages/sec
Data Loss	10%	50%	messages
Current Consumption	5.0	1.5	mA

CHAPTER 6: SYSTEM PERFORMACE

6.1 Verification of System Design

A comparison of technical specification and measured results for 2.4 GHz pressure sensor system is shown in Table 6.1.

Table 6.1: System Specification Comparison – 2.4 GHz System

	Tech. Specification	Measured	Units
Pressure Sensitivity	0.99	1.14	cmH_2O/LSB
Current Consumption	4.0	5.0	mA

In the table above, it is seen that there is a difference between the sensitivity from specifications and measurements. This is mainly due to the errors from the ADC circuitry. A list of the errors from the components' manuals contributing to the pressure sensitivity loss in this system is shown in Table 6.2.

Table 6.2: List of Errors in 2.4 GHz Pressure Sensor System

	Error
Pressure Sensor Error	0.20%
Voltage Reference Error	4.20%
Gain Error	3.60%

The most significant source of errors is from the ADC circuitry in the microcontroller. Only limited data in the microcontroller's manual is given about errors in the analog circuit. By taking into considerations of errors that introduced by the hardware, the measured pressure sensitivity is considered within the technical specification of the 2.4 GHz pressure sensor system.

Next, the technical specification and measured results of 125 kHz RFID dual sensor system is shown in Table 6.3. And a list of possible error sources in shown in Table 6.4. These values are from the manual specifications of each component. There is no information given in the temperature sensor's manual.

Table 6.3: System Specification Comparison – 125 kHz System

	Tech. Specification	Measured	Units
Pressure Sensitivity	2.93	2.92	cmH_2O/LSB
Temperature Sensitivity	0.086	0.069	°C/LSB
Current Consumption	1.0	1.5	mA

Table 6.4: List of Errors in 125 kHz Dual Sensor System

	Error
Pressure Sensor Error	0.20%
Voltage Reference Error	9.10%
Gain Error (Differential)	2.50%
Gain Error (Single-Ended)	0.50%

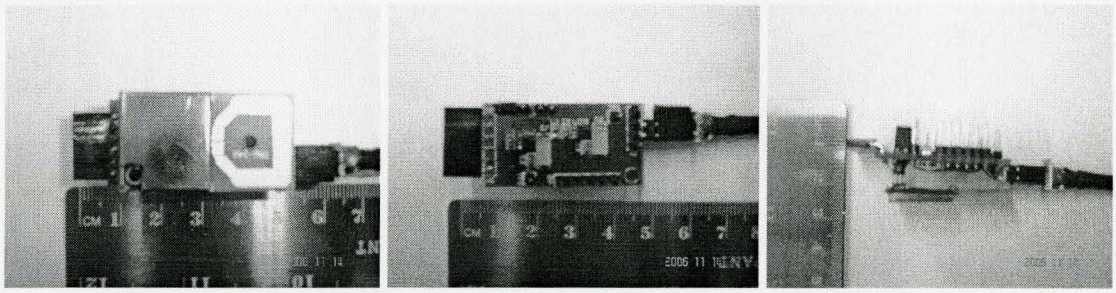
In this system, the measured pressure sensitivity and temperature sensitivity have met the technical specifications when hardware errors are considered.

6.2 Trade-offs in the systems

In Table 6.5, parameters of both systems are given as a comparison. From this table, the trade-offs between consumption, sensitivity, size, and transmission range can be seen clearly. Despite the consumption of the wireless transmitter in the 2.4 GHz system, three times more current consumption (5 mA compared to 1.5 mA) has led to an increase in the pressure sensitivity approximately three times. Moreover, the system dimension of the 2.4 GHz system is limited by the size of the wireless transmitter. On the other hand, the system dimension of the 125 kHz system is only limited by the arrangement of components on the PCB. The transmission range of the 125 kHz system is significantly shorter than the range of the 2.4 GHz system because of the technology difference. The transmission mechanism in the 125 kHz is considered passive while the transmitter in the 2.4 GHz system transmits messages all the time. Pictures of the 2.4 GHz pressure sensor system are shown in Figure 6.1, and a picture of the 125 kHz dual sensor system is shown in Figure 6.2. Lastly, the components cost of the 2.4 GHz system is $24 CAD and the components cost of the 125 kHz system is $9 CAD. Unit prices of each component can be found in Appendix C.

Table 6.5: System Performances of Both Systems

	2.4 GHz Pressure Sensor System	125 kHz RFID Dual Sensor System	Units
Pressure Sensitivity	1.14	2.92	cmH_2O/LSB
Detection Mechanism	Firmware	Software	
Temperature Sensitivity	N/A	0.069	°C/LSB
System Dimension	75 .0 × 20 .5 × 17 .5	30 .0 × 15 .0 × 15 .0	mm^3
Current Consumption	5.0	1.5	mA
Transmitter Consumption	10.5	N/A	mA
Transmission Range	>5	0.02	metres



(a) Top view (b) Bottom view (c) Side view

Figure 6.1: Pictures of 2.4 GHz Pressure Sensor System

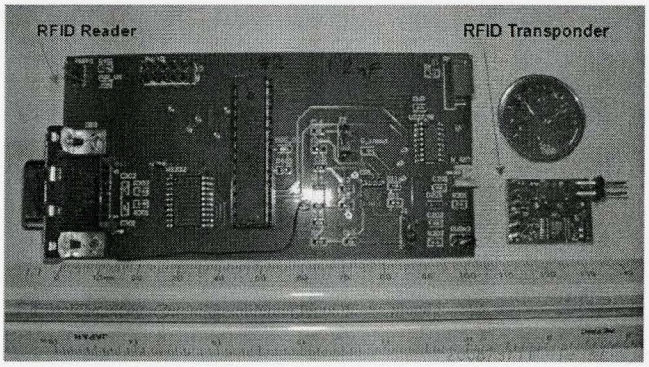


Figure 6.2: Picture of 125 kHz RFID Dual Sensor System.

CHAPTER 7: CONCLUSION

In this thesis, a detailed study of implantable microelectronic systems for sensor-based measurement was presented. A 2.4 GHz ISM band wireless pressure sensor and a 125 kHz RFID wireless dual sensor systems were designed towards one goal of in vivo monitoring of bladder pressure. Their functionalities were studied and quantified.

7.1 Summary

The first system designed in this work was the 2.4 GHz wireless pressure sensor system. A prototype of the system is designed to acquire pressure data using a piezoresistive pressure sensor, digitalize the data, process it and transmit it wirelessly. The integrated prototype was built with only off-the-shelf components and it was built on a commercially available printed-circuit-board. An overall system size of 75.0 × 20.5 × 17.5 mm^3 was achieved and a current consumption of 5 mA was measured. A pressure sensitivity of 1.14 cmH_2O/LSB from the pressure sensor was obtained. A transmission range of at least 5 metre was observed. The processed data is transmitted using a frequency range of 2.4-2.524 GHz with a data rate of 2 messages per second (limited by the amount of transmitted data in [32]). In the design, it was found that the current consumption (10.5 mA) and the size of the wireless transceiver pose greatest constraints in the miniaturization of the system.

The 125 kHz RFID dual sensor system was the second system designed in this work. This system is designed to acquire both pressure and temperature data using the same pressure sensor and a low power CMOS integrated-circuit temperature sensor respectively. Again, the integrated system was built on a commercially available printed-circuit-board. An overall system dimension of 30 × 15 × 15 mm^3, without the antenna, was achieved using only off-the-shelf components. A pressure sensitivity of 2.92 cmH_2O/LSB and a temperature sensitivity of 0.069 ^{o}C/LSB were achieved. A transmission range of 2 cm

was observed. The acquired sensory data was transmitted at average data rate of 5 messages per second. And a current consumption of 1.5 mA was measured.

7.2 Future Recommendations

The work performed in this research is only a preliminary study of implantable microelectronic systems for medical use. Several recommendations to further improve the system are described in this section.

First, a pressure sensor with low power consumption and adequate sensitivity should be used in the future research. One way to achieve lower power consumption is to utilize a piezocapacitive pressure sensor or a digital pressure module, such as the MS5536 from Intersema Sensoric SA. The MS5536 pressure module a standby current consumption of 3.5 μA and a conversion current consumption of 1 mA at a 3 V supply voltage while offering a pressure sensitivity of 0.1 cmH_2O/LSB [29].

Next, if RFID communication method is further explored in the future, an extensive study in the field of RFID antenna design is required in order to replace the transponder's 11 × 13.5 cm^2 square-loop antenna with a suitably sized antenna. For RFID purpose, a spiral wounded antenna should be considered if the desired system has a cylindrical shape and such an antenna can be designed and embedded in the system. In [47], a set of equations has be derived to calculate the optimum turns and current of the reader loop antenna by assuming a specific transponder loop antenna radius and a specific reading range. First, equation (7.1) is used to calculate the magnetic strength of arrival signal, which is given by

$$V_o = 2\pi f N S Q B_o \cos\alpha \tag{7.1}$$

where V_o is the required induced voltage at the transponder's loop antenna, f is the frequency of the signal, N is the number of turns of transponder's loop antenna, S is the cross-section area of the transponder's loop antenna, Q is the quality factor of the

transponder's loop antenna, B_o is the required magnetic strength of arrival signal, and α is the orientation of the transponder's loop antenna with respect to the reader's loop antenna.

Then, equation (7.2) can be used to calculate the require reader loop antenna turns and current, which is given by

$$B_z = (\mu_o N_r I a^2) / 2(a^2 + r^2)^{3/2} \tag{7.2}$$

where B_z is the magnetic field produced by a loop antenna, I is the current, N_r is the number of turns of the loop, a is the radius of the loop, r is the distance from the centre of the loop, and μ_o is the permeability of free space.

And finally, equation (7.3) should be applied to calculate the optimum radius of the reader loop antenna, which is given as

$$a = \sqrt{2}\, r \tag{7.3}$$

where a is the radius of the reader's loop antenna and r is the desired reading distance.

For example, if a reading range of 10 cm is desired, the reader's loop antenna should have a radius of 14.14 cm calculated using equation (7.3). Next, if a specific induced voltage is required on the transponder's loop antenna, equation (7.1) can be used to calculate the required magnetic field strength. For instance, by assuming the following values, N=100, a = 1 cm, V_o= 4 V, $\alpha = 0^o$, Q = 30, and f= 125 kHz, the required magnetic field strength, B_o, can be calculated as 3.82 μT. Then, in order to produce this magnetic field, the number of turns and current of the reader's loop antenna can be calculated using equation (7.2) as 1.11 ampere-turns. This means that if the reader's loop antenna has 100 turns, the minimum current required to generate the 125 kHz signal is 11.1 mA. Finally, to calculate the maximum magnetic field strength generated by the reader's loop antenna, the distance, r, is assumed to be zero in the equation (7.2). And, by using the required number of turns and current, radius of the reader's antenna, and a distance of zero, the maximum generated magnetic field strength is calculated as 4.94 μT. According to Safety Code 6

published by Health Canada, the maximum magnetic field strength for this frequency range a person can be exposed to is 4.9 A/m or 6.16 μT. And the above calculation is assuming the transmission medium is air [46].

In the case of battery powered systems, other methods of wireless communication could also be considered. For example, ZL70250 ultra low power RF transceiver (less than 2 mA current consumption) from Zarlink Semiconductor which offers data rates up to 186 kbps in unlicensed frequency bands between 795 – 965 MHz [30] and ZL70101 Medical Implantable RF Transceiver (1 mA low power mode and 5 mA continuous mode) which offers data rates up to 800 kbps in the medical implant communication service frequency bands between 402-405 MHz [31] can both be considered. Moreover, a microcontroller integrated with ZigBee® technology which is a low-power short-distance wireless standard, could also be used to simplify the integration of a microcontroller and a wireless transceiver. And which may eventually lead to further miniaturization of the system.

Moreover, optimization of firmware in the system is also required to obtain a more power-efficient measurement method, such as a sleep-and-sniff function to sample sensor data only when reader is in proximity. Or alternatively, to turn on the sensors prior to each ADC conversion and turn off the sensor after each ADC conversion is another method of reducing consumption. Currently, both systems are configured to operate continuously to simplify the debugging process in the prototypes.

Finally, to transform these prototypes into practical systems, several recommendations are given. First, other possible wireless transmission method, such as using the ZL70101 Medical Implantable RF Transceiver, can be explored. Then, a custom design of the whole system (transducer units, signal processing unit, and data transmission unit) on a chip or in a package (SoC/SiP) can be considered to further miniaturize the system size. Moreover, plastic or resin could be used to enclose the system. Lastly, the system should be tested in a human body model to verify functions of the system.

REFERENCES

[1] Earl E. Bakken. "Timeline of Earl E. Bakken." Earl E. Bakken 2006 <http://www.earlbakken.com/content/timeline/timeline.html>

[2] Glenn M. Eisen. "The Economics of PillCam." Gastrointestinal Endoscopy Clinics of North America 16 (2006): 337-345.

[3] Walter H. Olson. "Basic concepts of instrumentation." Medical instrumentation Application and design. Boston: Houghton Mifflin, 1978. 1-42.

[4] Mohammad H. Asgarian. "Artificial Pacing." Cardiac Pacemakers. Ed. John G. Webster. New York: IEEE 1995. 105-131.

[5] Clark Graeme. Cochlear Implants: fundamentals and applications. New York: Springer, 2003.

[6] National Institute on Deafness and Other communication Disorders. "What is a cochlear implant?" Cochlear Implants. 2007. <http://www.nidcd.nih.gov/-health/hearing/coch.asp>

[7] Marcia Yu. "M2A™ Capsule Endoscopy A Breakthrough Diagnostic Tool for Small Intestine Imaging." Gastroenterology Nursing. 25(1) (2002): 24-27.

[8] The Canadian Continence Foundation. "Facts on Incontinence." Information about Incontinence. TCCF 2006 <http://www.continence-fdn.ca/consumers/facts.html>

[9] Mayo Foundation for Medical Education and Research. "Urinary Incontinence." Diseases and Conditions. MFMER. 2007.

[10] Donald R. Ostergard and Alfred E. Bent. Urogynecology and Urodynamics: Theory and Practice. Maryland: Williams and Wilkins, 1996.

[11] John W. Warren. "Catheter-Associated Urinary Tract Infections" Infectious Disease Clinics of North America. 11(3) (1997): 609-622.

[12] Chisato Enokawa, Yoshiharu Yonezawa, Hiromichi Maki, and Mikio Aritomo. "A Microcontroller-Based Implantable Telemetry System for Sympathetic Nerve Activity and ECG Measurement." IEEE Proceedings of the 19th International Conference on Engineering in Medicine and Biology society. 1997. 2232-2234.

[13] H. J. Park, H. W. Nam, B. S. Song, and J. H. Cho. "Design of Miniaturized Telemetry Module for Bi-Directional Wireless Endoscopy." IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science. E86-A(6) (2003): 1487-1491.

[14] Pietro Valdastri, Arianna Menciassi, Alberto Arena, Chiara Caccamo, and Paolo Fario. "An Implantable Telemetry Platform System for In Vivo Monitoring of Physiological Parameters." IEEE Transactions on Information Technology in Biomedicine. 8(3) (2004): 271-278.

[15] James F. Rorie, John K. Simmerman, Rafic Z. Makki, and Richard D. Piendl. "A High Level Language Implementation of a General Purpose Telemetry System for Biomedical Applications." IEEE Proceedings of the 28th Southeastern Symposium on System Theory. 1996. 338-342.

[16] Silicon Laboratories Inc., Datasheet, "Mixed-Signal 32KB ISP FLASH MCU Family", 2003.

[17] Merit Sensor Systems, Datasheet, "3000 Series Piezoresistive Pressure Sensor Die", 2004.

[18] Laipac Technology, Datasheet, "TRF-2.4G Transceiver", 2001.

[19] RFID Journal. "General RFID Information" RFID FAQS. RFID Journal, 2002. <http://www.rfidjournal.com/faq>

[20] Atmel Corporation, Datasheet, "8-bit AVR® Microcontroller with 2/4/8K Bytes In-System Programmable Flash: ATtiny24/4/84", 2007.

[21] National Semiconductor, Datasheet, "LM94022 – 1.5V, SC70, Multi-Gain Analog Temperature Sensor with Class-AB Output from the PowerWise® Family", 2007.

[22] Atmel Corporation, Datasheet, "Transponder Interface for Microcontroller: U3280M", 2007.

[23] Atmel Corporation, Datasheet, "Read/Write Base Station: U2270B", 2006.

[24] Atmel Corporation, Datasheet, "8-bit AVR® Microcontroller with 8K Bytes In-System Programmable Flash: ATmega8/ATmega8L", 2006.

[25] Maxim Integrated Products, Datasheet, "±15kV ESD-Protected, +5V RS-232 Transceivers", 2005.

[26] S. D. Eckford, R. Finney, S. R. Jackson, and P. Abrams. "Detection of Urinary Incontinence during Ambulatory Monitoring of Bladder Function by a Temperature-sensitive Device." British Journal of Urology. 77 (1996): 194-197.

[27] Wendy M. Fallis. "Monitoring Urinary Bladder Temperature in the Intensive Care Unit: State of the Science." American Journal of Critical Care. 11(1) (2002): 38-47.

[28] Wendy M. Fallis. "Monitoring Bladder Temperatures in the OR." AORN Journal. 76(3) (2002):467-489.

[29] Intersema Sensoric SA, Datasheet, "MS5536 SMD Gage Pressure MODULE", 2007.

[30] Zarlink Semiconductor, Datasheet, "ZL70250 Ultra Low Power RF Transceiver", 2008

[31] Zarlink Semiconductor, Datasheet, "ZL70101 Medical Implantable RF Transceiver", 2007

[32] Moussa Kfouri, Ognian Marinov, Paul Quevedo, Naser Faramarzpour, Shahram Shirani, Louis W.-C. Liu, Qiyin Fang, and M. Jamal Deen. "Toward a Miniaturized Wireless Fluorescence-Based Diagnostic Imaging System" IEEE Journal of Selected Topics in Quantumelectronics. 14(1) (2008): 226-234

[33] Federal Communication Commission. "Code of Federal Regulations - Title 47 - Volume One, Part 5.89 – Experimental radio service", (2007):709 <http://edocket.access.gpo.gov/cfr_2007/octqtr/47cfr5.89.htm>

[34] Cole-Parmer Canada Inc. "RK-68600-06 Handheld Differential Pressure Gauge." 2008. <http://www.coleparmer.ca/catalog/product_view.asp?sku=6860006>

[35] Kaïs Mnif. "Compensation is Critical in Fitting Analog Pressure Sensor to the Application." Integrated System Design Magazine. July (2001): 52-55

[36] Cole-Parmer Canada Inc. "RK-91100-4 Digi-Sense Dual Input J-T-E-K Thermocouple Thermometer." 2008. <http://www.coleparmer.ca/catalog/product_view.asp?sku=9110040>

[37] Corrado Cavalli. "Class for handling RS232 communication with VB.Net." 2001 <www.codeworks.it/net/index.htm>

[38] T. Yamada, K. Okada, K. Masu, A. Oki, and Y. Horiike. "Battery-less Wireless Communication System Using PIM for *in-vivo* Healthcare Chip." IEEE Workshop on Wireless Circuits and Systems. 2004.

[39] T. Yamada, H. Uesugi, K. Okada, K. Masu, A. Oki, and Y. Horiike "Preliminary Measurements for *in-vivo* Wireless Communication Chip." International Conference on Solid State Devices and Materials. 2003. 366-367

[40] K. Okada, T. Yamada, T. Uezono, K. Masu, A. Oki, and Y. Horiike. "Near Field Communication Chip using PIM for Bio MEMS Sensors." IEEE Asia-Pacific Conference on Advanced System Integrated Circuits. 2004. 440-441

[41] T. Yamada, T. Uezono, H. Sugawara, K. Okada, K. Masu, A. Oki, and Y. Horiike. "Battery-less Wireless Communication System through Human Body for *in-vivo* Healthcare Chip." IEEE Topical Meeting on Silicon Monolithic Integrated Circuits in RF Systems. 2004. 322-325

[42] SmartPill Corporation. "The SmartPill GI Monitoring System." 2003 <http://www.smartpillcorp.com/site/smartpill/assets/pdf/SmartPill_GI_Monitoring_System_Brochure.pdf>

[43] SmartPill Corporation. "pH.p Capsule Opeartional Specifications." 2003 <http://www.smartpillcorp.com/site/smartpill/assets/pdf/SmartPill_Operational_Specs.pdf>

[44] H. Bruschini, F. G. Almeida, and M. Srougi. "Upper and lower urinary tract evaluation of 104 patients with myelomeningocele without adequate urological management" World Journal of Urology. (2006) 24: 224-228

[45] D. Sinha, V. Nallaswamy, and A. S. Arunkalaivanan. "Value of Leak Point Pressure Study in Women with Incontinence" Journal of Urology. (2006) 176: 186-188

[46] Health Canada. "Limits of Human Exposure to Radiofrequency Electromagnetic Fields in the Frequency Range from 3 kHz to 300 GHz." Safety Code 6. 1999

[47] Microchip Technology Inc., Application Notes, "Antenna Circuit Design for RFID Applications", 2003

APPENDICES

Appendix A: 2.4 GHz System Firmware and MATLAB Scripts.

A.1 2.4 GHz Pressure Sensor System Firmware

main_pressure_MCU.asm

```
; This file is the core firmware of the system
; LONG_AVG and EXP_AVG routines can be found here.
;-------------------------------------------------------------------------------
; GLOBAL VARIABLES AND ASSIGNMENTS
;-------------------------------------------------------------------------------
$INCLUDE(C8051F000.INC)       ; Register definition file.
;include/remove keys
       include_simple_AVG     equ     'N'     ;'Y'es 'No', include subroutine AVG_ADC:
       LED            equ     P0.7
       CS             equ     P0.5 ; transmitter
       CE             equ     P0.4 ; transmitter
       NSS     equ    P0.3 ; hard to VDD, always 1
       MOSI    equ    P0.2 ; SPI data out
       MISO    equ    P0.1 ; SPI data in, not used
       SCK     equ    P0.0 ; SPI clock
PUBLIC  LED, CS, CE, NSS, MOSI, MISO, SCK
if     include_simple_AVG='Y'
       AVG_EXP equ    3       ;used in AVG_ADC:
       NUM_AVG equ    (1 SHL (AVG_EXP-0)) ;used in AVG_ADC:
endif
       AVG_EXP_LONG equ 8                          ; for Normal AVG
       NUM_AVG_LONG   equ     (1 SHL (AVG_EXP_LONG-0))
       EXP_EXP_AVG equ 4                          ; for exp AVG
       NUM_EXP_AVG equ        (1 SHL (EXP_EXP_AVG-0))
       PRESSURE_THRESHOLD     equ     4
;-------------------------------------------------------------------------------
; EXTERNAL CODE
;-------------------------------------------------------------------------------
EXTRN   CODE(Init_MCU) ; initialization routine in Config_MCU.asm
;-------------------------------------------------------------------------------
; STACK SEGMENT
;-------------------------------------------------------------------------------
Stck    segment IDATA
            rseg       Stck              ; switch to this code segment.
Stack:  ds      64                   ; reserve 64B space
; ===========================================================
RAM_BYTES       segment DATA
                rseg    RAM_BYTES
if     include_simple_AVG='Y'
ADC_CNT:        ds      1       ;used in ADC_inrterrupt: in conjuction with AVG_ADC:
ADC_AVG:        ds      2         ;used in AVG_ADC:
ADC_AVGL        equ     ADC_AVG ;used in AVG_ADC:
ADC_AVGH        equ     (ADC_AVG+1)     ;used in AVG_ADC:
endif

SPI_CNT: DS     1               ; used as counter to keep track of transmitted data
XRAM_DATA: DS 1         ; used to save XRAM DATA
PUBLIC SPI_CNT, XRAM_DATA
ADC_AVG_LONG_CNT: DS 1
ADC_AVG_LONG: DS 3         ;L H U
ADC_AVG_LONG_L equ ADC_AVG_LONG
ADC_AVG_LONG_H equ (ADC_AVG_LONG+1)
```

```
ADC_AVG_LONG_U equ (ADC_AVG_LONG+2)
ADC_AVG_LONG_RESULT: DS 2 ;L H-(H U of ADC_AVG_LONG)
ADC_AVG_LONG_RESULT_H equ ADC_AVG_LONG_RESULT
ADC_AVG_LONG_RESULT_L equ (ADC_AVG_LONG_RESULT+1)
EXP_AVG: DS 3        ;L H U
EXP_AVG_L equ EXP_AVG
EXP_AVG_H equ (EXP_AVG+1)
EXP_AVG_U equ (EXP_AVG+2)
TEMP: DS 3
TEMP_U equ TEMP
TEMP_H equ (TEMP+1)
TEMP_L equ (TEMP+2)
DIFF: DS 2          ; FOR DIFFERENCE CALCULATION
DIFF_L equ DIFF
DIFF_H equ (DIFF+1)
; public results for preparing message
if     include_simple_AVG='Y'
PUBLIC ADC_AVGH, ADC_AVGL
endif
PUBLIC ADC_AVG_LONG_RESULT_H, ADC_AVG_LONG_RESULT_L
PUBLIC EXP_AVG_U, EXP_AVG_H, EXP_AVG_L
PUBLIC DIFF_H, DIFF_L
TMRR: ds 2
TMRL equ TMRR
TMRH equ (TMRR+1)
; pointer in external RAM (last used byte, increment DPTR before appending data)
DPL_XRAM:      ds     1
DPH_XRAM:      ds     1
PUBLIC  DPH_XRAM,DPL_XRAM
DP_XRAM_MAX    equ    (0x0800-256)   ;max address to store results in external RAM
SCAN_XRAM_L:   ds     1      ;to scan external RAM via DAC
SCAN_XRAM_H:   ds     1      ;used in XRAM_to_DAC
Time_Out_L:    ds     1      ; about several seconds timeout after reset or to turn-off
the LED
Time_Out_H:    ds     1
Time_Out       equ    (0x0CFFF)      ;timeout duration (# of timer3 interrupts)
REF_PRESSURE:  ds     2      ;holds 1 atm pressure after reset timeout
REF_PRESSURE_H equ    REF_PRESSURE
REF_PRESSURE_L equ    (REF_PRESSURE+1)
PUBLIC REF_PRESSURE_H, REF_PRESSURE_L
;------------------------------------------------------------------------------
; BIT DATA SEGMENT
;------------------------------------------------------------------------------
Flags   segment BIT
                rseg   Flags        ; switch to this code segment
if     include_simple_AVG='Y'
flag_neg_ADC:  dbit   1             ; flag used in subroutine AVG_ADC:
endif
flag_new_ADC: dbit  1   ; NEW DATA FROM ADC
flag_exp_EMPTY: dbit 1   ; for exp_avg use
flag_new_exp_ADC: dbit 1
flag_new_exp_ADC_done: dbit 1
flag_doing_RESET_TimeOut:     dbit    1;=1 after reset for several seconds, then 0
;------------------------------------------------------------------------------
; INTERRUPT VECTOR CODE
;------------------------------------------------------------------------------
       cseg    AT 0000h
       ljmp   Begin_after_RESET
       ; Place jump to reset handler and interrupt service routines here.
;cseg   AT     002Bh
;ljmp   Timer2_interrupt
       cseg   AT     0073h
       ljmp   Timer3_interrupt
       cseg    AT 007Bh
       ljmp    ADC_interrupt
       org 0B3h                    ; End of Interrupt Vector space.
       cseg    AT    0B4h
```

```
        using  0
Begin_after_RESET:
        mov       SP,#Stack-1           ; stack pointer to indirect RAM
        call     Init_MCU               ; Config_MCU.asm
;-------------------------------------------------------------------------------
; MAIN PROGRAM CODE
;-------------------------------------------------------------------------------
Main:                    ; main code routines here...
if      include_simple_AVG='Y'
        mov       ADC_CNT,#0              ; set # of adc conversions =0
        ;used in ADC_inrterrupt: in conjuction with AVG_ADC:
endif
;Time_Out      equ     (0)     ;timeout duration (# of timer3 interrupts)
        mov     Time_Out_L,#0  ;ds     1
                ; about several seconds timeout after resed or to turn-off the LED
        mov     Time_Out_H,#0  ;       ds      1
        setb    flag_doing_RESET_TimeOut
                ;=1 after reset for several seconds, then 0
        setb    EA                              ; enable interrupts
        clr             flag_new_ADC   ; no ADC result available
        clr             flag_new_exp_ADC       ; no ADC avg available. Do not do exp avg
        setb  flag_exp_EMPTY
        mov     R5,#0                           ; counter for data transmitted
Main_Loop:
        inc             R5
        mov       R7,#0ffh
Main_Loop1:
        mov       R6,#00fh
Main_Loop2:
if      include_simple_AVG='Y'
        call    AVG_ADC
endif
        call  LONG_AVG                ; calculates long average here
        call  EXP_AVG_CODE            ; calculates exp. average here
        CALL  DIFF_CODE                       ; calculates difference between long and exp.
        djnz    R6,Main_Loop2
        djnz    R7,Main_Loop1
        mov             SPI_CNT, R5
        call    Init_RF ;this is in file "SPI_RF.asm"
        call    Send_RF ;this is in file "SPI_RF.asm"
        jmp       Main_Loop
;=============================================================
ADC_interrupt:
        push    ACC
        mov       A,R0
        push    ACC

        clr       ADCINT                        ; clear interrupt flag
if      include_simple_AVG='Y'
        mov       A,ADC0L                       ;low byte of ADC
        mov             R0,ADC_CNT
        movx  @R0,A
        mov       A,ADC0H                       ; high byte of ADC
        inc       R0
        movx  @R0,A
        inc       R0
        mov       A,#00eh                       ; prepare ADC_CNT for next ADC
        anl       A,R0
        mov       ADC_CNT,A
endif
        pop       ACC
        mov   R0,A
        pop       ACC
        setb  flag_new_ADC     ; DBIT  1      ; NEW DATA FROM ADC
        reti                                          ; end of ADC_inrterrupt
;=============================================================
;Timer 3 Overflow
```

```
        TemperatureBox.Image = temp_pic

        If BI = (Max_x1 - 1) Then
            BitmapIndex = 0
        End If

    End Sub

#End Region

#Region " Timer1 - running but not used in program"
    Public Sub Timer1_INT(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Timer1.Tick
        'Make_Picture(160, 100, BitmapIndex)
        'BitmapIndex = BitmapIndex + 1
    End Sub
#End Region

#Region " File Functions "

    Private Sub CreateFile(ByVal filename As String)
        ' Create new file with title row for each column
        Dim fs As New FileStream(filename, FileMode.Create, FileAccess.Write)
        Dim s As New StreamWriter(fs)
        s.BaseStream.Seek(0, SeekOrigin.End)
        s.WriteLine("-1,ID       Byte,Data       Counter,P.H.Byte,P.L.Byte,T       H_Byte,T
L_Byte,CheckSum,Validity, Time, Time(ms),Calculated Pressure, Calculated Temperature") '
        s.Close()
    End Sub

    Private Sub WriteLineToFile(ByVal txt As String, ByVal filename As String)
        ' Write one row(message) to file
        Dim fs As New FileStream(filename, FileMode.Append, FileAccess.Write)
        Dim s As New StreamWriter(fs)
        s.BaseStream.Seek(0, SeekOrigin.End)
        s.WriteLine(txt)
        s.Close()
    End Sub

    Private Sub MaxMsgNum_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MaxMsgNum.SelectedIndexChanged
        ' Changes maximum number of messages saved in one file
        ' only changable when system is not running
        'Public MaxData_Num As Integer
        If Run_Flag Then
            MaxMsgNum.SelectedIndex = 0
            MaxMsgNum.Text = MaxMsgNum.SelectedItem
            MaxData_Num = Int32.Parse(MaxMsgNum.Text)
        Else
            MaxMsgNum.Text = MaxMsgNum.SelectedItem
            MaxData_Num = Int32.Parse(MaxMsgNum.Text)
        End If

    End Sub

    Private Sub MaxFileNum_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MaxMsgNum.SelectedIndexChanged
        'Public MaxData_Num As Integer
        'Public MaxFile_Num As Integer
        If Run_Flag Then
            Return
        Else
            MaxFileNum.Text = MaxFileNum.SelectedItem
            MaxFile_Num = Int32.Parse(MaxFileNum.Text)
        End If

    End Sub
```

```
#End Region

#Region " Reading Bytes from File "
    Private Function ReadFile(ByVal filename As String) As String
        Dim fs As New FileStream(filename, FileMode.Open, FileAccess.Read)
        Dim s As New StreamReader(fs)
        Dim temp As String
        temp = s.ReadLine()
        s.Close()
        Return temp
    End Function
#End Region

#Region " FORM EXIT - MENU STRIP "

    Private Sub Menu_File_Exit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Menu_File_Exit.Click
        System.Windows.Forms.Application.Exit()
    End Sub

#End Region

#Region " Config - Import or Create config file"

    Private Sub Get_Config()

        Try
            Using MyReader As New
Microsoft.VisualBasic.FileIO.TextFieldParser(config_filename)
                MyReader.TextFieldType = FileIO.FieldType.Delimited
                MyReader.SetDelimiters(Delimiter)
                Dim currentRow As String()
                If MyReader.EndOfData Then

                End If
                While Not MyReader.EndOfData
                    Try
                        currentRow = MyReader.ReadFields()
                        Dim currentField As String
                        Dim i = 0
                        Dim cal_coef_val As String

                        For Each currentField In currentRow
                            i = i + 1
                            'MsgBox(i & ": " & currentField)
                            If i = 1 Then
                                Try
                                    cal_coef_val = Double.Parse(currentField)
                                Catch
                                    cal_coef_val = "0"
                                End Try
                            End If
                            If i = 2 Then
                                Try
                                    If currentField = "ZoomRate" Then ZoomRate.Text =
cal_coef_val
                                    If currentField = "TrackTime" Then TrackTime.Text =
cal_coef_val
                                    If currentField = "PCC" Then PCC.Text = cal_coef_val
                                    If currentField = "TCC" Then TCC.Text = cal_coef_val
                                    If currentField = "TOffset" Then TOffset.Text =
cal_coef_val
                                Catch ex As Exception
                                    MsgBox("can't write in textbox " & currentField)
                                End Try
                            End If
                        Next
```

```
                    Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
                        MsgBox("Line " & ex.Message & "is not valid and will be skipped.")
                    End Try
                End While
            End Using
        Catch ex As Exception
            MsgBox("No File: " & config_filename)
            MakeConfig("Make New Config", config_filename)
        End Try

        'Me.ZoomRate = New System.Windows.Forms.ToolStripTextBox
        'Me.TrackTime = New System.Windows.Forms.ToolStripTextBox
        'Me.PCC = New System.Windows.Forms.ToolStripTextBox
        'Me.TCC = New System.Windows.Forms.ToolStripTextBox
        'Me.TOffset = New System.Windows.Forms.ToolStripTextBox
    End Sub


    Private Sub MakeConfig(ByVal txt As String, ByVal filename As String)
        Dim fs_config As New FileStream(filename, FileMode.Create, FileAccess.Write)
        Dim s_config As New StreamWriter(fs_config)
        s_config.BaseStream.Seek(0, SeekOrigin.End)
        txt = ZoomRate.Text & Delimiter & "ZoomRate" & Delimiter & " for Pressure Graph"
        s_config.WriteLine(txt)
        txt = TrackTime.Text & Delimiter & "TrackTime" & Delimiter & " for Pressure"
        s_config.WriteLine(txt)
        txt = PCC.Text & Delimiter & "PCC" & Delimiter & "Calibration Coefficient for
Pressure in cmH2O/LSB"
        s_config.WriteLine(txt)
        txt = TCC.Text & Delimiter & "TCC" & Delimiter & "Calibration Coefficient for
Temperature in oC/LSB"
        s_config.WriteLine(txt)
        txt = TOffset.Text & Delimiter & "TOffset" & Delimiter & "Offset Calibration
Coefficient for Temperature in oC"
        s_config.WriteLine(txt)
        s_config.Close()
        'Me.ZoomRate = New System.Windows.Forms.ToolStripTextBox
        'Me.TrackTime = New System.Windows.Forms.ToolStripTextBox
        'Me.PCC = New System.Windows.Forms.ToolStripTextBox
        'Me.TCC = New System.Windows.Forms.ToolStripTextBox
        'Me.TOffset = New System.Windows.Forms.ToolStripTextBox
    End Sub

    Private Sub OpenConfig_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles OpenConfig.Click
        Get_Config()
    End Sub

    Private Sub SaveConfig_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SaveConfig.Click
        MakeConfig("Make New Config", config_filename)
    End Sub

#End Region

End Class
```

```
Timer3_interrupt:
        mov  TMR3CN, #006h     ; Timer 3 Control Register, clear interrupt flag
        call    XRAM_to_DAC                    ; scan XRAM to DAC
        call    Update_Time_Out        ; do several jobs at timeout
        jb       ADBUSY,ADC_is_converting
        setb ADBUSY                ; start ADC
ADC_is_converting:
        reti    ;Timer3_interrupt
;=================================================================
if      include_simple_AVG='Y'
AVG_ADC:
        mov     ADC_AVGL,#0
        mov     ADC_AVGH,#0
        mov     R1,#0
        mov     R2,#NUM_AVG
AVG_ADC_sum_loop:
        movx    A,@R1                                          ; get one of LB
        add     A,ADC_AVGL
        mov     ADC_AVGL,A
        inc     R1
        movx    A,@R1                                          ; get one of HB
        addc    A,ADC_AVGH
        mov     ADC_AVGH,A
        inc     R1
        djnz    R2,AVG_ADC_sum_loop
        mov     flag_neg_ADC,C
        mov     R2,#AVG_EXP
AVG_ADC_divide_by_NUM_AVG:
        mov C,flag_neg_ADC
        mov     A,ADC_AVGH
        rrc     A
        mov ADC_AVGH,A
        mov     A,ADC_AVGL
        rrc     A
        mov ADC_AVGL,A
        djnz    R2,AVG_ADC_divide_by_NUM_AVG
        ret                                                                    ; AVG_ADC
endif   ;include_simple_AVG='Y'
;=================================================================
LONG_AVG:
        jnb    flag_new_ADC,LONG_AVG_END
        mov     A,ADC0L                        ; low byte of ADC
        add     A,ADC_AVG_LONG_L
        mov     ADC_AVG_LONG_L,A
        ; ADC_AVG_LONG_L = ADC_AVG_LONG_L + ADC0L
         mov    A,ADC0H                        ; high byte of ADC
        addc    A,ADC_AVG_LONG_H
        mov     ADC_AVG_LONG_H,A
        ; ADC_AVG_LONG_H = ADC_AVG_LONG_H + ADC0H with carry
        mov     A,#0
        addc    A,ADC_AVG_LONG_U
        ; ADC_AVG_LONG_U = ADC_AVG_LONG_U + carry
        mov     ADC_AVG_LONG_U,A
        djnz  ADC_AVG_LONG_CNT,LONG_AVG_END
        ; decrement and jump to LONG_AVG_END if not zero
        mov   ADC_AVG_LONG_CNT,#(Low NUM_AVG_LONG)
        ; assign ADC_AVG_LONG_CNT = 0, NUM_AVG_LONG = 0010 0000b
        mov       R2,#(8-AVG_EXP_LONG)         ; assign R2 with 8 - 8 = 0
LONG_AVG_SHIFT_LOOP:
        mov       A,R2
        jz    LONG_AVG_SHIFT_LOOP_END
        ; jump to LONG_AVG_SHIFT_LOOP_END if A = 0
        mov   A,ADC_AVG_LONG_L
        rlc   A
        mov   ADC_AVG_LONG_L,A
        mov   A,ADC_AVG_LONG_H
        rlc   A
```

```
        mov   ADC_AVG_LONG_H,A
        mov   A,ADC_AVG_LONG_U
        rlc   A
        mov   ADC_AVG_LONG_U,A
        dec   R2
        jmp   LONG_AVG_SHIFT_LOOP
LONG_AVG_SHIFT_LOOP_END:
        mov   ADC_AVG_LONG_RESULT_L,ADC_AVG_LONG_H
        mov   ADC_AVG_LONG_RESULT_H,ADC_AVG_LONG_U
        setb  flag_new_exp_ADC  ; enable exp_avg
        clr     flag_new_exp_ADC_done
        jnb flag_doing_RESET_TimeOut,LONG_AVG_REF_PRESSURE_UPDATED
        mov     REF_PRESSURE_L,ADC_AVG_LONG_RESULT_L
        mov     REF_PRESSURE_H,ADC_AVG_LONG_RESULT_H
;REF_PRESSURE  ds      2       ;holds 1 atm pressure after reset timeout
LONG_AVG_REF_PRESSURE_UPDATED:
        mov   ADC_AVG_LONG_L,#0
        mov   ADC_AVG_LONG_H,#0
        mov   ADC_AVG_LONG_U,#0
LONG_AVG_END:
        clr  flag_new_ADC   ; clear flag
 ret                  ; LONG_AVG:
;=============================================================
EXP_AVG_CODE:
        jb              flag_new_exp_ADC,EXP_AVG_BEGIN
        ret     ;exit from EXP_AVG_CODE, do nothing
EXP_AVG_BEGIN:
        jnb   flag_exp_EMPTY,EXP_AVG_BEG
        mov   EXP_AVG_U,ADC_AVG_LONG_RESULT_H
        mov   EXP_AVG_H,ADC_AVG_LONG_RESULT_L
        mov   EXP_AVG_L,#0
        clr   flag_exp_EMPTY
EXP_AVG_BEG:
        mov   A,ADC_AVG_LONG_RESULT_L
        clr   C
        subb  A,EXP_AVG_H
        mov   TEMP_L,A
        mov   A,ADC_AVG_LONG_RESULT_H
        subb  A,EXP_AVG_U
        mov   TEMP_H,A
        mov   A,#0
        subb  A,#0
        mov   TEMP_U,A
        mov   R2,#(8-EXP_EXP_AVG)
        mov   A,R2
        jz    EXP_AVG_ROT_END
EXP_AVG_ROT:
        clr   C
        mov   A,TEMP_L
        rlc   A
        mov   TEMP_L,A
        mov   A,TEMP_H
        rlc   A
        mov   TEMP_H,A
        mov   A,TEMP_U
        rlc   A
        mov   TEMP_U,A
        djnz  R2,EXP_AVG_ROT
EXP_AVG_ROT_END:
        mov   A,TEMP_L
        add   A,EXP_AVG_L
        mov   EXP_AVG_L,A
        mov   A,TEMP_H
        addc  A,EXP_AVG_H
        mov   EXP_AVG_H,A
        mov   A,TEMP_U
        addc  A,EXP_AVG_U
```

```
        mov   EXP_AVG_U,A
        setb    flag_new_exp_ADC_done
;       mov     DAC0L,EXP_AVG_H
;       mov     DAC0H,EXP_AVG_U
        jb      flag_doing_RESET_TimeOut,MAX_XRAM_DONE
        mov     DPH,DPH_XRAM    ; store pressure in external RAM
        mov     DPL,DPL_XRAM
        inc     DPTR
;       mov     A,EXP_AVG_H             ; low byte of exp average
        mov     TEMP_L,ADC_AVG_LONG_RESULT_L  ;low byte of long average
        mov     TEMP_H,ADC_AVG_LONG_RESULT_H  ;high byte of long average
        clr     C
        mov     A,TEMP_L
        subb    A,REF_PRESSURE_L
        mov     TEMP_U,A
        mov     A,TEMP_H
        subb    A,REF_PRESSURE_H
        rlc     A       ;get the sign of difference
        mov     A,TEMP_U
        rl      A
        cpl     C
        rrc     A
;       add     A,#0x080
        movx @DPTR,A
        mov XRAM_DATA, A
        ; move ACC data to XRAM_DATA for output
        mov     DPH_XRAM,DPH
        mov     DPL_XRAM,DPL
        clr     C       ; limit use of external RAM up to DP_XRAM_MAX
; DP_XRAM_MAX ;max address to store results in external RAM
        mov     A,#low(DP_XRAM_MAX)
        subb    A,DPL_XRAM
        mov     A,#high(DP_XRAM_MAX)
        subb    A,DPH_XRAM
        jnc     MAX_XRAM_DONE
        mov     DPL_XRAM,#low(DP_XRAM_MAX)
        mov     DPH_XRAM,#high(DP_XRAM_MAX)
MAX_XRAM_DONE:

EXP_AVG_END:
        clr   flag_new_exp_ADC   ; clear flag
ret                     ; EXP_AVG_CODE:
;======
DIFF_CODE:
        jnb             flag_new_exp_ADC_done,DIFF_CODE_END
        mov   A,ADC_AVG_LONG_RESULT_L
        clr     C
        subb  A,EXP_AVG_H
        mov   DIFF_L,A
        mov     A,ADC_AVG_LONG_RESULT_H
        subb  A,EXP_AVG_U
        mov   DIFF_H,A
        jnc     DIFF_CODE_ABS
        mov     A,#0
        clr     C
        subb    A,DIFF_L
        mov     DIFF_L,A
        mov     A,#0
        subb    A,DIFF_H
        mov   DIFF_H,A
DIFF_CODE_ABS:
        clr     flag_new_exp_ADC_done
        mov     A,DIFF_L
        clr     C
        subb    A,#PRESSURE_THRESHOLD
        jc              DIFF_CODE_END
        cpl     LED
```

```
DIFF_CODE_END:
        ret
;================ scan XRAM and send to DAC ====
XRAM_to_DAC:    ; show stored data for pressure at DAC output
        push    PSW             ; called from Timer3_iterrupt:
        push    ACC
        push    DPH
        push    DPL
        mov     DPH,SCAN_XRAM_H
        mov     DPL,SCAN_XRAM_L
        inc     DPTR
        movx A,@DPTR
;data format [DAC0H,DAC0L]: [000 dddd, dddd dddd]
;       mov     DAC0L,A
;       mov     DAC0H,#0
;data format [DAC0H,DAC0L]: [dddd dddd, dddd 0000]
        mov     DAC0L,#0
        mov     DAC0H,A
        mov     SCAN_XRAM_H,DPH
        mov     SCAN_XRAM_L,DPL
        pop             DPL
        pop             DPH
        pop             ACC
        pop             PSW
        ret     ;XRAM_to_DAC; called from Timer3_iterrupt:
;=======================================
Update_Time_Out:        ; called from Timer3_iterrupt:
        push    PSW             ; called from Timer3_iterrupt:
        push    ACC
;Time_Out       equ     (0)     ;timeout duration (# of timer3 interrupts)
;Time_Out_L:    ds      1
        ; about several seconds timeout after reset or to turn-off the LED
;Time_Out_H:    ds      1
        mov             A,#1            ; increment timeout counter
        add             A,Time_Out_L
        mov             Time_Out_L,A
        mov             A,#0
        addc    A,Time_Out_H
        mov             Time_Out_H,A
        clr             C       ;check for expired timeout
        mov             A,#low(Time_Out)
        subb    A,Time_Out_L
        mov             A,#high(Time_Out)
        subb    A,Time_Out_H
        jnc             Update_Time_Out_End
        mov     Time_Out_L,#0  ;ds     1
        ; about several seconds timeout after reset or to turn-off the LED
        mov     Time_Out_H,#0  ;       ds      1
        clr     flag_doing_RESET_TimeOut
        clr     LED
Update_Time_Out_End:
        pop             ACC
        pop             PSW
        ret     ;Update_Time_Out:
;=======================================
EXTRN   CODE (Init_RF)          ; code segment in SPI_RF.asm
EXTRN   CODE (Send_RF) ; code segment in SPI_RF.asm
END
```

Config_MSU.asm

```
; This file contains the microncontroller configurations and initilizations.
; It is originally generated using the development kit from Silicon Laboratory Inc.
; Changes have been done to this file. (Adding initializations)
; ---------------------------------------------------------------
; CYGNAL Integrated Products
```

```
;
; Assembly Code Configuration Tool: F005 INITIALIZATION/CONFIGURATION CODE
;-----------------------------------------------------------------
; This file is read only. To insert the code into your
; application, simply cut and paste or use the "Save As"
; command in the file menu to save the file in your project
; directory.
;-----------------------------------------------------------------
;-----------------------------------------------------------------
; GLOBAL VARIABLES AND ASSIGNMENTS
;-----------------------------------------------------------------
PUBLIC Init_MCU

$INCLUDE(C8051F000.INC)       ; Register definition file.

Init_MCU1      segment CODE
                      rseg    Init_MCU1

;Config:
Init_MCU:
;-----------------------------------------------------------------
; Watchdog Timer Configuration
;
; WDTCN.[7:0]: WDT Control
;   Writing 0xA5 enables and reloads the WDT.
;   Writing 0xDE followed within 4 clocks by 0xAD disables the WDT
;   Writing 0xFF locks out disable feature.
;
; WDTCN.[2:0]: WDT timer interval bits
;   NOTE! When writing interval bits, bit 7 must be a 0.
;
;  Bit 2 | Bit 1 | Bit 0
;------------------------
;    1   |   1   |   1      Timeout interval = 1048576 x Tsysclk
;    1   |   1   |   0      Timeout interval =  262144 x Tsysclk
;    1   |   0   |   1      Timeout interval =   65636 x Tsysclk
;    1   |   0   |   0      Timeout interval =   16384 x Tsysclk
;    0   |   1   |   1      Timeout interval =    4096 x Tsysclk
;    0   |   1   |   0      Timeout interval =    1024 x Tsysclk
;    0   |   0   |   1      Timeout interval =     256 x Tsysclk
;    0   |   0   |   0      Timeout interval =      64 x Tsysclk
;------------------------

    mov WDTCN, #007h   ; Watchdog Timer Control Register
    mov WDTCN, #0DEh    ; Disable WDT
    mov WDTCN, #0ADh

;-----------------------------------------------------------------
; CROSSBAR REGISTER CONFIGURATION
;
; NOTE: The crossbar register should be configured before any
; of the digital peripherals are enabled. The pinout of the
; device is dependent on the crossbar configuration so caution
; must be exercised when modifying the contents of the XBR0,
; XBR1, and XBR2 registers. For detailed information on
; Crossbar Decoder Configuration, refer to Application Note
; AN001, "Configuring the Port I/O Crossbar Decoder".
;-----------------------------------------------------------------

; Configure the XBRn Registers

       mov XBR0, #002h        ; XBAR0: Initial Reset Value
       mov XBR1, #000h        ; XBAR1: Initial Reset Value
       mov XBR2, #040h        ; XBAR2: Initial Reset Value

; Select Pin I/O
```

```
; NOTE: Some peripheral I/O pins can function as either inputs or
; outputs, depending on the configuration of the peripheral. By default,
; the configuration utility will configure these I/O pins as push-pull
; outputs.
                    ; Port configuration (1 = Push Pull Output)
       mov PRT0CF, #0B5h ; Output configuration for P0
       mov PRT1CF, #000h ; Output configuration for P1
       mov PRT2CF, #000h ; Output configuration for P2
       mov PRT3CF, #000h ; Output configuration for P3

; View port pinout

              ; The current Crossbar configuration results in the
              ; following port pinout assignment:
              ; Port 0
              ; P0.0 = SPI Bus SCK      (Push-Pull Output)
              ; P0.1 = SPI Bus MISO     (Push-Pull Output)
              ; P0.2 = SPI Bus MOSI     (Push-Pull Output)
              ; P0.3 = SPI Bus NSS      (Open-Drain Output/Input)
              ; P0.4 = GP I/O           (Open-Drain Output/Input)
              ; P0.5 = GP I/O           (Push-Pull Output)
              ; P0.6 = GP I/O           (Open-Drain Output/Input)
              ; P0.7 = GP I/O           (Push-Pull Output)
              ; Port 1
              ; P1.0 = GP I/O           (Open-Drain Output/Input)
              ; P1.1 = GP I/O           (Open-Drain Output/Input)
              ; P1.2 = GP I/O           (Open-Drain Output/Input)
              ; P1.3 = GP I/O           (Open-Drain Output/Input)
              ; P1.4 = GP I/O           (Open-Drain Output/Input)
              ; P1.5 = GP I/O           (Open-Drain Output/Input)
              ; P1.6 = GP I/O           (Open-Drain Output/Input)
              ; P1.7 = GP I/O           (Open-Drain Output/Input)
              ; Port 2
              ; P2.0 = GP I/O           (Open-Drain Output/Input)
              ; P2.1 = GP I/O           (Open-Drain Output/Input)
              ; P2.2 = GP I/O           (Open-Drain Output/Input)
              ; P2.3 = GP I/O           (Open-Drain Output/Input)
              ; P2.4 = GP I/O           (Open-Drain Output/Input)
              ; P2.5 = GP I/O           (Open-Drain Output/Input)
              ; P2.6 = GP I/O           (Open-Drain Output/Input)
              ; P2.7 = GP I/O           (Open-Drain Output/Input)
              ; Port 3
              ; P3.0 = GP I/O           (Open-Drain Output/Input)
              ; P3.1 = GP I/O           (Open-Drain Output/Input)
              ; P3.2 = GP I/O           (Open-Drain Output/Input)
              ; P3.3 = GP I/O           (Open-Drain Output/Input)
              ; P3.4 = GP I/O           (Open-Drain Output/Input)
              ; P3.5 = GP I/O           (Open-Drain Output/Input)
              ; P3.6 = GP I/O           (Open-Drain Output/Input)
              ; P3.7 = GP I/O           (Open-Drain Output/Input)

;-----------------------------------------------------------------
; Comparators Register Configuration
;
; Bit 7  | Bit 6  | Bit 5  | Bit 4  | Bit 3 | Bit 2 | Bit 1 | Bit 0
;-----------------------------------------------------------------
;  R/W   |  R     | R/W    | R/W    | R/W   | R/W   | R/W   | R/W
;-----------------------------------------------------------------
; Enable | Output | Rising | Falling|  Positive     |  Negative
;        | State  | Edge   | Edge   |  Hysterisis   |  Hysterisis
;        | Flag   | Int.   | Int.   |  00: Disable  |  00: Disable
;        |        | Flag   | Flag   |  01:  5mV     |  01:  5mV
;        |        |        |        |  10: 10mV     |  10: 10mV
;        |        |        |        |  11: 20mV     |  11: 20mV
; ----------------------------------------------------------------

       mov CPT0CN, #000h        ; Comparator 0 Control Register
```

```
       mov CPT1CN, #000h        ; Comparator 1 Control Register
       ;Comp1 marker
;-----------------------------------------------------------------
; Oscillator Configuration
;-----------------------------------------------------------------
       mov OSCXCN, #030h      ; External Oscillator Control Register
       mov OSCICN, #004h      ; Internal Oscillator Control Register
;-----------------------------------------------------------------
; Reference Control Register Configuration
;-----------------------------------------------------------------
       mov REF0CN, #003h      ; Reference Control Register
;-----------------------------------------------------------------
; SPI Configuration
;-----------------------------------------------------------------
       mov SPI0CN, #002h      ; SPI Control Register
       mov SPI0CFG, #007h     ; SPI Configuration Register
       mov SPI0CKR, #000h     ; SPI Clock Rate Register
       orl SPI0CN, #001
;-----------------------------------------------------------------
; DAC Configuration
;-----------------------------------------------------------------
       mov DAC0CN, #080h      ; DAC0 Control Register
       mov DAC0L, #050h       ; DAC0 Low Byte Register
       mov DAC0H, #005h       ; DAC0 High Byte Register
       mov DAC1CN, #000h      ; DAC1 Control Register
       mov DAC1L, #000h       ; DAC1 Low Byte Register
       mov DAC1H, #000h       ; DAC1 High Byte Register
;-----------------------------------------------------------------
; UART Configuration
;-----------------------------------------------------------------
       mov SCON, #000h               ; Serial Port Control Register
       mov PCON, #000h               ; Power Control Register
;-----------------------------------------------------------------
; SMBus Configuration
;-----------------------------------------------------------------
       mov SMB0CN, #000h      ; SMBus Control Register
       mov SMB0ADR, #000h     ; SMBus Address Register
       mov SMB0CR, #000h      ; SMBus Clock Rate Register
;-----------------------------------------------------------------
; PCA Configuration
;-----------------------------------------------------------------
       mov PCA0MD, #000h    ; PCA Mode Register
       mov PCA0CN, #000h    ; PCA Control Register
       mov PCA0L, #000h     ; PCA Counter/Timer Low Byte
       mov PCA0H, #000h     ; PCA Counter/Timer High Byte
       ;Module 0
       mov PCA0CPM0, #000h    ; PCA Capture/Compare Register 0
       mov PCA0CPL0, #000h    ; PCA Counter/Timer Low Byte
       mov PCA0CPH0, #000h    ; PCA Counter/Timer High Byte
       ;Module 1
       mov PCA0CPM1, #000h    ; PCA Capture/Compare Register 1
       mov PCA0CPL1, #000h    ; PCA Counter/Timer Low Byte
       mov PCA0CPH1, #000h    ; PCA Counter/Timer High Byte
       ;Module 2
       mov PCA0CPM2, #000h    ; PCA Capture/Compare Register 2
       mov PCA0CPL2, #000h    ; PCA Counter/Timer Low Byte
       mov PCA0CPH2, #000h    ; PCA Counter/Timer High Byte
       ;Module 3
       mov PCA0CPM3, #000h    ; PCA Capture/Compare Register 3
       mov PCA0CPL3, #000h    ; PCA Counter/Timer Low Byte
       mov PCA0CPH3, #000h    ; PCA Counter/Timer High Byte
       ;Module 4
       mov PCA0CPM4, #000h    ; PCA Capture/Compare Register 4
       mov PCA0CPL4, #000h    ; PCA Counter/Timer Low Byte
       mov PCA0CPH4, #000h    ; PCA Counter/Timer High Byte
```

```
;-------------------------------------------------------------------
; ADC Configuration
;-------------------------------------------------------------------
       mov AMX0CF, #061h      ; AMUX Configuration Register
       mov AMX0SL, #000h      ; AMUX Channel Select Register
       mov ADC0CF, #064h      ; ADC Configuration Register
       mov ADC0CN, #0C0h      ; ADC Control Register
       mov ADC0LTH, #000h     ; ADC Less-Than High Byte Register
       mov ADC0LTL, #000h     ; ADC Less-Than Low Byte Register
       mov ADC0GTH, #0FFh     ; ADC Greater-Than High Byte Register
       mov ADC0GTL, #0FFh     ; ADC Greater-Than Low Byte Register
;-------------------------------------------------------------------
; Timer Configuration
;-------------------------------------------------------------------
       mov CKCON, #000h       ; Clock Control Register
       mov TH0, #000h         ; Timer 0 High Byte
       mov TL0, #000h         ; Timer 0 Low Byte
       mov TH1, #000h         ; Timer 1 High Byte
       mov TL1, #000h         ; Timer 1 Low Byte
       mov TMOD, #000h               ; Timer Mode Register
       mov TCON, #000h               ; Timer Control Register
       mov RCAP2H, #000h      ; Timer 2 Capture Register High Byte
       mov RCAP2L, #000h      ; Timer 2 Capture Register Low Byte
       mov TH2, #0EAh         ; Timer 2 High Byte
       mov TL2, #060h         ; Timer 2 Low Byte
       mov T2CON, #00Ch       ; Timer 2 Control Register
       mov TMR3RLL, #0FEh     ; Timer 3 Reload Register Low Byte
       mov TMR3RLH, #0FCh     ; Timer 3 Reload Register High Byte
       mov TMR3H, #000h       ; Timer 3 High Byte
       mov TMR3L, #000h       ; Timer 3 Low Byte
       mov TMR3CN, #006h      ; Timer 3 Control Register
;-------------------------------------------------------------------
; Reset Source Configuration
;
; Bit 7  | Bit 6  | Bit 5  | Bit 4  | Bit 3 | Bit 2 | Bit 1 | Bit 0
;-------------------------------------------------------------------
;    R   |   R/W  |  R/W   |  R/W   |   R   |   R   |  R/W  |  R
;-------------------------------------------------------------------
;  JTAG  |Convert | Comp.0 | S/W    | WDT   | Miss. | POR   | HW
; Reset  |Start   | Reset/ | Reset  | Reset | Clock | Force | Pin
; Flag   |Reset/  | Enable | Force  | Flag  | Detect| &     | Reset
;        |Enable  | Flag   | &      |       | Flag  | Flag  | Flag
;        |Flag    |        | Flag   |       |       |       |
;-------------------------------------------------------------------
; NOTE! : Comparator 0 must be enabled before it is enabled as a
; reset source.
;
; NOTE! : External CNVSTR must be enalbed through the crossbar, and
; the crossbar enabled prior to enabling CNVSTR as a reset source
;-------------------------------------------------------------------
;      mov RSTSRC, #000h      ; Reset Source Register
       mov    A,RSTSRC
       anl    A,#2
       jz     clear_XRAM_end
EXTRN  BIT(LED)
       setb   LED
       mov DPTR,#0    ;on power-on reset only
clear_XRAM:
       mov    A,#0
       movx @DPTR,A
       inc    DPTR
       mov    A,DPL   ;data pointer low byte
       jnz    clear_XRAM
       mov    A,DPH   ;data pointer high byte
       jnz    clear_XRAM
EXTRN  DATA(DPH_XRAM,DPL_XRAM)
       ; pointer in external RAM(last used byte, increment DPTR before appending data)
```

```
        mov     DPL_XRAM,#0
        mov     DPH_XRAM,#0
        mov     A,#0ffh
        mov DPTR,#0     ;on power-on reset only
        movx @DPTR,A
clear_XRAM_end:
        mov     DPH,DPH_XRAM
        mov     DPL,DPL_XRAM
        inc     DPTR
        mov     A,#0            ;set label for new sequence in XRAM
        movx @DPTR,A
        mov     DPH_XRAM,DPH
        mov     DPL_XRAM,DPL
        clr     LED
;---------------------------------------------------------------
; Interrupt Configuration
;---------------------------------------------------------------
        mov IE, #000h           ;Interrupt Enable
        mov IP, #000h           ;Interrupt Priority
        mov EIE1, #000h                 ;Extended Interrupt Enable 1
        mov EIE2, #003          ;Extended Interrupt Enable 2
        mov EIP1, #000h                 ;Extended Interrupt Priority 1
        mov EIP2, #000h                 ;Extended Interrupt Priority 2
; other initialization code here...
        ret
END
```

SPI_RF.asm

```
; SPI_RF.asm
; controls SPI and RF for pressure sensor
$ INCLUDE (c8051f000.inc)      ; Include register definition file
        include_simple_AVG      equ     'N'     ;'Y'es 'No', include subroutine AVG_ADC:
        ;LED            equ     P0.7
        ;CS             equ     P0.5 ; transmitter
        ;CE             equ     P0.4 ; transmitter
        ;NSS    equ     P0.3 ; hard to VDD, always 1
        ;MOSI   equ     P0.2 ; SPI data out
        ;MISO   equ     P0.1 ; SPI data in, not used
        ;SCK    equ     P0.0 ; SPI clock
        EXTRN   bit (LED,CS,CE,NSS,MOSI,MISO,SCK)      ; command code and identifier
        EXTRN DATA (SPI_CNT, XRAM_DATA)
        EXTRN DATA (ADC_AVG_LONG_RESULT_H, ADC_AVG_LONG_RESULT_L)
        EXTRN DATA (EXP_AVG_U, EXP_AVG_H, EXP_AVG_L, DIFF_H, DIFF_L)
        EXTRN DATA (REF_PRESSURE_H, REF_PRESSURE_L)
if include_simple_AVG = 'Y'
        EXTRN DATA (ADC_AVGH, ADC_AVGL)
endif
;-------------------------------------------------------------------------------
; CODE SEGMENT
;-------------------------------------------------------------------------------
Transceiver     segment CODE
        rseg    Transceiver     ; switch to this code segment.
        using   0               ; specify register bank for the
                                ; following program code.
PUBLIC  Init_RF
Init_RF:
        ;clr EA ;stop interrupts
        mov             R3,#5 ;5 ~ 5us delay
        djnz    R3,$    ;make delay

; Configure Laipac transceivers
        mov             R7,#3;(ADW_CRC-CHS_RD+1); LPCFGBS      ; number of bytes to send
        mov             DPTR,#ADW_CRC; LPCFG    ; set data pointer to address of first byte
        clr     CE
        nop
```

```
        nop
        setb    CS              ; set chip select line (configure)
        mov             R3,#5 ;5 ~ 5us delay
        djnz    R3,$    ;make delay
NXTLPC:
        clr             A
        movc    A,@A+DPTR       ; get next byte
        jb              TXBSY,$         ; wait for transmit to finish
        mov             SPI0DAT,A       ; send byte to SPI
        inc             DPTR            ; increment pointer to next byte
        djnz    R7,NXTLPC       ; check if more to send
        jb              TXBSY,$         ; wait for transmit to finish

        clr             CS              ; unset chip select line
        mov             R3,#5 ;48 ~ 5us delay
        djnz    R3,$    ;make delay
        ;setb EA ;enable interrupts
        ret
PUBLIC  Send_RF
Send_RF:
        clr     EA ;stop interrupts
        ;SPI_TX_LEN equ 31                      ;Length of RF message

;       Prepare message
        mov     R1,#SPI_TX
        ; SPI_TX contain address in RAM used to store SPI messages (31 bytes)
        mov     R7,#SPI_TX_LEN
        mov     @R1,#RF_ADR_H  ;equ    0E7h    ; high byte of RF receiver address
        inc     R1
        dec     R7
        mov     @R1,#RF_ADR_L  ;equ    042h    ; low byte of RF receiver address
        inc     R1
        dec     R7
        mov     @R1,#EXT_MSG   ;equ 0F0h  ; code F0 for external message in Pill2
        inc     R1
        dec     R7
; ---------------------------- Message Format -------------------------------
;       [FD 00 00 SPI_CNT(1B) LONG_AVG_RESULT(2B) EXP_RESULT(4B) 1ATM(2B) DR(2B) 00 DF]
        mov     @R1,#PRS_COD   ;equ 0FDh       ; code FD for data from pressure sensor
        inc     R1
        dec     R7
        mov     @R1,#000h
        inc     R1
        dec     R7
        mov     @R1,#000h
        inc     R1
        dec     R7
        mov     @R1,SPI_CNT     ; add a counting byte here to keep track of data transmitted
        inc     R1
        dec     R7
        mov     @R1,ADC_AVG_LONG_RESULT_H      ; High byte result from ADC_AVG_LONG
        inc     R1
        dec     R7
        mov     @R1,ADC_AVG_LONG_RESULT_L      ; Low byte result from ADC_AVG_LONG
        inc     R1
        dec     R7
        mov     @R1,#000h
        inc     R1
        dec     R7
        mov     @R1,EXP_AVG_U                  ; Upper byte result from EXP_AVG
        inc     R1
        dec     R7
        mov     @R1,EXP_AVG_H                  ; High byte result from EXP_AVG
        inc     R1
        dec     R7
        mov     @R1,EXP_AVG_L                  ; Low byte result from EXP_AVG
        inc     R1
```

```
        dec     R7
        mov     @R1,REF_PRESSURE_H          ; High byte of REF_PRESSURE
        inc     R1
        dec     R7
        mov     @R1,REF_PRESSURE_L          ; Low byte of REF_PRESSURE
        inc     R1
        dec     R7
        mov     @R1,DIFF_H                  ; High bye result from DIFF_CODE
        inc     R1
        dec     R7
        mov     @R1,DIFF_L                  ; Low bye result from DIFF_CODE
        inc     R1
        dec     R7
        mov     @R1,#000h
        inc     R1
        dec     R7
        mov     @R1,XRAM_DATA               ; XRAM_DATA
        inc     R1
        dec     R7
        mov     @R1,#000h                   ; 00 DF => END OF MESSAGE 2 BYTES
        inc     R1
        dec     R7
        mov     @R1,#0DFh
        inc     R1
        dec     R7
; ------------------------------ End ----------------------------------
Send_RF_load_data:
        mov     A,R7 ;here you have to choose data
        add     A,#0
;NOTE do not put important data in the last byte, because the last bit of this byte is
;always zero in Pill2
        mov     @R1,A
        inc     R1
        djnz    R7,Send_RF_load_data
Send_RF_load_data_done:

;Send the ready message
        ;clr EA ;stop interrupts
        clr     CS
        nop
        nop
        setb    CE      ;activate data loading into transmitter
        mov             R3,#5 ;5 ~ 5us delay
        djnz    R3,$    ;make delay
        mov     R1,#SPI_TX
        mov     R7,#SPI_TX_LEN
Send_RF_send:
        jb              TXBSY,$         ; wait for transmit to finish
        mov     SPI0DAT,@R1 ;send a byte
        inc     R1
        djnz    R7,Send_RF_send
        jb              TXBSY,$         ; wait for transmit to finish
;       mov     R3,#5 ;5 ~ 5us delay
;       djnz    R3,$    ;make delay
        clr             CE
        ; data loading into transmitter finished, send the message
        mov             R3,#5 ;5 ~ 5us delay
        djnz    R3,$    ;make delay
        setb    EA ;enable interrupts
        ret
;-------------------------------------------------------------------------------
; CONSTANTS
;-------------------------------------------------------------------------------
; Laipac configuration registers
LPCFGBS equ     15      ; Number of configuration bytes
LPCFG:
;DATA2_W:       DB      000h            ; Data width ch2, in bits
```

```
;DATA1_W:       DB      (SPIBFL*8)      ; Data width ch1, in bits
;ADDR2_4:       DB      000h            ; Ch2 address RX
;ADDR2_3:       DB      000h            ; Ch2 address RX
;ADDR2_2:       DB      000h            ; Ch2 address RX
;ADDR2_1:       DB      000h            ; Ch2 address RX
;ADDR2_0:       DB      000h            ; Ch2 address RX
;ADDR1_4:       DB      000h            ; Ch1 address RX
;ADDR1_3:       DB      000h            ; Ch1 address RX
;ADDR1_2:       DB      000h            ; Ch1 address RX
;ADDR1_1:       DB      RXADHI          ; Ch1 address RX
;ADDR1_0:       DB      RXADLO          ; Ch1 address RX
ADW_CRC:        DB      041h
; Addr width, CRC length, CRC enable;;01000001b
; Address width (b7:b2), CRC mode/enable (b1:b0)
RF_CFG:         DB      06Fh
;single receiver, shock-burst, 1Mb/s, 16MHz crystal(3bit),max power(2bit);; 01101100b
; RF configuration
CHS_RD:         DB      010h    ;chan #9 (7bits), transmit mode;; 00000000b
; Channel select, RX enable (b0)
RF_ADR_H        equ     0E7h    ; high byte of RF receiver address
RF_ADR_L        equ     042h    ; low byte of RF receiver address
EXT_MSG         equ 0F0h  ; code F0 for external message in Pill2
PRS_COD         equ 0FDh        ; code FD for data from pressure sensor
RF_TX_BUF       segment IDATA
                rseg    RF_TX_BUF
SPI_TX_LEN      equ 31                  ;Length of RF message
SPI_TX:         DS      SPI_TX_LEN
RF_AD_H         equ SPI_TX                      ; high byte of RF receiver address
RF_AD_L         equ (SPI_TX+1) ; low byte of RF receiver address
EXT_MES         equ (SPI_TX+2) ; holds code F0 for external message in Pill2
PRS_TYP         equ (SPI_TX+3) ; holds code FD for data from pressure sensor
; the rest to (SPI_TX+30))    holds data
END
```

A.2 2.4 GHz Pressure Sensor System MATLAB Scripts

Frame_Decode.m

```
% This indented code segment was originally developed by Dr. O. Marinov for
% decoding image frames from the capsule imaging device
        %function Frame_Decode
        clc;clear all;close all;
        %Reads binalry files with extension *.frm
        %-------------------------------------------------------------
        Matlab_program='Frame_Decode.m';Log_file_name=[Matlab_program '.log'];
        %-------------------------------------------------------------
        % Choose folder and files with data with frames and options for frame show
        frame_directory='Decoded_frames\'; %Directory with frames found in data from Files
        from RF receiver
        file_name_prefix='Default';     % Files created with RF_to_USB_Decode.m
        file_name_prefix='zzz';     % Files created with RF_to_USB_Decode.m
        frame_ext='frm'; %Extension for files with frames
        max_num_frames=999999; %Do not comment this line
        max_num_frames=1000;  %that many frames to be analyzed
        max_num_pictures=2; %that many pictures to be shown
        max_num_pictures=min([max_num_pictures max_num_frames]);
        % The frame files *.frm contain separated single frames with messages from
        % the from imager. The format of the *.frm files is
        % One frame delimiter record, Data records (32 bytes each), and second
        % delimiter record, e.g.
        % FF FF SYSTIME FF FF FF ... FF -> begin of frame (32) bytes
        % FF FF SYSTIME xx xx xx ... xx -> 1st record in the frame with data xx...
        % FF FF SYSTIME xx xx xx ... xx -> 2nd record in the frame with data xx...
        % ...  ...  ...  ... ...  ... -> next records in the frame with data xx...
        % FF FF SYSTIME xx xx xx ... xx ->last record in the frame with data xx...
        % FF FF SYSTIME FF FF FF ... FF -> end of frame (32) bytes
        % Note that several messages (about 10% in a frame) have not been received
        % by the RF receiver, and they are missing in the frame data
        %
        % The format of the recors is:
        % FF FF SYSTIME xx xx xx ... xx, where xx are 29 bytes received from RF link
        %-------------------------------------------------------------
        max_records_in_frame=1+1+1+480+1+1;
        %FF_frame_delim+sync_mess+ext_mess+480_HDs+5th_FIFO+FF_frame_delim
        Sys_Time_step=0.1; % 100 ms from receiver MCU
        pix_in_row=160; %Number of pixels in one imager row %=160 for a QQVGA horizontal
        line
        first_img_byte_num_VHDL=0;% Starting number of counting bytes in VHDL code of
        image compression
        %YUV colors for picture background
                YUV_background_color=[192,128,128]; % grey--> Y=192, U=128, V=128
        %         YUV_background_color=[ 16,128,128]; %black--> Y= 16, U=128, V=128
        %         YUV_background_color=[235,128,128]; %white--> Y=235, U=128, V=128
        %         YUV_background_color=[145, 54, 34]; %green--> Y=145, U= 54, V= 34
        %         YUV_background_color=[ 41,240,110]; % blue--> Y= 41, U=240, V=110
        Ymin_max=[16, 235]; %limits for Y-code
        Umin_max=[16, 240]; %limits for U-code
        Vmin_max=[16, 240]; %limits for V-code
        %-------------------------------------------------------------
        log_fid=fopen(Log_file_name,'a'); %log file
        log_message=sprintf('\n====     Run     Matlab     program     "%s"     on     %s
        ====',Matlab_program,datestr(now));
        disp(log_message); fprintf(log_fid,'\n%s',log_message);
        fclose(log_fid); clear log_fid;
        %-------------------------------------------------------------

        % get filenames of required type with single-frame data
```

```
data_files_type=[frame_directory, file_name_prefix, '*.',frame_ext];
disp(['==== Read frame data from files = dir(' data_files_type ') ===='])
files = dir(data_files_type);
[num_files dummy]=size(files); clear dummy;
if num_files<1
    disp(sprintf('No file of type "%s" is found',data_files_type))
    disp(sprintf('====    Matlab    program    "%s"    is    canceled    with    error
====',Matlab_program))
    break
end

file_names=[];for i=1:num_files;file_names=[file_names;files(i).name];end
file_names=sortrows(file_names); clear files

num_frames_to_process=min([num_files max_num_frames]);

All_Frame_file_data=[]; % Each frame is in a column
All_num_records=[]; % Row vector with number of 32-byte records in each frame
for i=1:num_frames_to_process %read frame files
    A=fclose('all');
    fid_s=fopen([frame_directory file_names(i,:)]);
    disp(sprintf('--> Reading file %s',file_names(i,:)))
    A = uint8(fread(fid_s));
    A_num_records=length(A)/32;
    if A_num_records>max_records_in_frame
        A_num_records
        disp(sprintf(...
'Warning: --> Frame #%g from file "%s" contains more than %g records,\n   ==>
Excess records skipped.',...
            i,file_names(i,:),max_records_in_frame))
        log_fid=fopen(Log_file_name,'a'); %log file
        fprintf(log_fid,...
'\nWarning: --> Frame #%g from file "%s" contains more than %g records,\n   ==>
Excess records skipped.',...
            i,file_names(i,:),max_records_in_frame);
        fclose(log_fid); clear log_fid;
    end
    Frame_file_data=uint8(ones(max_records_in_frame*32,1)*255);
    Frame_file_data=[A;Frame_file_data];
    Frame_file_data=Frame_file_data(1:max_records_in_frame*32);
    All_Frame_file_data=[All_Frame_file_data,Frame_file_data];
    A_num_records=min([A_num_records max_records_in_frame]);
    All_num_records=[All_num_records,A_num_records];
end %for i=1:num_frames_to_process %read frame files
clear A_num_records Frame_file_data fid_s
A=fclose('all');clear A

%==== Here, each frame is in a column in ==============================
All_Frame_file_data; % Each frame data is in a column ======
All_num_records; % Row vector with number of 32-byte records in each frame
%=======================================================================
disp('Check for frame delimiters in the header of the frames')
%                All_Frame_file_data(1,4)=3; %Add error
%                All_Frame_file_data(1,3)=3; %Add error
%                All_Frame_file_data(2,2)=3; %Add error
%                All_Frame_file_data(2,3)=3; %Add error
%                All_Frame_file_data(4,1)=3; %Add error
            A=find(255~=All_Frame_file_data(1,:));
            A=[A,find(255~=All_Frame_file_data(2,:))];
            for i=4:32
                A=[A,find(255~=All_Frame_file_data(i,:))];
            end
            if ~isempty(A) % Bad header found
                Amax=max(A);A=sort(A);
                for i=1:length(A)
                    Arep=find(A(i)==A);
                    if length(Arep)>1
```

```
                        Arep=Arep(2:length(Arep));
                        A(Arep)=Amax+1;
                    end
                end
                A=sort(A);Arep=find((Amax+1)==A);
                if ~isempty(Arep)
                    A=A(1:Arep(1)-1);
                end
                clear Arep;

                log_fid=fopen(Log_file_name,'a'); %log file
                for i=A
                    log_message=sprintf(...
'Error: --> Frame #%g from file "%s" does not have heading frame delimiter "FF FF
xx FF FF...',...
                        i,file_names(i,:));
                    disp(log_message); fprintf(log_fid,'\n%s',log_message);
                end
                log_message=sprintf('==== Matlab program "%s" is canceled with
error on %s ====',...
                    Matlab_program,datestr(now));
                disp(log_message); fprintf(log_fid,'\n%s',log_message);
                fclose(log_fid); clear log_fid;
                break
            end %if ~isempty(A) % Bad header found
disp('end of Check for frame delimiters in the header of the frames')

disp('Check for 32-byte record delimiters in the frames')
%                 All_Frame_file_data(1,4)=3; %Add error
%                 All_Frame_file_data(34,2)=3; %Add error
%                 All_Frame_file_data(2,2)=3; %Add error
%                 All_Frame_file_data(34,5)=3; %Add error
%                 All_Frame_file_data(33,5)=3; %Add error
%                 All_Frame_file_data(34,1)=3; %Add error
            A=[];
            for i=1:32:max_records_in_frame*32
                A=[A,find(255~=All_Frame_file_data(i,:))];
                A=[A,find(255~=All_Frame_file_data(i+1,:))];
            end % for i=1:32:max_records_in_frame*32
            if ~isempty(A) % Bad record delimiter found
                Amax=max(A);A=sort(A);
                for i=1:length(A)
                    Arep=find(A(i)==A);
                    if length(Arep)>1
                        Arep=Arep(2:length(Arep));
                        A(Arep)=Amax+1;
                    end
                end
                A=sort(A);Arep=find((Amax+1)==A);
                if ~isempty(Arep)
                    A=A(1:Arep(1)-1);
                end
                clear Arep;

                log_fid=fopen(Log_file_name,'a'); %log file
                for i=A
                    log_message=sprintf(...
'Error: --> Frame #%g from file "%s" does not have one or more 32-byte record
delimiter "FF FF xx xx...',...
                                        i,file_names(i,:));
                    disp(log_message); fprintf(log_fid,'\n%s',log_message);
                end
                log_message=sprintf('==== Matlab program "%s" is canceled with
error on %s ====',...
                    Matlab_program,datestr(now));
                disp(log_message); fprintf(log_fid,'\n%s',log_message);
                fclose(log_fid); clear log_fid;
```

```
                    break
                end %if ~isempty(A) % Bad record delimiter found
        disp('end of Check for 32-byte record delimiters in the frames')

        %==== Here, each frame is in a column in ==============================
        All_Frame_file_data; % Each frame data is in a column ======
        All_num_records; % Row vector with number of 32-byte records in each frame
        %=======================================================================
```

```
%The following code segment was added to extracted pressure data from image messages
%=======================================================================
% Taking Pressure DATA out from All_Frame_file_data
%=======================================================================
affd_size = size(All_Frame_file_data);
if ((All_Frame_file_data(69,1)==253) & (All_Frame_file_data(86,1)==223))
    Pressure_Data_0all = double(All_Frame_file_data(69:86,1));
else
    Pressure_Data_0all = double([253, zeros(16,1), 223]);
end
for pscnt = 2:affd_size(2)
    if ((All_Frame_file_data(69,pscnt)==253) & (All_Frame_file_data(86,pscnt)==223))
    Pressure_Data_0all   = double([Pressure_Data_0all, All_Frame_file_data(69:86,pscnt)]);
    end
end
Pressure_Data_1begin     = double(Pressure_Data_0all(1:2,1:length(Pressure_Data_0all)));
Pressure_Data_2SPI_cnt   = double(Pressure_Data_0all(3:4,1:length(Pressure_Data_0all)));
Pressure_Data_3LONG_AVG  = double(Pressure_Data_0all(5:6,1:length(Pressure_Data_0all)));
Pressure_Data_4EXP       = double(Pressure_Data_0all(7:10,1:length(Pressure_Data_0all)));
Pressure_Data_51ATM      = double(Pressure_Data_0all(11:12,1:length(Pressure_Data_0all)));
Pressure_Data_6DIFF      = double(Pressure_Data_0all(13:14,1:length(Pressure_Data_0all)));
Pressure_Data_7XRAM      = double(Pressure_Data_0all(15:16,1:length(Pressure_Data_0all)));
Pressure_Data_8end       = double(Pressure_Data_0all(17:18,1:length(Pressure_Data_0all)));
break
%=======================================================================
```

pressure_plots.m

```
% remove wrongful data in Pressure_Data_0all before running this script
scale_code_press=1.0;%
pd_size = size(Pressure_Data_0all);
refatm = 256*Pressure_Data_0all(11,2)+Pressure_Data_0all(12,2);

PDplot=256*Pressure_Data_3LONG_AVG(1,1:pd_size(2))+...
Pressure_Data_3LONG_AVG(2,1:pd_size(2));
time = 0.4*[1:pd_size(2)];
yy=PDplot-refatm;
yyy=yy*scale_code_press; %Long Avg Pressure in cm H2O
PDexp=256*Pressure_Data_4EXP(2,1:pd_size(2)) + Pressure_Data_4EXP(3,1:pd_size(2));
yexp=PDexp-refatm;
yyexp=yexp*scale_code_press; %Exp Avg Pressure in cm H2O
led=Pressure_Data_6DIFF(2,1:pd_size(2)); %Long - Exp Avg Pressure in cm H2O
threshold=4*ones(1,pd_size(2));

diff2 = (Pressure_Data_7XRAM(2,1:pd_size(2))-128)*2; %XRAM

subplot(221);
plot(time,yyy,'.');title('Long Averaging Output') ;
xlabel('Time(second)') ;ylabel('cm H_2O');
axis([0 time(pd_size(2)) min(yyy) max(yyy)]); grid on;

subplot(222);plot(time,yyexp,'.');title('Exponential Averaging Output')
;xlabel('Time(second)');ylabel('cm H_2O');
axis([0 time(pd_size(2)) min(yyexp) max(yyexp)]); grid on;
```

```
subplot(224);plot(time,led);title('LED Status');
xlabel('Time(second)');ylabel('Difference value');hold on;
subplot(224);plot(time,threshold,'r-'); axis([0 time(pd_size(2)) 0 75]); hold off; grid on;

subplot(223);plot(time,diff2,'r.');
title('Difference between reference pressure and current pressure');
xlabel('Time(second)'); ylabel('Differece(cmH_2O)');
axis([0 time(pd_size(2)) min(diff2) max(diff2)]); grid on;

Long_Exp_XRAM_diff=[];
Long_Exp_XRAM_diff=[Long_Exp_XRAM_diff time']; %time of reading
Long_Exp_XRAM_diff=[Long_Exp_XRAM_diff yyy']; %Long Avg Pressure in cm H2O
Long_Exp_XRAM_diff=[Long_Exp_XRAM_diff yyexp']; %Exp Avg Pressure in cm H2O
Long_Exp_XRAM_diff=[Long_Exp_XRAM_diff diff2'*scale_code_press/2]; %XRAM
Long_Exp_XRAM_diff=[Long_Exp_XRAM_diff led'*scale_code_press];
%Long - Exp Avg Pressure in cm H2O

xlswrite('Pressure_Data.xls', Long_Exp_XRAM_diff); % writes matrix Long_Exp_XRAM_diff to
the Excel file Pressure_Data.
```

Appendix B: 125 kHz System Firmwares and GUI Software.

B.1 125 kHz Dual Sensor System Transponder Firmware

RFID_Transponder_v2.asm

```
; RFID Wireless Sensing System For In-Vivo Applications
; Transponder Assembly Code
; Developed by Kurt Huang
;
; This firmware system is consisted of 4 files,
; 1. RFID_Transponder_v2.asm,        ; Main structure of the system
; 2. RFID_init.asm                   ; Initilizations of the system
; 3. RFID_ADC.asm                    ; ADC sample routines
; 4. RFID_MOD.asm                    ; Data transmission - controlling MOD pin on U3280M

.include "tn24def.inc"
.equ NRST = 8
.equ SRamHEAD          =       $0060
.equ SRamEND           =       $00DF

.def PCount                    =       r18
.def DataCount         =       r19
.def Pressure0         =       r20
.def PressureH         =       r21
.def PressureL         =       r22
.def TemperatureH      =       r23
.def TemperatureL      =       r24
.def CheckSum          =       r25

.org 0x0000
       rjmp reset
.org 0x000D
```

```
ADC_Complete:
reti

reset:
        ldi r16, low(RAMEND)
        out SPL, r16

main:
        clr DataCount
        rcall Initialize
main_loop:
                                        ;rcall BitToggle
        clr CheckSum
        rcall delay_long
        ;sbi PORTA, PA4
        rcall ADC_pressure
        rcall ADC_temperature
        rcall ADC_vcc
        ;cbi PORTA, PA4
        inc DataCount
        ;in r16, SREG
        ;push r16
        rcall RFID_MOD_send_data
        ;pop r16
        ;out SREG, r16
        ;brcs main
        rjmp main_loop
BitToggle:
        sbis PORTA, PA5
        sbi PORTA, PA5
        sbic PORTA, PA5
        cbi PORTA, PA5
ret
.include "RFID_init.asm"
.include "RFID_ADC.asm"
.include "RFID_MOD.asm"

.DSEG
TvsVCC_low:             .BYTE 1                 ; ADC of temperature sensor VS 2.7V single-
ended
TvsVCC_high:    .BYTE 1                 ; ADC of temperature sensor VS 2.7V single-ended
```

RFID_init.asm

```
; ATTiny24 Initializations
Initialize:
        sei     ; global interrupt enable

        ; MCU Control Register Setup (p38 & 53)
        ldi r16, 0b00001011             ; sleep mode = ADC Noise Reduction
        out MCUCR, r16
        ldi r16, 0b00001010             ; shut down TIMER1 and USI
        out PRR, r16

        ; PORT A setup DDRA[7..0] = DDRA[N/C OUT N/C OUT IN IN IN IN]
        ldi r16, 0b01110000
        out DDRA, r16
        ldi r16, 0b01010000
        out PORTA, r16



        ; PORT B setup DDRB[X X X X PB3..0] = DDRB[X X X X IN IN OUT OUT]
        ldi r16, 0b00000011
        out DDRB, r16
```

```
        ldi r16, 0b00000000
        out PORTB, r16


        ; Timer0 Setup (p84, 88 & 89)
        ;ldi r16, 31
        ; compare match frequency = 125KHz/32 = ~3.91KHz
        ;out OCR0A, r16
        ; compare match frequency = 125KHz/16 = ~7.82KHz,
        ; USICLK_frequency = CM_Frequecy / 2.
        ; if r16 = 31, then USI frequency     = ~1.95 KHz
        ; if r16 = 15, then USI frequency     = ~3.90 KHz
        ;ldi r16, 0b00000010          ; WGM02:00 = 0b010 -> CTC mode
        ;out TCCR0A, r16
        ;ldi r16, 0b00000010
        ; CS02:00 = 0b111 -> external clock rising-edge trigger on T0
        ;out TCCR0B, r16
        ; CS02:00 = 0b010 -> 1M / 8 = 125KHz
        ; CS02:00 = 0b000 -> timer stopped
        ;ldi r16, 0b00000010          ; enable Compare Match A Interrupt
        ;out TIMSK0, r16
        ; ADC Control and Status Register A
        ;ldi r16, 0b00001110
        ; disable ADC and select frequeny division of 64. 1MHz/64 = 15KHz
        ldi r16, 0b00001100
        ; disable ADC and select frequeny division of 64. 1MHz/16 = 62.5KHz
        out ADCSRA, r16
        cbi ADCSRB, 4
        ; ADC result is right adjusted = 000000XX RESULTXX <- {ADCH, ADCL}
        ;sbi ADCSRB, 4
        ; ADC result is left adjusted  = XXRESULT XX000000 <- {ADCH, ADCL}

        ;ldi r16, 0b00001111
        ;out DIDR0, r16               ; disable digital input buffer on PA0-3 pins

ret
```

RFID_ADC.asm

```
; ADC routines

; Obtain ADC value for Pressure Sensor
ADC_pressure:
                                        ;rcall BitToggle
                                        ;rcall BitToggle

        ldi r16, 0b10001001
        ; REF1..0 = AREF control, [10] selects internal 1.1V reference voltage
        ; MUX5..0 = channel selection,
        ; 001001 - Differential ADC0+/ADC1- with 20X gain
        out ADMUX, r16          ; Enable ADC Complete Interrupt

        sbi ADCSRA, ADEN
        sbi ADCSRA, ADSC

        ADC_pressure_loop:
                in r17, ADCSRA
                sbrs r17, ADIF                  ; check for ADC interrupt Flag
                rjmp ADC_pressure_loop

                mov r16, PressureL
                mov r17, PressureH
                clc
                ror r17
                ror r16                         ; divide by 2
```

```
                clc
                ror r17
                ror r16                         ; divide by another 2

                in r30, ADCL
                sub r30, r16            ; difference between low byte ADC- Pressure/4
                in r16, ADCH
                sbc r16, r17            ; difference between high byte ADC- Pressure/4
                add PressureL, r30      ; add low byte of the difference
                adc PressureH, r16      ; add high byte of the difference

        cbi ADCSRA, ADEN        ; disable ADC
                                                ;rcall BitToggle
ret

; Obtain ADC value for Temperature Sensor
ADC_temperature:
                                                ;rcall BitToggle
        ;ldi r16, 0b00000011   ; REF1..0 = AREF control,
        ldi r16, 0b10010000     ; REF1..0 = AREF control,
        ;       [10] selects internal 1.1V reference voltage
        ;       [00] 2.7V Voltage Supply as reference
        ; MUX5..0 = channel selection,
        ; 010000 - Differential ADC2+/ADC3- with 1X gain
        ; 000011 - Single Channel ADC3
        out ADMUX, r16

        sbi ADCSRA, ADEN

        sbi ADCSRA, ADSC        ; sets ADSC - ADC Start Conversion bit
        ADC_temperature_loop:
                in r17, ADCSRA
                sbrs r17, 4
                rjmp ADC_temperature_loop
                in TemperatureL, ADCL
                in TemperatureH, ADCH

        cbi ADCSRA, ADEN        ; disable ADC
                                                ;rcall BitToggle
ret

ADC_vcc:
;TvsVCC_low:    .BYTE 1                 ; ADC of temperature sensor VS 2.7V single-ended
;TvsVCC_high:   .BYTE 1                 ; ADC of temperature sensor VS 2.7V single-ended

        ldi r16, 0b00000011     ; REF1..0 = AREF control,
        ;ldi r16, 0b10010000    ; REF1..0 = AREF control,
                                ;       [10] selects internal 1.1V reference voltage
                                ;       [00] 2.7V Voltage Supply as reference
                                ; MUX5..0 = channel selection,
                                ; 010000 - Differential ADC2+/ADC3- with 1X gain
                                ; 000011 - Single Channel ADC3(output of temperature sensor)
        out ADMUX, r16
        sbi ADCSRA, ADEN
        sbi ADCSRA, ADSC
ADC_vcc_loop:
                in r17, ADCSRA
                sbrs r17, 4
                rjmp ADC_vcc_loop
                in r16, ADCL
                sts TvsVCC_low, r16
                in r16, ADCH
                sts TvsVCC_high, r16
        cbi ADCSRA, ADEN
ret
```

RFID_MOD.asm

```
; Transmitting each bit by controlling the MOD pin on U3280M Transponder Chip

; Assumptions: Data to be transferred is stored in register R16, before calling the
routine

RFID_MOD_Start:
; RFID_MOD_Start adds a prefix start bit & a synchronize bit to the data stream
        rcall delay_2  ;     1
        cbi PORTA, PA6     ; MOD   |
        rcall delay_2   ;     0   --
        sbi PORTA, PA6
        rcall delay_2   ;     1      --    --
        cbi PORTA, PA6      ; MOD   |  |  |  |
        rcall delay_2   ;     0   --    --
        sbi PORTA, PA6
RFID_MOD_Send:
        clr PCount
RFID_MOD_loop:
        sbrc r16, 7
        rjmp RFID_MOD_Bit1
RFID_MOD_Bit0:
        sbi PORTA, PA6
        rcall delay_1
        cbi PORTA, PA6
        rcall delay_3
        rjmp RFID_MOD_End
RFID_MOD_Bit1:
        sbi PORTA, PA6
        rcall delay_3
        cbi PORTA, PA6
        rcall delay_1
        rjmp RFID_MOD_End
RFID_MOD_End:
        lsl r16
        inc PCount
        cpi PCount, 8
        brne RFID_MOD_loop
RFID_MOD_loop_end:
        ret

; 1st preamble [00] byte + 2nd prefix [FF] Byte
;     1 --    --    --    --    --    --    --    --    --       --    --    --    --    -
-    --    --    -
; MOD     |S |  |0 |  |0 |  |0 |  |0 |  |0 |  |0 |  |0 |  |0   1|  | 1|  | 1|  | 1|  | 1|
| 1|  | 1|  | 1| ...
;     0    --    --    --    --    --    --    --    --    -- --    --    --    --    --
--    --    --
; 3rd prefix [FF] Byte + 4th ID [C3] Byte
;      1      -    --    --    --    --    --    --    --    --    --    -- --    --
--    -- --    --
; MOD    ... | 1|  | 1|  | 1|  | 1|  | 1|  | 1|  | 1|  | 1|  | 1|  | 1|     |0 |  |0    1|
| 1|    0|  | 0|
;     0       --    --    --    --    --    --    --    --    --    --       --    -- --
--       --    --
; 4th DataCount Byte + sensor outputs, etc

; MESSEAGE LENGTH = 12 Bytes
;                          MESSEAGE                          FORMAT                          =
[000000000][11111111][11111111][11000011][databyte][databyte][databyte][databyte][datacoun
t][11000011][11111111][11111111][STOP]
;                  ^Preamble^ ^-- Sync. Byte --^  ^- ID -^  ^- PressureDATA -^  ^-- Temp.
Data --^ ^DataCount^ ^- ID -^

RFID_MOD_send_data:
```

```
;TvsVCC_low:          .BYTE 1        ; ADC of temperature sensor VS 2.7V single-ended
;TvsVCC_high:  .BYTE 1               ; ADC of temperature sensor VS 2.7V single-ended
       lds r28, TvsVCC_low
       lds r29, TvsVCC_high

       ldi r16, 6
TvsVCC_loop1:
       clc
       rol r28
       rol r29
       dec r16
       brne TvsVCC_loop1            ; making TvsVCC (high/low) bytes left adjusted

       mov r16, r29
       andi r16, $0F
       or r28, r16

       swap r28

       mov r30, PressureH                          ; High Byte of Pressure Data
       mov r31, PressureL                          ; Low Byte of Pressure Data
       clc
       ror r30
       ror r31
       clc
       ror r30
       ror r31

       or r30, r28                  ; 6 LSB from TvsVCC become 6 MSB of PressureH

       mov r28, TemperatureH
       andi r29, $F0
       or r28, r29                  ; 4 MSB from TvsVCC become 4 MSB of TemperatureH
       mov TemperatureH, r28

       ldi r16, 0x00                ; Preamble Byte [00000000]
       rcall RFID_MOD_Start

       ldi r16, 0xC3          ; load USI Data Register with 1100 0011 for Identification
       rcall RFID_MOD_Send

    ldi CheckSum, 0                 ; clear CheckSum

       mov r16, DataCount           ; <-- Data Counter for tracking data order.
       rcall RFID_MOD_Send
       add CheckSum, DataCount

       mov r16, r30
       rcall RFID_MOD_Send
       add CheckSum, r30;, PressureH

       mov r16, r31                 ;, PressureL   ; Low Byte of Pressure Data
       rcall RFID_MOD_Send
       add CheckSum, r31            ;, PressureL

       mov r16, TemperatureH        ; High byte of Temperature Data
       rcall RFID_MOD_Send
       add CheckSum, TemperatureH

       mov r16, TemperatureL        ; Low byte of Temperature Data
       rcall RFID_MOD_Send
       add CheckSum, TemperatureL

       mov r16, CheckSum            ; CheckSum Byte
       rcall RFID_MOD_Send

       rcall delay_16
```

```
        sbi PORTA, PA6
        ret

; ============================
; Delay Routines
; ============================
.equ t = 40                     ; delay loop 36 times is about 125us
; 1t DELAY
delay_1:
        ldi r17, 1*t
loop_delay_1:
        dec r17
        brne loop_delay_1
ret
; 2t DELAY
delay_2:
        ldi r17, 2*t
loop_delay_2:
        dec r17
        brne loop_delay_2
ret
; 3t DELAY
delay_3:
        ldi r17, 3*t
loop_delay_3:
        dec r17
        brne loop_delay_3
ret
; 6t COMPARE MATCH INTERRUPT DELAY
delay_16:
        ldi r17, 6*t
loop_delay_16:
        dec r17
        brne loop_delay_16
ret
; VERY LONG DELAY - 50t
delay_long:
        ldi r16, 50
        ldi r17, t
loop_delay_long:
        dec r17
        brne loop_delay_long
        dec r16
        brne loop_delay_long
ret
```

B.2 125 kHz Dual Sensor System Reader Firmware

RFID_BaseStation_v5.asm

```
; RFID Wireless Sensing System For In-Vivo Applications
; Base Station Assembly Code
; ooo~Version 5~ooo
; Developed by Kurt Huang
;
; This firmware uses Timer2 to send data to a PC periodically, Timer1 to recover clock,...
; and Timer0 to trigger timeout reset.
; External interrupt, EXT_INT0 is used to received data from the U2270B reader chip.
; Validity of received data from wireless link is done by comparing ID and checksum bytes
; UART function of the microcontroller is used to communicate with a PC.

; === INCLUDES ===
.include "m8def.inc"

; === REGISTER DEFINITION ===
.def CheckSum = r18    ; Counter for messages sent to PC
.def DBuffer2  =r19    ; Data Buffer 2
.def ByteCount =r20    ; used to count number of edges (external interrupts)
.def DataByte  =r21    ; Data Byte to send
.def TimeL     =r22    ; low byte of Time - used in Clock_Recover routine
.def TimeH     =r23    ; high byte of Time

.org 0x00
       rjmp RESET_V
.org 0x01
       rjmp EXT_INT0
.org 0x04
       rjmp TIM2_OVF
.org 0x06
       rjmp TIM1_COMPA
.org 0x09
       rjmp TIM0_OVF
.org 0x1F

TIM2_OVF:      ; timer 2 overflow interrupt
                                          ;rcall BitToggle
                                          ;rcall BitToggle
       push r16
       in r16, SREG
       push r16
       nop
       nop
       lds r16, cnt_communic
       inc r16
       cpi r16, cnt_communic_max
       brne TIM2_OVF1
;.equ stat1_snd_PC = 0               ; bit in stat1; =1 sending data to PC
;stat1:              .BYTE 1               ; staus byte with flags
;cnt_communic: .BYTE 1               ; counter of number of timer2 overflow interrupts
between sending data
       lds r16, stat1
       sbr r16, (1<<stat1_snd_PC)
       sts stat1, r16
                                          ;rcall BitToggle
                                          ;rcall BitToggle
       clr r16
TIM2_OVF1:
       sts cnt_communic, r16
       nop
       nop
```

```
        pop r16
        out SREG, r16
        pop r16
        nop
        nop

reti

TIM0_OVF:
        push r16
                                                ;rcall BitToggle
                                                ;rcall BitToggle
        rcall INT0_RESET
        pop r16
reti

TIM1_COMPA:
        push r16
                                                ;rcall BitToggle
                                                ;rcall BitToggle
        rcall Decode_Start

        lds r16, BitNum
        inc r16
        sts BitNum, r16
        cpi r16, MaxBitNum
        brne TIM1_COMPA1
        rcall INT0_RESET

TIM1_COMPA1:

        in r16, TCCR1B
        cbr r16, (1<<WGM12)                     ; turn off Compare Match Mode
        out TCCR1B, r16

        in r16, TIMSK
        cbr r16, (1<<OCIE1A)                    ; disable Compare Match A Interrupt
        out TIMSK, r16

        in r16, TIFR
        sbr r16, (1<<OCF1A)                     ; clear Compare Match A Flag
        out TIFR, r16
        pop r16
reti

EXT_INT0:
                                                ;rcall BitToggle
                                                ;rcall BitToggle
        in r24, TCNT1L ; low byte
        in r25, TCNT1H ; high byte

        push r16
        clr r16
        out TCNT1H, r16
        out TCNT1L, r16
        out TCNT0, r16 ; reset no communication timeout

        lds r16, EdgeNum
        inc r16
        sts EdgeNum, r16

        cpi r16, 1
        brne First_Edge_NOT
        rcall Reset_Timer1
        pop r16
        reti
```

```
First_Edge_NOT:
                                                ;rcall BitToggle
                                                ;rcall BitToggle
        cpi r16, 10
        brsh Tenth_Edge_Up
        rcall Clock_Recover
        pop r16
        reti

Tenth_Edge_Up:
        rcall CTCA1_ON
        pop r16
        reti

Reset_Timer1:
        clr r16
        out TCNT1H, r16
        out TCNT1L, r16
ret

Clock_Recover:
        cpi r25, PW_short
        brlo Sync_error

        cpi r25, PW_long
        brsh Sync_error

        clc
        ror r25
        ror r24

        clc
        ror r25
        ror r24

        clc
        ror r25
        ror r24

        clc
        ror r25
        ror r24

        add TimeL, r24
        adc TimeH, r25

ret

Sync_error:
                                                ;rcall BitToggle
                                                ;rcall BitToggle
        rcall INT0_RESET
reti

CTCA1_ON:
        out OCR1AH, TimeH       ; high byte
        out OCR1AL, TimeL       ;

        in r16, TIFR
        sbr r16, (1<<OCF1A)                     ; clear Compare Match A Flag
        out TIFR, r16

        in r16, TCCR1B
        sbr r16, (1<<WGM12)                     ; turn on Compare Match Mode
        out TCCR1B, r16

        in r16, TIMSK
```

```
        sbr r16, (1<<OCIE1A)                    ; enable Compare Match A Interrupt
        out TIMSK, r16

ret

RESET_V:
        ldi     r16, high(RAMEND)
        out SPH, r16
        ldi r16, low(RAMEND)
        out SPL, r16
main:
        rcall INT0_RESET
        rcall Initialize

main_loop:
;.equ stat1_snd_PC = 0                  ; bit in stat1; =1 sending data to PC
;stat1:                .BYTE 1                  ; staus byte with flags
        ;cli
        lds r16, stat1
        sbrc r16, stat1_msg_in
        rcall Store_msgbuffer

        lds r16, stat1
        sbrc r16, stat1_msg_check
        rcall Chk_msg

        lds r16, stat1
        sbrc r16, stat1_msg_valid
        rcall update_rsbuffer

        lds r16, stat1
        sbrc r16, stat1_snd_PC      ;skip if tim2in did not set the flag to send data to PC
        rcall SND_PC                ;sends data to PC on timer2
        ;sei
        rjmp main_loop

Decode_Start:
        push r16
        in r16, PIND                ; read PORTD value to r16, PD2 is OUTPUT pin of U2270B
        sec                         ; set carry
        sbrs r16, DATAPIN
        clc
        rol DBuffer2
        brcc Decode_Start1
        rcall Save_A_Byte
        ldi DBuffer2, 1
Decode_Start1:
        pop r16
        ret

Save_A_Byte:
        inc ByteCount

        cpi ByteCount, 1        ; Check if first byte is ID byte
        brne Save_A_Byte1
        cpi DBuffer2, $C3
        brne Sync_Error                 ; if not ID byte, reset INT0

Save_A_Byte1:
        st X+, DBuffer2
        ldi DBuffer2, 1
        cpi ByteCount, MaxByteNum
        brne Save_A_Byte2

;.equ stat1_snd_PC = 0                  ; bit in stat1; =1 sending data to PC
;.equ stat1_msg_in = 1                  ; bit in stat1; =1 msg received
;.equ stat1_msg_valid = 2               ; bit in stat1; =1 update msg to send
```

```
;stat1:                .BYTE 1                 ; staus byte with flags
;Msg_Buffer:           .BYTE 8                 ; array for received message
        lds r16, stat1
        sbr r16, (1<<stat1_msg_in)
        sts stat1, r16
        ldi XL, low(SRAMHEAD)
        ldi XH, high(SRAMHEAD)
                                                ;rcall BitToggle
                                                ;rcall BitToggle
        ;rcall Send_Data ;moved to tim2int -> SND_PC:     ;sends data to PC on timer2
Save_A_Byte2:
ret

Store_msgbuffer:
        cli
                                                ;rcall BitToggle
                                                ;rcall BitToggle
        ldi YL, low(SRAMHEAD)
        ldi YH, high(SRAMHEAD)
        ldi ZL, low(Msg_Buffer)
        ldi ZH, high(Msg_Buffer)
        ldi r16, MaxByteNum
        sts byte_rs_cnt, r16

Store_msgbuffer_loop:
        ld r17, Y+
        st Z+, r17
        dec r16
        sts byte_rs_cnt, r16
        brne Store_msgbuffer_loop
        lds r17, stat1
        cbr r17, (1<<stat1_msg_in)
        sbr r17, (1<<stat1_msg_check)
        sts stat1, r17
        sei
        ret

Chk_msg:
                                                ;rcall BitToggle
                                                ;rcall BitToggle
        ldi YL, low(Msg_Buffer)
        ldi YH, high(Msg_Buffer)
        ld r17, Y+
        cpi r17, $C3
        brne Chk_msg_end
        ldi r16, (MaxByteNum-2)
        sts byte_rs_cnt, r16
        clr CheckSum
Chk_msg_loop:
        ld r17, Y+
        add CheckSum, r17
        lds r16, byte_rs_cnt
        dec r16
        sts byte_rs_cnt, r16
        brne Chk_msg_loop

        ld r17, Y+
        cp CheckSum, r17
        brne Chk_msg_end
        lds r17, stat1
        cbr r17, (1<<stat1_msg_check)
        sbr r17, (1<<stat1_msg_valid)
                                                ;rcall BitToggle
                                                ;rcall BitToggle
        sts stat1, r17
        rjmp Chk_msg_end2
Chk_msg_end:
```

```
        lds r17, stat1
        cbr r17, (1<<stat1_msg_check)
        cbr r17, (1<<stat1_msg_valid)
        sts stat1, r17
Chk_msg_end2:
        ret

update_rsbuffer:
                                                ;rcall BitToggle
                                                ;rcall BitToggle
        ldi YL, low(Msg_Buffer)
        ldi YH, high(Msg_Buffer)
        ldi ZL, low(Rs_Buffer)
        ldi ZH, high(Rs_Buffer)
        ldi r16, MaxByteNum
        sts byte_rs_cnt, r16
update_rsbuffer_loop:
        ld r17, Y+
        st Z+, r17
        dec r16
        sts byte_rs_cnt, r16
        brne update_rsbuffer_loop
        lds r17, stat1
        cbr r17, (1<<stat1_msg_valid)
        sts stat1, r17
        ret

USART_transmit:
;                                               rcall BitToggle
;                                               rcall BitToggle
        sbis UCSRA, UDRE                ; Wait for empty transmit buffer
        rjmp USART_transmit
        out UDR, DataByte               ; Put data into buffer, and send it

ret

SND_PC:     ;sends data to PC on timer2
                                                ;rcall BitToggle
                                                ;rcall BitToggle
;.equ PC_mess_len = 7                   ; number of bytes of the message to PC
; byte_rs_cnt: .BYTE 1                  ; counter for data bytes to PC
; SRAMHEAD:           .BYTE 8                   ; SRAM HEAD Address; array with message to PC
        ;ldi XL, SRAMHEAD                       ; begin address of data
        ;ldi YL, Msg_Buffer                     ; begin address of data
        ldi ZL, Rs_Buffer                       ; begin address of Rs Buffer
        ldi r16, PC_mess_len
        sts byte_rs_cnt, r16            ; load the counter of bytes

;       ldi r16, $AA
;       sts SRAMHEAD, r16
;       rcall   USART_transmit          ;send one byte to PC

SND_PC_loop:
        ;ld DataByte, X+
        ;ld DataByte, Y+
        ld DataByte, Z+
        rcall   USART_transmit          ;send one byte to PC
        lds r16, byte_rs_cnt
        dec r16
        sts byte_rs_cnt, r16
        brne SND_PC_loop

        cli
        lds r16, stat1
        cbr r16, (1<<stat1_snd_PC)              ; data to PC has been sent, clear the flag
        sts stat1, r16
        sei
```

```
                                        ;rcall BitToggle
ret ;end of SND_PC:     ;sends data to PC on timer2

INT0_RESET:
                                        ;rcall BitToggle
                                        ;rcall BitToggle
        in r16, MCUCR
        sbr r16, 3                      ; set ISC01 and ISC00 bits.
        out MCUCR, r16          ; INT0 on Rising-Edge

        clr r16
        sts BitNum, r16         ; point to data BitNum = 0
        sts EdgeNum, r16        ; point to data EdgeNum = 0

        in      r16, TCCR1B
        sbr r16, (1<<CS10)              ; set CS10 bit - choose prescaler to 1 for TIMER1
        out TCCR1B, r16

        in r16, TIMSK
        cbr r16, (1<<OCIE1A)    ; disable Compare Match A Interrupt
        out TIMSK, r16

        in r16, TIFR
        sbr r16, (1<<OCF1A)             ; clear Compare Match A Flag
        out TIFR, r16

        in r16, GICR
        sbr r16, (1<<INT0)              ; enable INT0
        out GICR, r16

        clr TimeL
        clr TimeH

        ldi XL, low(SRAMHEAD)
        ldi XH, high(SRAMHEAD)
;       clr YH
;       ldi YL, Msg_Buffer
;       clr ZH
;       ldi ZL, Rs_Buffer

        ldi DBuffer2, 1
        clr ByteCount

ret

Initialize:
        ; PORT Configuration
        ldi r16, (1<<PD7)|(1<<PD6)|(1<<PD5)|(1<<PD4)|(1<<PD3)|(0<<PD2)|(1<<PD1)|(0<<PD0)
; ldi r16, (0<<PD2)|(1<<PD1)|(1<<PD4)|(0<<PD0)
; PD1(TxD) is the output for UART, PD0(RxD) is input
        out DDRD, r16
; PD2 = Input(INT0), connected to OUTPUT pin on U2270B

        clr r16
        ldi r16, (1<<PD7)|(1<<PD6)|(1<<PD5)|(1<<PD4)|(0<<PD3)|(0<<PD2)|(0<<PD1)|(0<<PD0)
        out PORTD, r16

        ser r16
        out DDRB, r16           ; PB = led Output
        ldi r16, $F0
        out PORTB, r16

        ; Timer1 Config
        ldi r16, 0b00000000     ; WGM13:0 = 0b0000 -> Normal mode, TOP = OCR1A
        out TCCR1A, r16

        in  r16, TCCR1B
```

```
        ldi r16, 0b00000001    ; CS12:0 = 0b001 -> prescaler fclk/1
        out TCCR1B, r16
        ldi r16, 0b00000000    ; disable Compare Match A Interrupt
        out TIMSK, r16
        ldi r16, 0b00010000    ; clear CTC-A Interrupt
        out TIFR, r16

        ; USART Initialization
        ldi r16, 0b00011000
        out UCSRB, r16
        ldi r16, 12            ; set USART Baud Rate Register to 12 to obtain 38.4 kbps
        out UBRRL, r16
        ldi r16, (1<<URSEL)|(0<<USBS)|(3<<UCSZ0)|(0<<UPM0)
        out UCSRC,r16          ; Set frame format: 8data, 1stop bit, no parity

        ; Timer0 Config
        ldi r16, 5
        out TCCR0, r16
        clr r16
        out TCNT0, r16
        ldi r16, 0b00000000
        out TIFR, r16
        ldi r16, 0b00000001
        out TIMSK, r16

        ; Timer2 Config
        ldi r16, 0b00000111
                        ; WGM23:0 = 0b0000 -> Normal mode, TOP = OCR1A, frequency/1024
        out TCCR2, r16
        in r16, TIMSK
        sbr r16, (1<<TOIE2)                    ; enable timer2 overflow interrupt
        out TIMSK, r16

        ; Global Interrupt Enable
        sei

        ; RAM DATA Initialize
        clr r16
        sts stat1, r16
        sts cnt_communic, r16
        sts byte_rs_cnt, r16
        sts BitNum, r16
        sts EdgeNum, r16

        clr XH
        ldi XL, SRAMHEAD
        clr YH
        ldi YL, Msg_Buffer
        clr ZH
        ldi ZL, Rs_Buffer
ret

BitToggle:
        push r16
        in r16, PINB
        swap r16
        out PORTB, r16
        pop r16
ret

; === VARIABLE DEFINITION ===

.equ MaxByteNum = 7                    ; MAX Byte Numbers per DataStream
.equ MaxBitNum         = MaxByteNum*8 ; MAX Bit Numbers to sample
.equ DATAPIN   = 2                     ; DATAIN is PORTD2
.equ cnt_communic_max = 3      ; number of timer2 ovf interrupts between sending data to PC
.equ stat1_snd_PC = 0                  ; bit in stat1; =1 sending data to PC
```

```
.equ stat1_msg_in = 1                ; bit in stat1; =1 msg received
.equ stat1_msg_check = 2             ; bit in stat1; =1 check msg validity
.equ stat1_msg_valid = 3             ; bit in stat1; =1 update msg to send
.equ PC_mess_len = 7                 ; number of bytes of the message to PC

.equ PW_short= $0F                   ; pulse duration limit for short pulses
.equ PW_long = $12                   ; pulse duration limit for long pulses
; RAM Data Segment
.DSEG
stat1:                 .BYTE 1               ; staus byte with flags
cnt_communic:  .BYTE 1
; counter of number of timer2 overflow interrupts between sending data to PC
byte_rs_cnt:   .BYTE 1
; counter for data bytes to PC

BitNum:                .BYTE 1       ; bit 0 is sync bit
EdgeNum:               .BYTE 1
SRAMHEAD:              .BYTE 10      ; SRAM HEAD Address; array for storing message
Msg_Buffer:            .BYTE 10      ; array for received message
Rs_Buffer:             .BYTE 10      ; array for message to PC
CheckSum1:             .BYTE 1       ; CheckSum1, for checking msg validity in firmware

;var1:    .BYTE 1            ; reserve 1 byte to var1
;table:   .BYTE tab_size     ; reserve tab_size bytes
```

B.3 125 kHz Dual Sensor System Graphical User Interface Software

Form_Main.vb

```
' This file contains the GUI software functions, including COM port initialization and _
' listening, file initilization and save, graph initilaization and display, etc.
' Moreover, a class for handling RS232 comunication with VB.Net, CRS232.vb created by
' Corrado Cavalli, is used. Reference -> [37]
Imports System.Text
Imports System.IO
Public Class Form_Main
    Inherits System.Windows.Forms.Form

#Region " Declarations "
    ' Declare necessary class variables
    Private WithEvents miRS232 As New Rs232
#End Region

#Region " Initialze Component Call by Windows Form Designer "
    Public Sub New()
        ' This call is required by the Windows Form Designer.
        InitializeComponent()
        ' Add any initialization after the InitializeComponent() call.
        StatusBox.Items.Clear()
        StatusBox.Items.Add(" RFID GUI by KURT HUANG ")
    End Sub
#End Region

#Region " Global Variables "

    ' START/STOP Control Variables '
    Public Run_Flag As Boolean = False      ' Indicates the data is being processed

    ' RFID Message Variables '
    Public RFID_Data_Valid As Boolean = False ' Indicates the validity of received data
    Public RFID_msg_last(7) As Byte             'Holds last found RF message
    ' RFID Data File Variables '
    Public Data_Folder As String = ".\RFID_Data\"
    Public Save_File_Name As String = Data_Folder & "RFID_Default_01.csv" ' default file
name
    Public Save_File_Num As Integer
    Public Open_File_num As Integer
    Public Delimiter As String = ","
    Public Data_Num As Integer = 0
    Public MaxData_Num As Integer
    Public MaxFile_Num As Integer

    Public config_filename As String = "RFID_GUI_settings.csv"

    ' COM Port Variables '
    Public miComPort As Integer = 2

    'Timer Variables
    Public IntervalCount As Integer = 0

    'Bitmap Variable
    Public BitmapIndex As Integer = 0
    Public Pressure_middle As Double = 0
    Public zoom_pressure As Double = 4
    Public calculated_pressure As Double = 0
    Public calculated_temperature As Double = 0

#End Region
```

```
#Region " START/STOP Events "
    Private Sub StartButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StartButton.Click
        ' START routine
        '   1.  Sets up COM port parameters and initilize the communication
        '       COM port communication is done using the RS232 Class, defined in CRs232.vb
        '   2.  Delete \RFID_Data\*.csv - removes existing RFID data and create ...
        '       a new file for saving
        '       File is defined by parsing the text defined in SaveTo.Text
        Run_Flag = True
        StartButton.Enabled = False
        StopButton.Enabled = True
        Timer1.Start()
        'zoom_pressure = Double.Parse(ZoomP.Text)
        StatusBox.Items.Clear()
        InputString.Text = " Listening on COM" & miComPort.ToString() & "..."
        Me.Text = "RFID GUI - Acquiring"
        Try
            ' Setup com port parameters
            With miRS232
                .Port = miComPort
                .BaudRate = 38400
                .DataBit = 8
                .StopBit = Rs232.DataStopBit.StopBit_1
                .Parity = Rs232.DataParity.Parity_None
                .Timeout = Int32.Parse(TimeoutBox.Text)
            End With
            ' Initilize port
            miRS232.Open()
            ' Set state of RTS / DTS
            miRS232.Rts = True
            miRS232.Dtr = True
            ' Enable RS232 Events
            If COMEvent.Checked Then
                miRS232.EnableEvents()
            Else
                miRS232.DisableEvents()
            End If

        Catch Ex As Exception
            MessageBox.Show(Ex.Message, "Connection Error", MessageBoxButtons.OK)
            Timer1.Stop()
            Run_Flag = False
            StartButton.Enabled = True
            StopButton.Enabled = False
            StatusBox.Items.Clear()
            StatusBox.Items.Add("COM" & miComPort.ToString() & " Error")
            StatusBox.Items.Add("Check COM Port Connection!")
        End Try

        ' Sensor data file creation
        Dim filename As String
        filename = Data_Folder & SaveTo.Text.Substring(0, SaveTo.Text.IndexOf("*"))
        Try
            Kill(filename & "*.csv")
        Catch ex As Exception
        End Try
        Save_File_Num = 0
        filename = Data_Folder & SaveTo.Text.Substring(0, SaveTo.Text.IndexOf("*"))
        Save_File_Num = Save_File_Num + 1
        filename = filename & "_" & Format(Save_File_Num, "00") & ".csv"
        Save_File_Name = filename
        CreateFile(Save_File_Name)

    End Sub
```

```
    Private Sub StopButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StopButton.Click
        '   1.  Calls Make_STOP() function to stop the program
        Make_STOP()
    End Sub
    Private Sub Make_STOP()
        '   1.  Stops COM port communication, and
        '   2.  Reset data number (index for each message saved)
        Run_Flag = False
        Timer1.Stop()
        StartButton.Enabled = True
        StopButton.Enabled = False
        miRS232.DisableEvents()
        miRS232.Close()
        'StatusBox.Items.Clear()
        InputString.Text = InputString.Text & "... Stopped ..."
        Me.Text = "RFID GUI - Stopped"
        Data_Num = 0
    End Sub
#End Region

#Region " COM Port Listening "

    Private Sub miRS232_CommEvent(ByVal source As Rs232, ByVal Mask As Rs232.EventMasks)
Handles miRS232.CommEvent
        ' This routine is called upon each COM port event.
        '   1.  Prepare filename and filenumber to save receive message
        '   2.  Transfer data from RS232 receiving buffer(source.InputStream) to
RFID_msg_buffer()
        '   3.  Display each received message in StatusBox
        '   4.  Check each received message for ID byte and checksum, and save to file if
message is valid
        Dim RFID_msg_buffer() As Byte    'Collects one RFID message
        Dim sBuf As String
        Dim iPnt As Integer
        Dim ID_Byte As Byte = 195
        Dim CheckSum As Integer = 0
        Dim time As Date
        Dim filename As String

        If Data_Num > MaxData_Num Then
            filename = Data_Folder & SaveTo.Text.Substring(0, SaveTo.Text.IndexOf("*"))
            Save_File_Num = Save_File_Num + 1
            filename = filename & "_" & Format(Save_File_Num, "00") & ".csv"
            Save_File_Name = filename
            CreateFile(Save_File_Name)
            Data_Num = 0
            If Save_File_Num > MaxFile_Num - 1 Then
                Save_File_Num = 99
            End If
        End If

        If (Mask And Rs232.EventMasks.RxChar) > 0 Then
            InputString.Text = "Receiving..."
            RFID_msg_buffer = source.InputStream
            If RFID_msg_buffer.Length = 7 Then
                For iPnt = 0 To RFID_msg_buffer.Length - 1
                    StatusBox.Items.Add(iPnt.ToString & ControlChars.Tab &
RFID_msg_buffer(iPnt).ToString)
                    If iPnt > 0 And iPnt < 6 Then
                        If CheckSum + RFID_msg_buffer(iPnt) > 255 Then
                            CheckSum = CheckSum + RFID_msg_buffer(iPnt) - 256
                        Else
                            CheckSum = CheckSum + RFID_msg_buffer(iPnt)
                        End If
                    End If
                Next
```

```
                StatusBox.SelectedIndex = StatusBox.Items.Count - 1
                If      (RFID_msg_buffer(0)      =      ID_Byte)      And      CheckSum      =
RFID_msg_buffer(RFID_msg_buffer.Length - 1) Then
                    RFID_msg_last = RFID_msg_buffer                 ' RFID_msg_last holds
valid data
                    Make_Picture(RFID_msg_last, BitmapIndex)
                    BitmapIndex = BitmapIndex + 1
                    InputString.Text = CheckSum.ToString
                    time = Now()
                    sBuf = Data_Num.ToString & Delimiter & RFID_msg_buffer(0).ToString & _
     Delimiter & RFID_msg_buffer(1).ToString & Delimiter & RFID_msg_buffer(2).ToString & _
     Delimiter & RFID_msg_buffer(3).ToString & Delimiter & RFID_msg_buffer(4).ToString & _
     Delimiter & RFID_msg_buffer(5).ToString & Delimiter & RFID_msg_buffer(6).ToString & _
     Delimiter & "V" & Delimiter & time.ToString & _
     Delimiter & Now().Millisecond.ToString & _
     Delimiter & PValue.Text & Delimiter & TValue.Text
                    Data_Num = Data_Num + 1
                    WriteLineToFile(sBuf, Save_File_Name)
                Else
                    InputString.Text = "Not Valid"              ' if not valid
                    Make_Picture(RFID_msg_last, BitmapIndex)      ' output the last valid
data
                    BitmapIndex = BitmapIndex + 1
                    time = Now()
                    sBuf = Data_Num.ToString & Delimiter & RFID_msg_buffer(0).ToString & _
     Delimiter & RFID_msg_buffer(1).ToString & Delimiter & RFID_msg_buffer(2).ToString & _
     Delimiter & RFID_msg_buffer(3).ToString & Delimiter & RFID_msg_buffer(4).ToString & _
     Delimiter & RFID_msg_buffer(5).ToString & Delimiter & RFID_msg_buffer(6).ToString & _
     Delimiter & "X" & Delimiter & time.ToString & _
     Delimiter & Now().Millisecond.ToString & _
     Delimiter & PValue.Text & Delimiter & TValue.Text
                    WriteLineToFile(sBuf, Save_File_Name)
                    Data_Num = Data_Num + 1

                End If
            End If
        Else
            InputString.Text = "No data in RX Buffer"
        End If

    End Sub

#End Region

#Region " COM Settings "

    Private Sub RadioButton_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Com1.CheckedChanged, Com2.CheckedChanged, Com3.CheckedChanged
        ' Gets COM port select if changed and change it
        If (sender Is Com1) Then
            miComPort = 1
        ElseIf (sender Is Com2) Then
            miComPort = 2
        Else
            miComPort = 3
        End If
    End Sub

    Private Sub COMEvent_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles COMEvent.CheckedChanged
        ' This routine controls the COM port events (EnableEvent() or DisableEvent())
        If (miRS232.IsOpen() And COMEvent.Checked) Then
            miRS232.EnableEvents()
        Else
            miRS232.DisableEvents()
        End If
    End Sub
```

```
#End Region

#Region " Update Graph Routines "
    ' 1. prepare pressure and temperature graph and display them
    ' 2. display receive data values in their text boxes.
    Public Sub Make_Picture(ByVal RFID_msg() As Byte, ByVal BI As Integer)

        Dim g1 As System.Drawing.Graphics = PressureBox.CreateGraphics
        Dim g2 As System.Drawing.Graphics = TemperatureBox.CreateGraphics

        Dim x1, y1, xloc1 As Integer
        Dim x2, y2, xloc2 As Integer

        Dim Or_x1, Or_y1, Max_x1, Max_y1 As Integer
        Dim Or_x2, Or_y2, Max_x2, Max_y2 As Integer

        Dim pressure_raw, temperature_raw As Integer
        Dim TemperatureH, TemperatureL As Byte
        Dim TemperatureD As Double

        Dim PressureH, PressureL As Byte
        Dim PressureD As Double

        Dim TvsVCCH, TvsVCCL As Byte
        Dim TvsVCC_raw As Integer

        ' Get and display temperature data byte from RFID_msg() buffer here
        TemperatureL = RFID_msg(5)
        TemperatureH = RFID_msg(4)
        TemperatureD = 256 * TemperatureH + TemperatureL
        TemperatureD = (TemperatureD / 1024 - Int(TemperatureD / 1024)) * 1024

        Try
            temperature_raw = TemperatureD
        Catch
        End Try

        TValueRaw.Text = temperature_raw.ToString

        ' Get and display pressure data byte from RFID_msg() buffer here
        PressureL = RFID_msg(3)
        PressureH = RFID_msg(2)
        PressureD = 256 * PressureH + PressureL
        PressureD = (PressureD / 1024 - Int(PressureD / 1024)) * 1024

        Try
            pressure_raw = PressureD
        Catch
        End Try

        PValueRaw.Text = pressure_raw.ToString

        ' Get and display TvsVCC (temperature vs VCC) bytes from RFID_msg()
        TvsVCCL = RFID_msg(2)
        TvsVCCL = TvsVCCL / 2 ^ 2
        TvsVCCH = RFID_msg(4)
        TvsVCCH = TvsVCCH / 2 ^ 4
        TvsVCC_raw = TvsVCCH * 64 + TvsVCCL
        TvsVCC.Text = TvsVCC_raw.ToString

        Or_x1 = PressureBox.Left
        Or_y1 = PressureBox.Top
        Max_x1 = PressureBox.Left
        Max_y1 = PressureBox.Height

        Or_x2 = TemperatureBox.Left
```

```
Or_y2 = TemperatureBox.Top
Max_x2 = TemperatureBox.Width
Max_y2 = TemperatureBox.Height

' Draw new data starting here
' First define new bitmap for each image
Dim pres_pic As Bitmap = PressureBox.Image
pres_pic = PressureBox.Image
Dim temp_pic As Bitmap = TemperatureBox.Image
temp_pic = TemperatureBox.Image

Max_x1 = pres_pic.Width
Max_y1 = pres_pic.Height
Max_x2 = temp_pic.Width
Max_y2 = temp_pic.Height

x1 = BI
x2 = BI
xloc1 = BI
xloc2 = BI

Dim setcolor As Color
Dim Ro, Go, Bo As Single

' Grey leading column
Ro = 128 : Go = 128 : Bo = 128 'Grey
setcolor = Color.FromArgb(255, Ro, Go, Bo)
If BI = (Max_x1 - 1) Then : xloc1 = 0 : xloc2 = 0 : End If
For y1 = 0 To Max_y1 - 1
    pres_pic.SetPixel(xloc1 + 1, y1, setcolor)
    temp_pic.SetPixel(xloc2 + 1, y1, setcolor)
Next

'Black line in the middle
y1 = Int((Max_y1 - 1) / 2)
y2 = Int((Max_y2 - 1) / 2)
Ro = 0 : Go = 0 : Bo = 0 ' Black
setcolor = Color.FromArgb(255, Ro, Go, Bo)
pres_pic.SetPixel(x1, y1, setcolor)
temp_pic.SetPixel(x2, y2, setcolor)

'Green dots for trend
'Public Pressure_middle As Double = 0
Dim delta_y1 = 1024
Dim y1_double As Double
Dim zoom_rate As Double
Dim track_time As Double
Dim pres_coe As Double
Dim pressure_reading As Double
'TextBox1.Text = "here"

' Auto Zoom routine for pressure, uses zoom_rate, track_time and pres_coe
Try
    zoom_rate = Convert.ToDouble(ZoomRate.Text)
Catch
    zoom_rate = 1.01
End Try
If zoom_rate < 1.001 Then zoom_rate = 1.001
ZoomRate.Text = zoom_rate.ToString

Try
    track_time = Convert.ToDouble(TrackTime.Text)
Catch
    track_time = 256
End Try
If track_time < 16 Then track_time = 16
TrackTime.Text = track_time.ToString
```

```
Try
    pres_coe = Convert.ToDouble(PCC.Text)
Catch
    pres_coe = 3.3

End Try
If pres_coe < 0.064 Then pres_coe = 0.064
PCC.Text = pres_coe.ToString

If Pressure_middle = 0 Then
    delta_y1 = 1024
    Pressure_middle = 0.5
End If
y1_double = PressureD / delta_y1
Pressure_middle = Pressure_middle + (y1_double - Pressure_middle) / track_time
If (Math.Abs((y1_double - Pressure_middle) * zoom_pressure / 1)) > 0.4 Then
    zoom_pressure = zoom_pressure / zoom_rate
    If zoom_pressure < 1 Then zoom_pressure = 1
End If
If (Math.Abs((y1_double - Pressure_middle) * zoom_pressure / 1)) < 0.3 Then
    zoom_pressure = zoom_pressure * zoom_rate
    If zoom_pressure > 8 Then zoom_pressure = 8
End If

ZoomP.Text = zoom_pressure.ToString
pressure_reading = (PressureD - Pressure_middle * delta_y1) * pres_coe
PValue.Text = pressure_reading.ToString
y1 = Int(Max_y1 * (0.5 - (PressureD / delta_y1 - Pressure_middle) * zoom_pressure))
If y1 < 0 Then y1 = 0
If y1 > Max_y1 - 1 Then y1 = Max_y1 - 1 ' clipping pressure

' Temperature data is processed here, using temp_coe and temp_offset
Dim delta_y2 = 1024
Dim temp_coe As Double
Dim temp_offset As Double
Try
    temp_coe = Double.Parse(TCC.Text)

Catch
    temp_coe = 0.078
End Try
If temp_coe < 0.00001 Then temp_coe = 0.00001
TCC.Text = temp_coe.ToString
Try
    temp_offset = Double.Parse(TOffset.Text)
Catch
    temp_offset = -2
End Try
If temp_offset > 100 Then temp_offset = 100
If temp_offset < -100 Then temp_offset = -100
TOffset.Text = temp_offset.ToString

y2 = Int(Max_y2 * (1 - TemperatureD / delta_y2))
If y2 < 0 Then y2 = 0
If y2 > Max_y2 - 1 Then y2 = Max_y2 - 1 ' clipping temperature
TValue.Text = (temp_coe * TemperatureD + temp_offset).ToString

Ro = 0 : Go = 255 : Bo = 0 'green
setcolor = Color.FromArgb(255, Ro, Go, Bo)

pres_pic.SetPixel(x1, y1, setcolor)
g1.DrawImage(pres_pic, 0, 0)
PressureBox.Image = pres_pic

temp_pic.SetPixel(x2, y2, setcolor)
g2.DrawImage(temp_pic, 0, 0)
```

```
        TemperatureBox.Image = temp_pic

        If BI = (Max_x1 - 1) Then
            BitmapIndex = 0
        End If

    End Sub

#End Region

#Region " Timer1 - running but not used in program"
    Public Sub Timer1_INT(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Timer1.Tick
        'Make_Picture(160, 100, BitmapIndex)
        'BitmapIndex = BitmapIndex + 1
    End Sub
#End Region

#Region " File Functions "

    Private Sub CreateFile(ByVal filename As String)
        ' Create new file with title row for each column
        Dim fs As New FileStream(filename, FileMode.Create, FileAccess.Write)
        Dim s As New StreamWriter(fs)
        s.BaseStream.Seek(0, SeekOrigin.End)
        s.WriteLine("-1,ID       Byte,Data       Counter,P.H.Byte,P.L.Byte,T       H_Byte,T
L_Byte,CheckSum,Validity, Time, Time(ms),Calculated Pressure, Calculated Temperature") '
        s.Close()
    End Sub

    Private Sub WriteLineToFile(ByVal txt As String, ByVal filename As String)
        ' Write one row(message) to file
        Dim fs As New FileStream(filename, FileMode.Append, FileAccess.Write)
        Dim s As New StreamWriter(fs)
        s.BaseStream.Seek(0, SeekOrigin.End)
        s.WriteLine(txt)
        s.Close()
    End Sub

    Private Sub MaxMsgNum_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MaxMsgNum.SelectedIndexChanged
        ' Changes maximum number of messages saved in one file
        ' only changable when system is not running
        'Public MaxData_Num As Integer
        If Run_Flag Then
            MaxMsgNum.SelectedIndex = 0
            MaxMsgNum.Text = MaxMsgNum.SelectedItem
            MaxData_Num = Int32.Parse(MaxMsgNum.Text)
        Else
            MaxMsgNum.Text = MaxMsgNum.SelectedItem
            MaxData_Num = Int32.Parse(MaxMsgNum.Text)
        End If

    End Sub

    Private Sub MaxFileNum_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MaxMsgNum.SelectedIndexChanged
        'Public MaxData_Num As Integer
        'Public MaxFile_Num As Integer
        If Run_Flag Then
            Return
        Else
            MaxFileNum.Text = MaxFileNum.SelectedItem
            MaxFile_Num = Int32.Parse(MaxFileNum.Text)
        End If

    End Sub
```

```
#End Region

#Region " Reading Bytes from File "
    Private Function ReadFile(ByVal filename As String) As String
        Dim fs As New FileStream(filename, FileMode.Open, FileAccess.Read)
        Dim s As New StreamReader(fs)
        Dim temp As String
        temp = s.ReadLine()
        s.Close()
        Return temp
    End Function
#End Region

#Region " FORM EXIT - MENU STRIP "

    Private Sub Menu_File_Exit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Menu_File_Exit.Click
        System.Windows.Forms.Application.Exit()
    End Sub

#End Region

#Region " Config - Import or Create config file"

    Private Sub Get_Config()

        Try
            Using MyReader As New
Microsoft.VisualBasic.FileIO.TextFieldParser(config_filename)
                MyReader.TextFieldType = FileIO.FieldType.Delimited
                MyReader.SetDelimiters(Delimiter)
                Dim currentRow As String()
                If MyReader.EndOfData Then

                End If
                While Not MyReader.EndOfData
                    Try
                        currentRow = MyReader.ReadFields()
                        Dim currentField As String
                        Dim i = 0
                        Dim cal_coef_val As String

                        For Each currentField In currentRow
                            i = i + 1
                            'MsgBox(i & ": " & currentField)
                            If i = 1 Then
                                Try
                                    cal_coef_val = Double.Parse(currentField)
                                Catch
                                    cal_coef_val = "0"
                                End Try
                            End If
                            If i = 2 Then
                                Try
                                    If currentField = "ZoomRate" Then ZoomRate.Text =
cal_coef_val
                                    If currentField = "TrackTime" Then TrackTime.Text =
cal_coef_val
                                    If currentField = "PCC" Then PCC.Text = cal_coef_val
                                    If currentField = "TCC" Then TCC.Text = cal_coef_val
                                    If currentField = "TOffset" Then TOffset.Text =
cal_coef_val
                                Catch ex As Exception
                                    MsgBox("can't write in textbox " & currentField)
                                End Try
                            End If
                        Next
```

```
                    Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
                        MsgBox("Line " & ex.Message & "is not valid and will be skipped.")
                    End Try
                End While
            End Using
        Catch ex As Exception
            MsgBox("No File: " & config_filename)
            MakeConfig("Make New Config", config_filename)
        End Try

        'Me.ZoomRate = New System.Windows.Forms.ToolStripTextBox
        'Me.TrackTime = New System.Windows.Forms.ToolStripTextBox
        'Me.PCC = New System.Windows.Forms.ToolStripTextBox
        'Me.TCC = New System.Windows.Forms.ToolStripTextBox
        'Me.TOffset = New System.Windows.Forms.ToolStripTextBox
    End Sub


    Private Sub MakeConfig(ByVal txt As String, ByVal filename As String)
        Dim fs_config As New FileStream(filename, FileMode.Create, FileAccess.Write)
        Dim s_config As New StreamWriter(fs_config)
        s_config.BaseStream.Seek(0, SeekOrigin.End)
        txt = ZoomRate.Text & Delimiter & "ZoomRate" & Delimiter & " for Pressure Graph"
        s_config.WriteLine(txt)
        txt = TrackTime.Text & Delimiter & "TrackTime" & Delimiter & " for Pressure"
        s_config.WriteLine(txt)
        txt = PCC.Text & Delimiter & "PCC" & Delimiter & "Calibration Coefficient for
Pressure in cmH2O/LSB"
        s_config.WriteLine(txt)
        txt = TCC.Text & Delimiter & "TCC" & Delimiter & "Calibration Coefficient for
Temperature in oC/LSB"
        s_config.WriteLine(txt)
        txt = TOffset.Text & Delimiter & "TOffset" & Delimiter & "Offset Calibration
Coefficient for Temperature in oC"
        s_config.WriteLine(txt)
        s_config.Close()
        'Me.ZoomRate = New System.Windows.Forms.ToolStripTextBox
        'Me.TrackTime = New System.Windows.Forms.ToolStripTextBox
        'Me.PCC = New System.Windows.Forms.ToolStripTextBox
        'Me.TCC = New System.Windows.Forms.ToolStripTextBox
        'Me.TOffset = New System.Windows.Forms.ToolStripTextBox
    End Sub

    Private Sub OpenConfig_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles OpenConfig.Click
        Get_Config()
    End Sub

    Private Sub SaveConfig_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SaveConfig.Click
        MakeConfig("Make New Config", config_filename)
    End Sub

#End Region

End Class
```

Appendix C: Components Cost

C.1 2.4 GHz Pressure Sensor System – Components Prices

Components	Quantity	Unit Price	Subtotal
C8051F005 Microcontroller	1	16.39	16.39
Capacitior 10 nF	4	0.04	0.16
Capacitior 1 uF	2	0.10	0.21
Capacitor 4.7 uF	2	1.83	3.66
Resistor 470 ohm	1	0.60	0.60
Resistor 4.75 kohm	1	0.08	0.08
Votlage Regulator 2.7V	1	0.41	0.41
Voltage Regulator 3V	2	1.17	2.34
		Total	**23.84** **CAD**

C.2 125 kHz Dual Sensor System – Components Prices

Components	Quantity	Unit Price	Subtotal
ATTiny24 Microcontroller	1	2.06	2.06
Capacitor 2.2uF	1	0.26	0.26
Capacitor 0.47uF	2	0.18	0.35
Capacitor 681pF	1	0.06	0.06
Resistor 330 kohm	3	0.09	0.27
Votlage Regulator 2.7V	1	1.70	1.70
Battery-Holder	1	1.03	1.03
CR1025 Battery	1	1.17	1.17
U3280M Transponder IC	1	2.10	2.10
		Total	**9.00** **CAD**