A PORTABLE BIOTELEMETER FOR BATTERYLESS GERD SENSORS

by

SANDEEP BATTULA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2009

ACKNOWLEDGEMENTS

First, I want to thank my supervising professor, Dr. Chiao for giving me the opportunity to be part of his excellent research work. He has supported me throughout my thesis with his patience and knowledge whilst allowing me the room to work in my own way. He was always there to listen and to give advice regarding my research and career.

My special thanks to Dr. Carter and Dr. Vasilyev for their interest in my research and for taking time to be on my thesis defense committee.

I thank my parents, my brothers and my fiancée for unconditional support and encouragement to pursue my interests, even when the interests went beyond boundaries.

Last but not least, I offer my thanks and regards to all iMEMS group, roommates and friends who supported me during the completion of the project.

November 21, 2009

ABSTRACT

A PORTABLE BIOTELEMETER FOR BATTERYLESS GERD SENSORS

Sandeep Battula, M.S.

The University of Texas at Arlington, 2009

Supervising Professor: Jung-Chih Chiao

In many medical problems, a continuous monitoring of the disease in vivo is essential to prevent further deterioration of health. The monitoring usually exceeds 24 hours. With the help of current advanced technologies, micro-medical devices are built to sense the internal physiological conditions in patients. Telemetry is used to transmit information to an external monitoring device. For the convenience and comfort of the patient, these monitoring systems should have low power consumption and be portable.

This work discusses the design of a low power portable biotelemeter for batteryless Gastro esophageal reflux disease (GERD) sensors which can detect acid and non-acid refluxes in esophagus. When a batteryless GERD sensor is placed in the working region of the telemeter, it should wirelessly power up the sensor and receive a modulated physiological signal from the implant. The sensor signals should be extracted from this modulated signal using a demodulation circuit. In the GERD monitoring, depending on the modulated frequency changes, the acid refluxes in esophagus then can be detected.

A microcontroller was introduced to measure, process and record this information signals. To record this data to a secured digital (SD) card, the Serial Peripheral Interface (SPI) mode communication link was setup between the card and microcontroller. In the SD card, the measured signal is recorded with its corresponding time stamp in a TEXT file format. Using the

microcontroller, the total power consumption was reduced by controlling the timings of powering up the batteryless sensor. Experiments were conducted to study the effect of different environmental parameters including distances, motion artifacts, and misalignment on the measured signals. Performance and stability of the system were also studied in the experiments.

The designed biotelemeter has a reading range up to 6 cm along z-axis, with less than 5% change in the measured frequency. All the frequencies measured at every point within the reading range are repeatable. When sensor is not perfectly aligned to telemeter at a distance of 1.5cm, the maximum allowable tilt is $\pm 60^{o}$ about the x-axis, $\pm 55^{o}$ about the y-axis and no change about the z-axis. The shift in the frequency because of the refluxes is much higher than the change in the frequency with any environmental parameter, so the signal to noise ratio is maintained. The frequency measured is stable with the battery voltage, time and any other parameter. The power consumption of the device is improved with a closed loop timing control using the microcontroller by 34 times.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In the modern medicine, non-invasive or minimally invasive diagnosis methods are preferred to invasive methods or bulky systems for diagnosis of many medical problems. For diagnosis of these problems, continuous monitoring of the disease condition is essential to prevent further deterioration of health which usually exceeds 24 hours. With the help of current advanced technologies like Micro Electro Mechanical Systems (MEMS), Wireless Communications, Radio Frequency Identification (RFID) and Circuit Designing, many monitoring systems are being designed for different medical problems. Present work discusses the design of a biotelemeter for Gastro esophageal reflux disease (GERD) monitoring system.

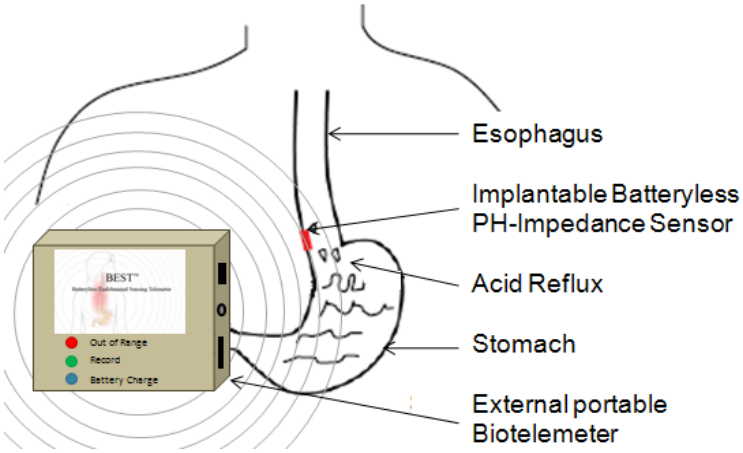


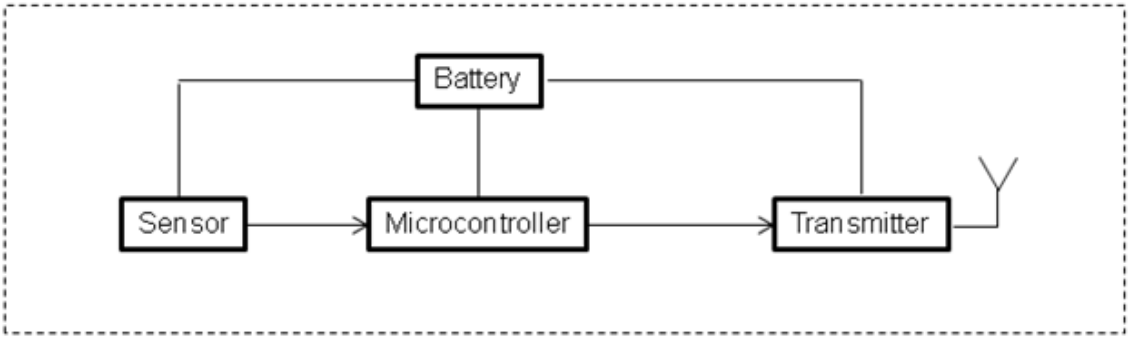
Figure 1.1 Schematics of an implantable battery less impedance-pH sensor and an external bio-telemeter

1.1 Monitoring System for GERD

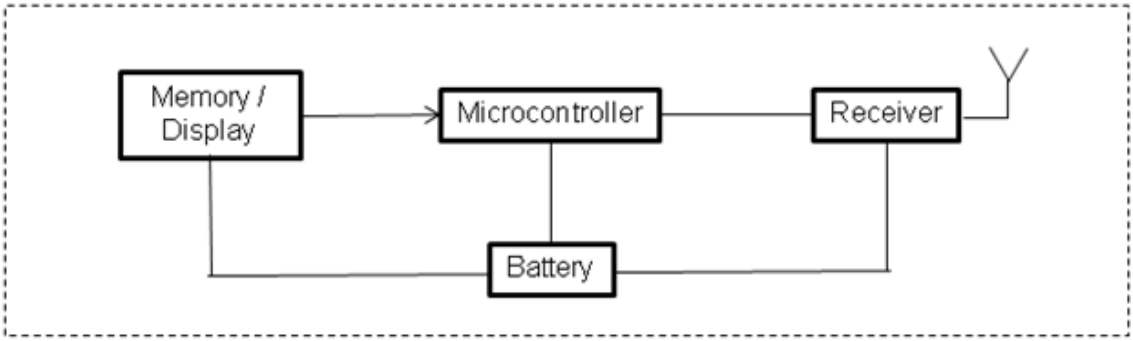
Gastro esophageal reflux disease (GERD) or acid reflux is a condition in which the liquid content of the stomach regurgitates (backs up or refluxes) into the esophagus. The acidic nature of these refluxes damages the mucosal lining of the esophagus causing heart burn, dysphasia etc. A continuous esophageal impedance-pH monitoring is one of the best ways of

diagnosing GERD [1.1]. This continuous monitoring is important because an early detection of GERD prevents cancer.

A wireless solution for monitoring GERD includes design of Implantable GERD sensor and an external telemeter. This implantable sensor detects acid refluxes by sensing PH or Impedance or both. An external telemeter for GERD sensors is a device which continuously monitors and records the refluxes in the esophagus. This impedance-pH monitoring system consists of an Implantable sensor and an external biotelemeter as shown in Fig 1.1.



(a)



(b)

Figure 1.2 Block diagram of (a) an implantable microsystem and (b) an external biotelemeter.

1.2 Architecture of Currently Available Monitoring System for GERD

A currently available wireless monitoring system for GERD is Bravo^{TM} pH monitoring system. This system includes an implantable microsystem and an external pH recording telemeter. Physically, this system is similar to the system proposed in previous section (Sec.1.1), but working of these systems are completely different.

A typical architecture of a battery based wireless medical monitoring systems is shown in Fig.1.2. An implantable microsystem is a device which electrically senses a medical problem using a sensor electrode or multiple sensor electrodes. This sensing signal is processed on a microcontroller and transmitted to an external monitoring system using RF/IR transmitters. A battery is included in this micro-system to power up all these internal circuitry. An external sensing telemeter receives the information modulated signal from microsystem, processes and records the sensing information. Even on external side a battery is required to power up all the circuitry of telemeter. Usually, this information is displayed on the telemeter or an interface is provided to display the recorded information on a computer [1.2].

1.3 Obstacles in Current Implantation

There are various obstacles or limitations in implementation of these wireless medical monitoring systems, which are broadly classified in to obstacles in application and obstacles in electrical design.

Some of the obstacles in application include:

1. Size of the sensor
2. Sensing modality
3. Calibration of sensors
4. Patient comfort with implantable system or external monitoring system
5. Biodegradable implantable systems

Some of the obstacles in electrical design include:

1. Life of the product: This is greatly determined by battery life of the implantable microsystem.
2. Cost of the final product: This is usually high because of development of Special devices (Ex. Bravo data link, Media pump, Calibration stand etc.) and Special Software (Ex. Bravo polygram NET software)

1.4 Previous Work

In the previous work a new impedance-pH monitoring system was proposed based on radio frequency identification (RFID) technology including a transponder and a reader [1.3].

1.4.1 Working Principle

In this design, a RFID transponder acts as a Batteryless Implantable Microsystem and a RFID reader acts as an External Monitoring Telemeter. The complete working of the system is shown in Fig.1.3.

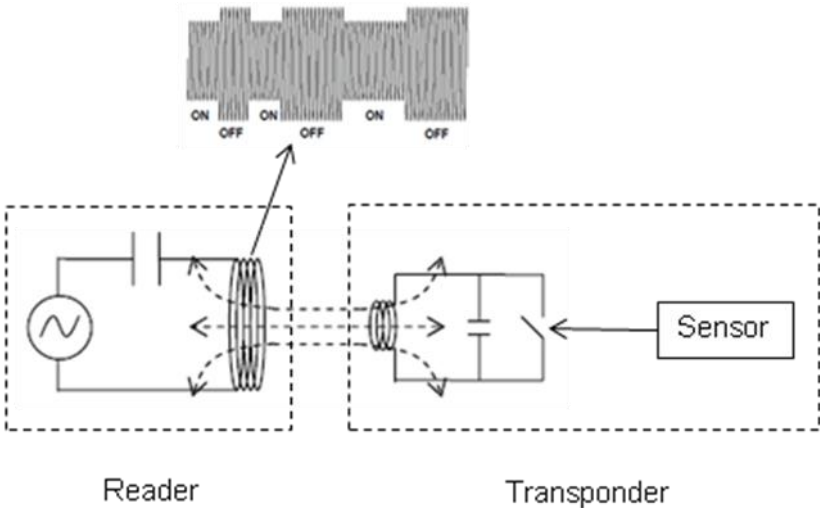
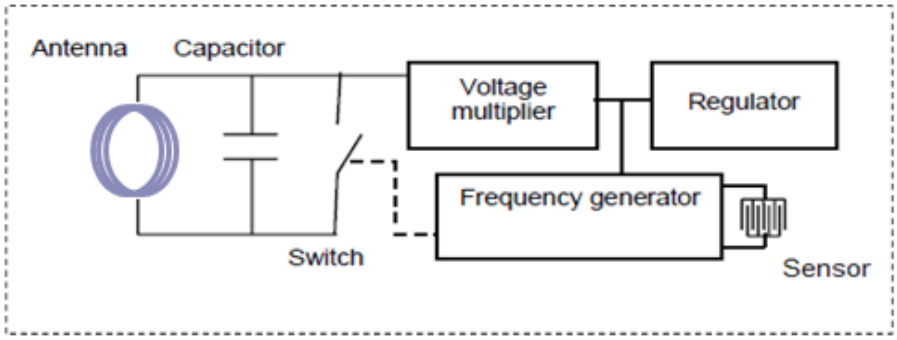


Figure 1.3 Schematic diagrams of Transponder and Reader.

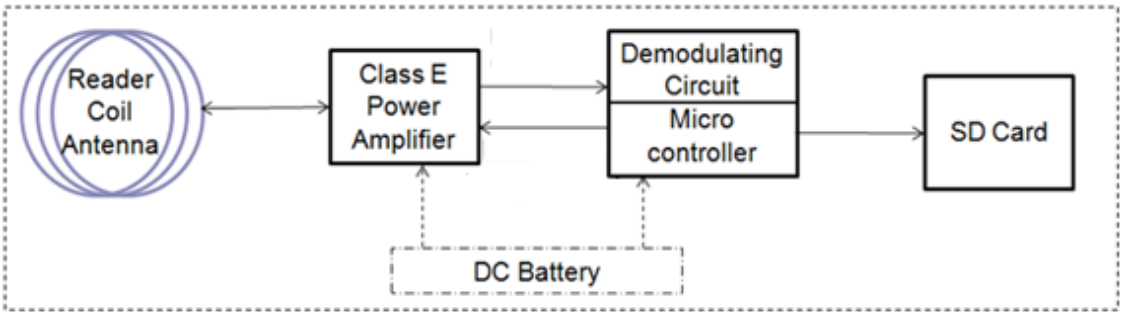
The coil antennas on both sides are connected to a capacitors forming resonance at the same frequency. At resonant frequency, the reader coil generates maximum electromagnetic fields coupling into the transponder coil. In the near field region, the impedance seen by the reader coil changes, when the switch at the transponder opens or closes. This load modulation alters the voltage level at the reader coil. This modulation frequency is usually much less than that of the resonant carrier frequency of the system [1.4].

Figure 1.3 shows the modulated signals at the reader coil at different switching frequencies. Signal amplitude at the reader is high, when the switch at the transponder is open (OFF), due to low loading effects. It is low, when the switch is close (ON) as the transponder

loads the reader coil and the reader is shifted from resonance condition [1.5]. The switching frequency at the transponder varies with sensor impedance and pH conditions. Hence a wireless batteryless impedance-pH system is achieved [1.6].



(a)



(b)

Figure 1.4 Architecture of the proposed impedance-pH monitoring system with (a) implantable batteryless GERD sensor and (b) batteryless endoluminal sensing telemeter.

The architecture of this new impedance-pH monitoring system based on working principle explained above is shown in Fig.1.4. Unlike other implantable microsystems, this is a passive sensor, making it “Implantable Batteryless GERD sensor” as shown in Fig.1.4 (a). As discussed in previous section, this external telemeter is based on RFID reader which powers up and communicates with battery-less sensors, making it a “Batteryless Endoluminal Sensing Telemeter” (BEST^{TM}) as shown in Fig.1.4 (b).

1.4.2 Architecture of Implantable Batteryless Sensors

An implantable batteryless sensor is the transponder side of RFID technology. This can be designed either for single sensor or multiple sensors. The complete block diagram of transponder for single sensor and multiple sensors is shown in the Fig. 1.5 (a) and Fig. 1.5 (b) respectively [1.7].

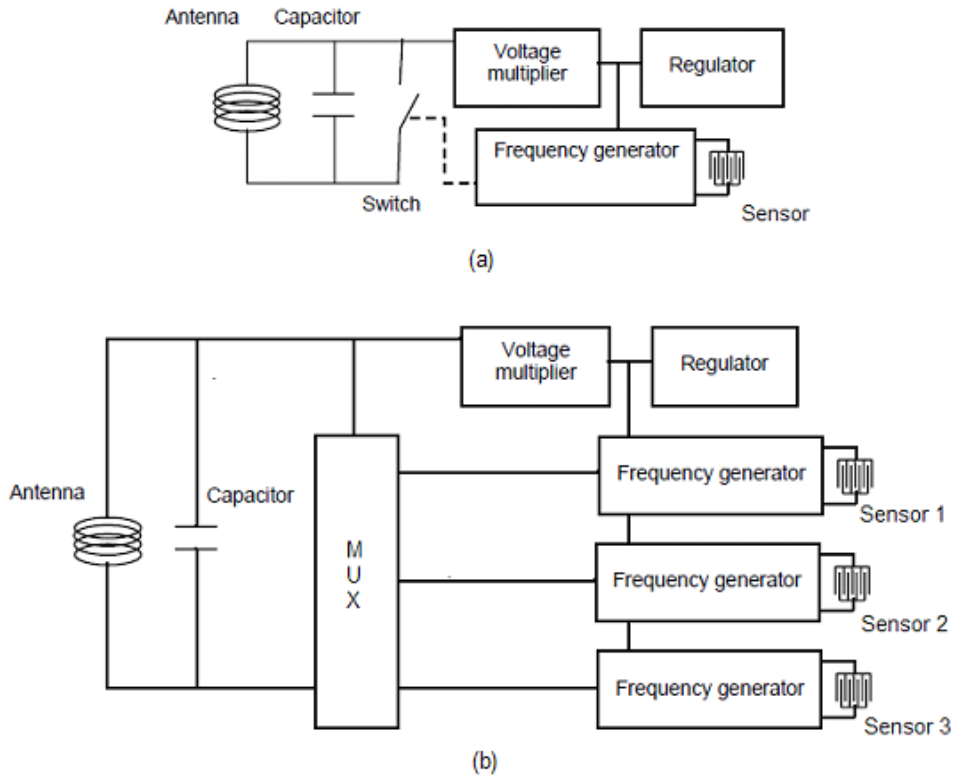


Figure 1.5 Block diagram of (a) single sensor platform and (b) multiple sensor platform.

The antenna and the capacitor shown in the Fig. 1.5(a) and Fig. 1.5(b) forms a resonant circuit to receive powers and modulate the data. The voltage multiplier and the regulator build up a constant DC level to operate the rest of the circuit. The frequency generator generates a varying frequency signal depending upon the impedance or pH or both in the circuit. By connecting the electrodes to the frequency generator, the output frequency can be controlled by the impedance of the materials on the electrodes [1.8].

Voltage multiplier and Regulator: A 4 stage voltage multiplier circuit was designed and optimized experimentally to achieve the longest read range [1.9]. The output of the voltage multiplier was connected to a 2.5-V CMOS regulator and a storage capacitor. A 12-V zener diode was added to protect the regulator from high voltages in case of close proximities of transponder and reader.

Frequency generator: A frequency generator generates a square wave signal whose frequency depends on the condition of the sensing element. For example, the pH sensor electrodes produce a potential difference depending upon the medium [1.11]. So the frequency generator output frequency varies with this potential difference [1.12]. The number of frequency generators required depends upon the number of sensors. These frequency generators are designed such that they work over a large frequency range and their frequency ranges should not overlap [1.13].

For an Impedance sensor, a relaxation oscillator was designed for a frequency range of 9 KHz to 12 KHz. As the Impedance was changed from 0 ohm (short circuit) to infinity ohm (open circuit), the output of the frequency generator changed from 9 KHz to 12 KHz.

For a pH sensor, a relaxation oscillator was designed as shown in Fig. 3.6 for a frequency range of 15 KHz to 23 KHz. This oscillator was designed to generate output signal depending upon the potential difference between the terminals of pH sensor electrodes. As the pH was changed from 1 to 12, the output of the frequency generator changed from 9 KHz to 12 KHz [1.14].

Modulator: For modulation of signal from single frequency generator, a MOSFET was used as switch shown in the Fig. 1.6 (a). For modulation of signals from multiple frequency generators, a simple time division multiplexing scheme was used as switch shown in the Fig. 1.6 (b) [1.15].

The prototypes of the single sensor (impedance) and dual sensor (impedance and PH) circuitries were assembled on a 4-layer PCB as shown in Fig. 1.6 (a) and Fig. 1.6 (b) respectively. The prototypes were coated with Polydimethylsiloxane (PDMS), a biocompatible polymer as it shown in Fig 1.6 [1.7].

(a)

(b)

Figure 1.6 (a) A prototype of the wireless impedance sensor. (b) A prototype of the wireless impedance and pH sensor.

1.5 Present Work

Our present work here is design of an External Monitoring Telemeter for above discussed battery less GERD sensors. All the blocks of biotelemeter are discussed in detail in next chapters.

CHAPTER 2

BIOTELEMETER FOR GERD SENSORS

A biotelemeter is a device which monitors or records or displays the sensing parameter by wirelessly communicating with the implanted device. Our current biotelemeter design not only wirelessly communicates with implantable sensor but also wirelessly powers it. So this specially designed bio-telemeter is called “Batteryless Endoluminal Sensing Telemeter” (BEST^{TM}). The main features of BEST^{TM} are

- Wireless power to batteryless sensors
- Wireless communication
- Signal processing
- Recording to secured digital (SD) card
- Low power consumption
- Sensor position detection
- Battery charging indication

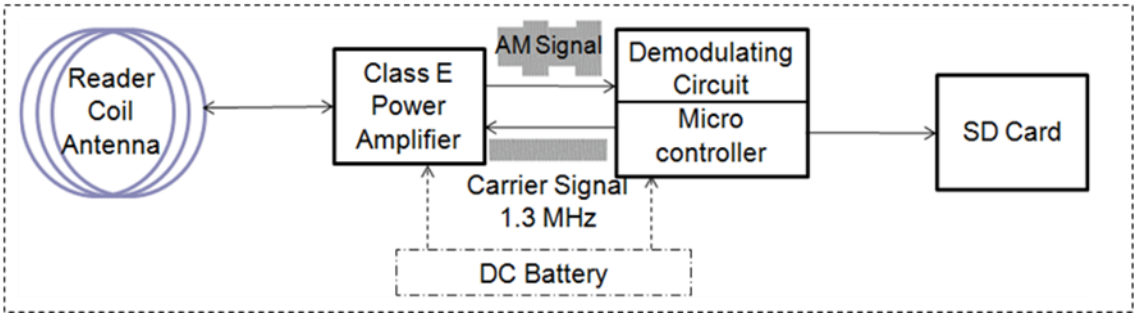


Figure 2.1 Block diagram of BEST^{TM}

2.1 Wireless Power to Battery-less Sensors

The batteryless sensors are powered up wirelessly by inductively coupling them with BEST™.

So to power up the sensors wirelessly involves two major designs:

- BEST™ coil antenna design
- Class E power amplifier design

2.1.1. BEST™ Coil Antenna Design

Consider a Tag Antenna and a BEST™ Coil Antenna (or simply Reader Antenna) which are inductively coupled as shown in Fig 2.2.

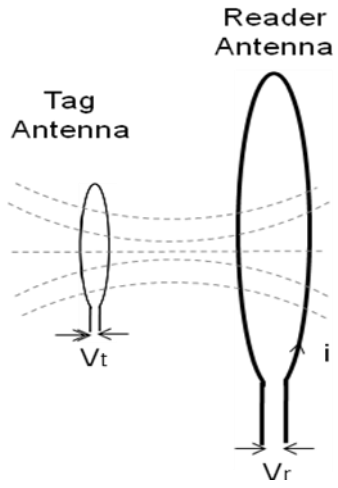


Figure 2.2 Inductively coupled reader and tag antennas.

V_t – Voltage induced in tag coil antenna

V_r – Voltage across reader coil antenna

$$V_t \propto \emptyset f \quad (1)$$

$\emptyset$ – Magnetic flux produced by the reader coil

f – Operating frequency =1.3MHz

$$\emptyset \propto Ni \quad (2)$$

N – Number of turns of reader coil

i – Current in reader coil

From (1) & (2)

$\Rightarrow V_t \propto Nif$

$\Rightarrow V_t \propto Ni$ (3)

(As f = 1.3MHz)

Also $V_r = j\omega Li$

$\Rightarrow V_r \propto Li$ (4)

From (3) & (4)

$\Rightarrow V_t \propto \frac{NV_r}{L}$

$\Rightarrow V_t \propto \frac{N}{L}$

$V_t \rightarrow$ High

$\frac{N}{L} \rightarrow$ High

Wire Diameter of Reader Coil Antenna

This was determined from the AC resistance R_{ac} of the coil antenna [2.1].

$$R_{ac} = \frac{l}{\sigma(w+t)\delta}$$

l – length of the coil

σ – Conductivity of material

w – Width of the coil

t – Thickness of the coil

δ – Skin depth

For 1.3 MHz, AWG # 16 was chosen [2.8]

Radius of Reader Coil Antenna

The Optimum Coil radius (a) was determined from the reading distance (r) [2.7].

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

Reader coil radius was calculated as 5cm for a reading distance of 3.5cm.

Number of Turns of Antenna

With the radius of reader coil antenna as 5cm and coil diameter as AWG#16, different antennas with turns number N = 1, 2, 3, 4, 5, … were designed and corresponding inductance of the coil was determined using LCR meter.

From the highest $\frac{N}{L}$ ratio, Number of turns was determined as N=3 & Inductance was 2.5uH.

2.1.2. Class E Power Amplifier Design

To transfer power from Reader coil to Tag coil, a class E Power amplifier was chosen. This was chosen because of its high efficiency than compare other power amplifier [2.2].

Theory:

From the inductance of the reader coil antenna (L) determine in Sec.3.1.1, its corresponding capacitance (C) was determined to resonate at 1.3MHz frequency [2.3].

$$f = \frac{1}{2\pi\sqrt{LC}}$$

f – Operating frequency =1.3MHz

L – Inductance of the reader coil antenna

C – Resonant capacitance

⇨ $C = 6nF$ for $L = 2.5uH$

Simulation:

IRF510 MOSFET was chosen after simulating various RF MOSFETs for low switching loss in the circuit [2.4]. A 3.8V battery was chosen for simulation of circuit because of commercial availability of 3.8V high capacity batteries. The circuit shown in Fig.2.3 was simulated with 1.3MHz square wave input signal. A 100uH inductor or RF choke was placed in

series with 3.8V supply to maintain constant input current [2.5]. A 10nF capacitor was placed in parallel to IRF510 MOSFET to pass the current when MOSFET was open [2.6].

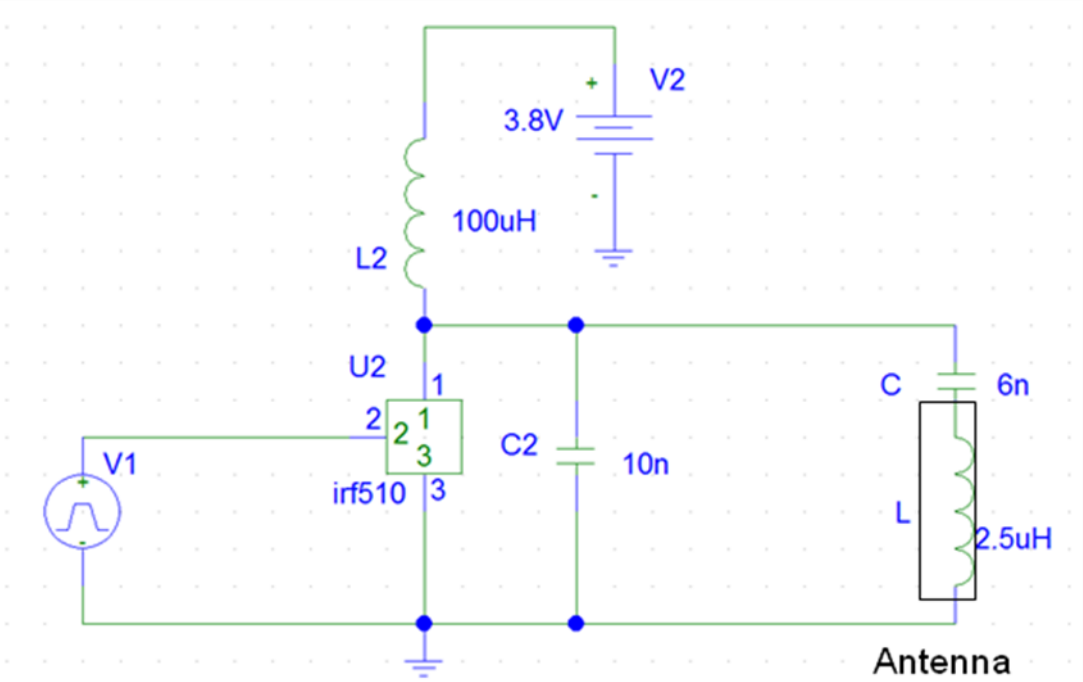


Figure 2.3 Circuit diagram of design class E Amplifier

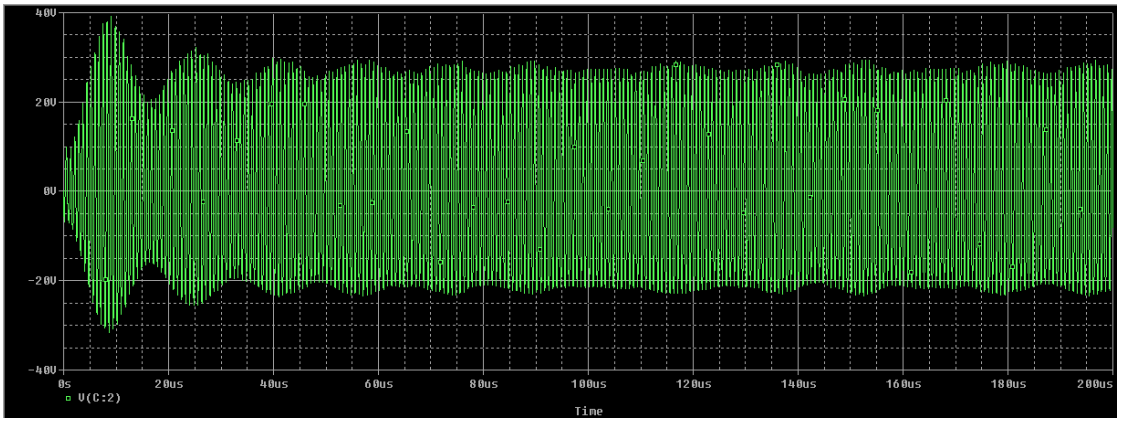


Figure 2.4 Simulation result of voltage across coil in class E amplifier

Practical Circuit

The above simulated Class E power amplifier was designed on PCB with slight tuning in capacitance for resonance at 1.3MHz.

DC Batteries

Though all the circuit components were designed to work at 3.8V, two separate DC batteries were chosen. One was high capacity (1300mAh) rechargeable battery and other one was normal capacity battery (500mAh) as shown in Fig.2.5.

As discussed earlier, a class E power amplifier requires high power to transfer wireless power to batteryless sensors. So a high capacity battery supplies power to this power amplifier while the other battery supplies power to low power devices (op-amps, microcontroller & SD card) on board.

Main advantage of using two batteries is for user interface. So, BEST^{TM} indicates the user to recharge the battery for every few days, while whole device works for few months on a normal non rechargeable battery. This gives an additional feature to BEST^{TM}, called "Battery Recharge indicator". Working of this Indicator is discussed in detail in Sec.3.2.6.

Using two batteries also isolates the high current class E amplifier circuit from low current microcontroller circuit.

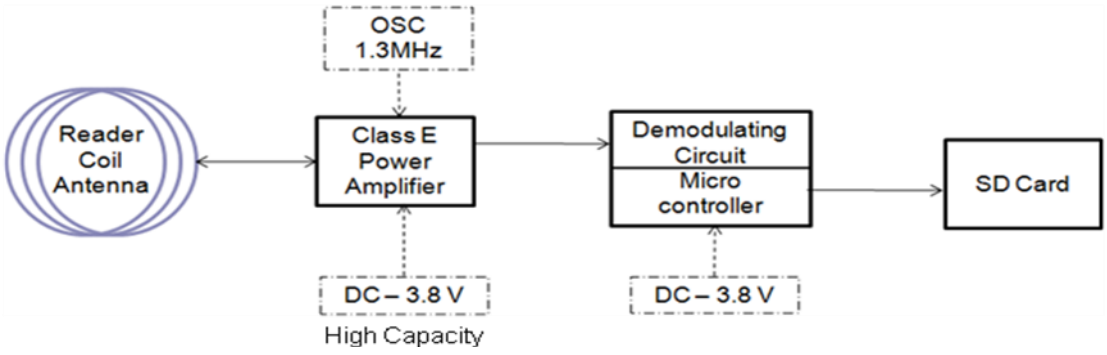


Figure 2.5 BEST^{TM} with a high capacity battery and a normal battery.

With continuous operation a 1300^{mAh} battery lasted for 5 hours

⇨ DC Current input = 260mA

Table 2.1 Comparison of practical and theoretical values

Parameter	Theoretical	Practical
Voltage (V)	3.8	3.8
Current (mA)	80	260
Power Consumption (mW)	663	980
Inductance (uH)	2.5	2.54
Capacitance (nF)	8	10.2
Operating Frequency (KHz)	1.3	1.315
Maximum Reading Distance (cm)	3.5 (Optimum)	5.5 (up to)

2.2 Wireless Communication with Implantable sensors

Another important feature of BEST^{TM} is wireless communication with implanted sensor. As discussed in Sec.1.3.1, the same reader coil designed for wireless power transfer itself acts as an antenna for communication. When the Tag sensor is at operating distance from BEST^{TM} the voltage across the reader coil antenna is modulated with information signal from the sensor. This modulated signal is processed and information is extracted using Demodulating Circuit.

As shown in Fig.2.1 a Demodulating Circuit has

1. Passive envelope detector
2. Active band pass filter
3. Non inverting amplifier
4. Comparator

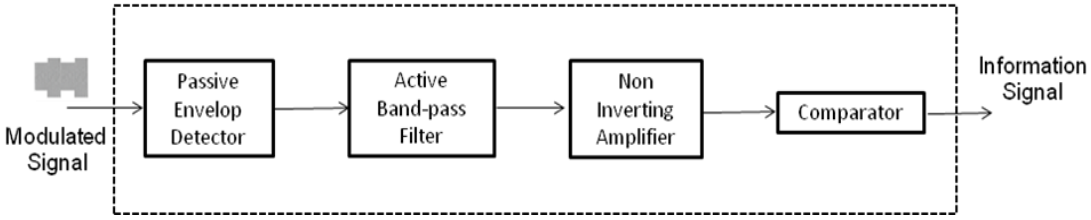


Figure 2.6 Stages in Demodulating Circuit.

Passive Envelope Detector

A envelop detector was designed to extract the envelope signal from carrier signal. In this, carrier signal across the reader coil was first rectified using a diode and the high frequency signal was suppressed by the RC network shown in Fig.2.6. Finally this signal was passed through a buffer to reduce the DC signal.

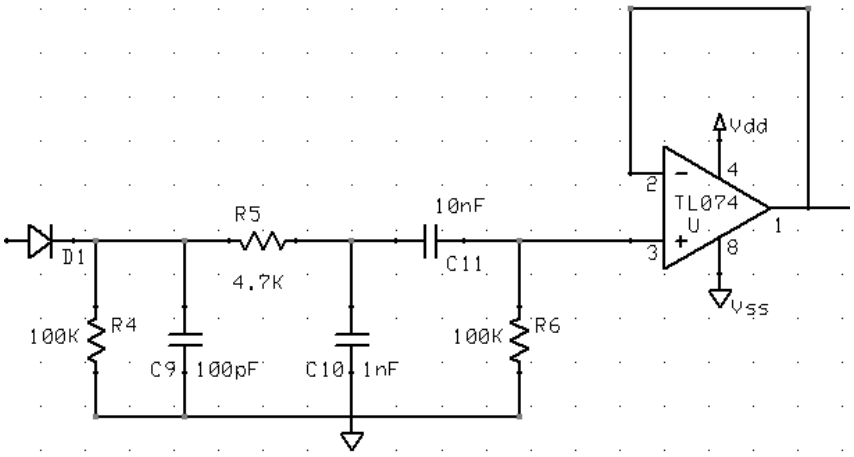


Figure 2.7 Circuit diagram of passive envelope detector.

.

Active Band-pass filter

To eliminate all the frequencies other than information signal frequency, an active band pass filter was designed as shown in Fig.2.3. This unity gain Butterworth band-pass filter was designed for frequency range of 500Hz to 75 kHz.

Non Inverting Amplifier

Hence obtained information signal was amplified to required voltage level using non inverting amplifiers as shown in Fig. 2.4.

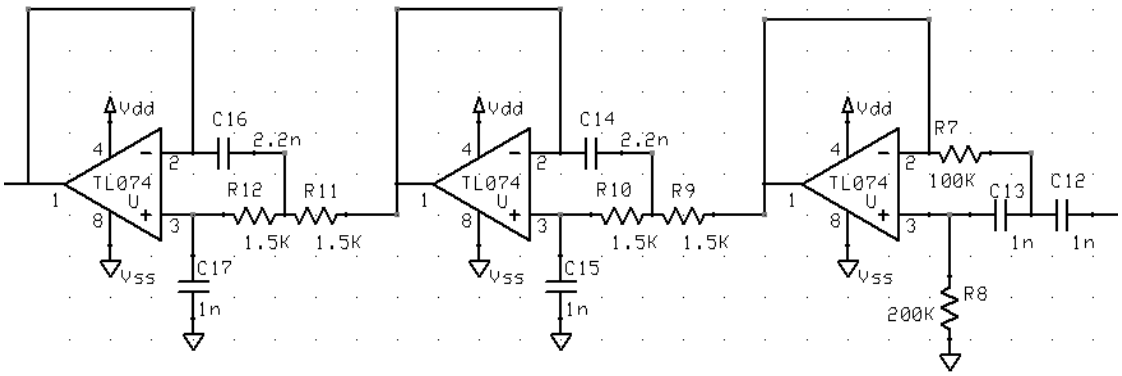


Figure 2.8 Circuit diagram of band-pass filter.

Comparator

This was basically designed to convert the analog signal obtained to digital signal for further processing by microcontroller.

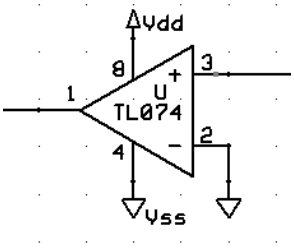


Figure 2.9 Interfacing of microcontroller with reader circuit.

CHAPTER 3

DATA PROCESSING AND RECORDING USING MICROCONTROLLER

The digital signal output of demodulating circuit was connected to a microcontroller for data processing and recording to SD card.

3.1 Microcontroller Hardware

A microcontroller was programmed to process the output frequency from the demodulating circuit. After processing the frequency signal it was recorded on SD card along with its corresponding time in a text (.TXT) file format. For this purpose a digital signal processing microcontroller from Microchip, dsPIC30f4013 was chosen. This microcontroller along with SD Card is called "Processing and Recording Unit" as shown in Fig. 3.1.

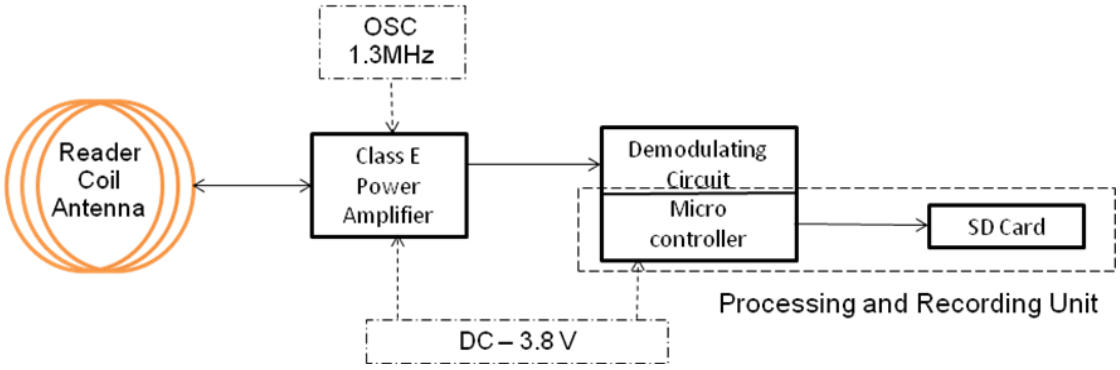


Figure 3.1 Recording and Processing unit of BEST™.

The desired features of microcontroller for choosing dsPIC30f4013 are:

1. 40 pin, PDIP package
2. Low-power consumption
3. Wide operating voltage range (2.5v to 5.5v)
4. 16-bit timers/counters
5. SPI module

6. C compiler optimized instruction set architecture

The circuit of the processing and recording unit is basically

1. Microcontroller circuit
2. Microcontroller circuit interfaced with SD card

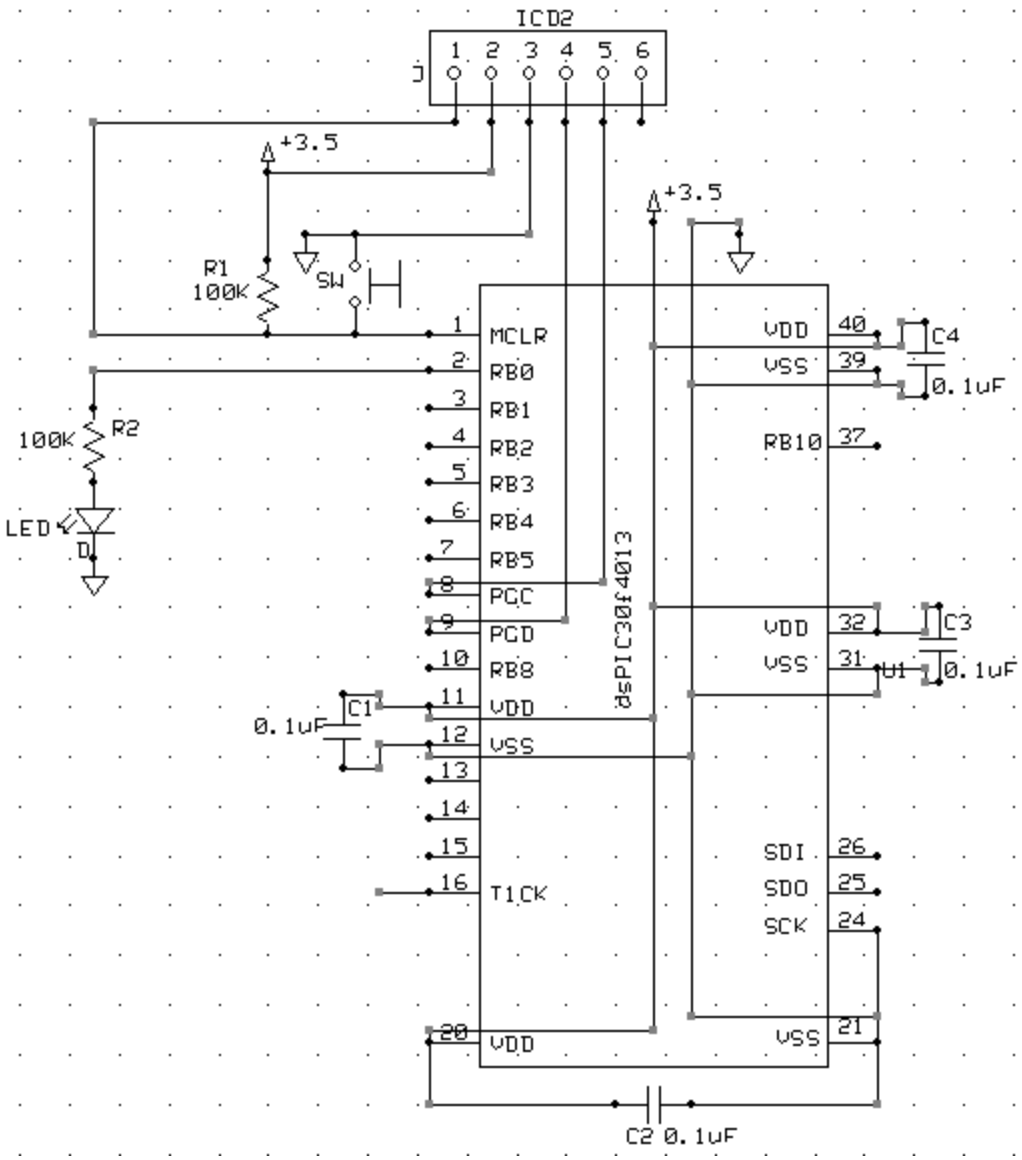


Figure 3.2 Circuit diagram of microcontroller

3.1.1 Microcontroller Circuit

The microcontroller circuit was designed as shown in Fig.3.2 with a supply voltage of 3.8V and storage capacitors of 0.1uF. For on-board programming a MPLAD ICD2 connection was provided. A RESET switch was provided to reset the system, a red LED was connected to pin 2 to indicate working of program to the user, and a yellow LED was connected to indicate power supply.

The output of last stage op-amp of demodulating circuit was connected to T1CK pin (pin 14) of microcontroller. This connection was connected though a ME switch (Memory Erase Switch). The purpose of ME switch was to erase memory by keeping it open and resetting the system.

3.1.2 Microcontroller and Secured Digital Card Interface

To record the processed data from microcontroller, a 16MB Sandisk SD card was chosen. Some main system features are:

- Up to 16MB of data storage.
- SD Card protocol compatible.
- Supports SPI Mode.
- Secured and non-secured data storage.
- Voltage range:
    - Basic communication (CMD0, CMD15, CMD55, ACMD41): 2.0—3.6V.
    - Other commands and memory access: 2.7—3.6V.
- Variable clock rate 0—25 MHZ.
- Up to 12.5 MB/sec data transfer rate (using 4 parallel data lines).
- Correction of memory field errors.
- Password protected of cards (not on all models).
- Write protect feature using mechanical switch.
- Built-in write protection features (permanent and temporary).

- Card detection (Insertion/Removal).
- Application specific commands.

SD card was connected to microcontroller using a 9-pin connector in SPI mode as shown in Table 3.1. The serial peripheral pins of microcontroller were connected to serial communication pins (Data in & Data out) of SD card as shown in Fig.3.3. Power supply to SD card was also provided by pin 3 of microcontroller. For SD card selection, pin 5 of microcontroller was connected to SD card chip select pin.

Table 3.1 Pins of SD card.

Pin #	Name	Type	SPI Description
1	Chip Select (CS)	Input	Active low
2	Data In	Input	Data and commands from microcontroller
3	VSS1	Supply	Supply voltage ground
4	VDD	Supply	Supply Voltage (3.8V)
5	Clock (CLK)	Input	Clock signal from microcontroller serial port to synchronize communication
6	VSS2	Supply	Supply voltage ground
7	Data Out	Output	Data transfer from SD card to microcontroller
8	Reserved		
9	Reserved		

According to the power requirements shown in Table 3.2, supply voltage of the whole system was chosen as 3.8V (Maximum withstand voltage of SD card is 3.8V). Table 3.2 also shows current drawn during different modes of operation. As stated in Table 3.2, a 100K Ohm resistor was connected to Data Out pin to prevent bus floating. Storage capacitors 3.3uF and

0.1uF are connected in parallel to supply voltage (3.8V) from microcontroller, to avoid any voltage fluctuations.

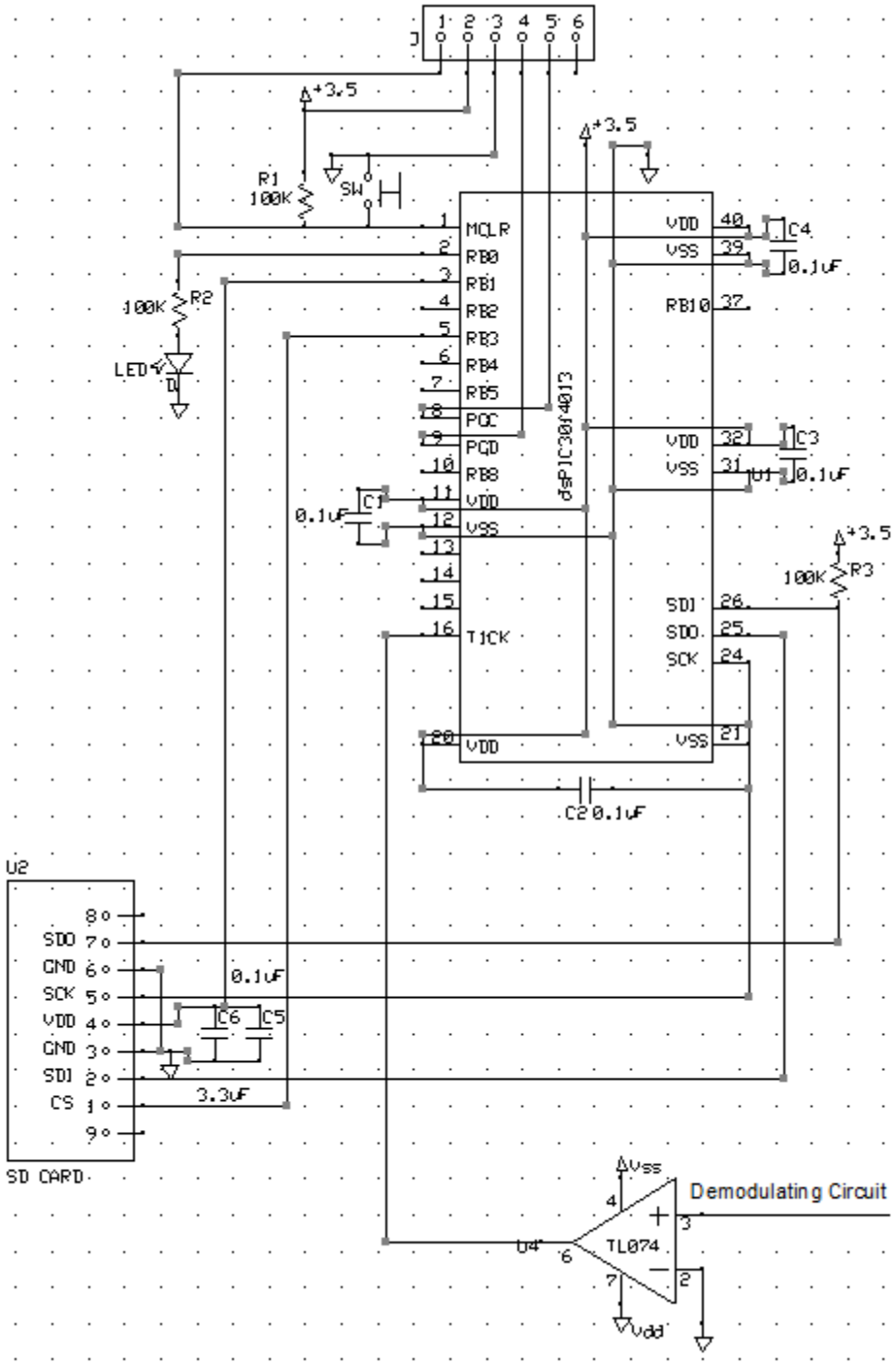


Figure 3.3 Circuit diagram of microcontroller interfaced to Secured Digital card.

Table 3.2 Power requirements of SD card

VDD (ripple: max, 60 mV peak to peak)	2.7 V – 3.6 V

(Ta = 25°C @3 V)

	Value	Measurement	Notes
Sleep	250	uA	Max
Read	65	mA	Max
Write	75	mA	Max

Table 3.3 Resistors and capacitors required for SD card interfacing with microcontroller

Parameter	Symbol	Min.	Max.	Unit	Remark
Pull-up resistance	R_{CMD} R_{DAT}	10	100	kΩ	To prevent bus floating
Bus signal line capacitance	C_L		250	pF	$f_{PP} \leq 5$ MHz, 21 cards
Bus signal line capacitance	C_L		100	pF	$f_{PP} \leq 20$ MHz, 7 cards
Single card capacitance	C_{CARD}		10	pF	
Maximum signal line inductance			16	nH	$f_{PP} \leq 20$ MHz
Pull-up resistance inside card (pin 1)	R_{DAT3}	10	90	kΩ	May be used for card detection

3.2 Microcontroller Software

The following development tools were used to program the dsPIC30f4013 microcontroller.

Programming Language: C programming

Language Tool-suite for C Compiler: Microchip C30 Tool-suite

Development software: MPLAB IDE v8.10

Programmer: MPLAB ICD2

The dsPIC30f4013 microcontroller was programmed to perform following functions:

1. Frequency counting
2. Data processing
3. Frequency recording on SD card
4. Timing control

5. Sensor position detection
6. Battery charging Indication
7. Memory Erase
8. Read/ Busy Indication

Microcontroller Configuration

In the program, p30f4013.h header file was included and the following configuration of microcontroller, shown in Table 3.4 was chosen. The corresponding code to this is given in Code 1 (Appendix A).

Table 3.4 Microcontroller configuration settings

CATEGORY	**SETTING**
Oscillator	7.37 MHz Internal RC oscillator; 8x PLL
Fail-Safe Clock Monitor (FSCM)	Disabled
Clock Switching	Disabled
Watchdog Timer (WDT)	Disabled
Programmable Brown-out Reset (PBOR)	Disabled
POR Timer Value	64ms
Brown out Voltage	2.0V
Master Clear	Enable
General Code segment write protect	Disabled
General segment code protection	Disabled
Communication Channel Select	PGC & PGD

Table 3.5 Microcontroller pin definitions

PINS	DEFINITION
RB0	LED
RB10	FG
RB1	SD_PWR
RB3	SD_CS
RF2	SDI
RF3	SDO
RF6	SCK

Microcontroller pin definition

Microcontroller pins were defined in the program as shown in Table 3.5. The corresponding code to this is given in Code 2 (Appendix A).

3.2.1 Frequency Counter

As mention in Sec.3.1.1, frequency signal was fed to pin 16 of microcontroller for frequency counting. This signal frequency can measured either by logic 1 or logic 2 discussed below.

Logic 1: Count the number of pulses in 1sec (1000msec) as shown in Fig.3.4

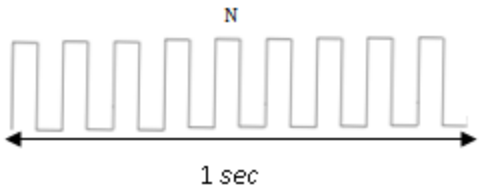


Figure 3.4 Frequency counting using logic 1.

If N is number of pulses in 1 sec

=> Frequency (f) = N Hz

Logic 2: Count the number of pulses in 100msec and the measured number was multiplied by 10 to get frequency of the signal as shown in Fig.3.5.

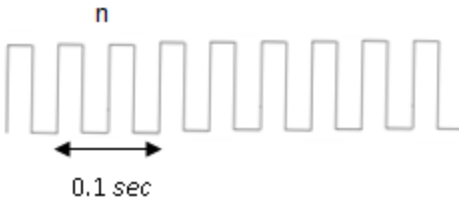


Figure 3.5 Frequency counting using logic 2.

If n is number of pulses in 0.1 sec

=> Pulses in 1 sec is 10n

=> Frequency (f) =10n Hz

The logic 2 was implemented for faster response and minimum operation time. This means logic 2 updates frequency for every 100ms while logic 1 updates for every 1 sec. So, by using logic 1 measurement is 10 times faster than using logic 2 (though there is tradeoff with accuracy, faster response is more desired).

The information signal frequency counting was realized by using two timers Timer 1 and Timer 2.

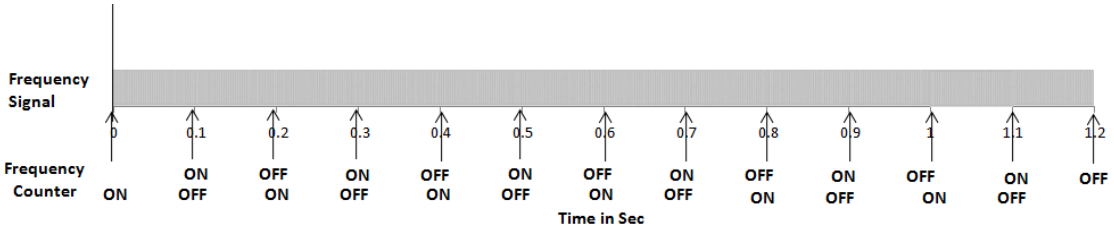


Figure 3.6 Frequency counter using timer 1 and timer 2.

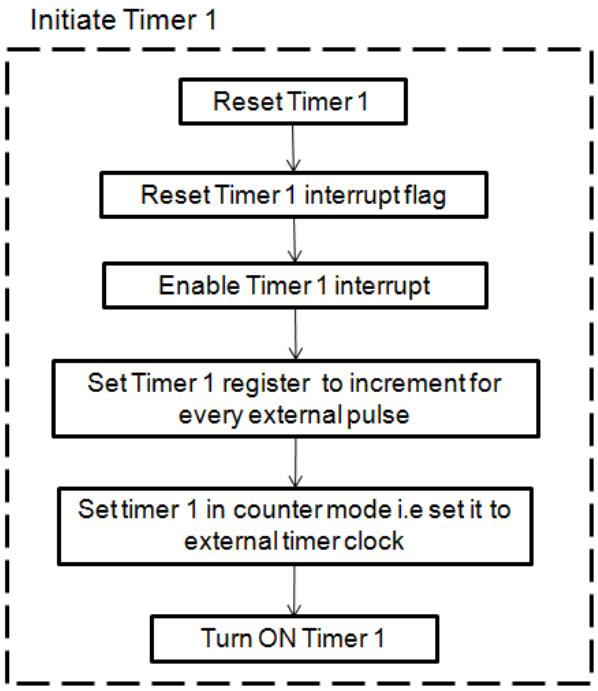


Figure 3.7 Flow chart of timer 1 initiation.

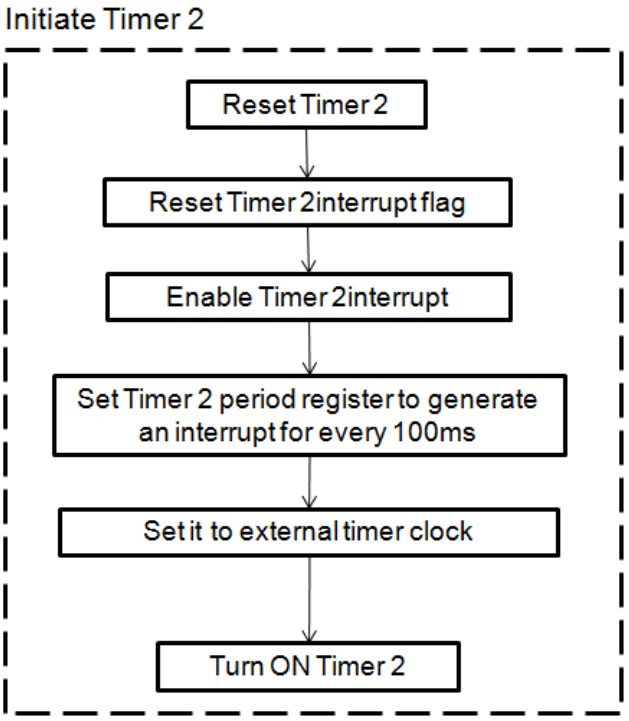


Figure 3.8 Flow chart of timer 2 initiation.

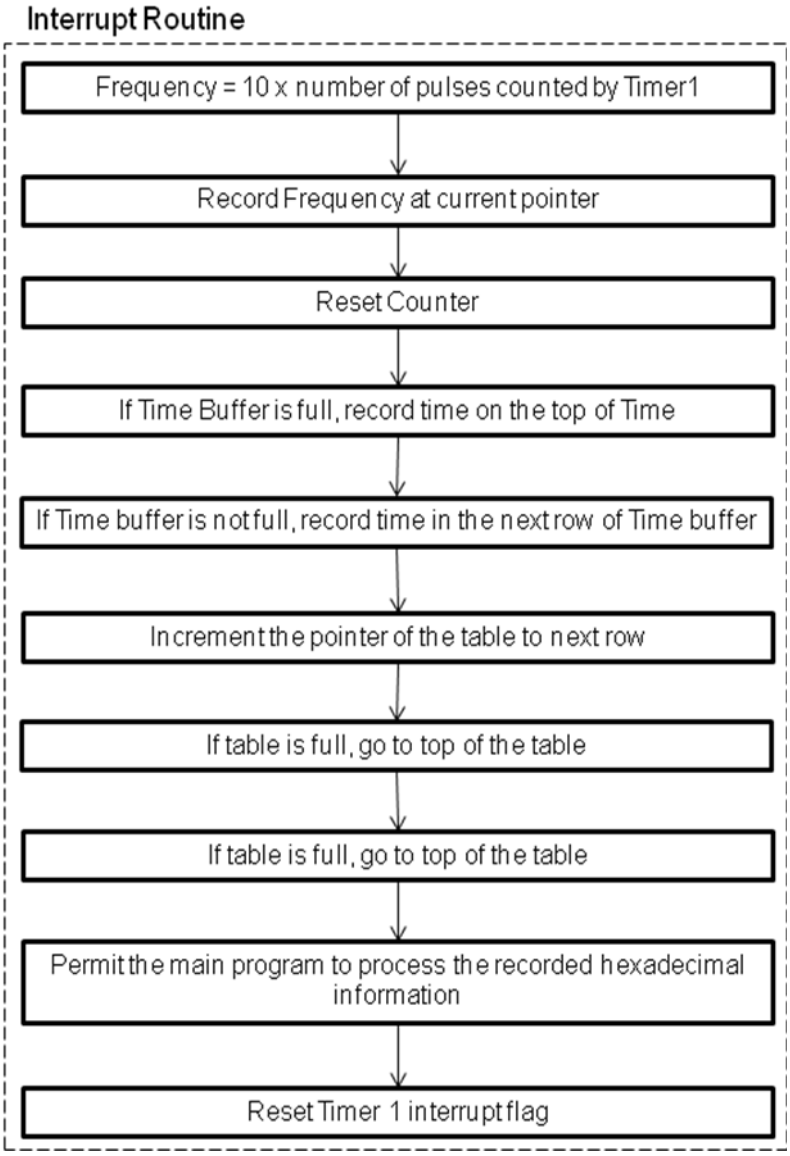


Figure 3.9 Flow chart of Interrupt Routine.

Timer 1: This was initiated as a counter by resetting interrupt flag, set to increment for every pulse. The flow chart to initiate Timer 1 is shown in Fig.3.6 and corresponding code is given in Code 3 (Appendix A).

Timer 2: It was initiated as a timer to interrupt for every 100ms by resetting timer and interrupt flag, setting period register to generate interrupt. The flow chart of Timer 1 is shown in Fig.3.7 and corresponding code is given Code 4 (Appendix A).

Data Table

m	Time (ms)	Frequency (Hz)
0	100	7150
1	200	7110
2	300	7210
. . . . .	. . . . .	. . . . .
47	400	7750
48	500	7740
49	600	7890

Figure 3.10 Unprocessed Data Table.

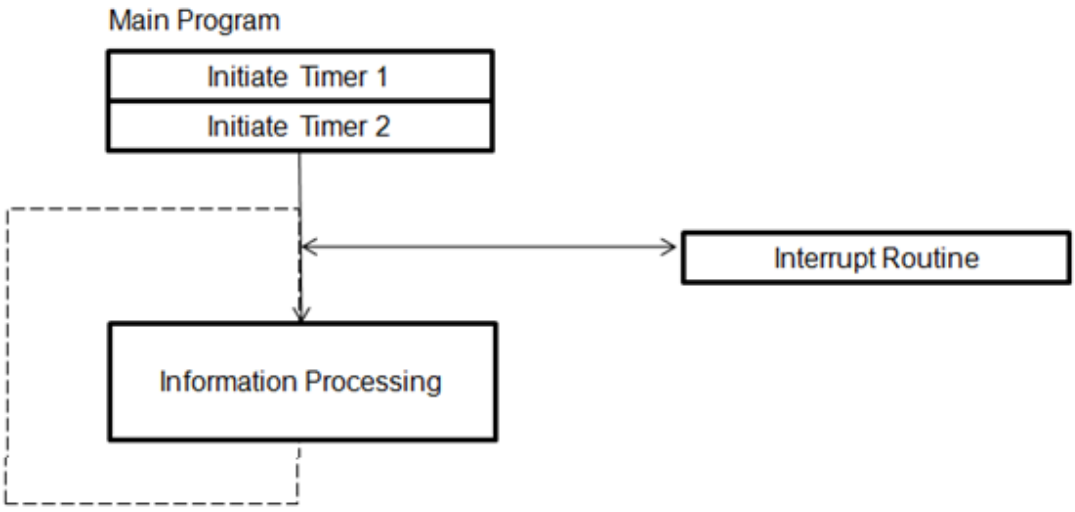


Figure 3.11 Complete working of frequency counter with all blocks together.

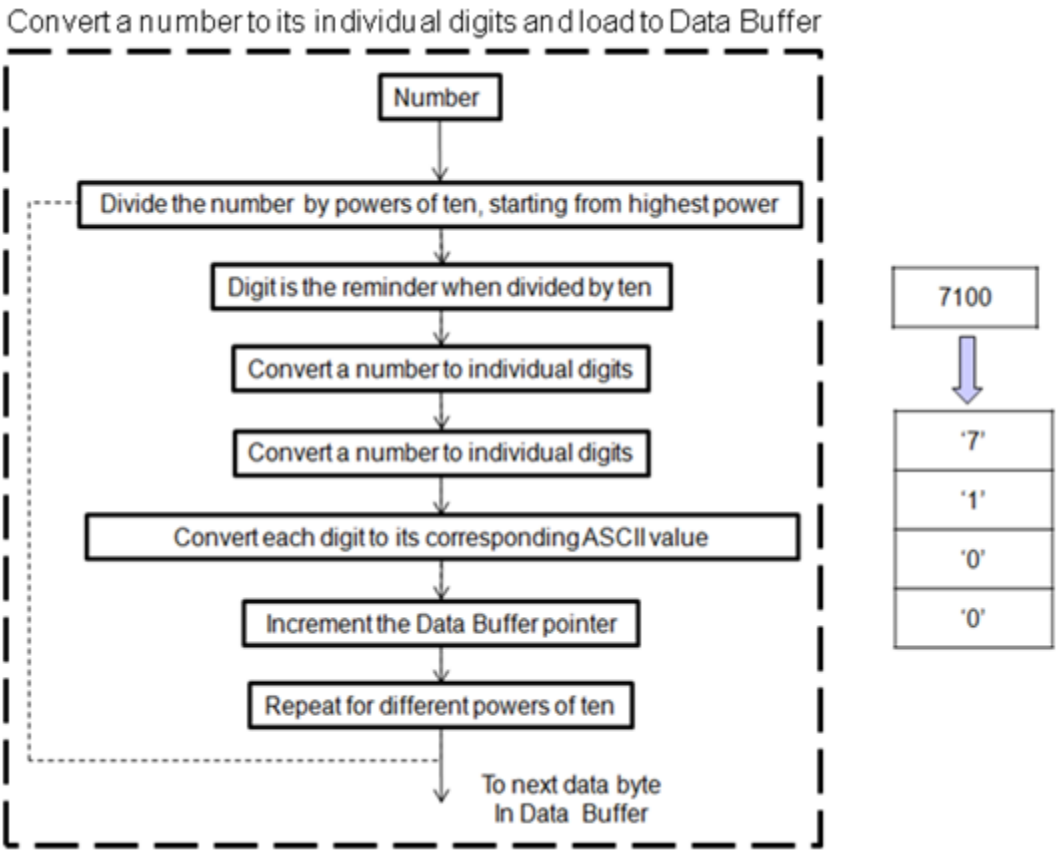


Figure 3.12 Conversion of a number to its ASCII digits flow chart.

Interrupt Routine

For measuring the frequency of the signal, Timer 1 was turned ON and Timer 2 (Counter) was turned ON immediately. After 100msec in the interrupt routine, the number of pulses (n) counted by Timer 2 was multiplied by 10(f = 10n) to determine the actual frequency of the signal. Later this frequency with its corresponding time was transferred to Unprocessed Data table as shown in Fig.3.10. This Data table was made using arrays. Once this data table was filled in interrupt routine, the program pointer was returned to its last point in the main/function program. The flow chart of Interrupt Routine is shown in Fig.3.9 and corresponding code is given Code 5 (a) (Appendix A).

The complete working of frequency counter is shown in Fig.3.11 and corresponding code is given Code 5 (Appendix A).

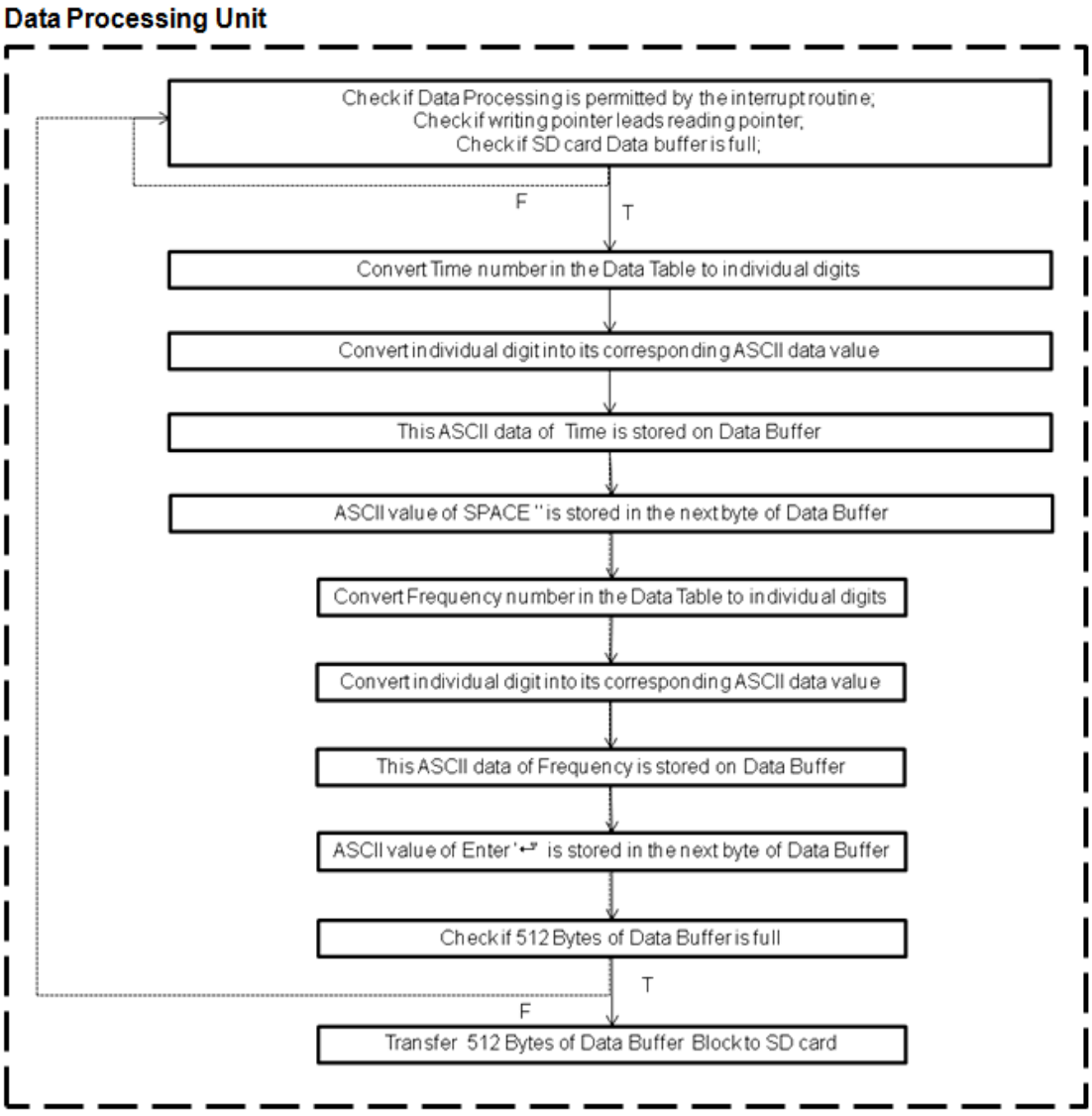


Figure 3.13 Flowchart of complete data processing.

3.2.2 Data processing unit

Here the data recorded on unprocessed data table was processed for the following purposes:

1. Keep a 512 bytes data block (data buffer) ready to transfer to SD card
2. Convert data in Data Table to data that can be displayed on a TEXT(.TXT) file

A 512 byte data buffer was created as a global unsigned character array. The Time and Freq (Frequency) headings were preprocessed in the main program before timers initialization. The loading of these headings to Data Buffer is given in Code 6 (Appendix A).

Microcontroller was programmed to start data processing once it received a permit from interrupt routine and a new value was found in the Data Table. This permit was cleared once data processing started to avoid reentrancy. The logic flow shown in Fig.3.12 was followed for converting data in Data Table to TEXT file displayable ASCII data.

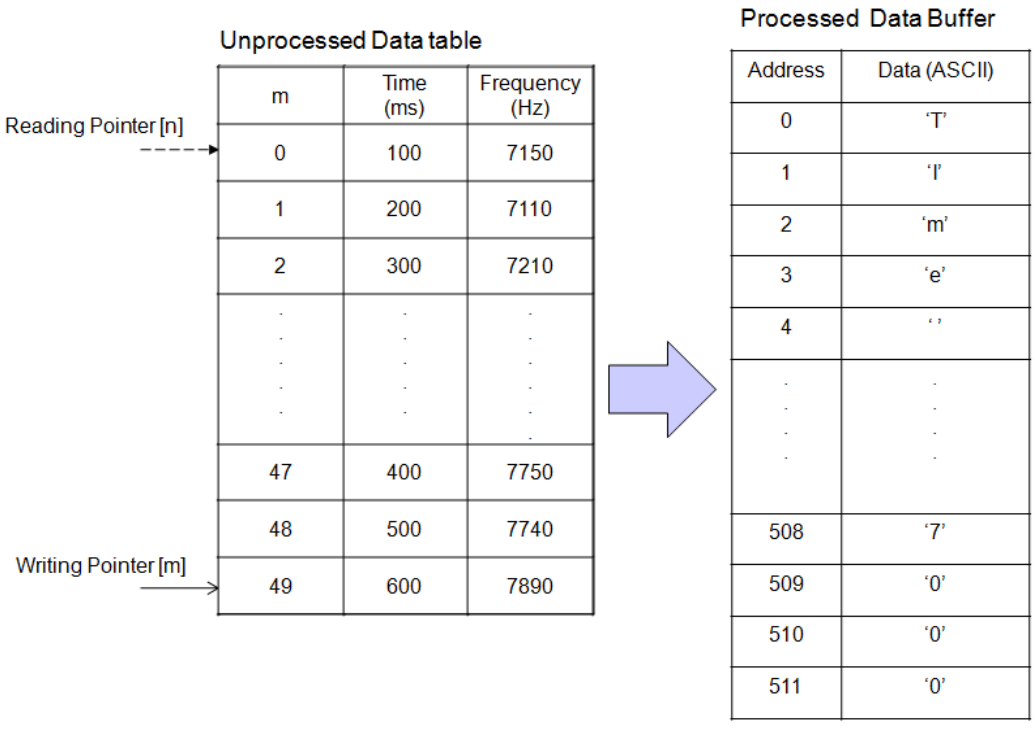


Figure 3.14 Conversion of unprocessed data table to processed data buffer using data processing.

In this logic every number was divided by the powers of ten and their reminder were collected. These reminders were then added to integer number 30 to get ASCII values of the number. This was first done to Time data and then to frequency data in data table. These ASCII data bytes were loaded to 512 byte Data Buffer mentioned above, but with a SPACE ‘ ’ byte between Time ASCII data and Frequency ASCII data to display space in the final TEXT file. This was repeated for whole Data Table and every time this was done, a ENTER ‘↵ ‘byte was loaded to Data Buffer to display data on the next line in the final TEXT file. Once this 512 bytes of Data Buffer was about to full, two ENTER ‘↵ ‘bytes were loaded to skip next line. This whole data processing was shown in Fig 3.13 and corresponding code is given in Code 7 (Appendix A). An example of a Data Buffer converted from a Data Table using Data processing is shown Fig.3.14

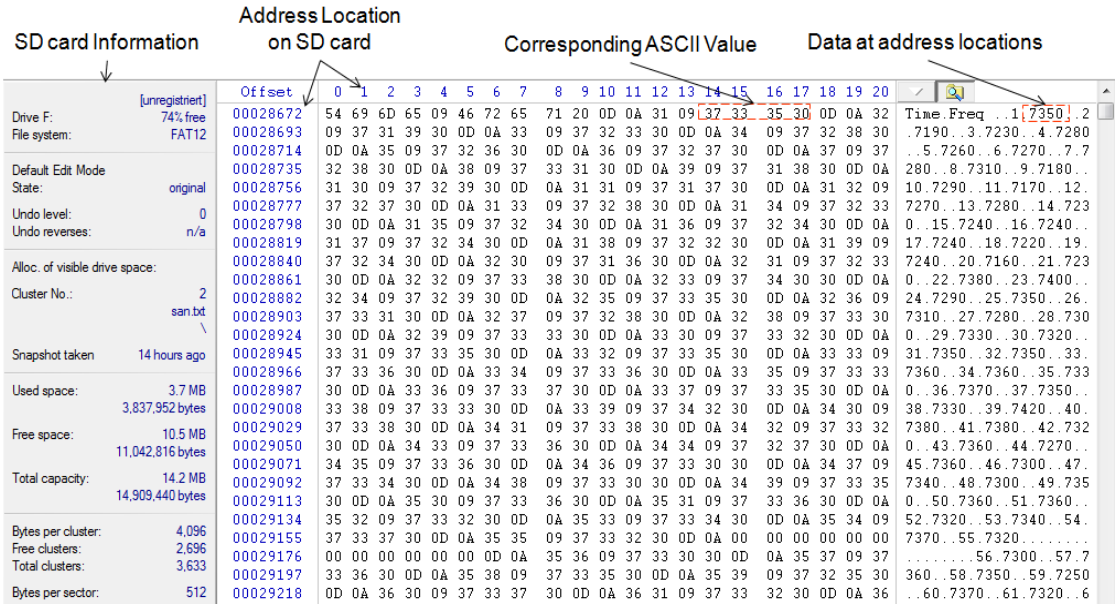


Figure 3.15 A TEXT file on SD card opened on computer using WinHex.

.

3.2.3 Data recording to Secured Digital card

A Serial Peripheral Interface (SPI) communication mode was setup between microcontroller and SD card. The 512 byte data kept ready in Data Buffer was sent to a desired address location (here it is 0028672) on SD card. This address location was determined by saving a TEXT file to SD card and reading memory of the same, using WinHex software.

Fig.3.15 shows the memory map of TEXT file on SD card using WinHex. This software shows address locations and ASCII data which is on SD card memory. It also shows the data displayed on TEXT file on the extreme right of the Fig.3.15.

After recording the data to SD card a new set of data was kept ready by the data processing unit, repeating the whole processing again and again till the device was RESET or turned OFF. But after every recording to SD card, the memory address of the card was increased by 512 to transfer data to next memory location.

A sample recorded TEXT file on SD card with frequency measurement in shown in Fig.3.16.

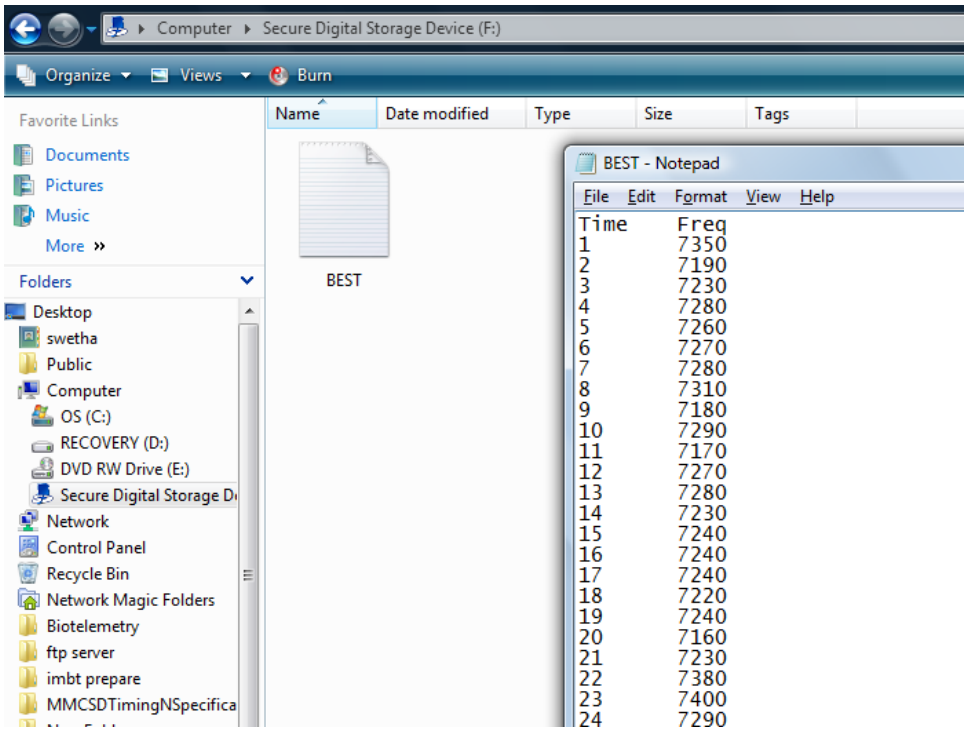


Figure 3.16 A recorded TEXT file (named as BEST) on Secured Digital card .

3.2.4 Timing control

The class E amplifier consumes a great extent of power as it has to power up the batteryless sensor. So there a lot of power can be saved if the class E amplifier switching timings are controlled. Depending upon power transferred to batteryless sensor continuously or intermittently, there are two types of controls.

1. Open loop control
2. Closed loop control

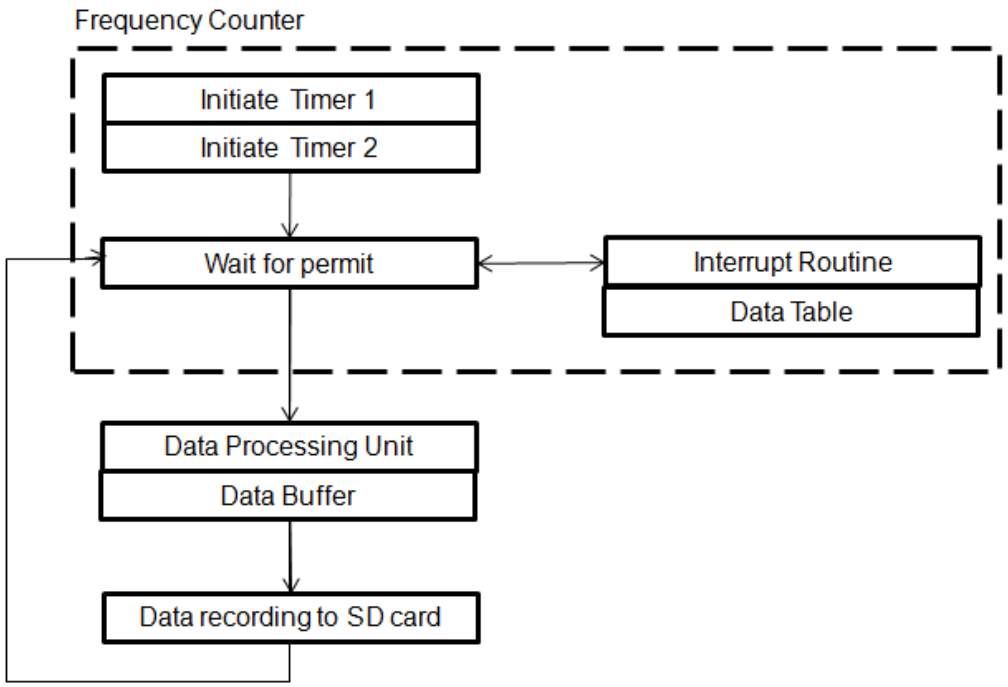


Figure 3.17 Flow chart of open loop control

3.2.4.1 Open Loop Control

Our discussion so far was based open loop control, in which complete signal flow was unidirectional as shown in Fig.3.18. A 1.3MHz carrier signal was generated from oscillator (OSC) which was modulated with information signal from sensor using inductive coupling. This modulated signal was demodulated using demodulating circuit designed. A microcontroller was

programmed to receive this signal, process and record it on a SD card. Here frequency of the signal was measured continuously without any pause, as oscillator was not controlled. This led to very high power consumption in open loop control. The complete flow chart of Open Loop Control by integrating all the blocks discussed so far is shown in Fig.3.17. Corresponding code for this control is given in Code 8 (Appendix A). The timings of different signals are shown in Fig.3.19 and complete circuit diagram in open loop is shown in Fig.3.20.

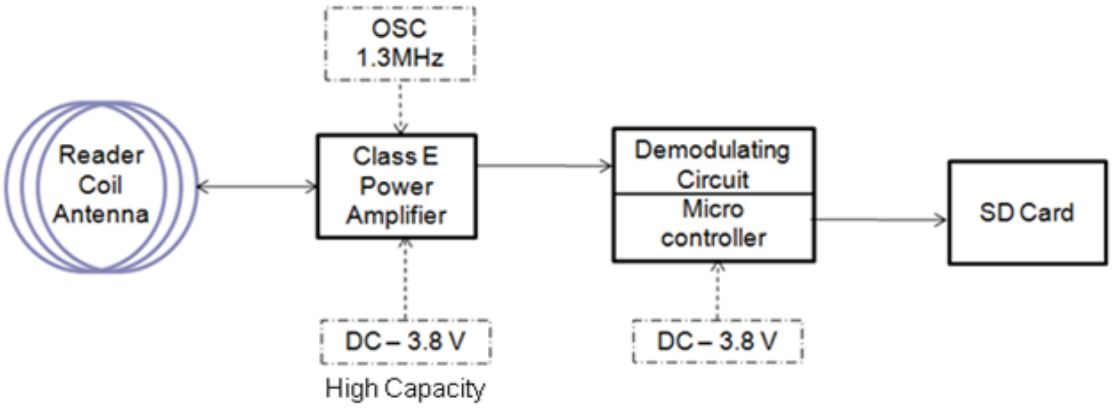


Figure 3.18 BEST^{TM} in open loop control.

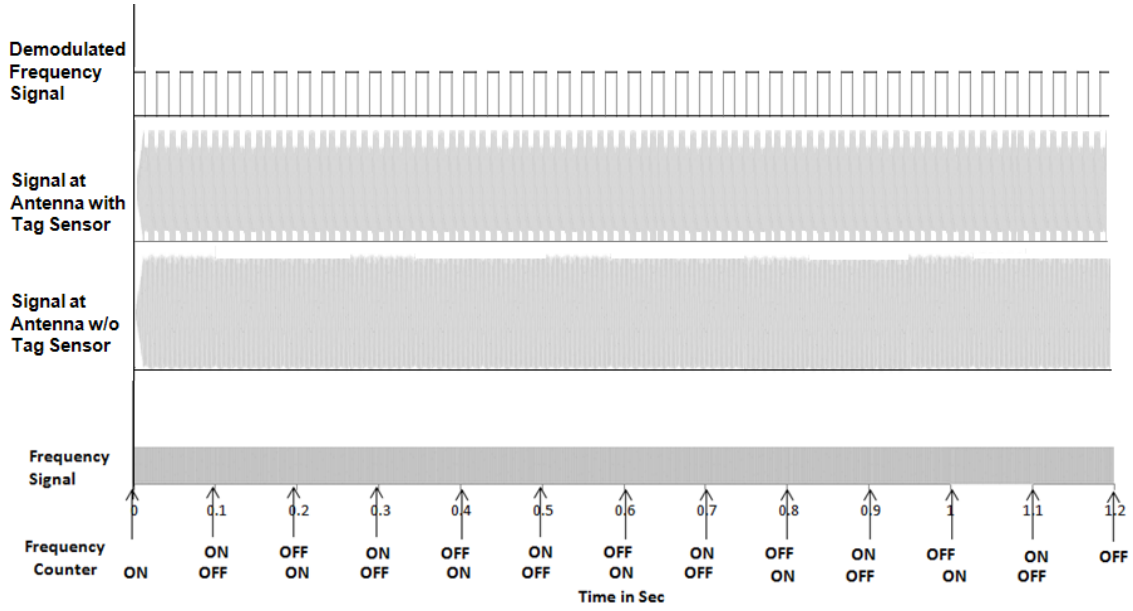


Figure 3.19 Different signal timings in open loop control.

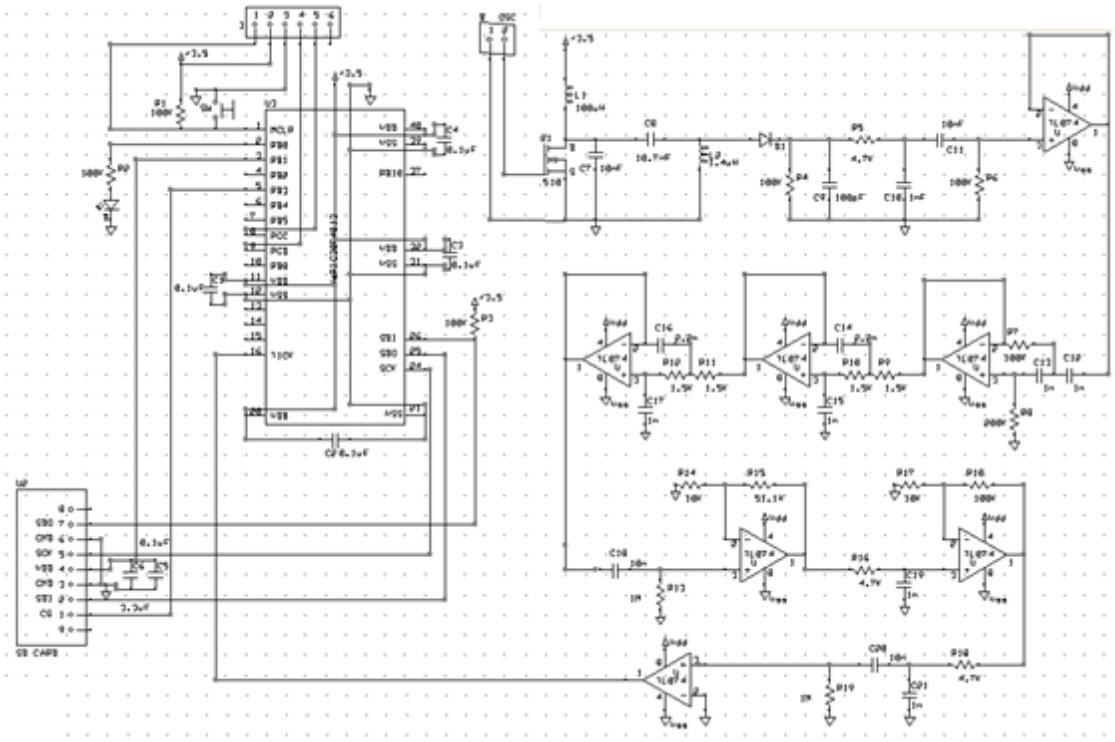


Figure 3.20 Complete circuit diagram of BEST^{TM} in open loop.

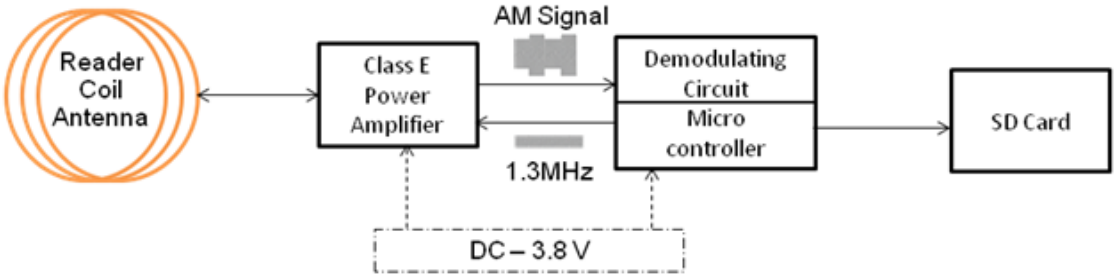


Figure 3.21 BEST^{TM} in closed loop control.

3.2.4.2 Closed Loop Control

To save power and have better control of device, closed loop control was introduced as shown in Fig.3.21. Here the oscillator block was completely removed and microcontroller was programmed to generate 1.315MHz carrier frequency. Microcontroller was programmed to

power up the reader coil antenna and received the information signal from demodulating circuit. This signal was processed and recorded on SD card similar to open loop control. The main advantage of this control was reduction in power consumption. This was achieved by introducing some delay in the carrier signal which powers up the antenna. So it was programmed to work for 100ms and 900ms delay (pause) time. The complete flow chart of closed loop control with all the blocks discussed so far together is shown in Fig 3.22. Corresponding code for this control is given in Code 9 (Appendix A). The timings of different signals are shown in Fig.3.23 and complete circuit diagram in open loop is shown in Fig.3.24.

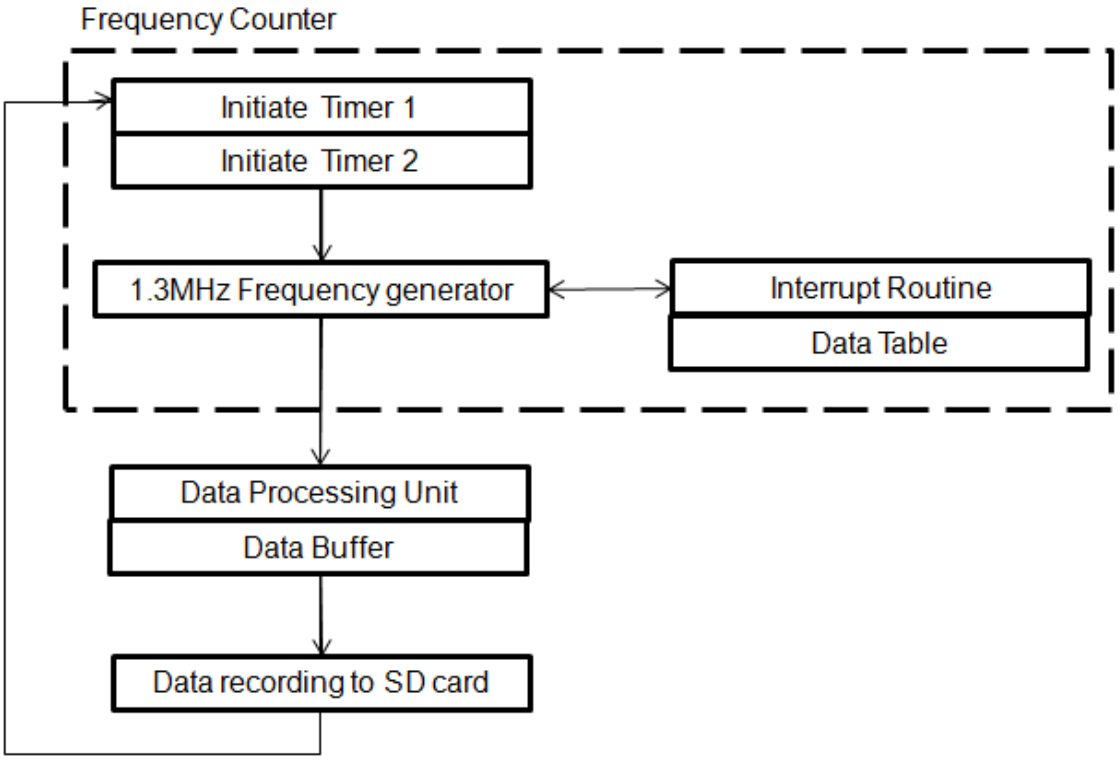


Figure 3.22 Flow chart of closed loop control.

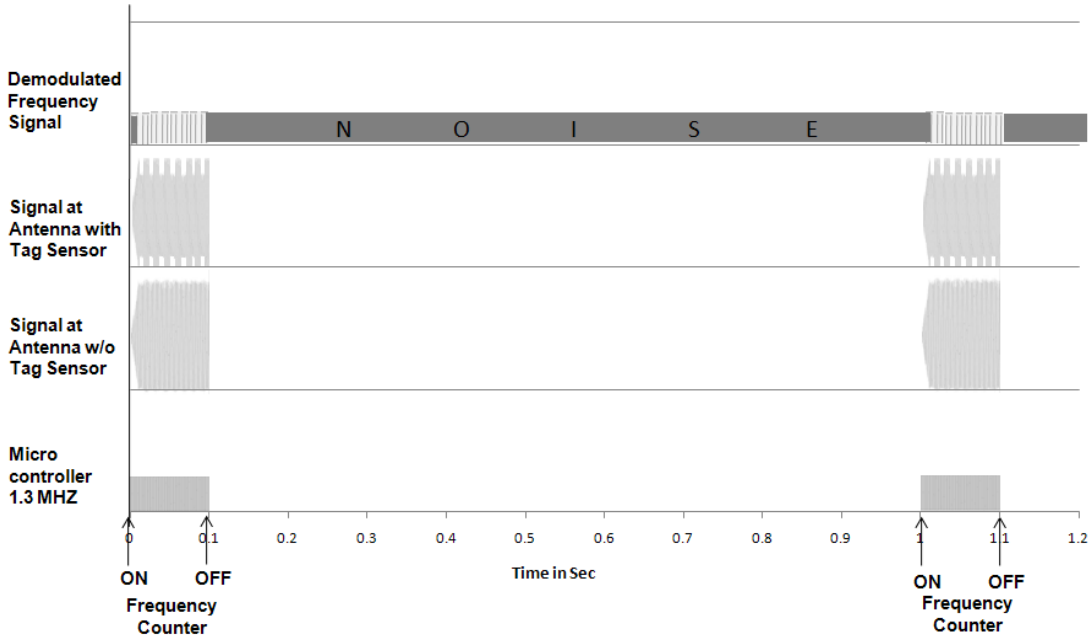


Figure 3.23 Different signal timings in closed loop control.

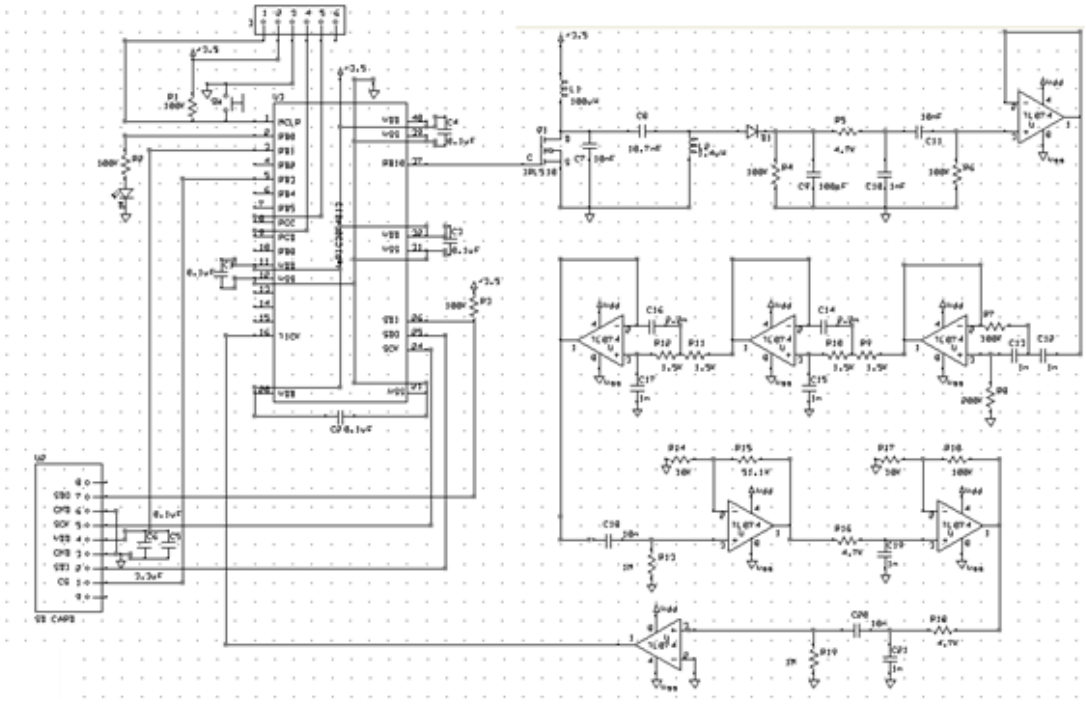


Figure 3.24 Complete circuit diagram of BEST^{TM} in closed loop.

3.2.5 Sensor position detection

Once sensor is implanted, to avoid misalignment between sensor and BEST™, this additional feature was included. Here microcontroller was programmed to detect the out of range frequencies or Noise and turn ON an indicator. The sensor position detect code is given in Code 10 (Appendix A) and its corresponding flow chart is shown in Fig.3.25.

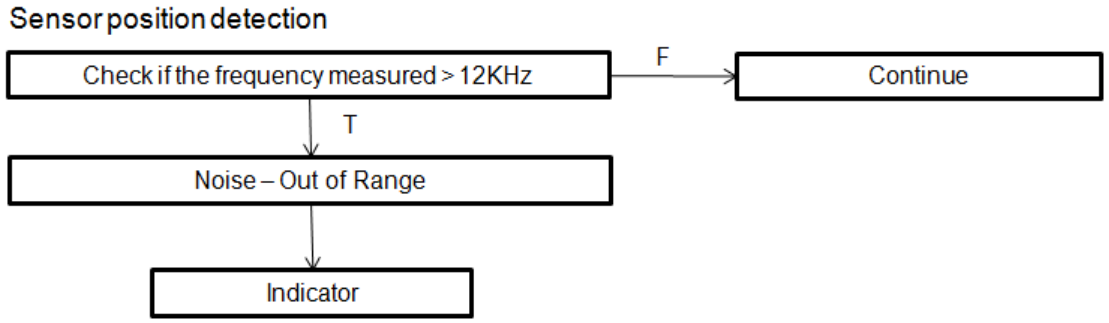


Figure 3.25 Flow chart of sensor position detection.

3.2.6 Battery Charging Indication

This is also an additional feature provided on BEST™. As shown in Fig. 3.26 if there is no battery the frequency goes below 7 KHz. Based on this, microcontroller was programmed to turn ON an indicator, in case of battery outage / improper battery connection.

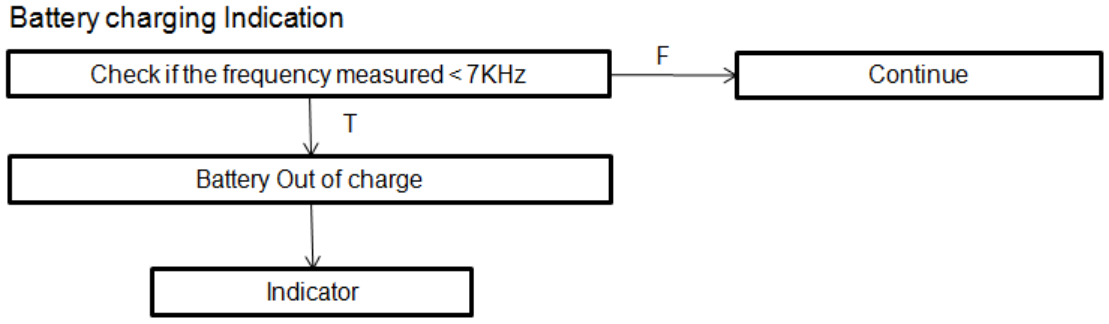


Figure 3.26 Flow chart of battery charging indicator.

CHAPTER 4

COMMUNICATION BETWEEN MICROCONTROLLER AND SECURED DIGITAL CARD

This communication is a part of data recording to SD card discussed in Sec.3.2.3. To transfer the 512 bytes data buffer obtained from data processing to SD card, a communication link was setup between microcontroller and SD card. For this communication a Serial Peripheral Interface (SPI) port on the microcontroller was chosen.

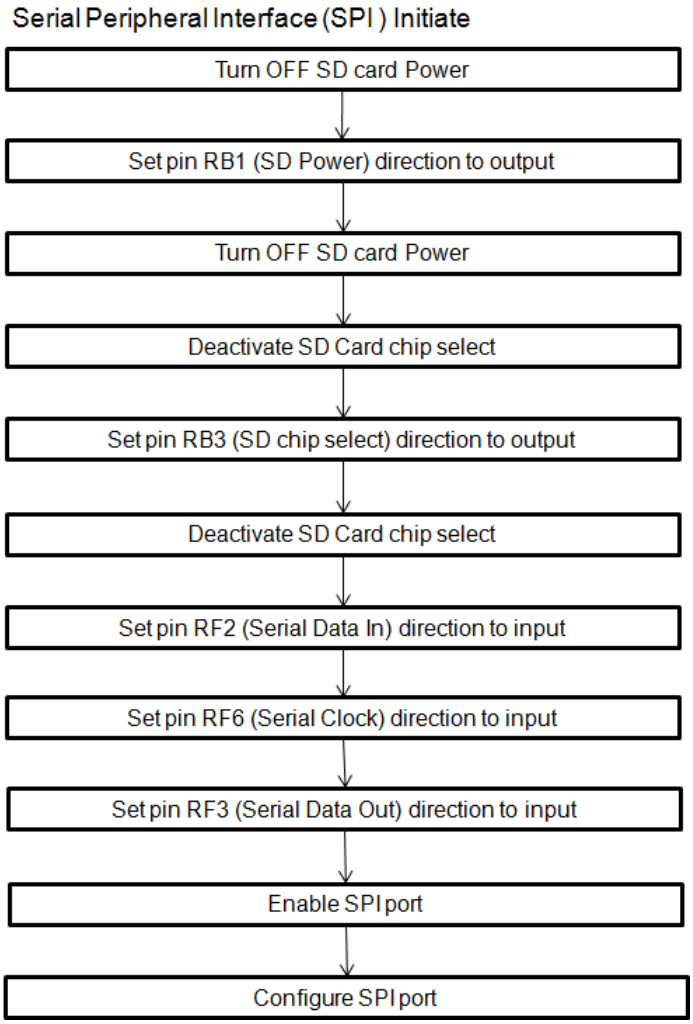


Figure 4.1 Flow chart of SPI initiation on microcontroller.

4.1 SPI Mode Setup on Microcontroller

To initiate communication with SD card, its power was turned OFF and chip select was deactivated, as shown in Fig.4.1. Later appropriate port directions were chosen and the following configuration of SPI was chosen using the microcontroller code 11 (Appendix A)

SPI peripherals Configuration of microcontroller

- Set SPI port to slowest setting
- Master mode
- 8 bit
- Idle state for clock is high level
- Primary prescaler 64:1
- Secondary prescaler 8:1

SD card was initiated as shown in flow chart below (Fig.4.1) and Code 11(Appendix A) to setup the communication link between SD card and microcontroller.

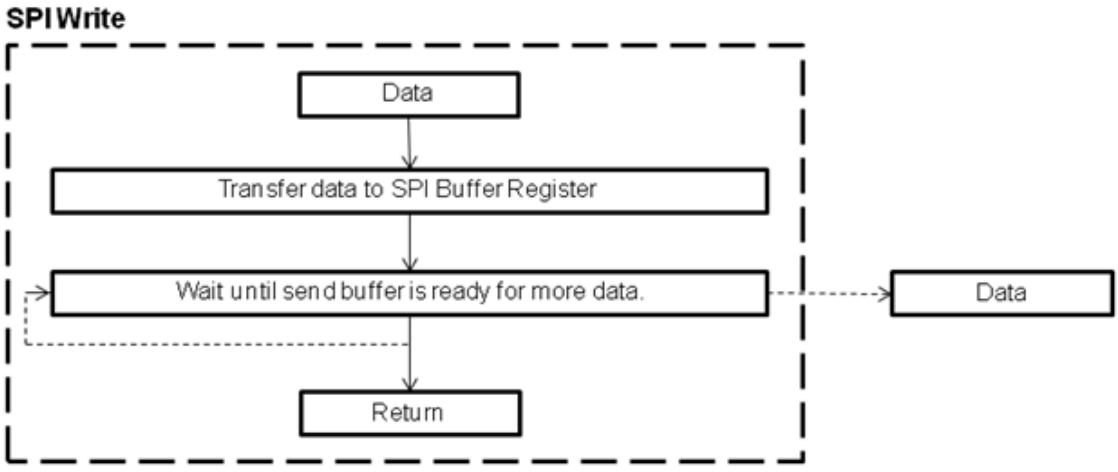


Figure 4.2 Flow chart of data transfer to SD card using SPI Write.

4.1.1 SPI Write

A byte of data can be written to SD card using SPI Write, logic flowchart shown in Fig.4.2. Here data is transferred to SPI buffer register (SPI1BUF) and let the microcontroller wait till data is transferred to SD card. The code used to SPI write data is given in Code 12 (Appendix A).

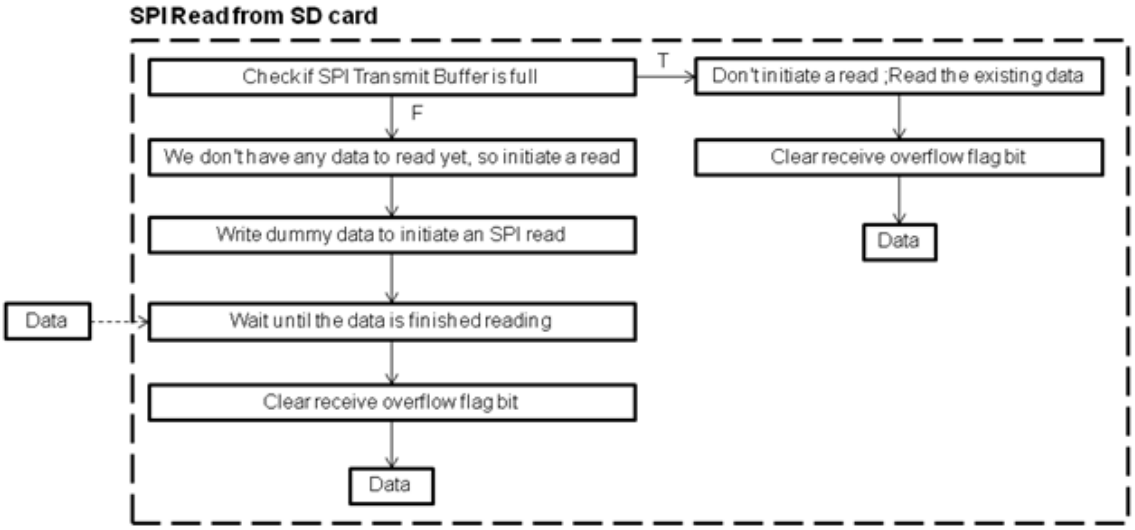


Figure 4.3 Flow chart of data transfer from using SD card SPI Read.

4.1.2 SPI Read

Similarly, using SPI communication a data byte can be read from SD card as shown in SPI Read flowchart (Fig.4.3). Here microcontroller reads the data without initiating a SPI Read, if any data is available already in SPI buffer register. Else, a new Read is initiated by writing a dummy data to register and then let it wait for till buffer is full. In the end this data is returned to main program. The code used to SPI read is shown here.

4.2 SD Card SPI Mode Communication

To communicate with SD card, byte oriented SPI bus protocol was followed. In this protocol commands were sent on Data In/ SDO line and response or data was received on Data out/ SDI line from SD card.

Before going any further to understand SD card SPI mode communication, it is important to know the following (in SPI Mode).

- Command Tokens
- Responses Tokens
- Data Tokens

4.2.1 Command Tokens

A command is a token that starts an operation. It is sent from the microcontroller to a SD card, serially on the Data In / CMD line. This command format is as shown in Fig.4.4.

Byte 1				Bytes 2—5		Byte 6	
7	6	5	0	31	0	7	0
0	1	Command		Command Argument		CRC	1

Figure 4.4 Format of command token.

Here

Command [Byte 1, Bit 5-0] : There are 64 commands that can be sent to SD card from microcontroller. Out of these only few are available in SPI mode and some used for programming here are listed in Table 4.1.

Command Argument [Byte 2-5, Bit 31-0] : Depending upon the function to be performed appropriate command argument corresponding to its command can be chosen from Table 4.1 below.

CRC [Byte 6 Bit 7-1] : Usually CRC for all the commands are ignored, except for CMD0. If required CRC can be activated any time in the program. If activated, the following is the logic

Table 4.1 SD card Commands

CMD	Command Abbreviation	Command Description	Argument	Res.
0	GO_IDLE_STATE	Resets the SD Card	None	R1
1	SEND_OP_COND	Activates the card's initialization process	None	R1
9	SEND_CSD	Asks card to send card-specific data	None	R1
10	SEND_CID	Asks card to send card identification	None	R1
12	STOP_TRANSMISSION	Forces card to stop transmission during multi-block read	None	R1
13	SEND_STATUS	Asks card to send its status register	None	R1
16	SET_BLOCKLEN	Selects block length for all subsequent block commands (default is 512)	[31:0] block length	R1
17	READ_SINGLE_BLOCK	Reads a block of the size specified by SET_BLOCKLEN	[31:0] data address	R1
18	READ_MULTIPLE_BLOCK	Continuously transfers data until interrupted by STOP_TRANSMISSION	[31:0] data address	R1
24	WRITE_BLOCK	Writes a block of the size specified by SET_BLOCKLEN	[31:0] data address	R1
25	WRITE_MULTI_BLOCK	Continuously writes blocks of data until a stop token	[31:0] data address	R1

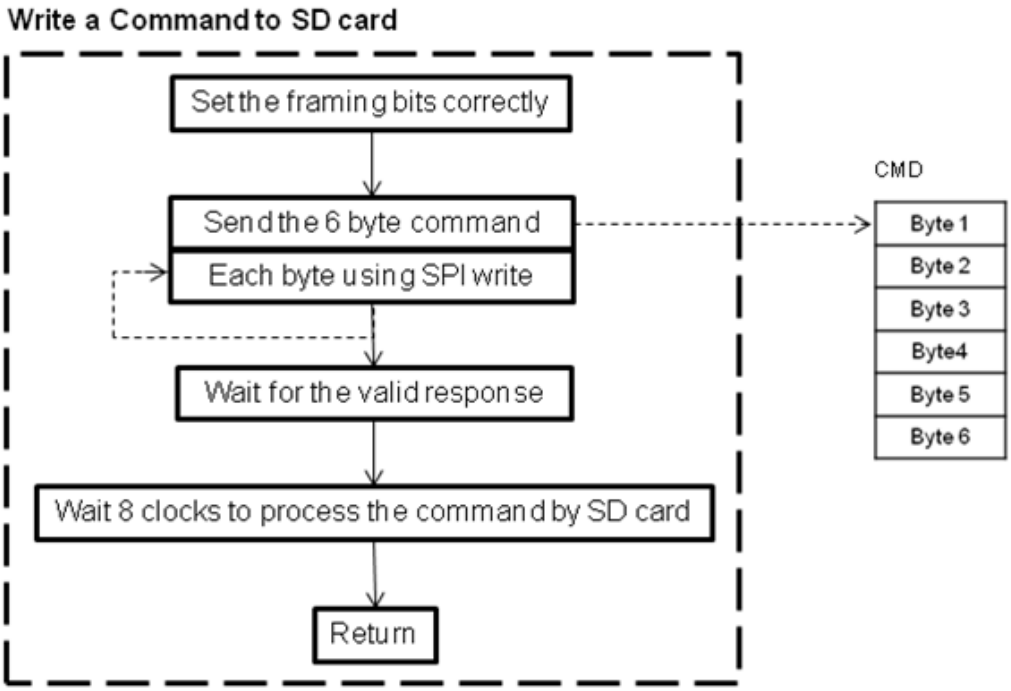


Figure 4.5 Flow chart of writing a command to SD card.

4.2.1.1 Write a command to SD card

To perform various tasks these commands, discussed above are sent to SD card. The framing of these 6 byte commands should be set correctly. Each byte of this command is written to SD card using SPI write discussed in detail, in Sec.4.1.1. Now, program waits for a valid response from SD card (these responses are discussed in detail in next Sec.4.2.1.2). Before returning to main program, a write dummy byte to SD card to provide up to 10 clock cycles time to it, to process the command. The flow chart of writing a command to SD card is shown in Fig.4.5 and corresponding code is given Code 14 (Appendix A).

4.2.2 Response Token

A response is a token that is sent from a SD card, to the microcontroller as an answer to a previously received command. A response is transferred serially on the Data Out line. A typical command and response communication between microcontroller and SD card is shown in Fig.4.6.

In SPI Mode of SD card there are five types of response formats. Those formats are

1. Format R1
2. Format R1b
3. Format R2
4. Format R3
5. Data Respons

Of all these responses only Format R1 and Data response formats were used in the

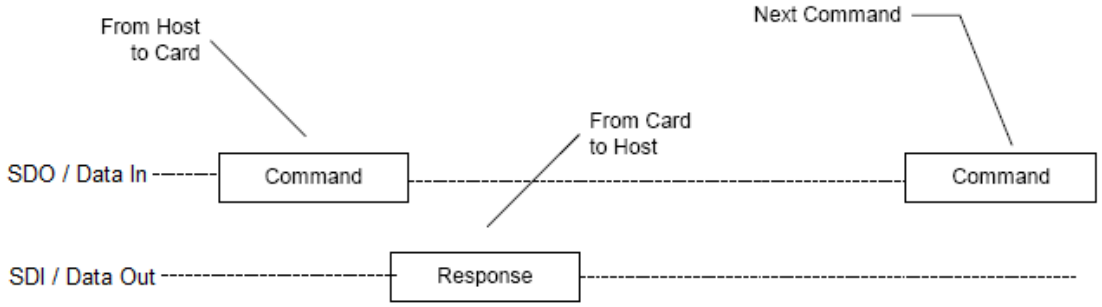


Figure 4.6 SPI mode communication between microcontroller and SD card.

Format R1

This is an automatically generated 8 bit (1byte) response for every command sent (except for SEND_STATUS commands).

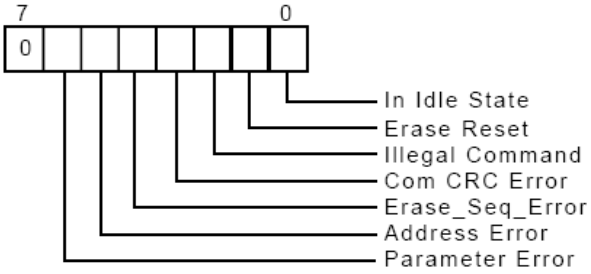


Figure 4.7 Bit format of Response R1.

Here

Bit	Error	Description
0	In idle state	The card is in idle state and running initializing process
1	Erase reset	An erase sequence was cleared before executing because an out of erase sequence command was received
2	Illegal command	An illegal command code was detected.
3	Communication CRC error	The CRC check of the last command failed
4	Erase sequence error	An error in the sequence of erase commands occurred
5	Address error	A misaligned address, which did not match the block length was used in the Command
6	Parameter error	The command's argument (e.g., address, block length) was out of the allowed range for this card
7	-	Always low ('0')

The bit format of Response R1 is shown in Fig.4.7. If there is an error, the bit corresponding to the error is set high ('1'). These R1 response bits are defined in the program as given in code 15 (Appendix A). These bits are compared with received response from SD card to detect its status.

Data Response

This is an automatically generated 8 bit (1 byte) response format received for every data block transferred to SD card. Of all the 8 bits only bits 1, 2 & 3 determines the status of the data sent as shown in Fig.4.8.

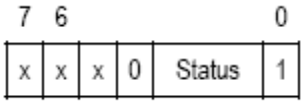


Figure 4.8 Bit format of Data Response

Here

Bit	Description
0	Always high
1	‘010’—Data accepted.
2	‘101’—Data rejected due to a CRC error.
3	’110’—Data Rejected due to a Write Error
4	Always low
5	-
6	-
7	-

4.2.3 Data Tokens

Data is transferred between SD card and microcontroller via data tokens. Data tokens are 4 to 515 bytes long depending upon the length of the data block. There are two types of data token formats.

1. For single block read/write and multiple block read
2. For multiple block write

Here the format for Single Block Write data tokens is shown in Fig.4.9.

Byte 1	Byte 2-513(depends on data block length)	Last two bytes
1 1 1 1 1 1 1 0	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXX
Start Block Token	User data	16bit CRC

Figure 4.9 Byte formats of single block write data tokens.

4.3 SD Card Initiation

4.3.1 SD card RESET

To initiate SD card in SPI mode, microcontroller was programmed to deactivate SD card, turn OFF and turn ON SD card power. But before and after turning it ON, a time delay was created to give time for SD card to change voltage levels. It is essential to give SD Card about a hundred clock cycles to boot up (which include supply ramp up time and eliminate power up synchronization problems) as shown in Fig.4.10, so a dummy data byte was written16 times to SD card.

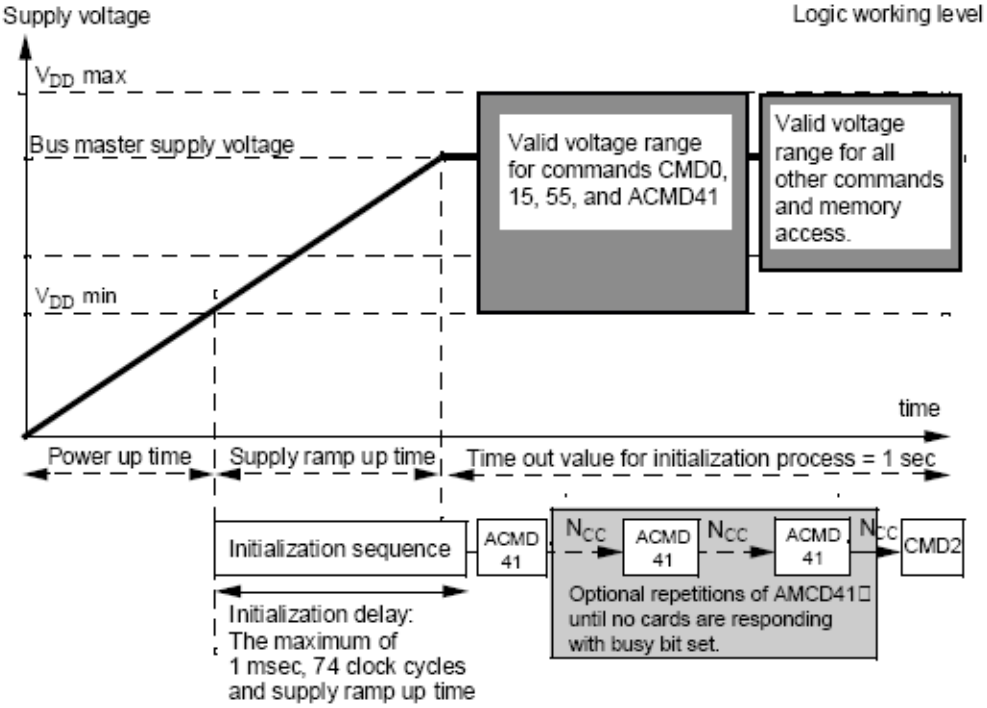


Figure 4.10 Power-up Diagram of SD card.

4.3.2 SPI Mode Selection

After SD card was RESET, SPI mode was selected by asserting CS signal during transmission of command GO_IDLE_STATE (CMD0). The command token was formatted as discussed in Sec.4.2.1. So the entire command sequence was 40 00 00 00 00 95 (hexadecimal). This command was sent to SD card using SPI Command Write discussed in

Sec.4.2.1.1. Immediately a status response was returned from SD card for this command. This response was compared with a R1 response format to determine the status of the SD card as discussed in Sec.4.2.2.1. SPI Mode selection was confirmed, if an appropriate response (00h byte – No errors) was received. Else command CMD0 was written to SD card again and again till a valid response is received.

Once SD card was set in SPI communication mode, command CMD1 was sent to SD card, to activate the card's initialization process. The end of initialization was determined by the no error (00h byte) response from SD card. Then command CMD55 was sent to SD card to notify next coming application specific commands. The response of this command was ignored. Finally an application specific command ACMD41 was sent to SD card to activate the card's initialization process. The end of initialization or SD card Idle or SD card ready state was determined by the no error (00h byte) response from SD card. The complete SD card initiation process is shown in Fig.4.11 and corresponding code is given in code 13 (Appendix A).

4.4 Data Write to SD card

Once SD card was initialized, microcontroller was programmed to write the 512 byte data stored in data buffer to SD card. The address location to write data was determined using WinHex software as discussed in Sec.3.2.3. A single block write command CMD24 was sent to SD card to transfer data. By default data block length on SD card was set to 512 bytes, so a command to set the block length was not sent.

The command token for single block write command CMD24 was determined from the command formats discussed in Sec.4.2.1. Here, command was taken as 24 (decimal), command argument (in this case, address location to write data as shown in Tab.4.1) was taken as 28672 (decimal) and CRC bytes were taken as FF (hexadecimal) as they are ignored. This command token was sent to SD card and response was read from SD card. Similar to any other command token writing to SD card, this was written to card till a No error (00h byte) R1 response was received.

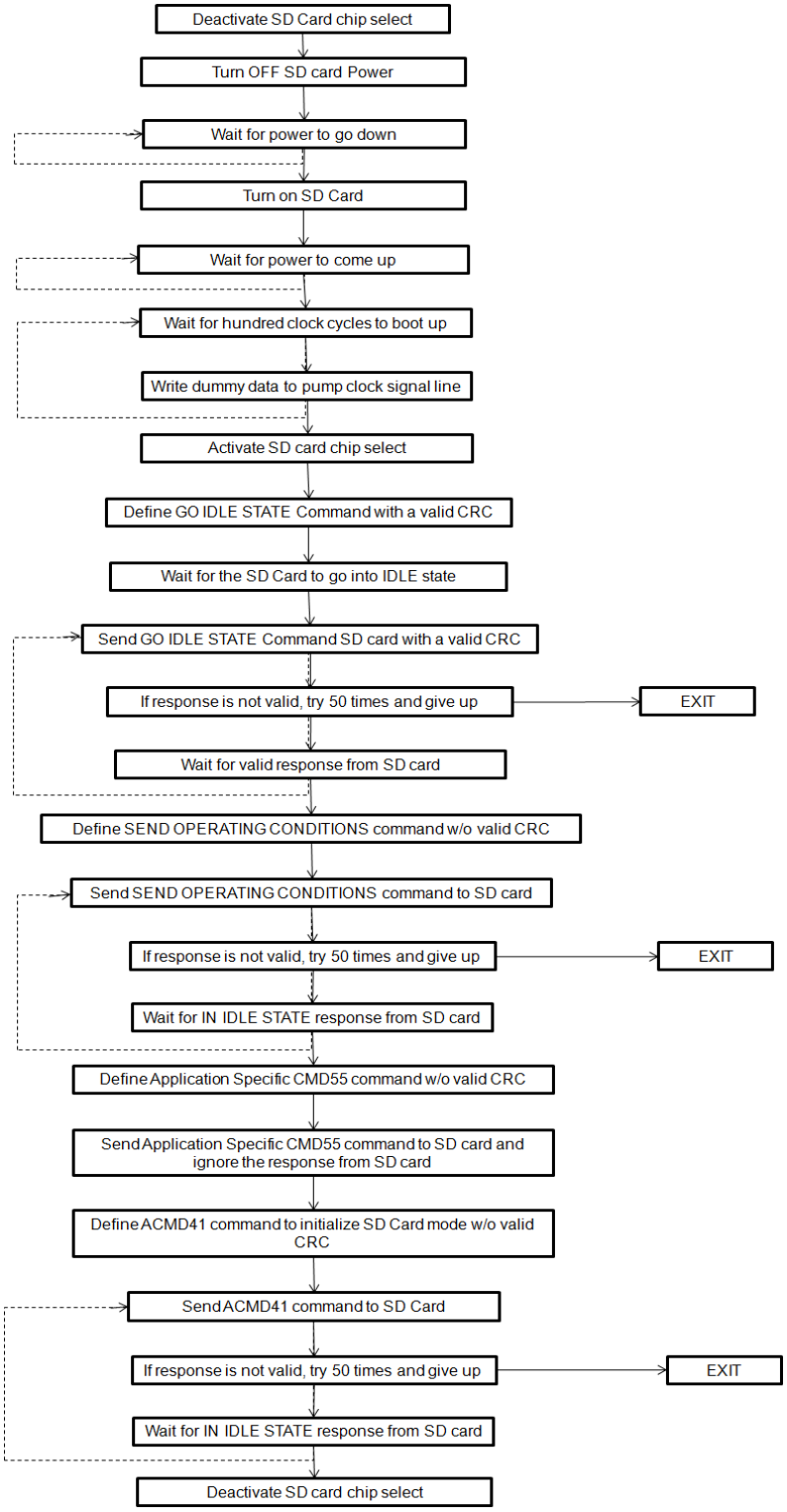


Figure 4.11 Flow chart of SD card SPI mode initiation.

As soon as SD card was ready for data transfer, data token (formatted as discussed in Sec.4.2.3) was sent using SPI Write. In this data token, a start block token (FEh byte) was sent, followed by 512 byte Data Buffer and invalid 16 bit CRC (FFh bytes). The data response was read from SD card using SPI Read. Microcontroller was left waiting until a valid (Data accepted) data response was read from the card. The end of data writing to card was determined by this data response. Then SD card chip select was deactivated and activated with a delay of 8 clock cycles. Finally SD card was read using SPI Read before deactivating the card again.

The complete SPI Mode data write to SD card from microcontroller was shown in Fig.4.12 and corresponding timings diagram was shown in Fig.4.13. Entire flow chart of data write to SD card is shown in Fig.4.14 and its corresponding code is given in Code 17 (Appendix A).

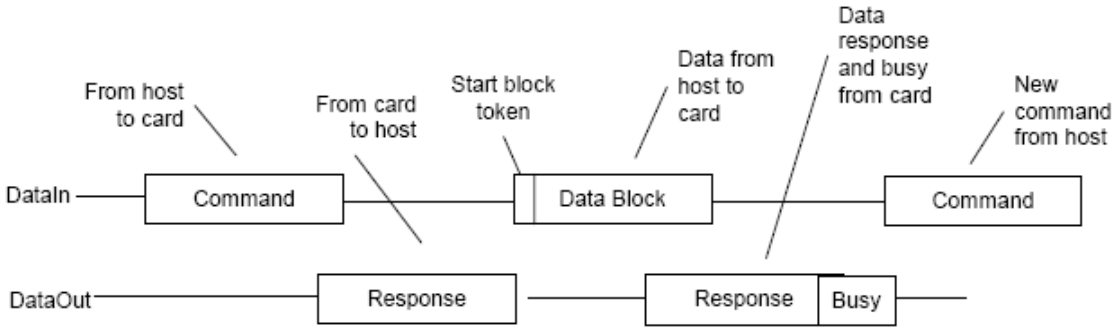


Figure 4.12 Communication for data transfer between microcontroller and SD card.

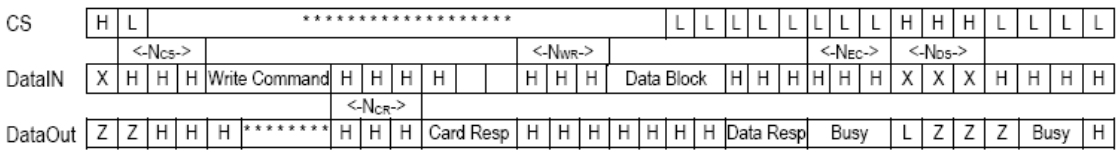


Figure 4.13 Timing diagram of data transfer communication between microcontroller and card.

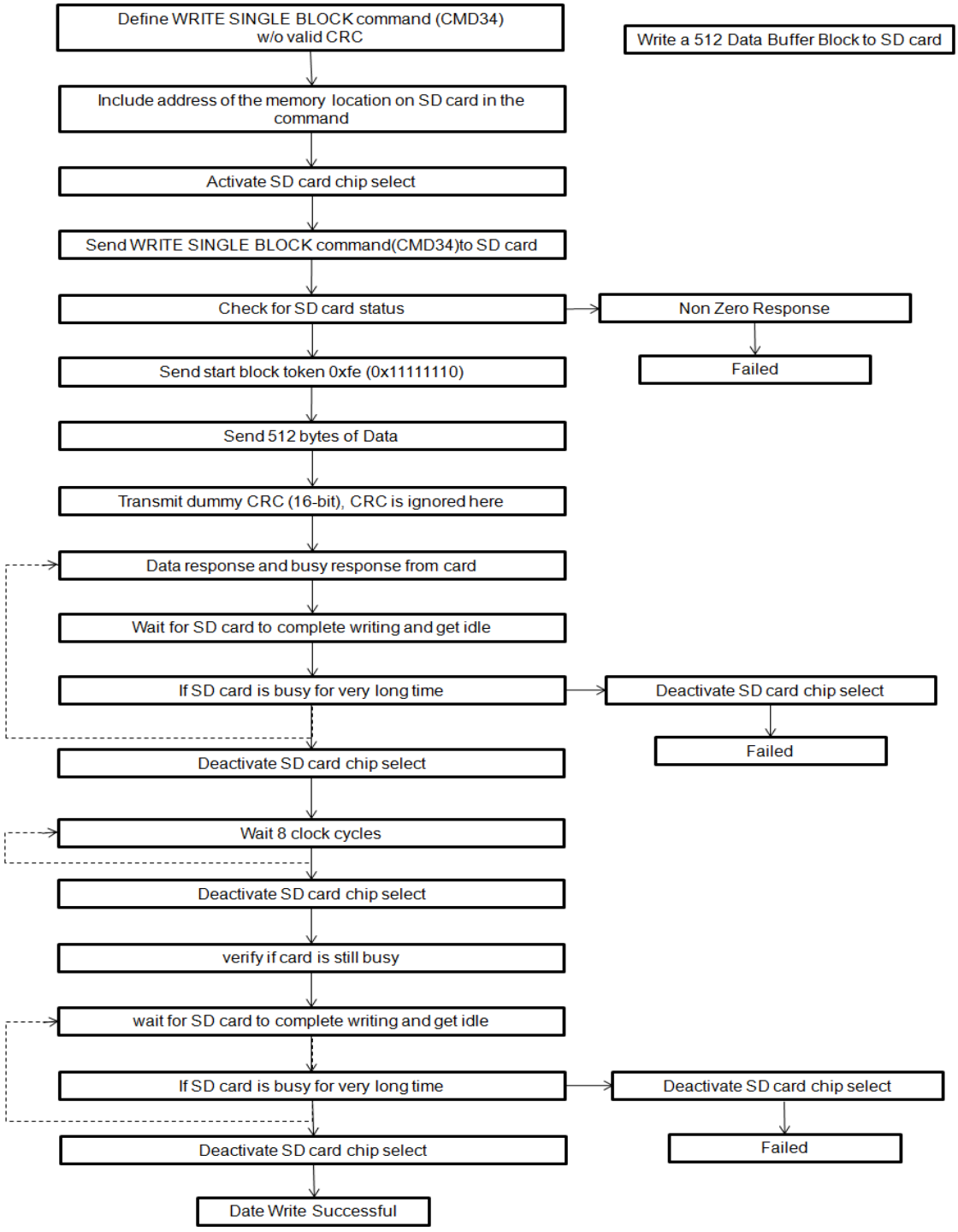


Figure 4.14 Flow chart of data write to SD card.

CHAPTER 5

EXPERIMENTS AND RESULTS

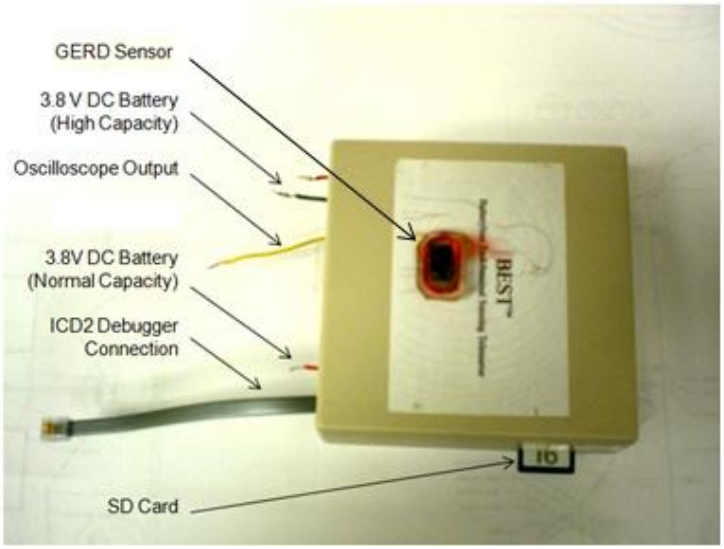
The final working device, its internal circuit, reader antenna and batteries used are shown in Fig.5.1 (b). As mention in Sec.1.3.1, this BEST^{TM} for GERD sensors is based on inductive coupling, which depends on distance and medium between them. As magnetic field is not uniform everywhere around the BEST^{TM}, signal received is not same everywhere around it. So it is essential to study the variation and repeatability of the signal in different medium and at different distances. At the same time stability and performance of device with time are studied. All the experiments conducted can be classified into motion artifact tests and performance and stability tests, which include the following.

I. Motion Artifact Tests

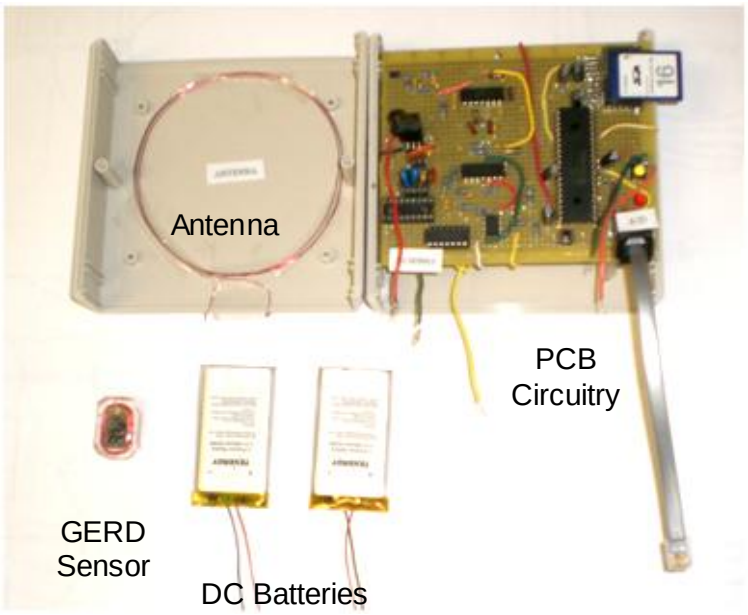
1. Effect of relative motion between sensor and BEST^{TM} on the received signal in different directions.
2. Repeatability in measured value with change in distance between BEST^{TM} and sensor.
3. Effect of relative motion between sensor and BEST^{TM} on the received signal at different angles.
4. Repeatability in measured value with change in angle between BEST^{TM} and sensor.
5. Effect of sensor environment on the received signal in different direction.
6. Determination of maximum allowable change in received signal.
7. Determination of working region of sensor around BEST^{TM}.

II. Performance and Stability Tests

1. Stability of the GERD monitoring system with drop in battery voltage (effect of battery voltage on measured frequency signal).



(a)



(b)

Figure 5.1 (a) BEST^{TM} with GERD sensor; (b) Internal PCB circuit, Antenna and High Capacity batteries of BEST^{TM}

2 Stability of the GERD monitoring system with time (effect of any other parameters on measured frequency signal).

3 Determination of power consumption or battery life with different optimization codes.

All the measurements were conducted with distance between BEST^{TM} and sensor starting from 1cm. For our application, study of variation in signal at father distances is more important than at close proximities.

5.1 Motion Artifact Tests

The main objective of these experiments is to study the effects of relative motion between BEST^{TM} and sensor, on measured frequency signal.

5.1.1 Effect of relative motion between sensor and BEST^{TM} on the received signal in different directions

To study the effects of motion in different directions on the measured frequency signal on the SD card, the GERD sensor was moved along all X, Y & Z axis keeping the BEST^{TM} fixed at a place. The recorded frequency signal on SD card was analyzed.

Experimental setup: To move the sensor along X, Y & Z axis, the setup shown in Fig.5.2 was adopted. In this BEST^{TM} was fixed at a place on the experiment table. A 7.5 cm x 7.5 cm square foam board was divided into a matrix of 6 x 6 squares. This foam board was placed at the centre of BEST^{TM}, and was aligned to the center of antenna by measuring the distances on all the sides. Now keeping the foam board at 1.5cm distance from telemeter, GERD sensor was moved to the center of each square and corresponding frequency was recorded. The same procedure was repeated with increasing the distance between foam board and BEST^{TM} by 1cm, till signal was completely lost.

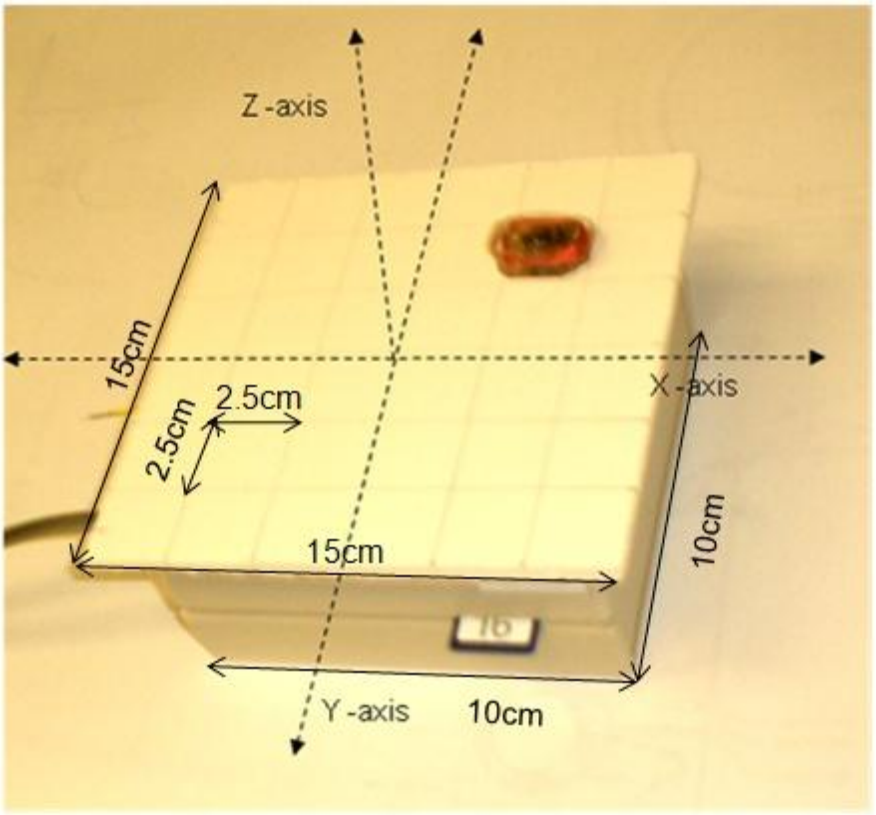


Figure 5.2 Experimental Setup # 1

*Experimental result: O*n every square and at different distances from BEST^{TM}, frequency signal was recorded on SD card. On every square there were up to 50 recordings measured, this data was processed and color coded as shown in Fig.5.3. From this, the working region of the whole system was determined depending upon the variation of signal from the average value.

Each square in Fig.5.3 shows the percentage change in frequency from the actual frequency. The change in frequency is acceptable, if it is below 5%. Here in the Fig.5.3, darker squares are out of range readings while lighter squares are within range readings.

1. BEST^{TM} measured signal up to 4.5 cm with <5% in the measured frequency.
2. At 4.5cm the area of coverage was 2.5 X 2.5 sq cm.

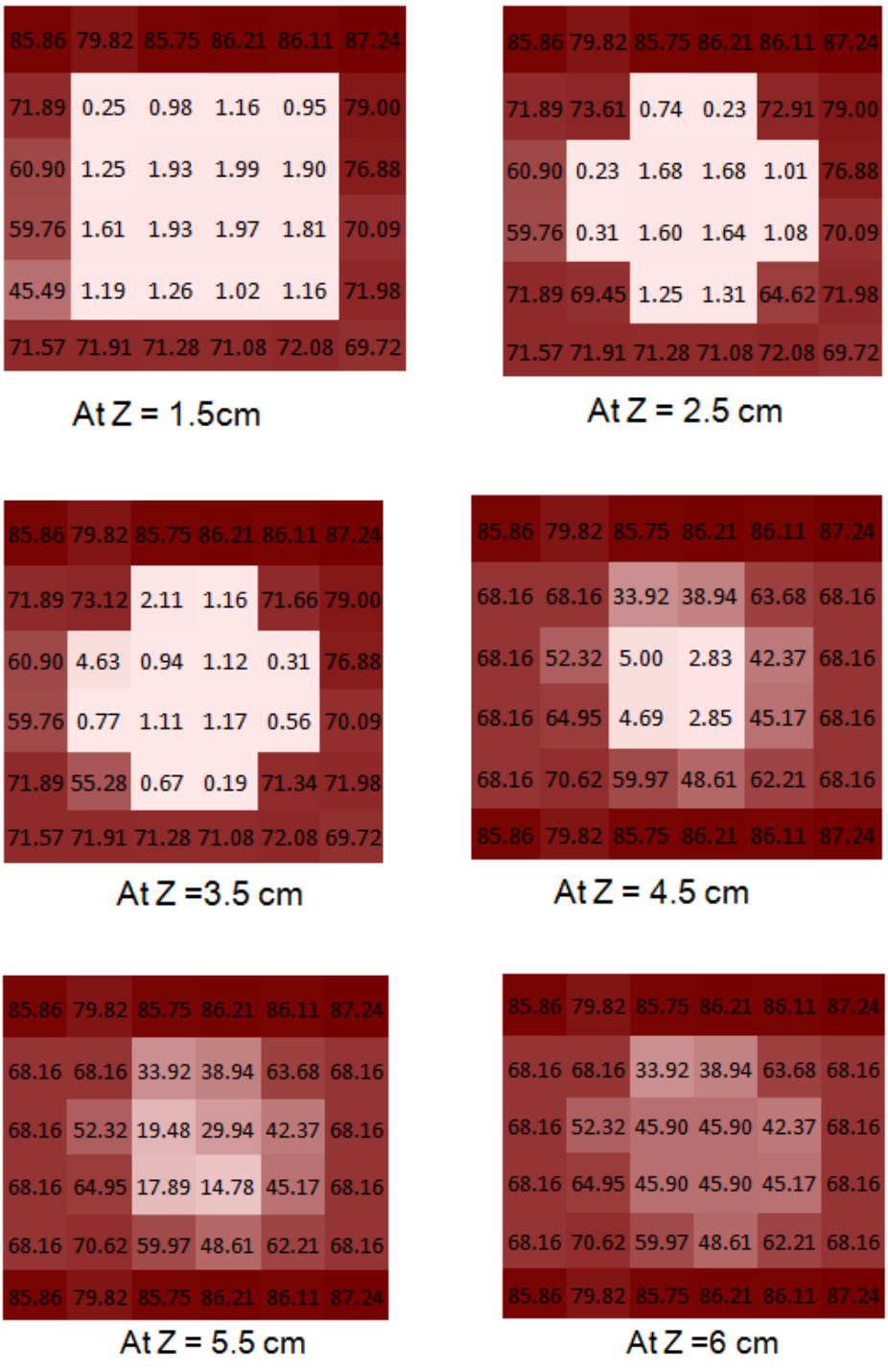


Figure 5.3 Percentage change in frequency in different directions.

5.1.2 Repeatability in measured value with change in distance between BEST™ and sensor

Experimental setup: To study the repeatability in the measured value, the setup shown in Fig.5.4 was adopted. BEST™ was fixed at placed and sensor was moved along Z-axis away from the sensor marked on the foam board scale (Here a metal scale should not used to measure the distance as it effects the magnetic field). The frequency signal measured at different distances was recorded on SD card. The same procedure was repeated again but this time sensor was moved towards the BEST™.

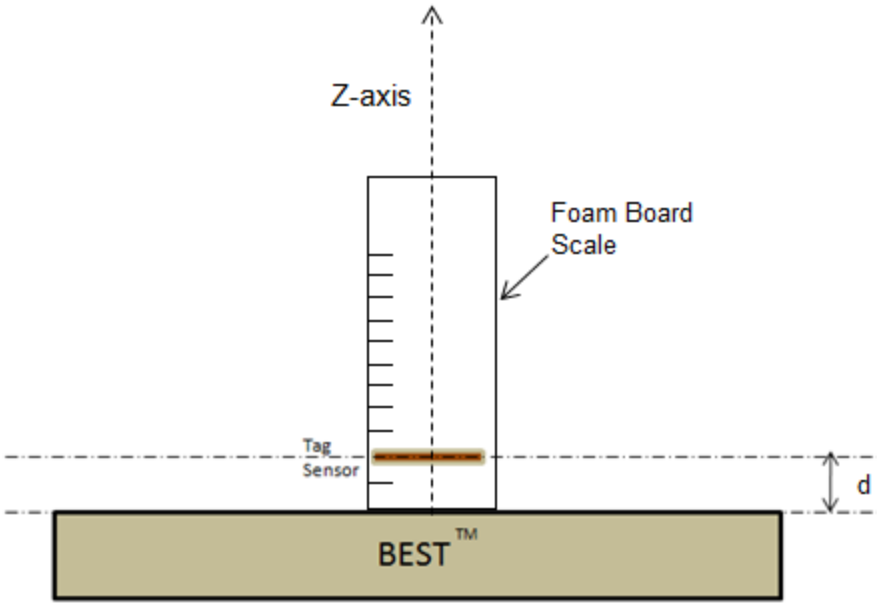


Figure 5.4 Experimental Setup # 2

Experimental result: The frequencies recorded on the SD card were processed and plotted on graph shown in Fig.5.5. Here Y-axis shows the frequency measured by the BEST™ and X-axis shows distance between BEST™ and sensor (motion along Z-axis) in steps of 0.5cm. Each step on X-axis is collection of up to 50 recordings or recordings for 5 sec. The corresponding percentage change in frequency with distance was plotted on Fig.5.6.

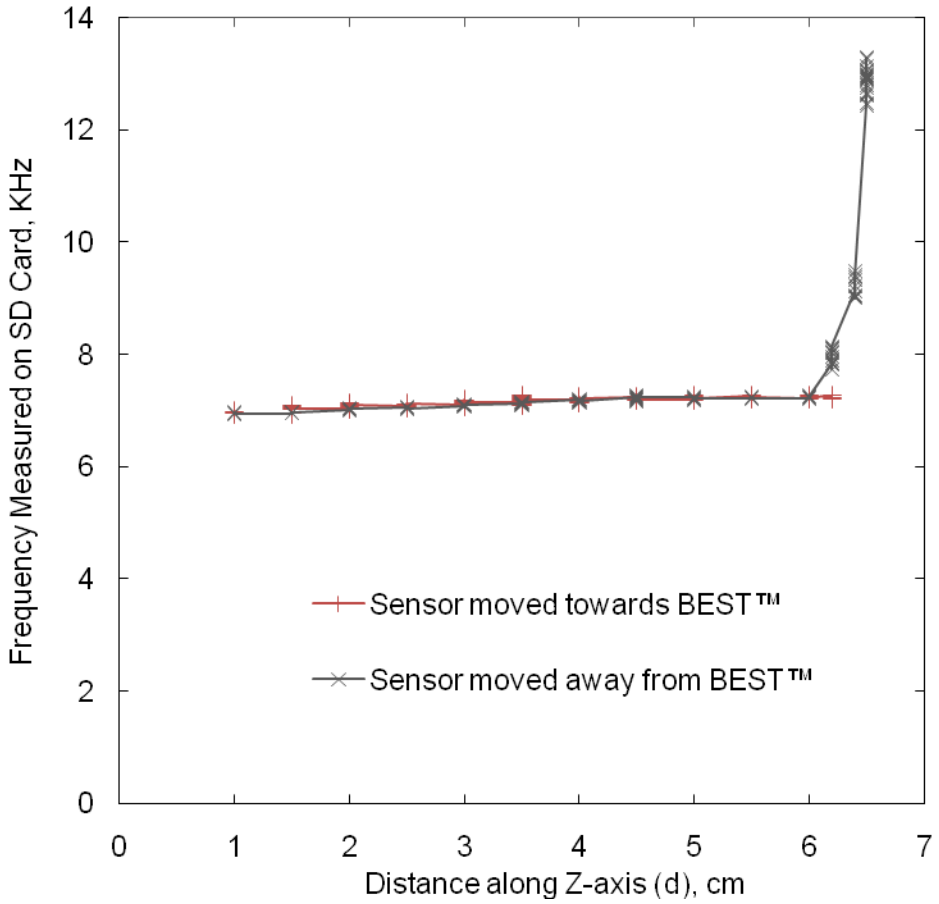


Figure 5.5 Repeatability of the sensor with change in distance along Z-axis

1. The frequency signal measured by BEST™ was repeatable with in working range.
2. With the experimental setup 2, frequency measured was in acceptable range till 5.5 cm
3. With the experimental setup 2, the percentage change in frequency is less than 5%.
4. The frequency shift from within the range measurement and out of range measurement was very drastic. (This is highly desired.)

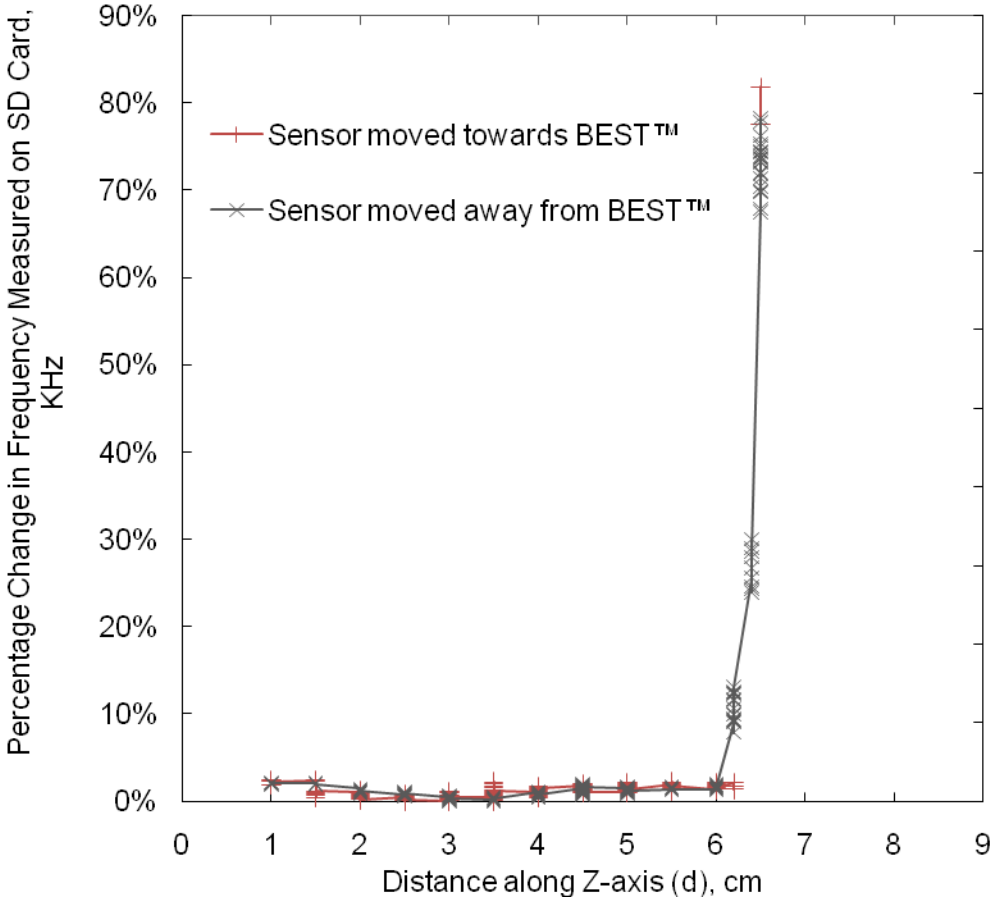


Figure 5.6 Percentage change in frequency with change in distance along Z-axis.

5.1.3 Effect of relative motion between sensor and BEST™ on the received signal at different angles

To study the effects of rotation on the measured frequency by *BEST™* the sensor was rotated about all the three axes.

Experimental Setup: To rotate the sensor about X, Y & Z axis, the setup shown in Fig.5.7 was adopted. Angle of rotation about X, Y and Z were called as angle α, β and γ respectively. In this setup, BEST™ was fixed at a place on the experiment table. On foam board, lines were drawn with increasing angle (w.r.t horizontal line) in steps of 30 deg. This foam board with angled lines was placed on the top of the BEST™.

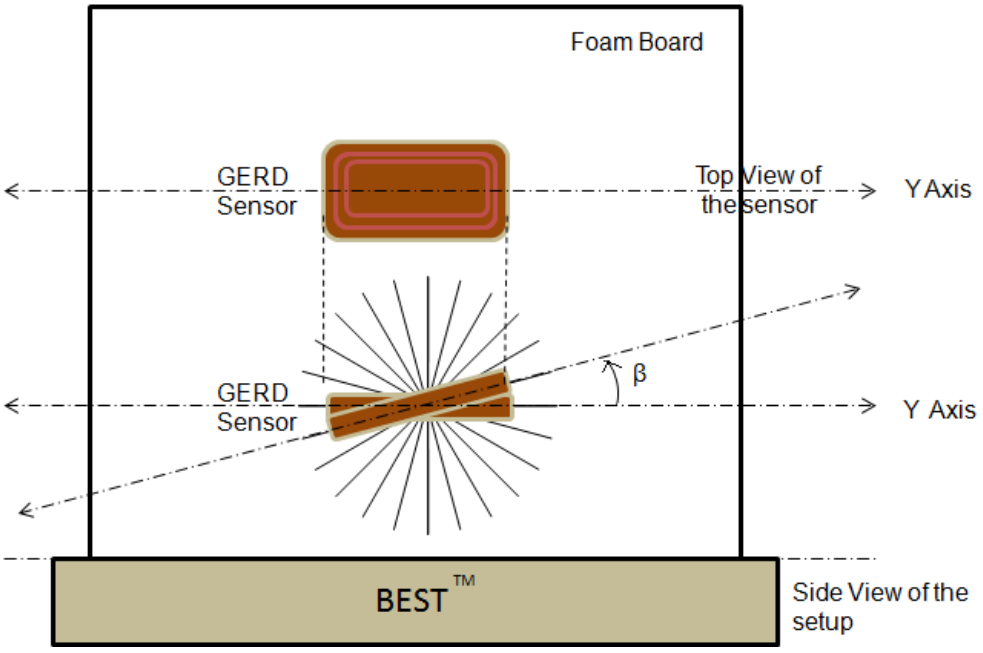


Figure 5.7 Experimental setup # 3

Rotation about X axis (change in angle β): Centre of sensor was aligned to centre of BEST™ and side of sensor was aligned to the horizontal line on the foam board as shown in Fig.5.8. Then sensor was rotated in steps of 15 deg by aligning to the lines on the foam board. At every angle about 50 frequency samples were recorded. Same procedure was repeated for different distances from BEST™.

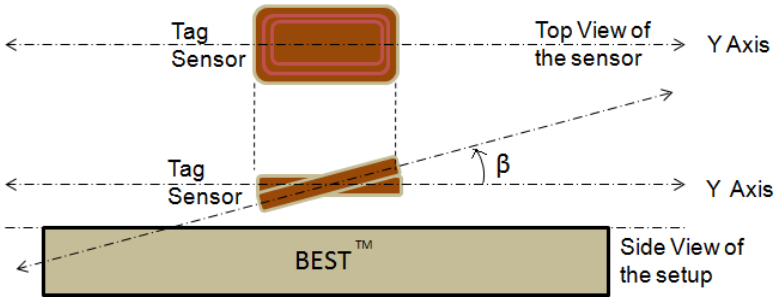


Figure 5.8 Rotation of sensor about X-axis

Rotation about Y axis (change in angle α): The same procedure mention above was repeated here but sensor was positioned as shown in Fig.5.9 about Y-axis.

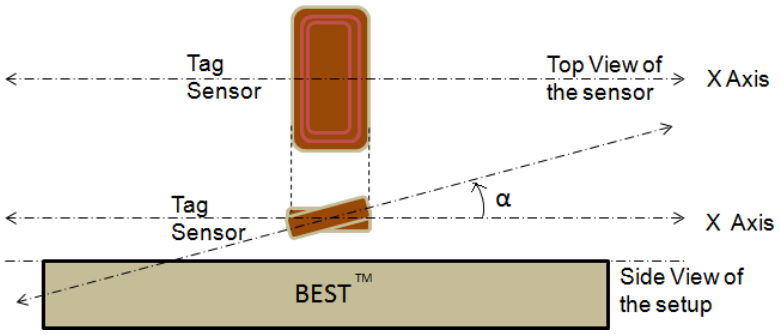


Figure 5.9 Rotation of sensor about Y-axis

Rotation about Z axis (change in angle γ): For this, rectangles of size of sensor were drawn on a foam board which are rotated 180 deg in steps of 15 .This board was fixed at a distance from BEST™ and sensor was placed in every rectangle. At every angle about 50 frequency samples were recorded. Same procedure was repeated for different distances from BEST™.

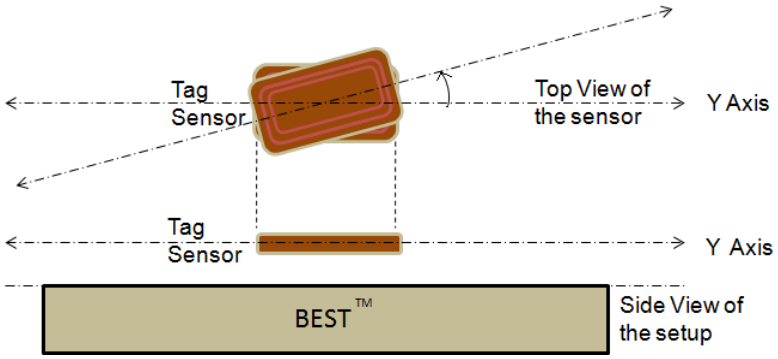


Figure 5.10 Rotation of sensor about Z-axis

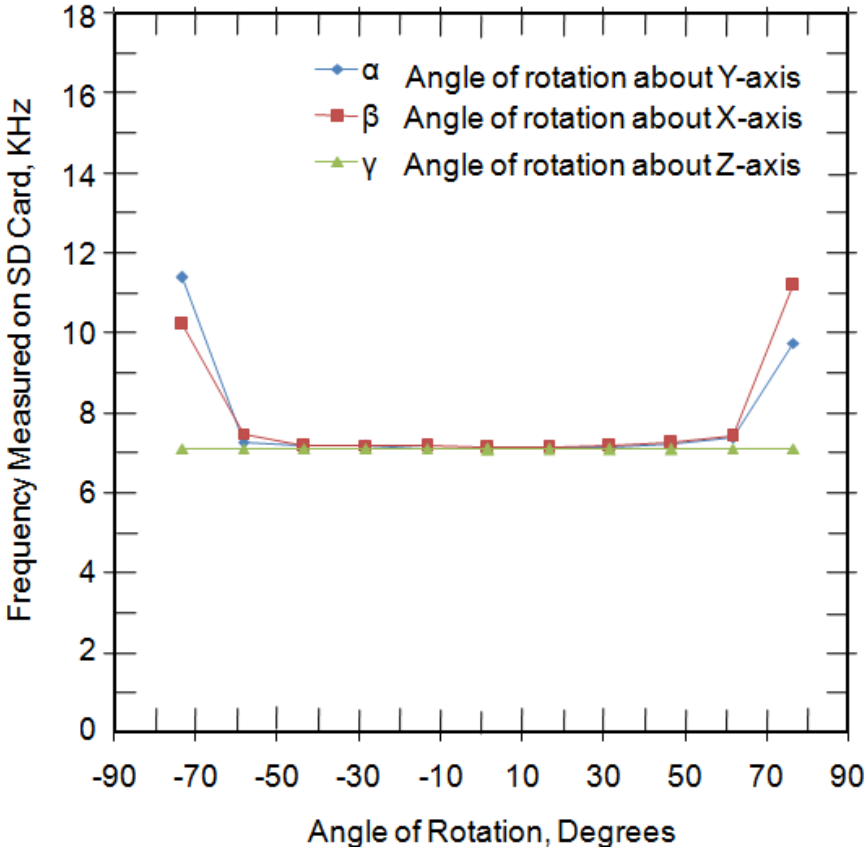


Figure 5.11 Rotation of Sensor about XYZ axes at Z =1.5cm

Experimental Result:

The frequencies recorded on SD card at every angle at fixed distance (Z= 1.5cm) were averaged and plotted on graph as shown in Fig.5.11. All the angles α, β & γ are plotted on same graph with frequency measured on Y-axis and angles in degrees on X-axis.

On another graph frequencies with change in angle about all axes and change in distance are plotted as shown in Fig.5.12.

1. When the sensor was rotated about X-axis, the maximum angle of rotation with respect to reference plane for frequency to be in acceptable range was +/- 60 deg.
2. When the sensor was rotated about Y-axis, the maximum angle of rotation with respect to reference plane for frequency to be in acceptable range was +/- 55 deg.

3. There was no change in frequency with rotation of sensor about Z-axis rotation with respect to reference plane.
4. As distance increased the max angle of rotation decreases slightly.
5. Acceptable orientation of sensor at a distance up to 4.5cm was +/-52 deg for a 7% change in frequency.

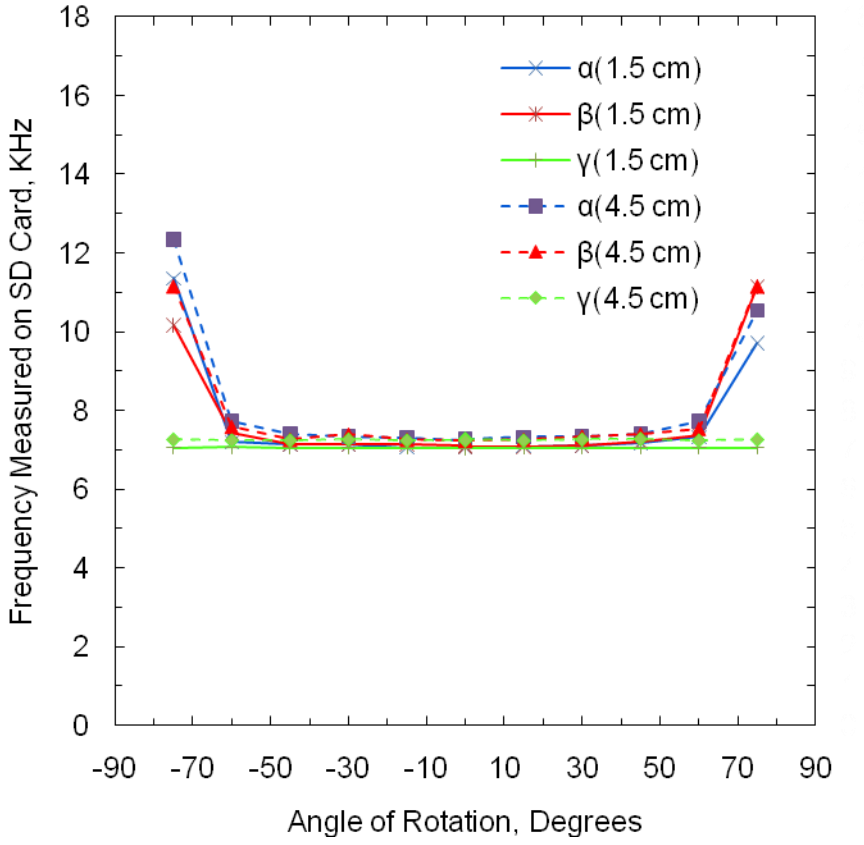


Figure 5.12 Rotation of Sensor at Z =1.5cm & Z = 4.5cm

5.1.4 Repeatability in measured value with change in angle between BEST^{TM} *and sensor*

Experimental setup: The setup similar to that mentioned above was adopted here to study the repeatability with rotation of sensor. In this setup keeping BEST^{TM} fixed, sensor was rotated 180 deg about X-axis in steps of 15 deg, in clockwise direction and anti clockwise direction at distance Z =1.5cm. Same procedure was repeated for Z=4.5 cm.

Experimental result: The results recorded on SD card are plotted with measured frequencies on Y- axis and angle of rotation on X-axis (here 0-90 deg is angle of rotation in both clockwise and anti clockwise direction).

1. When sensor was rotated about X-axis, the frequency measured by BEST^{TM} was repeatable up to 60 deg at a distance of Z =1.5cm.
2. The measured frequency was still repeatable up to 60deg even if distance was changed to Z = 4.5cm.

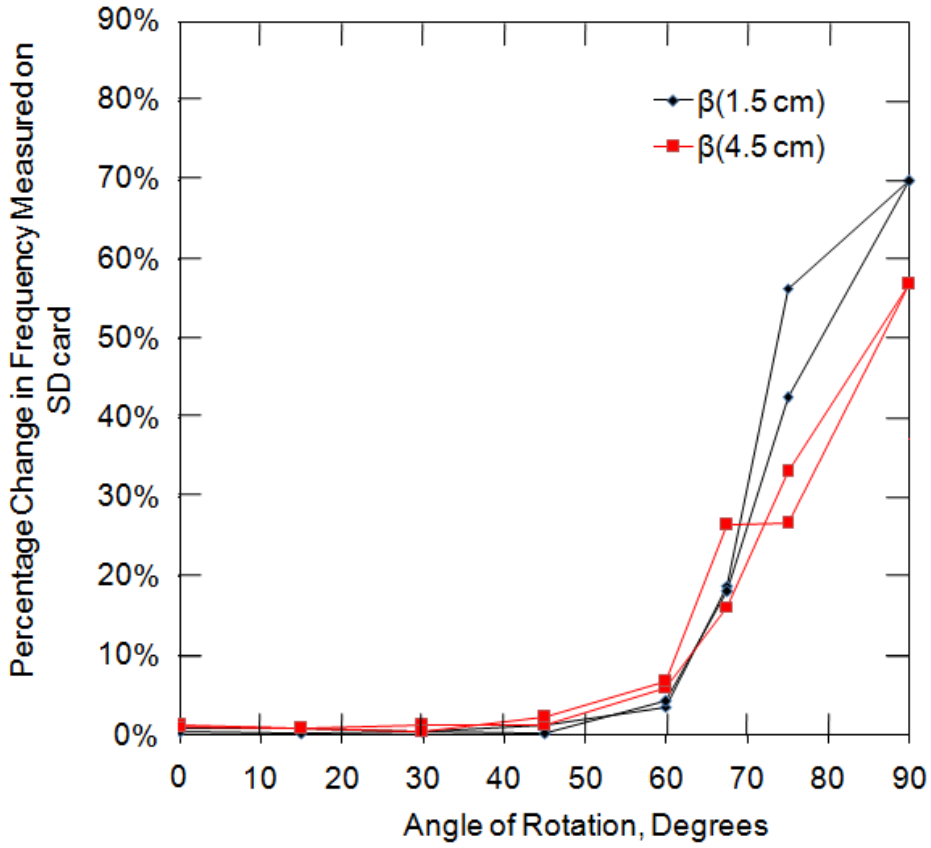


Figure 5.13 Repeatability with rotation of sensor

5.1.5 Effect on the received signal with change in sensor environments and at different distances

The GERD sensor was designed to work in different environments, so it is important to study the effect of environment on the measured signal. As mentioned in Sec.1.3.2, the acid reflux is detected by shift in frequency (Δ F), which should be much higher than change in frequency with motion/rotation (δ f).

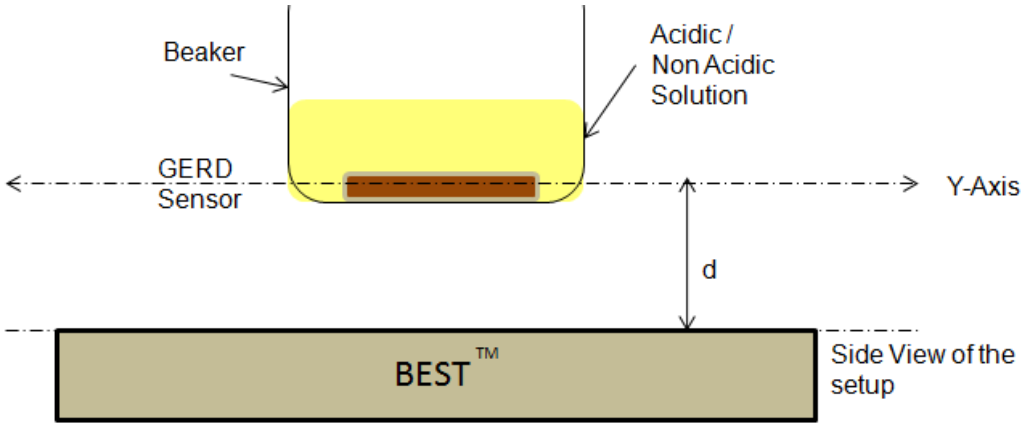


Figure 5.14 Experimental setup # 4

Experimental setup: This effect was studied with the experimental setup as shown in Fig.5.14. In this BEST™ was fixed at a position and sensor immersed in a solution (water / orange juice, short circuit and air) was moved away in steps of 0.5cm from 1cm to 6.5cm. The same procedure was repeated with sensor immersed in different medium. Here sensor electrodes were open circuited (air) and short circuited to determine the frequency shift (ΔF) in extreme possible conditions.

Experimental result: From the data recorded on the SD card, graphs similar to motion artifact plots were plotted, where X-axis was not linear scale but a stepped scale. And Y-axis was frequency measured in KHz. All the frequencies data obtained in different medium were plotted on same graph as shown in Fig.5.15.

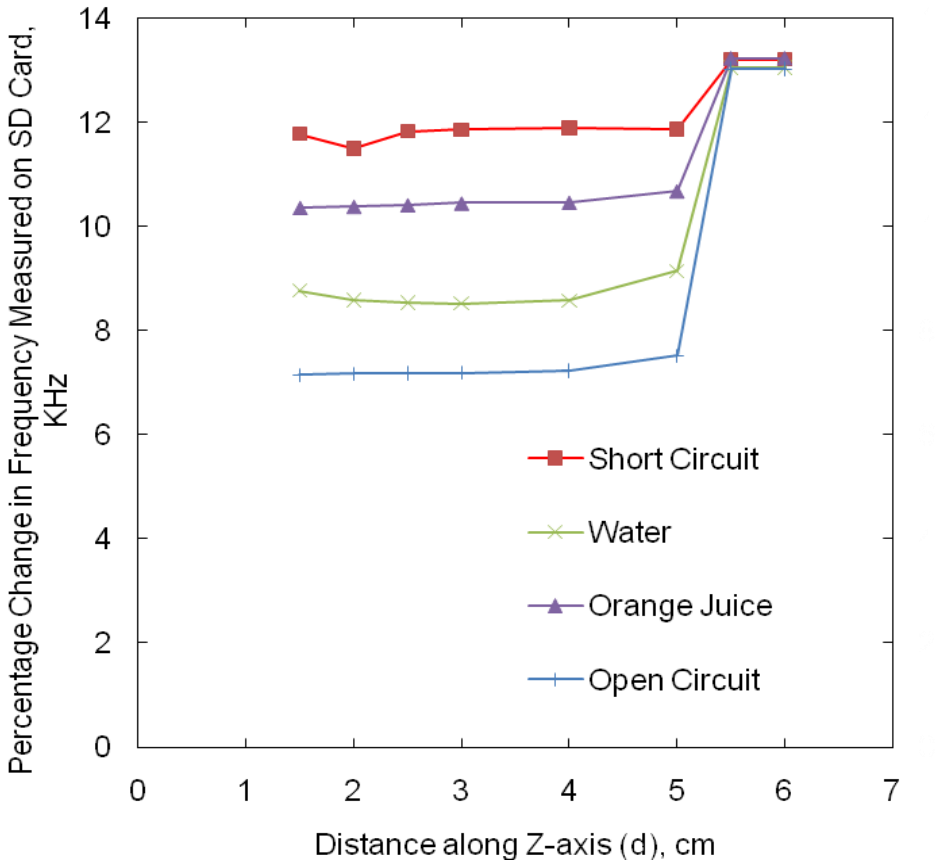


Figure 5.15 Frequency measured in different sensor environments and at different distances.

1. When sensor was in open air (OC) which is one extreme possible condition of the sensor, the frequency measured was in the range of 7-7.5KHz.
2. When sensor was short circuited (other possible extreme condition) the frequency was shifted to 12KHz.
3. A non acid frequency shift is in the range of 8-9 KHz
4. An acidic reflux is in the range of 10-11KHz
5. As distance increased the measured frequency also increased but not overlapped with the frequency range of other medium.

6. When sensor was out of range the frequency measured was >12 KHz but not any other defined frequencies.

5.1.6 Determination of maximum allowable change in received signal

Experimental setup: From the above experiments, the following results were studied.

Experimental result: From the data obtained from the previous experiment, the maximum and minimum frequency values for a given medium were determined. All the values between these boundaries were averaged to determine the average frequency for a given medium. The shift in the frequency because of the solution change (Δ F) and change in frequency with motion (δ f) are compared on Fig. 5.16.

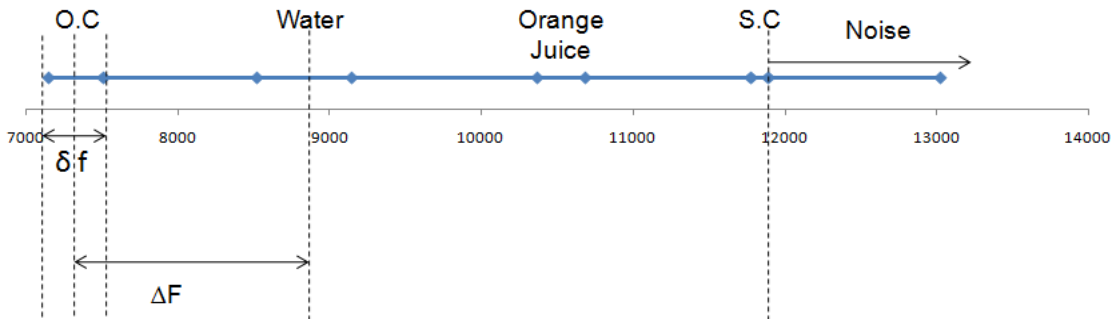


Figure 5.16 Frequency calibration

Δ F > δ f

δ f – Allowable range of frequencies of a liquid

Δ F – Shift in average frequency with different liquids

1. The shift in frequency due to change in medium was much higher than maximum change in frequency at a given frequency.
2. The complete frequency range of operation is 7-12KHz
3. Any value greater than 12KHz and less than 7KHz is noise

5.2 Performance and Stability Tests

These tests were conducted to determine the stability of the received signal with long operation time. These tests also included tests for battery life.

5.2.1 Stability of the GERD monitoring system with drop in battery voltage

As discussed in Sec.2.1.1, inductive coupling between reader and Tag senor antenna also depends on supply voltage of Class E amplifier. The purpose of the experiment was to study the effect of change in supply voltage on measured frequency.

Experimental setup: For this a setup similar to Fig.5.4 was adopted. Here sensor was fixed at a distance of Z=1.5cm and a full charged 3.8V DC battery (capacity 1300mAh) was connected to Class E amplifier. The change in battery voltage was recorded for every 1hour.

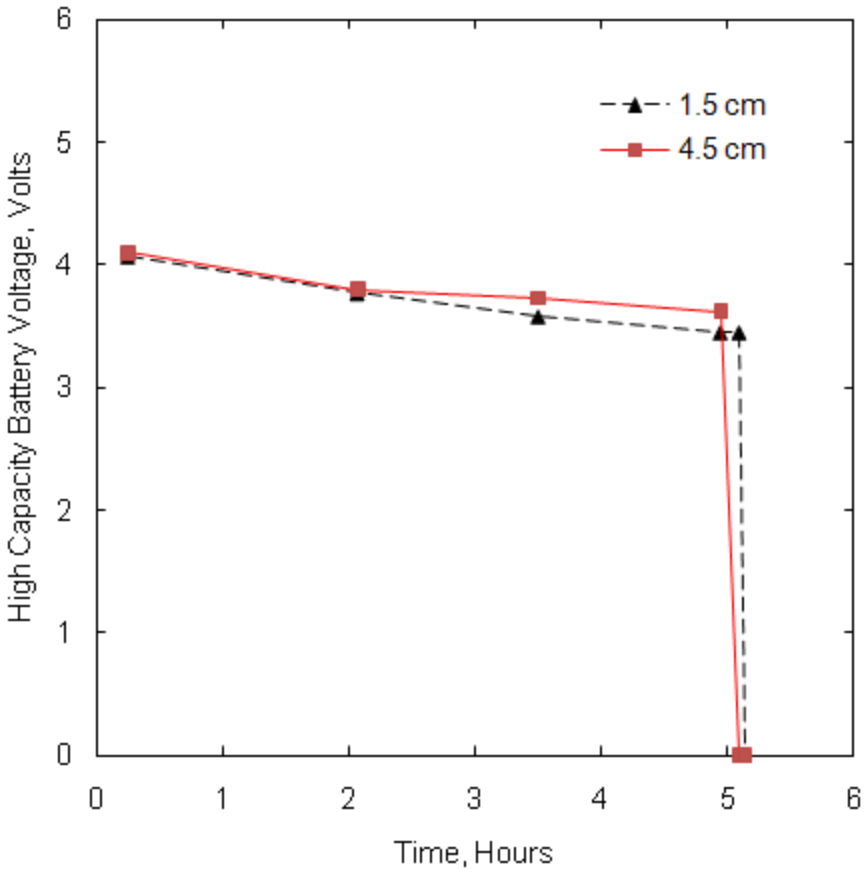


Figure 5.17 Effect of supply voltage on measured frequency.

Experimental result: To plot the change in frequency with change in battery supply voltage, the measured frequency was plotted on Y-axis and battery voltage was plotted on X-axis. The average frequency for every one hour at the two different distances was plotted on same graph as shown in Fig.5.17.

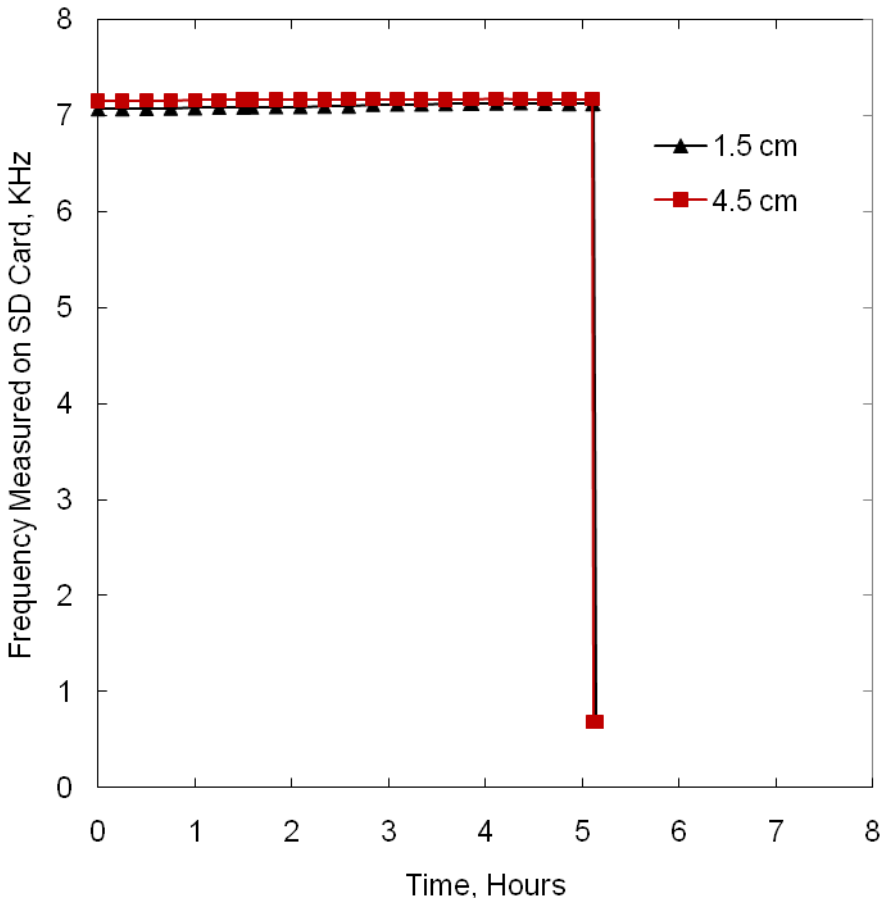


Figure 5.18 Variation of frequency signal with time in open loop.

5.2.2 Stability of the GERD monitoring system with time

It is important to determine, if there is any change in measured frequency signal when device was used continuously for several days.

Experimental setup: Same setup shown in Fig.5.4 was used but with closed loop control for longer life.

Experimental result: There was no effect of any other parameter other than voltage on the measured frequency as there was not much variation with time, as shown in Fig.5.18.

5.2.3 Determination of power consumption or battery life with different optimization codes

Experimental setup for open loop control of BEST^{TM}*:* A fully charged 3.8V and 1300mAh battery was used in class E amplifier circuit which was working in open loop control. The telemeter and sensor were placed as shown in Fig.5.4 at a distance of 1.5cm (close proximity) and this setup was left undisturbed till the battery was completely discharged. The recorded frequency on the SD card was plotted as shown in Fig.5.18. The same experiment was repeated again with a full charged battery again at 4.5 cm (farther distance) to study the change in power consumption or change in battery life with increase in distance between telemeter and sensor.

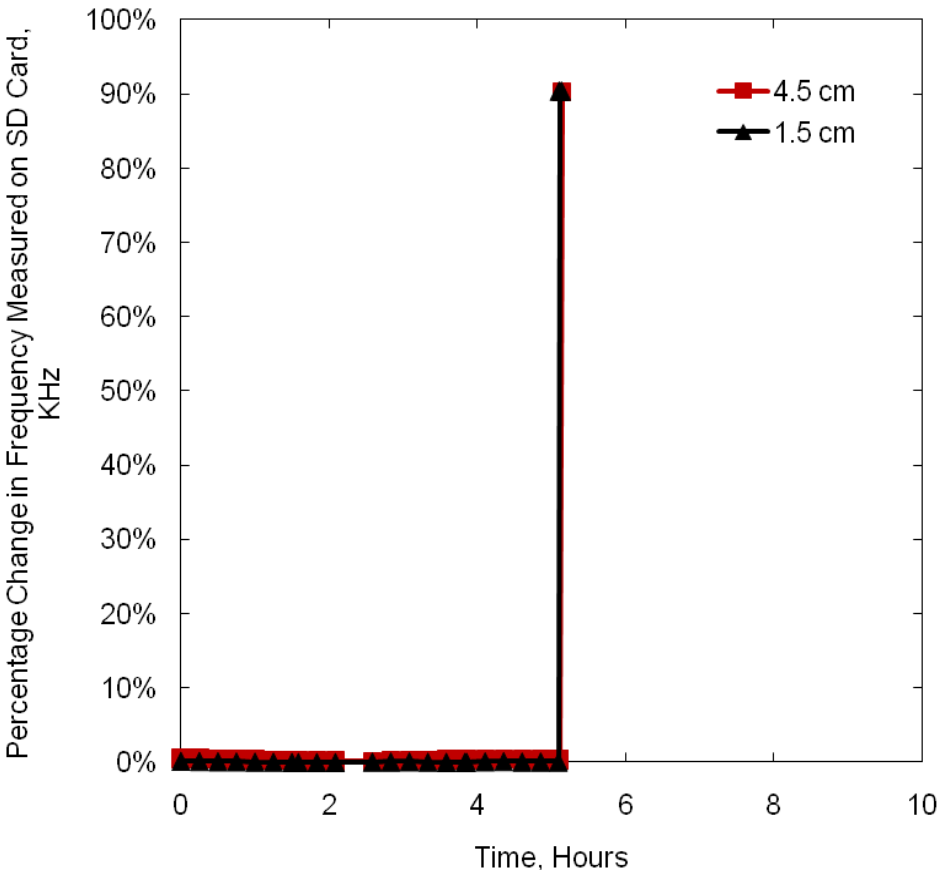


Figure 5.19 Percentage variation of frequency signal with time in open loop.

Experimental result:

1. A 3.8V and 1300mAh battery lasted 5 hour with distance between sensor and telemeter was 1.5 cm.
2. The same full charged battery lasted for same time with distance between sensor and telemeter was 4.5 cm.
3. There is no much change in battery life with change in distance

Experimental setup for closed loop control of BEST^{TM}: As mentioned in Sec. 3.2.4.2, the power consumption can be improved by implementing the closed loop control using microcontroller. This change in battery life with different controls was tested with the same experimental setup with distance of 1.5cm. The recorded frequency obtain from closed loop control experiment and open loop experiment were plotted on same graph shown in Fig.5..20 with Z = 1.5 cm.

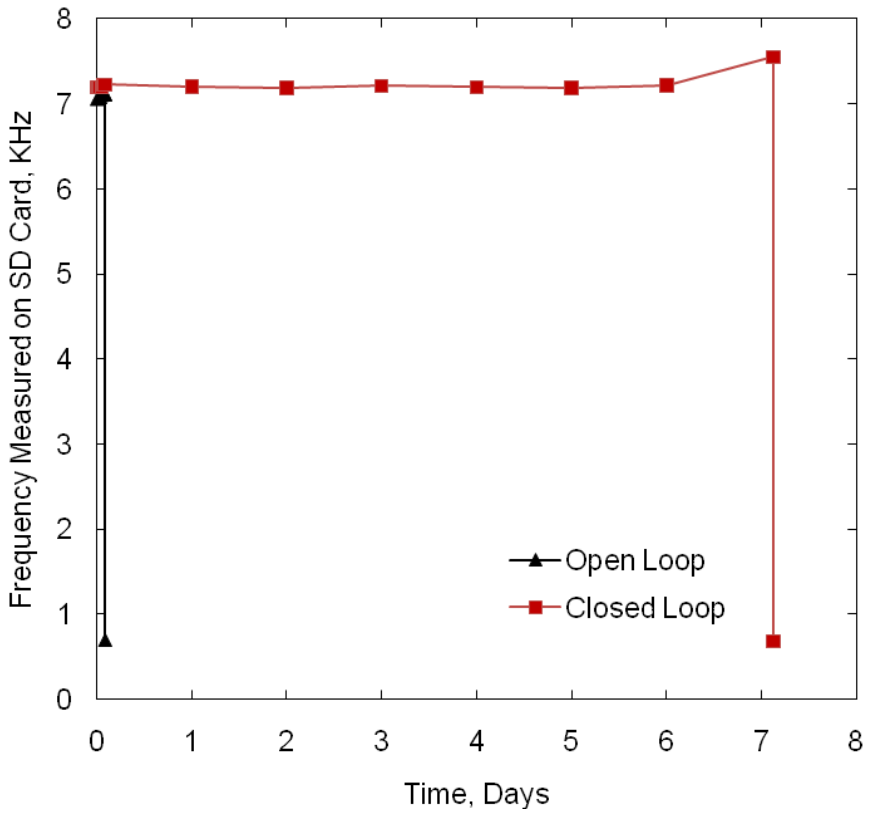


Figure 5.20 Comparison of variation of frequency signal with time in open loop and closed loop timing controls.

Experimental result:

1. There was a drastic change in battery life with different controls; the battery life was improved 34 times by implanting closed loop control.
2. But the measured frequency is higher than that was measured in open loop.

5.3 Conclusion

Experiments were conducted to study the effects of the different parameters on the measured signal and also performance and stability of the system. From the motion artifact tests it can be concluded that, sensor at 4.5cm from BEST^{TM}, the area of coverage was 2.5 X 2.5 sq cm about centre of system, with <5% change in the measured frequency. All the frequencies measured at every point within the reading range are repeatable. When sensor is not perfectly aligned to BEST^{TM} at a distance of 1.5cm the maximum allowable tilt is +/- 60 degrees about X-axis, +/-55 degrees about Y-axis and no change about Z-axis. The complete frequency range of operation of the monitoring system is 7-12 KHz and any frequency other than this noise because of sensor out of range or battery outage. In this range an acid reflux can be determined in 10-11 KHz range and a non reflux in 8-9 KHz range. The shift in the frequency (ΔF) because of the refluxes is much higher than the change in the frequency (δf) with any parameter. The following conclusions can be made from the performance and stability tests, the frequency measured on BEST^{TM} is stable with battery voltage, time and any other parameter. The performance of the device is improved with closed loop control using microcontroller. With this control, battery life of a 3.8V DC 1300mAh battery is improved 30 times from open loop control. Also it is important to notice there is no change in battery life with change in distance between BEST^{TM} and sensor.

CHAPTER 6

DISCUSSION AND CONCLUSION

A portable biotelemeter for batteryless Gastro esophageal reflux disease (GERD) sensors was designed. Wireless power to sensors and wireless communication with the same was achieved by inductively coupling with the telemeter. For this an antenna was designed to work at 1.3MHz with a reading range up to 5.5cm and a class E power amplifier using IRF510 MOSFET was designed to transfer power to antenna. A demodulating circuit was designed to demodulate the information signal from the 1.3MHz carrier signal. This demodulating circuit was integrated with microcontroller circuit to measure and process the information signal. A SD card was interfaced with the microcontroller in Serial Peripheral Interface (SPI) mode, to record the monitored signal with its corresponding time and displayed on a TEXT file. To reduce the power consumption of the biotelemeter (BEST^{TM}) different timing controls were implemented using microcontroller. Apart from all these some additional features like memory Erase, Read/Busy indication, Sensor position detection and Battery charging Indication were provided.

Experiments were conducted to study the effects of the different parameters on the measured signal and also performance and stability of the system. From the motion artifact tests it can be concluded that, sensor at 4.5cm from BEST^{TM}, the area of coverage was 2.5 X 2.5 sq cm about centre of system, with <5% change in the measured frequency. All the frequencies measured at every point within the reading range are repeatable. With centre of BEST^{TM} aligned to centre of sensor along Z-axis, can measure upto 5.5cm with less than 5% change in measurement. When sensor is not perfectly aligned to BEST^{TM} at a distance of 1.5cm the maximum allowable tilt is +/- 60 degrees about X-axis, +/-55 degrees about Y-axis and no change about Z-axis. Though this maximum tilt about X and Y axis decreases with increase in measuring distance, the maximum tilt at a distance of 4.5cm is +/- 52 deg with less than 7%

change in measurement. The frequency measurement is still repeatable when rotated about any axis and within maximum allowable tilt. The complete frequency range of operation of the monitoring system is 7-12 KHz and any frequency other than this noise because of sensor out of range or battery outage. In this range an acid reflux can be determined in 10-11 KHz range and a non reflux in 8-9 KHz range. The shift in the frequency (ΔF) because of the refluxes is much higher than the change in the frequency (δf) with any parameter. Also it has to be made sure that tilt in alignment is not in the range of 50-75 degrees. Any other position other than this can be detected with frequency measurement greater than 12 KHz on the telemeter.

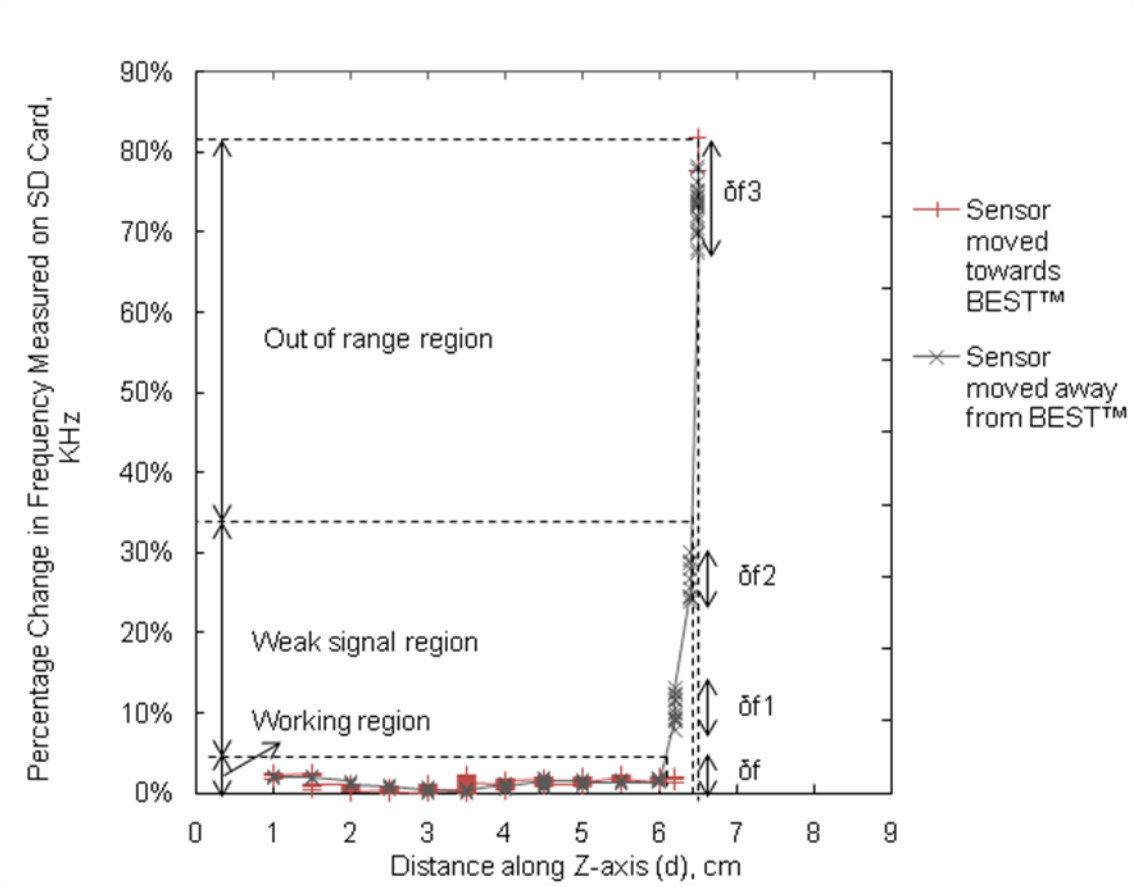


Figure 6.1 Different operating regions of the sensor.

.

When the sensor is moved away or towards the biotelemeter there are three operating regions. In working region, the batteryless sensor is powered up by the telemeter and

modulated signal is extracted. In this region the allowable change in frequency f is less than 5% and variation of signal at a distance is less. The second region is called weak signal region in which the sensor is powered up with a weak magnetic coupling but received signal has high noise. The change in frequency f1 and f2 at a distance of 6.2cm and 6.4 respectively, is greater than 1000 KHz (or greater than 5%) and variation in signal is also high. The third region is called out of range region in which the sensor is at distance that the telemeter fails to power it up. In this region the signal is just noise which is greater than 12 KHz with high variation in the received signal.

The following conclusions can be made from the performance and stability tests, the frequency measured on BEST^{TM} is stable with battery voltage, time and any other parameter. The performance of the device is improved with closed loop control using microcontroller. With this control, battery life of a 3.8V DC 1300mAh battery is improved 30 times from open loop control. The change battery life is because of the intermittent operation of the telemeter using microcontroller With this the battery life can be improved up to 30% depending upon the intermittent duration of operation [6.1]. Also, during continuous operation of class E amplifier the temperature of the MOSFET increases because of the switching losses. This increases the power consumption by changing the operating characteristics of amplifier. So with intermittent operation, the MOSFET is cooled during the pauses improving the battery life further. The operating characteristics of MOSFET IRF510 are shown in Appendix B. Also it is important to notice there is no change in battery life with change in distance between BEST and sensor.

So a low power, minimum motion artifact, wearable (less weight because of light weight of batteries and other circuitry) and portable biotelemeter for batteryless Gastro esophageal reflux disease (GERD) sensors was designed.

CHAPTER 7

FUTURE WORK

The portable bio-telemeter BEST™ was designed for detection of acid reflux in esophagus using impedance sensing (GERD impedance sensor) only. All the desired features of the BEST for the GERD application are given below.

- Portable
- Flexible
- Light weight
- Multi-sensing (pH and impedance)
- Record to a user friendly device
- Power Consumption
- Reading range
- Motion artifact
- Rechargeable

Of all the desired features of BEST™, basic features are implemented in current telemeter design. Some of the different ways of implementing the other features are discussed in the next sections. A next generation flexible belt model for BEST™ implementing all the features can be designed as shown in Fig.7.1. This model includes a bigger antenna and telemetry circuit embedded in a flexible cloth belt.

7.1 Flexibility

The BEST™ can be made flexible by reducing the size of telemeter circuitry to smallest size possible and embedding it in a flexible cloth belt. So all the telemeter circuitry, Secured Digital (SD) card and battery on the current design should made to fit in the inner dark box shown in Fig.7.1. The telemeter circuit can be reduced by redesigning at SMD level or replacing

whole circuit with microchip which can be designed to perform all the telemeter functions. The size of the SD card can be reduced by replacing with a micro SD card. For reducing the size of the batteries, the current ones can be replaced with a less capacity batteries by trading off with duration of operation with full charging.

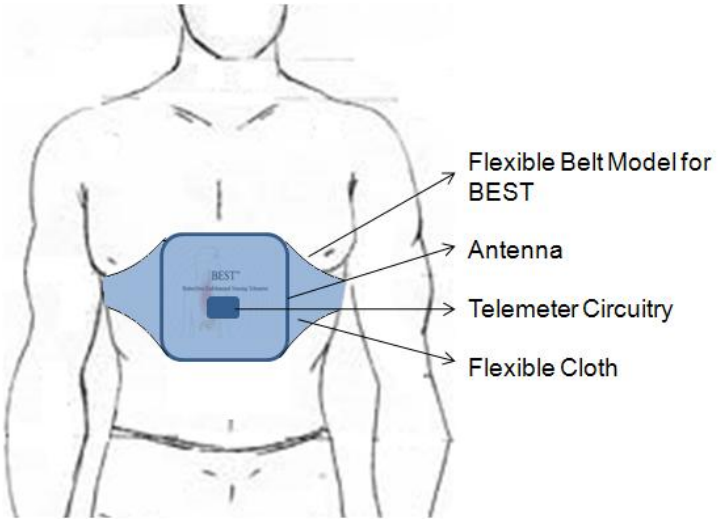


Figure 7.1 A next generation flexible belt model for BEST™.

7.2 Multi-sensing

The current telemeter was programmed to detect the acid reflux in esophagus by impedance change on GERD impedance sensor only. But for GERD diagnosis, it is essential to monitor both PH and impedance, so the telemeter has to be programmed to receive both signals from GERG Impedance-pH sensors as shown in Fig.1.5 (b). For this sensor, one frequency generator generates frequency depending upon the impedance of the medium and the other one generate depending upon PH value of the medium. The MUX switches between these two generators with 1Hz frequency. Now the microcontroller program on the telemeter has to be changed to processes these two signals and record to SD card.

7.3 Reading Distance

The current reading distance of the telemeter can be improved by increasing the radius of the antenna coil. At the same time the thickness of the wire and number of coil turns has to be small to make it flexible. An optimum size among all these parameters has to be determined.

7.4 Power Consumption

As some of the important parameters like antenna size are going to be changed from the current design, the power consumption of the device also changes. Another important parameter that effects the power consumption is microcontroller timing. The current design operates only for 100ms to measure the frequency signal and goes to sleep mode for next 900ms, this cycle repeats. But for the multi sensing the telemeter has to operate for at least 1 sec to measure both the signals. So to improve the power consumption microcontroller can be programmed to operate for 1sec and go to sleep mode for next 1sec repeating the cycle.

APPENDIX A

MICROCONTROLLER PROGRAM

Code 1: Microcontroller Configuration

```
#include "p30f4013.h"
_FOSC(CSW_FSCM_OFF & FRC_PLL8);
_FWDT(WDT_OFF);
_FBORPOR(PBOR_OFF & BORV_20 & PWRT_64 & MCLR_EN);
_FGS(CODE_PROT_OFF);
```

Code 2: Pin Definitions

```
#define LED PORTBbits.RB0
#define LED_DIR TRISBbits.TRISB0

#define FG PORTBbits.RB10
#define FG_DIR TRISBbits.TRISB10

#define SD_PWR PORTBbits.RB1
#define SD_PWR_DIR TRISBbits.TRISB1
#define SD_CS PORTBbits.RB3
#define SD_CS_DIR TRISBbits.TRISB3

#define SDI PORTFbits.RF2
#define SDI_DIR TRISFbits.TRISF2
#define SCK PORTFbits.RF6
#define SCK_DIR TRISFbits.TRISF6
#define SDO PORTFbits.RF3
#define SDO_DIR TRISFbits.TRISF3
```

Code 3: Timer 1 Initiation

```
void Init_Timer1( void )
{
T1CON = 0;                /* Reset Timer 1 */
IFS0bits.T1IF = 0;        /* Reset Timer 1 interrupt flag */
IEC0bits.T1IE = 1;        /* Enable Timer 1 interrupt */
PR1 = 0xFFFF;             /* Timer 1 period register was set to increment its register (TMR1) for
                          each external pulse */
T1CONbits.TCS = 0;        /* Set timer 1 in counter mode i.e set it to external timer clock */
T1CON =0x8006;            /* Turn ON Timer 1*/
}
```

Code 4: Timer 2 Initiation

```
void Init_Timer2( void )
{
T2CON = 0;                /* Reset Timer 2 */
 IFS0bits.T2IF = 0;       /* Reset Timer 2 interrupt flag */
IEC0bits.T2IE = 1;        /* Enable Timer 2 interrupt */
 PR2 = 0x591E;            /* Set Timer 2 period register to generate an interrupt for every 100ms
*/
T2CONbits.TCS = 0;        /* select external timer clock */
T2CON =0x8020;            /* enable Timer 2 and start the count */
}
```

Code 5: Complete working of frequency counter and data recording to Data Table

```
Main()
{
.
.
While(1)
{
Init_Timer1( );                    /*Turn ON Timer 1*/
Init_Timer2( );                    /*Turn ON Timer 2*/
.
.
/*Information Signal Processing*/
.
}
}
```

Code 5 (a):

```
void __attribute__((interrupt)) _T2Interrupt( void )
{
freq[m]=(TMR1*10);          /* Frequency = 10 x number of pulses counted ; Record Frequency
on Frequency Buffer*/
TMR1=0;                     /* Reset Counter */
if(m==0)                    /* If Time Buffer is full, record time on the top of Time Buffer */
        {
        time[0]=time[49]+1;
        }
```

```
if(m!=0)                    /* If Time buffer is not full, record time in the next row of Time buffer  */
        {
        time[m]=time[m-1]+1;
        }
m++;                        /* Increament the pointer of the table to next row*/
if(m==50)                   /* If table is full, go to top of the table */
        {
                m=0;
        }
flag=1;                     /* Permit the main program to process the recorded  information */
IFS0bits.T2IF = 0;          /* Reset Timer 1 interrupt flag */
}
```

Code 6: Preprocessing of Data Buffer

```
unsigned char buf[512];
int main (void)
{
                unsigned int p;
                p = 0;              //reset counter
                buf[0] ='T';
                buf[1]='i';
                buf[2]='m';
                buf[3]='e';
                buf[4]=0x09;
                buf[5]='F';
                buf[6]='r';
                buf[7]='e';
```

```
                buf[8]='q';
                buf[9]=0x20;
                buf[10]=0x0D;
                buf[11]=0x0A;
                p=12;
.

.
/*Data Processing*/
.
.
}
```

Code 7: Processing of data in Data Table

```
/*Check if writing pointer leads reading pointer; Check if SD card data buffer is full; Check if data
processing is permitted by the interrupt routine */
if((flag==1)&(p<=497)&((m-n)>0)|((m==0)&(n==49)))
{
        flag=0;                         /*Disable the permit to process information*/
/*Program to convert a Time number to its individual digits and store this data on to DATA
BUFFER*/
        div=1000000000;
        while(div>=1)
        {
                digit=time[n]/div;      /*Divide the number by powers of ten*/
                digit=(digit%10);       /*Digit is reminder when divided by ten*/
```

```
            if((digit!=0)|(flag==2))      /*Convert a number to individual digits*/
            {
                  flag=2;                 /*Check if zero is significant*/
                  buf[p]=digit+0x30; /*Convert each digit to its corresponding ASCII
value*/
                  p++;                    /*Increment the Data Buffer pointer*/
            }
            div/=10;                      /*Repeat for different powers of ten*/
      }
      flag=0;                             /*Disable the permit to process information*/
      buf[p]=0x09;  /*Give as space between Time digit data and frequency digit data*/
      p++;
      div=10000;                          /*Increment the Data Buffer pointer*/

/*Program to convert a Frequency number to its individual digits and store this data on to DATA
BUFFER; Repeat the above program again for Frequency*/
      while(div>=1)
      {
            digit=freq[n]/div;
            digit=(digit%10);
            if((digit!=0)|(flag==2))
            {
                  flag=2;//check for zeros
                  buf[p]=digit+0x30;
                  p++;
            }
```

```
            div/=10;
        }
        n++;
        if(n==50)                   /*Check if reading pointer is at bottom of the Data Table*/
        n=0;
        flag=0;
        buf[p]=0x0D;                /*ASCII value of 'Enter' to go to next line*/
        p++;
        buf[p]=0x0A;
        p++;
        flag=0;
        if(p>497)
        {
            for(;p<510;p++)
            buf[p]=0;
            buf[510]=0x0D;
            buf[511]=0x0A;
            p=0;
        }
}
```

Code 8: Complete Code for Open Loop Control

```
#include "p30f4013.h"
_FOSC(CSW_FSCM_OFF & FRC_PLL8);
_FWDT(WDT_OFF);
_FBORPOR(PBOR_OFF & BORV_20 & PWRT_64 & MCLR_EN);
_FGS(CODE_PROT_OFF);

// Pin definitions
#define LED PORTBbits.RB0
#define LED_DIR TRISBbits.TRISB0
#define FG PORTBbits.RB10
#define FG_DIR TRISBbits.TRISB10
#define SD_PWR PORTBbits.RB1
#define SD_PWR_DIR TRISBbits.TRISB1
#define SD_CS PORTBbits.RB3
#define SD_CS_DIR TRISBbits.TRISB3
#define SDI PORTFbits.RF2
#define SDI_DIR TRISFbits.TRISF2
#define SCK PORTFbits.RF6
#define SCK_DIR TRISFbits.TRISF6
#define SDO PORTFbits.RF3
#define SDO_DIR TRISFbits.TRISF3

// R1 Response Codes from SD Card
#define R1_IN_IDLE_STATE    (1<<0)   // Card is in idle state and running initializing process.
#define R1_ERASE_RESET      (1<<1)   // An erase sequence was cleared before executing.
```

```
#define R1_ILLEGAL_COMMAND  (1<<2)   // An illegal command code was detected
#define R1_COM_CRC_ERROR    (1<<3)   // The CRC check of the last command failed.
#define R1_ERASE_SEQ_ERROR  (1<<4)  // An error in the erase commands occurred.
#define R1_ADDRESS_ERROR    (1<<5)    // A misaligned in address
#define R1_PARAMETER        (1<<6)        // The command's argument was out of range

// Timer 1 Initiation
void Init_Timer1( void )
{
T1CON = 0;                /* Reset Timer 1 */
IFS0bits.T1IF = 0;        /* Reset Timer 1 interrupt flag */
IEC0bits.T1IE = 1;        /* Enable Timer 1 interrupt */
PR1 = 0xFFFF;             /* Timer 1 period register was set to increment its register (TMR1) for
                          each external pulse */
T1CONbits.TCS = 0;      /* Set timer 1 in counter mode i.e set it to external timer clock */
T1CON =0x8006;           /* Turn ON Timer 1*/
}

// Timer 2 Initiation
void Init_Timer2( void )
{
T2CON = 0;                /* Reset Timer 2 */
 IFS0bits.T2IF = 0;       /* Reset Timer 2 interrupt flag */
IEC0bits.T2IE = 1;        /* Enable Timer 2 interrupt */
 PR2 = 0x591E;            /* Set Timer 2 period register to generate an interrupt for every 100ms
*/
```

```
T2CONbits.TCS = 0;      /* select external timer clock */
T2CON =0x8020;          /* enable Timer 2 and start the count */
}

void SPIWrite(unsigned char data)
{
        SPI1BUF = data;                 /* Tranfer data to SPI Buffer Register*/
        while(SPI1STATbits.SPITBF);     /*Wait until send buffer is ready for more data*/
}

unsigned char SPIRead(void)
{
unsigned char data;
if(SPI1STATbits.SPIRBF)                 /*Check if SPI Transmit Buffer is full*/
        {
        data = SPI1BUF;                 /*Don't initiate a read ;Read the existing data */
        SPI1STATbits.SPIROV = 0;        /*Clear receive overflow flag bit*/
        return data;
        }
        /* No data availabel, initiate a read*/
        SPI1BUF = 0xFF;                 /*write dummy data to initiate an SPI read*/
        while(SPI1STATbits.SPITBF);   /* wait until the data is finished reading*?
        data = SPI1BUF;
        SPI1STATbits.SPIROV = 0;
        return data;
}
```

```
void SPI_Init(void)
{
        SD_PWR = 0;             /*Turn OFF SD card Power*/
        SD_PWR_DIR = 0;         /* Set pin RB1 direction to output*/
        SD_PWR = 0;             /*Turn OFF SD card Power*/
        SD_CS = 1;              /*Deactivate SD Card chip select */
        SD_CS_DIR = 0;          /*Set pin RB3 direction to output*/
        SD_CS = 1;              /*Deactivate SD Card chip select */
        SDI_DIR = 1;            /*Set pin RF2 direction to input*/
        SCK_DIR = 1;            /*Set pin RF6 direction to input*/
        SDO_DIR = 1;            /*Set pin RF3 direction to input*/
        SPI1STAT = 0x8000;      /*Enable SPI port*/
        // Configure SPI port
        // set SPI port to slowest setting
        // master mode
        // 8 bit
        // Idle state for Clock is high level
        // Primary prescaler 64:1
        // Secondary prescaler 8:1
        SPI1CON = 0x0060;
}
```

```
unsigned char SD_WriteCommand(unsigned char* cmd)
{
unsigned int i;
unsigned char response;
unsigned char savedSD_CS = SD_CS;

/*Set the framing bits correctly*/
cmd[0] |= (1<<6);
cmd[0] &= ~(1<<7);
cmd[5] |= (1<<0);

/*Send the 6 byte command*/
SD_CS = 0;
for(i = 0; i < 6; ++i)
{
        SPIWrite(*cmd);
        cmd++;
}

/* Wait for the valid response*/
i = 0;
do
{
response = SPIRead();
if(i > 100)
        {
```

```
                    break;
            }
            i++;
} while(response == 0xFF);
SD_CS = 1;

/*Wait 8 clocks to process the command by SD card*/
SPIWrite(0xFF);
SD_CS = savedSD_CS;
return(response);
}

unsigned char SD_Init()/*Initiate SD Card*/
{
unsigned int i = 0;
unsigned char status;
SD_CS = 1;                          /*Deactivate SD Card chip select*/
SD_PWR = 0;                         /*Turn OFF SD card Power*/

/*Wait for power to go down*/
for(i = 0; i; i++);
for(i = 0; i; i++);
for(i = 0; i; i++);
for(i = 0; i; i++);

SD_PWR = 1;                         /*Turn on SD Card*/
```

```
for(status = 0; status < 10; ++status)        /*Wait for power to come up*/
        {
        for(i = 0; i; i++);
        for(i = 0; i; i++);
        for(i = 0; i; i++);
        for(i = 0; i; i++);
        }

for(i = 0; i < 16; ++i)       /*We need to give SD Card about a hundred clock cycles to boot up*/
        {
        SPIWrite(0xFF); /*Write dummy data to pump clock signal line*/
        }
SD_CS = 0;                    /*Activate SD card chip select*/

unsigned char CMD0_GO_IDLE_STATE[] = {0x00,0x00,0x00,0x00,0x00,0x95};
/*Define GO IDLE STATE Command with a valid CRC*/

// Wait for the SD Card to go into IDLE state
i = 0;
do
        {
        status = SD_WriteCommand(CMD0_GO_IDLE_STATE);
         /*Send GO IDLE STATE Command SD card with a valid CRC*/
        if(i++ > 50)                        /*If response is not valid, try 50 times and give up */
                {return 1;}
```

```
        } while( status != 0x01 );  /*Wait for valid response from SD card*/


unsigned char CMD1_SEND_OP_COND[] = {0x01,0x00,0x00,0x00,0x00,0xFF};
/*Define SEND OPERATING CONDITIONS command w/o valid CRC*/
i = 0;
do
        {
        status = SD_WriteCommand(CMD1_SEND_OP_COND);
/*Send SEND OPERATING CONDITIONS command to SD card*/
        if(i++ > 50)                /*If response is not valid, try 50 times and give up */
                {return 2;}
        } while( (status & R1_IN_IDLE_STATE) != 0 );   /*Wait for IN IDLE STATE response
from SD card*/


unsigned char CMD55_APP_CMD[] = {55,0x00,0x00,0x00,0x00,0xFF};
/*Define Application Specific CMD55 command w/o valid CRC*/


status = SD_WriteCommand(CMD55_APP_CMD);        /*Send Application Specific CMD55
command to SD card and ignore the response from SD card*/


i = 0;
unsigned char ACMD41_SD_SEND_OP_COND[] = {41,0x00,0x00,0x00,0x00,0xFF};
/*Define ACMD41 command to initialize SD Card mode w/o valid CRC*/
do
        {
        status = SD_WriteCommand(ACMD41_SD_SEND_OP_COND);
```

```
        /*Send ACMD41 command to SD Card*/
        if(i++ > 50)          /*If response is not valid, try 50 times and give up */
                {return 3;}
        } while( (status & R1_IN_IDLE_STATE) != 0 );
        /*Wait for IN IDLE STATE response from SD card*/
SD_CS = 1;/*Deactivate SD card chip select*/
return 0;
}



unsigned char SD_WriteBlock(unsigned long addr, unsigned char *buf)/*Write a 512 Data Buffer
Block to SD card*/
{
unsigned char response;
unsigned int i, retry=0;
unsigned char CMD24_WRITE_SINGLE_BLOCK[] = {24,0x00,0x00,0x00,0x00,0xFF};
/*Define WRITE SINGLE BLOCK command(CMD34) w/o valid CRC*/
CMD24_WRITE_SINGLE_BLOCK[1] = ((addr & 0xFF000000) >> 24);
/*Include address of the memory location on SD card in the command*/
CMD24_WRITE_SINGLE_BLOCK[2] = ((addr & 0x00FF0000) >> 16);
CMD24_WRITE_SINGLE_BLOCK[3] = ((addr & 0x0000FF00) >> 8);
CMD24_WRITE_SINGLE_BLOCK[4] = ((addr & 0x000000FF));
SD_CS = 0;                              /*Activate SD card chip select*/
response = SD_WriteCommand(CMD24_WRITE_SINGLE_BLOCK);
/*Send WRITE SINGLE BLOCK command(CMD34)to SD card*/
```

```
if(response != 0x00)                    /*check for SD status: 0x00 - OK (No flags set)*/
return 1;

SPIWrite(0xFF);
SPIWrite(0xFE);                         /*Send start block token 0xfe (0x11111110)*/

for(i=0; i<512; i++)                    /*Send 512 bytes of Data*/
SPIWrite(buf[i]);

SPIWrite(0xff);     //Transmit dummy CRC (16-bit), CRC is ignored here
SPIWrite(0xff);

i=0;
do                                      /*Data response and busy response from card*/
        {
        response = SPIRead();
        i++;
        }while((response & 0x1f) != 0x05);
i=0;
while(!SPIRead())                       /*wait for SD card to complete writing and get idle*/
        {
        i++;
        if(retry++ > 0xfffe)            /*If SD card is busy for very long time*/
                {
                SD_CS = 1;              /*Deactivate SD card chip select*/
                return 1;
```

```
                }
        }
SD_CS = 1;                              /*Deactivate SD card chip select*/
SPIWrite(0xff);                         /*Wait 8 clock cycles*/
SD_CS = 0;                               /*Activate SD card chip select*/
//verify if card is still busy
while(!SPIRead())                       //wait for SD card to complete writing and get idle
         {
        i++;
         if(retry++ > 0xfffe)                 /*If SD card is busy for very long time*/
                {
                SD_CS = 1;                    /*Deactivate SD card chip select*/
                return 1;
                }
        }
SD_CS = 1;/*Deactivate SD card chip select*/
return 0;
}

unsigned char buf[512];
unsigned int flag,freq[50],m=0;
long long int digit,div,time[50];
int FG_ON=1;

int main (void)
{
```

```
        unsigned int i, j, x;
        unsigned int p;
        unsigned long y;
        unsigned char status;
        ADPCFG = 0xFFFF; // Force all ADC pins as digital I/O

        Init_Timer1( );
        Init_Timer2( );
        TMR2=0;
        TMR1=0;
//Preprocessing of Data Buffer
                p = 0; //reset counter
                buf[0] ='T';
                buf[1]='i';
                buf[2]='m';
                buf[3]='e';
                buf[4]=0x09;
                buf[5]='F';
                buf[6]='r';
                buf[7]='e';
                buf[8]='q';
                buf[9]=0x20;
                buf[10]=0x0D;
                buf[11]=0x0A;
                p=12;
```

```
        // Configure output pins
                LED_DIR = 0;
        LED = 1;
        y=28672;
int n=0;



while(1)
        {

        if((flag==1)&(p<=497)&((m-n)>0)|((m==0)&(n==49)))
                {
                flag=0;                 /*Disable the permit to process information*/
/*Program to convert a Time number to its individual digits and store this data on to DATA
BUFFER*/
        div=1000000000;
        while(div>=1)
        {
                digit=time[n]/div;          /*Divide the number by powers of ten*/
                digit=(digit%10);           /*Digit is reminder when divided by ten*/
                if((digit!=0)|(flag==2))    /*Convert a number to individual digits*/
                {
                        flag=2;             /*Check if zero is significant*/
```

```
                buf[p]=digit+0x30; /*Convert each digit to its corresponding ASCII
value*/
                p++;                /*Increment the Data Buffer pointer*/
            }
            div/=10;                /*Repeat for different powers of ten*/
        }
        flag=0;                     /*Disable the permit to process information*/
        buf[p]=0x09;  /*Give as space between Time digit data and frequency digit data*/
        p++;
        div=10000;                  /*Increment the Data Buffer pointer*/

/*Program to convert a Frequency number to its individual digits and store this data on to DATA
BUFFER; Repeat the above program again for Frequency*/
        while(div>=1)
        {
            digit=freq[n]/div;
            digit=(digit%10);
            if((digit!=0)|(flag==2))
            {
                flag=2;//check for zeros
                buf[p]=digit+0x30;
                p++;
            }
            div/=10;
        }
        n++;
```

```
        if(n==50)                   /*Check if reading pointer is at bottom of the Data Table*/
        n=0;
        flag=0;
        buf[p]=0x0D;                /*ASCII value of 'Enter' to go to next line*/
        p++;
        buf[p]=0x0A;
        p++;
        flag=0;
        if(p>497)
        {
                for(;p<510;p++)
                buf[p]=0;
                buf[510]=0x0D;
                buf[511]=0x0A;
                p=0;
        }
}

SPI_Init();                 /*Initiate SPI perpherals of microcontroller*/
status = SD_Init();             /*Initiate Communication with SD card and get the status of SD
card */
do
        {
        while(status)           /*Wiat till SD card is ready for Data tranfer */
                {
```

```
                status = SD_Init();/*Initiate Communication with SD card and get the status of
SD card */
                LED = 0;        /*Turn ON red LED in case of communication with SD card
failed*/
                };

        status = SD_WriteBlock(y,buf);/*Write the 512 bytes block of data(Data Buffer)to SD
Card at the memory location y and get status of SD card*/
        } while(status);        /*If failed to write try till Data Buffer Block is written*/
y+=512;                 /*Increment the Adress location on SD card by 512 to write next 512
bytes of data to it*/
}
}
i=0;
while(i <= 40)
        {
                LED = i & 1;
                for(j = 0; j < 40000; ++j);
                i++;
        }

}
        return 0;
}
```

```
void __attribute__((interrupt)) _T2Interrupt( void )
{
freq[m]=(TMR1*10);
TMR1=0;
if(m==0)
        {
        time[0]=time[49]+100;
        }
if(m!=0)
        {
        time[m]=time[m-1]+100;
        }
m++;
if(m==50)
        {m=0;}
flag=1;
/* reset Timer 1 interrupt flag */
IFS0bits.T2IF = 0;
}
```

Code 9: Complete Code for Closed Loop Control

```
#include "p30f4013.h"
_FOSC(CSW_FSCM_OFF & FRC_PLL8);
_FWDT(WDT_OFF);
_FBORPOR(PBOR_OFF & BORV_20 & PWRT_64 & MCLR_EN);
_FGS(CODE_PROT_OFF);

// Pin definitions
#define LED PORTBbits.RB0
#define LED_DIR TRISBbits.TRISB0
#define FG PORTBbits.RB10
#define FG_DIR TRISBbits.TRISB10
#define SD_PWR PORTBbits.RB1
#define SD_PWR_DIR TRISBbits.TRISB1
#define SD_CS PORTBbits.RB3
#define SD_CS_DIR TRISBbits.TRISB3
#define SDI PORTFbits.RF2
#define SDI_DIR TRISFbits.TRISF2
#define SCK PORTFbits.RF6
#define SCK_DIR TRISFbits.TRISF6
#define SDO PORTFbits.RF3
#define SDO_DIR TRISFbits.TRISF3

// R1 Response Codes from SD Card
#define R1_IN_IDLE_STATE    (1<<0)   // Card is in idle state and running initializing process.
#define R1_ERASE_RESET      (1<<1)   // An erase sequence was cleared before executing.
```

```
#define R1_ILLEGAL_COMMAND  (1<<2)   // An illegal command code was detected
#define R1_COM_CRC_ERROR    (1<<3)   // The CRC check of the last command failed.
#define R1_ERASE_SEQ_ERROR  (1<<4)  // An error in the erase commands occurred.
#define R1_ADDRESS_ERROR    (1<<5)    // A misaligned in address
#define R1_PARAMETER        (1<<6)        // The command's argument was out of range

// Timer 1 Initiation
void Init_Timer1( void )
{
T1CON = 0;                /* Reset Timer 1 */
IFS0bits.T1IF = 0;        /* Reset Timer 1 interrupt flag */
IEC0bits.T1IE = 1;        /* Enable Timer 1 interrupt */
PR1 = 0xFFFF;              /* Timer 1 period register was set to increment its register (TMR1) for
                          each external pulse */
T1CONbits.TCS = 0;      /* Set timer 1 in counter mode i.e set it to external timer clock */
T1CON =0x8006;           /* Turn ON Timer 1*/
}

// Timer 2 Initiation
void Init_Timer2( void )
{
T2CON = 0;                /* Reset Timer 2 */
 IFS0bits.T2IF = 0;       /* Reset Timer 2 interrupt flag */
IEC0bits.T2IE = 1;        /* Enable Timer 2 interrupt */
 PR2 = 0x591E;            /* Set Timer 2 period register to generate an interrupt for every 100ms
*/
```

```
T2CONbits.TCS = 0;      /* select external timer clock */
T2CON =0x8020;          /* enable Timer 2 and start the count */
}

void SPIWrite(unsigned char data)
{
        SPI1BUF = data;                 /* Tranfer data to SPI Buffer Register*/
        while(SPI1STATbits.SPITBF);     /*Wait until send buffer is ready for more data*/
}

unsigned char SPIRead(void)
{
unsigned char data;
if(SPI1STATbits.SPIRBF)                 /*Check if SPI Transmit Buffer is full*/
        {
        data = SPI1BUF;                 /*Don't initiate a read ;Read the existing data */
        SPI1STATbits.SPIROV = 0;        /*Clear receive overflow flag bit*/
        return data;
        }
        /* No data availabel, initiate a read*/
        SPI1BUF = 0xFF;                 /*write dummy data to initiate an SPI read*/
        while(SPI1STATbits.SPITBF);   /* wait until the data is finished reading*?
        data = SPI1BUF;
        SPI1STATbits.SPIROV = 0;
        return data;
}
```

```
void SPI_Init(void)
{
        SD_PWR = 0;             /*Turn OFF SD card Power*/
        SD_PWR_DIR = 0;         /* Set pin RB1 direction to output*/
        SD_PWR = 0;             /*Turn OFF SD card Power*/
        SD_CS = 1;              /*Deactivate SD Card chip select */
        SD_CS_DIR = 0;          /*Set pin RB3 direction to output*/
        SD_CS = 1;              /*Deactivate SD Card chip select */
        SDI_DIR = 1;            /*Set pin RF2 direction to input*/
        SCK_DIR = 1;            /*Set pin RF6 direction to input*/
        SDO_DIR = 1;            /*Set pin RF3 direction to input*/
        SPI1STAT = 0x8000;      /*Enable SPI port*/
        // Configure SPI port
        // set SPI port to slowest setting
        // master mode
        // 8 bit
        // Idle state for Clock is high level
        // Primary prescaler 64:1
        // Secondary prescaler 8:1
        SPI1CON = 0x0060;
}
```

```
unsigned char SD_WriteCommand(unsigned char* cmd)
{
unsigned int i;
unsigned char response;
unsigned char savedSD_CS = SD_CS;

/*Set the framing bits correctly*/
cmd[0] |= (1<<6);
cmd[0] &= ~(1<<7);
cmd[5] |= (1<<0);

/*Send the 6 byte command*/
SD_CS = 0;
for(i = 0; i < 6; ++i)
{
        SPIWrite(*cmd);
        cmd++;
}

/* Wait for the valid response*/
i = 0;
do
{
response = SPIRead();
if(i > 100)
        {
```

```
                    break;
            }
            i++;
} while(response == 0xFF);
SD_CS = 1;

/*Wait 8 clocks to process the command by SD card*/
SPIWrite(0xFF);
SD_CS = savedSD_CS;
return(response);
}

unsigned char SD_Init()/*Initiate SD Card*/
{
unsigned int i = 0;
unsigned char status;
SD_CS = 1;                          /*Deactivate SD Card chip select*/
SD_PWR = 0;                         /*Turn OFF SD card Power*/

/*Wait for power to go down*/
for(i = 0; i; i++);
for(i = 0; i; i++);
for(i = 0; i; i++);
for(i = 0; i; i++);

SD_PWR = 1;                         /*Turn on SD Card*/
```

```
for(status = 0; status < 10; ++status)        /*Wait for power to come up*/
        {
        for(i = 0; i; i++);
        for(i = 0; i; i++);
        for(i = 0; i; i++);
        for(i = 0; i; i++);
        }

for(i = 0; i < 16; ++i)       /*We need to give SD Card about a hundred clock cycles to boot up*/
        {
        SPIWrite(0xFF); /*Write dummy data to pump clock signal line*/
        }
SD_CS = 0;                    /*Activate SD card chip select*/

unsigned char CMD0_GO_IDLE_STATE[] = {0x00,0x00,0x00,0x00,0x00,0x95};
/*Define GO IDLE STATE Command with a valid CRC*/

// Wait for the SD Card to go into IDLE state
i = 0;
do
        {
        status = SD_WriteCommand(CMD0_GO_IDLE_STATE);
         /*Send GO IDLE STATE Command SD card with a valid CRC*/
        if(i++ > 50)                        /*If response is not valid, try 50 times and give up */
                {return 1;}
```

```
        } while( status != 0x01 );  /*Wait for valid response from SD card*/


unsigned char CMD1_SEND_OP_COND[] = {0x01,0x00,0x00,0x00,0x00,0xFF};
/*Define SEND OPERATING CONDITIONS command w/o valid CRC*/
i = 0;
do
        {
        status = SD_WriteCommand(CMD1_SEND_OP_COND);
/*Send SEND OPERATING CONDITIONS command to SD card*/
        if(i++ > 50)                /*If response is not valid, try 50 times and give up */
                {return 2;}
        } while( (status & R1_IN_IDLE_STATE) != 0 );    /*Wait for IN IDLE STATE response
from SD card*/


unsigned char CMD55_APP_CMD[] = {55,0x00,0x00,0x00,0x00,0xFF};
/*Define Application Specific CMD55 command w/o valid CRC*/


status = SD_WriteCommand(CMD55_APP_CMD);        /*Send Application Specific CMD55
command to SD card and ignore the response from SD card*/


i = 0;
unsigned char ACMD41_SD_SEND_OP_COND[] = {41,0x00,0x00,0x00,0x00,0xFF};
/*Define ACMD41 command to initialize SD Card mode w/o valid CRC*/
do
        {
        status = SD_WriteCommand(ACMD41_SD_SEND_OP_COND);
```

```
        /*Send ACMD41 command to SD Card*/
        if(i++ > 50)          /*If response is not valid, try 50 times and give up */
                {return 3;}
        } while( (status & R1_IN_IDLE_STATE) != 0 );
        /*Wait for IN IDLE STATE response from SD card*/
SD_CS = 1;/*Deactivate SD card chip select*/
return 0;
}



unsigned char SD_WriteBlock(unsigned long addr, unsigned char *buf)/*Write a 512 Data Buffer
Block to SD card*/
{
unsigned char response;
unsigned int i, retry=0;
unsigned char CMD24_WRITE_SINGLE_BLOCK[] = {24,0x00,0x00,0x00,0x00,0xFF};
/*Define WRITE SINGLE BLOCK command(CMD34) w/o valid CRC*/
CMD24_WRITE_SINGLE_BLOCK[1] = ((addr & 0xFF000000) >> 24);
/*Include address of the memory location on SD card in the command*/
CMD24_WRITE_SINGLE_BLOCK[2] = ((addr & 0x00FF0000) >> 16);
CMD24_WRITE_SINGLE_BLOCK[3] = ((addr & 0x0000FF00) >> 8);
CMD24_WRITE_SINGLE_BLOCK[4] = ((addr & 0x000000FF));
SD_CS = 0;                              /*Activate SD card chip select*/
response = SD_WriteCommand(CMD24_WRITE_SINGLE_BLOCK);
/*Send WRITE SINGLE BLOCK command(CMD34)to SD card*/
```

```
if(response != 0x00)                    /*check for SD status: 0x00 - OK (No flags set)*/
return 1;

SPIWrite(0xFF);
SPIWrite(0xFE);                         /*Send start block token 0xfe (0x11111110)*/

for(i=0; i<512; i++)                    /*Send 512 bytes of Data*/
SPIWrite(buf[i]);

SPIWrite(0xff);     //Transmit dummy CRC (16-bit), CRC is ignored here
SPIWrite(0xff);

i=0;
do                                      /*Data response and busy response from card*/
        {
        response = SPIRead();
        i++;
        }while((response & 0x1f) != 0x05);
i=0;
while(!SPIRead())                       /*wait for SD card to complete writing and get idle*/
        {
        i++;
        if(retry++ > 0xfffe)            /*If SD card is busy for very long time*/
                {
                SD_CS = 1;              /*Deactivate SD card chip select*/
                return 1;
```

```
                }
        }
SD_CS = 1;                              /*Deactivate SD card chip select*/
SPIWrite(0xff);                         /*Wait 8 clock cycles*/
SD_CS = 0;                               /*Activate SD card chip select*/
//verify if card is still busy
while(!SPIRead())                       //wait for SD card to complete writing and get idle
         {
        i++;
         if(retry++ > 0xfffe)                 /*If SD card is busy for very long time*/
                {
                SD_CS = 1;                    /*Deactivate SD card chip select*/
                return 1;
                }
        }
SD_CS = 1;/*Deactivate SD card chip select*/
return 0;
}

unsigned char buf[512];
unsigned int flag,freq[50],m=0;
long long int digit,div,time[50];
int FG_ON=1;

int main (void)
{
```

```
    unsigned int i, j, x;
    unsigned int p;
    unsigned long y;
    unsigned char status;
    ADPCFG = 0xFFFF; // Force all ADC pins as digital I/O

//Preprocessing of Data Buffer
        p = 0; //reset counter
        buf[0] ='T';
        buf[1]='i';
        buf[2]='m';
        buf[3]='e';
        buf[4]=0x09;
        buf[5]='F';
        buf[6]='r';
        buf[7]='e';
        buf[8]='q';
        buf[9]=0x20;
        buf[10]=0x0D;
        buf[11]=0x0A;
        p=12;


    // Configure output pins
        LED_DIR = 0;
```

```
            LED = 1;
            y=28672;
        int n=0;
            FG_DIR= 0;
            FG=0;


while(1)
        {
        FG_ON=1;
        Init_Timer1( );/*Turn ON Timer 1*/
        Init_Timer2( );/*Turn ON Timer 2*/
        TMR2=0;
        TMR1=0;

        while(FG_ON==1)
        {
           FG=1;
            FG=1;
            FG=1;
            FG=1;
            FG=0;
            FG=0;
            FG=0;
        }
```

```
        IEC0bits.T1IE = 0;
        IEC0bits.T2IE = 0;
if((flag==1)&(p<=497)&((m-n)>0)|((m==0)&(n==49)))
{
        flag=0;                         /*Disable the permit to process information*/
/*Program to convert a Time number to its individual digits and store this data on to DATA
BUFFER*/
        div=1000000000;
        while(div>=1)
        {
                digit=time[n]/div;      /*Divide the number by powers of ten*/
                digit=(digit%10);       /*Digit is reminder when divided by ten*/
                if((digit!=0)|(flag==2))    /*Convert a number to individual digits*/
                {
                        flag=2;             /*Check if zero is significant*/
                        buf[p]=digit+0x30; /*Convert each digit to its corresponding ASCII
value*/
                        p++;                /*Increment the Data Buffer pointer*/
                }
                div/=10;                    /*Repeat for different powers of ten*/
        }
        flag=0;                         /*Disable the permit to process information*/
        buf[p]=0x09;  /*Give as space between Time digit data and frequency digit data*/
        p++;
        div=10000;                          /*Increment the Data Buffer pointer*/
```

```
/*Program to convert a Frequency number to its individual digits and store this data on to DATA
BUFFER; Repeat the above program again for Frequency*/
        while(div>=1)
        {
                digit=freq[n]/div;
                digit=(digit%10);
                if((digit!=0)|(flag==2))
                {
                        flag=2;//check for zeros
                        buf[p]=digit+0x30;
                        p++;
                }
                div/=10;
        }
        n++;
        if(n==50)                       /*Check if reading pointer is at bottom of the Data Table*/
        n=0;
        flag=0;
        buf[p]=0x0D;                    /*ASCII value of 'Enter' to go to next line*/
        p++;
        buf[p]=0x0A;
        p++;
        flag=0;
        if(p>497)
        {
                for(;p<510;p++)
```

```
                buf[p]=0;
                buf[510]=0x0D;
                buf[511]=0x0A;
                p=0;
        }
}

SPI_Init();                     /*Initiate SPI perpherals of microcontroller*/
status = SD_Init();             /*Initiate Communication with SD card and get the status of SD
card */
do
        {
        while(status)            /*Wiat till SD card is ready for Data tranfer */
                {
                status = SD_Init();/*Initiate Communication with SD card and get the status of
SD card */
                LED = 0;     /*Turn ON red LED in case of communication with SD card failed*/
                };

        status = SD_WriteBlock(y,buf);/*Write the 512 bytes block of data(Data Buffer)to SD
Card at the memory location y and get status of SD card*/
        } while(status);      /*If failed to write try till Data Buffer Block is written*/
y+=512;                         /*Increment the Adress location on SD card by 512 to write next 512
bytes of data to it*/
}
}
```

```
i=0;
while(i <= 40)
        {
                LED = i & 1;
                for(j = 0; j < 40000; ++j);
                i++;
        }

}
        return 0;
}

void __attribute__((interrupt)) _T2Interrupt( void )
{
freq[m]=(TMR1*10);        /* Frequency = 10 x number of pulses counted ; Record Frequency on
Frequency Buffer*/
TMR1=0;                   /* Reset Counter */
if(m==0)                   /* If Time Buffer is full, record time on the top of Time Buffer */
        {
        time[0]=time[49]+1;
        }
if(m!=0)                  /* If Time buffer is not full,record time in the next row of Time buffer  */
        {
        time[m]=time[m-1]+1;
        }
m++;                    /* Increament the pointer of the table to next row*/
```

```
if(m==50)                   /* If table is full, go to top of the table */
        {
        m=0;
        }
flag=1;             /* Permit the main program to process the recorded hexadecimal information */
IFS0bits.T2IF = 0;      /* Reset Timer 1 interrupt flag */
FG_ON=0;                /*Turn OFF Frequency Generator*/
}
```

Code 10: Sensor position detect and Battery charge indicator

Interrupt routine Code:

```
void __attribute__((interrupt)) _T2Interrupt( void )
{
freq[m]=(TMR1*10);              /* Frequency = 10 x number of pulses counted ; Record Frequency
on Frequency Buffer*/
TMR1=0;                         /* Reset Counter */
LED=0;
If(freq[m]>12000)           /*Sensor Position detect*/
        {
        LED=1;
        return;
        }
If(freq[m]<6000)                /*Battery outage detect*/
        {
        LED=1;
        return;
```

```
        }
if(m==0)                    /* If Time Buffer is full, record time on the top of Time Buffer */
        {time[0]=time[49]+1;}
if(m!=0)                    /* If Time buffer is not full, record time in the next row of Time buffer  */
        {time[m]=time[m-1]+1;}
m++;                        /* Increment the pointer of the table to next row*/
if(m==50)                   /* If table is full, go to top of the table */
        {m=0;   }
flag=1;                     /* Permit the main program to process the recorded  information */
IFS0bits.T2IF = 0;          /* Reset Timer 1 interrupt flag */
}
```

Code 11:

```
void SPI_Init(void)
{
        SD_PWR = 0;             /*Turn OFF SD card Power*/
        SD_PWR_DIR = 0;         /* Set pin RB1 direction to output*/
        SD_PWR = 0;             /*Turn OFF SD card Power*/
        SD_CS = 1;              /*Deactivate SD Card chip select */
        SD_CS_DIR = 0;          /*Set pin RB3 direction to output*/
        SD_CS = 1;              /*Deactivate SD Card chip select */
        SDI_DIR = 1;            /*Set pin RF2 direction to input*/
        SCK_DIR = 1;            /*Set pin RF6 direction to input*/
        SDO_DIR = 1;            /*Set pin RF3 direction to input*/
        SPI1STAT = 0x8000;      /*Enable SPI port*/
        // Configure SPI port
        // set SPI port to slowest setting
```

```
        // master mode
        // 8 bit
        // Idle state for Clock is high level
        // Primary prescaler 64:1
        // Secondary prescaler 8:1
        SPI1CON = 0x0060;
}
```

Code 12: Write a byte of Data to SD card in SPI mode

```
void SPIWrite(unsigned char data)
{
        SPI1BUF = data;                 /* Tranfer data to SPI Buffer Register*/
        while(SPI1STATbits.SPITBF);     /*Wait until send buffer is ready for more data*/
}
```

Code13: Read a byte of Data from SD card in SPI mode

```
unsigned char SPIRead(void)
{
unsigned char data;
if(SPI1STATbits.SPIRBF)             /*Check if SPI Transmit Buffer is full*/
        {
        data = SPI1BUF;             /*Don't initiate a read ;Read the existing data */
        SPI1STATbits.SPIROV = 0;    /*Clear receive overflow flag bit*/
        return data;
        }
        /* No data availabel, initiate a read*/
```

```
        SPI1BUF = 0xFF;                 /*write dummy data to initiate an SPI read*/
        while(SPI1STATbits.SPITBF);   /* wait until the data is finished reading*?
        data = SPI1BUF;
        SPI1STATbits.SPIROV = 0;
        return data;
}
```

Code 14: Program to Write a command to SD card

```
unsigned char SD_WriteCommand(unsigned char* cmd)
{
unsigned int i;
unsigned char response;
unsigned char savedSD_CS = SD_CS;

/*Set the framing bits correctly*/
cmd[0] |= (1<<6);
cmd[0] &= ~(1<<7);
cmd[5] |= (1<<0);

/*Send the 6 byte command*/
SD_CS = 0;
for(i = 0; i < 6; ++i)
{
        SPIWrite(*cmd);
        cmd++;
}
```

```
/* Wait for the valid response*/
i = 0;
do
{
        response = SPIRead();
        if(i > 100)
        {
                break;
        }
        i++;
} while(response == 0xFF);
SD_CS = 1;

/*Wait 8 clocks to process the command by SD card*/
SPIWrite(0xFF);
SD_CS = savedSD_CS;
return(response);
}
```

Code 15: R1 Response format bit definitions

```
#define R1_IN_IDLE_STATE          (1<<0)
#define R1_ERASE_RESET            (1<<1)
#define R1_ILLEGAL_COMMAND  (1<<2)
#define R1_COM_CRC_ERROR     (1<<3)
#define R1_ERASE_SEQ_ERROR  (1<<4)
```

```
#define R1_ADDRESS_ERROR      (1<<5)
#define R1_PARAMETER          (1<<6)
```

Code 16: SD card initiation in SPI mode

```
unsigned char SD_Init()/*Initiate SD Card*/
{
unsigned int i = 0;
unsigned char status;
SD_CS = 1;  /*Deactivate SD Card chip select*/
SD_PWR = 0; /*Turn OFF SD card Power*/

/*Wait for power to go down*/
for(i = 0; i; i++);
for(i = 0; i; i++);
for(i = 0; i; i++);
for(i = 0; i; i++);

SD_PWR = 1;                         /*Turn on SD Card*/

for(status = 0; status < 10; ++status)       /*Wait for power to come up*/
        {
        for(i = 0; i; i++);
        for(i = 0; i; i++);
        for(i = 0; i; i++);
        for(i = 0; i; i++);
        }
```

```
for(i = 0; i < 16; ++i)       /*We need to give SD Card about a hundred clock cycles to boot up*/
        {
        SPIWrite(0xFF); /*Write dummy data to pump clock signal line*/
        }
SD_CS = 0;                    /*Activate SD card chip select*/

unsigned char CMD0_GO_IDLE_STATE[] = {0x00,0x00,0x00,0x00,0x00,0x95};
/*Define GO IDLE STATE Command with a valid CRC*/

// Wait for the SD Card to go into IDLE state
i = 0;
do
        {
        status = SD_WriteCommand(CMD0_GO_IDLE_STATE);
         /*Send GO IDLE STATE Command SD card with a valid CRC*/
        if(i++ > 50)                    /*If response is not valid, try 50 times and give up */
                {return 1;}
        } while( status != 0x01 );  /*Wait for valid response from SD card*/

unsigned char CMD1_SEND_OP_COND[] = {0x01,0x00,0x00,0x00,0x00,0xFF};
/*Define SEND OPERATING CONDITIONS command w/o valid CRC*/
i = 0;
do
        {
```

```
        status = SD_WriteCommand(CMD1_SEND_OP_COND);
/*Send SEND OPERATING CONDITIONS command to SD card*/
        if(i++ > 50)                /*If response is not valid, try 50 times and give up */
                {return 2;}
        } while( (status & R1_IN_IDLE_STATE) != 0 );    /*Wait for IN IDLE STATE response
from SD card*/

unsigned char CMD55_APP_CMD[] = {55,0x00,0x00,0x00,0x00,0xFF};
/*Define Application Specific CMD55 command w/o valid CRC*/

status = SD_WriteCommand(CMD55_APP_CMD);        /*Send Application Specific CMD55
command to SD card and ignore the response from SD card*/

i = 0;
unsigned char ACMD41_SD_SEND_OP_COND[] = {41,0x00,0x00,0x00,0x00,0xFF};
/*Define ACMD41 command to initialize SD Card mode w/o valid CRC*/
do
        {
        status = SD_WriteCommand(ACMD41_SD_SEND_OP_COND);
        /*Send ACMD41 command to SD Card*/
        if(i++ > 50)            /*If response is not valid, try 50 times and give up */
                {return 3;}
        } while( (status & R1_IN_IDLE_STATE) != 0 );
        /*Wait for IN IDLE STATE response from SD card*/
SD_CS = 1;/*Deactivate SD card chip select*/
return 0;
```

```
}
```

Code 17: Write a 512 bytes data block to SD card

```
unsigned char SD_WriteBlock(unsigned long addr, unsigned char *buf)/*Write a 512 Data Buffer
Block to SD card*/
{
unsigned char response;
unsigned int i, retry=0;
unsigned char CMD24_WRITE_SINGLE_BLOCK[] = {24,0x00,0x00,0x00,0x00,0xFF};
/*Define WRITE SINGLE BLOCK command(CMD34) w/o valid CRC*/
CMD24_WRITE_SINGLE_BLOCK[1] = ((addr & 0xFF000000) >> 24);
/*Include address of the memory location on SD card in the command*/
CMD24_WRITE_SINGLE_BLOCK[2] = ((addr & 0x00FF0000) >> 16);
CMD24_WRITE_SINGLE_BLOCK[3] = ((addr & 0x0000FF00) >> 8);
CMD24_WRITE_SINGLE_BLOCK[4] = ((addr & 0x000000FF));
SD_CS = 0;                              /*Activate SD card chip select*/
response = SD_WriteCommand(CMD24_WRITE_SINGLE_BLOCK);
/*Send WRITE SINGLE BLOCK command(CMD34)to SD card*/

if(response != 0x00)                    /*check for SD status: 0x00 - OK (No flags set)*/
return 1;

SPIWrite(0xFF);
SPIWrite(0xFE);                         /*Send start block token 0xfe (0x11111110)*/

for(i=0; i<512; i++)                    /*Send 512 bytes of Data*/
```

```
SPIWrite(buf[i]);


SPIWrite(0xff);     //Transmit dummy CRC (16-bit), CRC is ignored here
SPIWrite(0xff);


i=0;
do                                          /*Data response and busy response from card*/
        {
        response = SPIRead();
        i++;
        }while((response & 0x1f) != 0x05);
i=0;
while(!SPIRead())                           /*wait for SD card to complete writing and get idle*/
        {
        i++;
        if(retry++ > 0xfffe)                /*If SD card is busy for very long time*/
                {
                SD_CS = 1;                  /*Deactivate SD card chip select*/
                return 1;
                }
        }
SD_CS = 1;                                  /*Deactivate SD card chip select*/
SPIWrite(0xff);                             /*Wait 8 clock cycles*/
SD_CS = 0;                                  /*Activate SD card chip select*/
//verify if card is still busy
while(!SPIRead())                           //wait for SD card to complete writing and get idle
```

```
        {
        i++;
         if(retry++ > 0xfffe)                    /*If SD card is busy for very long time*/
                {
                SD_CS = 1;                       /*Deactivate SD card chip select*/
                return 1;
                }
        }
SD_CS = 1;/*Deactivate SD card chip select*/
return 0;
}.
```

APPENDIX B

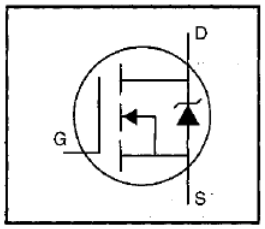
IRF510 MOSFET DATASHEET

International IOR Rectifier

IRF510

HEXFET® Power MOSFET

- Dynamic dv/dt Rating
- Repetitive Avalanche Rated
- 175°C Operating Temperature
- Fast Switching
- Ease of Paralleling
- Simple Drive Requirements



V_{DSS} = 100V

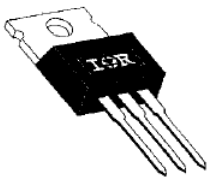
$R_{DS(on)}$ = 0.54Ω

I_D = 5.6A

Description

Third Generation HEXFETs from International Rectifier provide the designer with the best combination of fast switching, ruggedized device design, low on-resistance and cost-effectiveness.

The TO-220 package is universally preferred for all commercial-industrial applications at power dissipation levels to approximately 50 watts. The low thermal resistance and low package cost of the TO-220 contribute to its wide acceptance throughout the industry.



TO-220AB

DATA SHEETS

Absolute Maximum Ratings

	Parameter	Max.	Units
I_D @ T_C = 25°C	Continuous Drain Current, V_{GS} @ 10 V	5.6	A
I_D @ T_C = 100°C	Continuous Drain Current, V_{GS} @ 10 V	4.0	
I_{DM}	Pulsed Drain Current ①	20	
P_D @ T_C = 25°C	Power Dissipation	43	W
	Linear Derating Factor	0.29	W/°C
V_{GS}	Gate-to-Source Voltage	±20	V
E_{AS}	Single Pulse Avalanche Energy ②	100	mJ
I_{AR}	Avalanche Current ①	5.6	A
E_{AR}	Repetitive Avalanche Energy ①	4.3	mJ
dv/dt	Peak Diode Recovery dv/dt ③	5.5	V/ns
T_J T_{STG}	Operating Junction and Storage Temperature Range	-55 to +175	°C
	Soldering Temperature, for 10 seconds	300 (1.6mm from case)	
	Mounting Torque, 6-32 or M3 screw	10 lbf•in (1.1 N•m)	

Thermal Resistance

	Parameter	Min.	Typ.	Max.	Units
$R_{\theta JC}$	Junction-to-Case	—	—	3.5	°C/W
$R_{\theta CS}$	Case-to-Sink, Flat, Greased Surface	—	0.50	—	
$R_{\theta JA}$	Junction-to-Ambient	—	—	62	

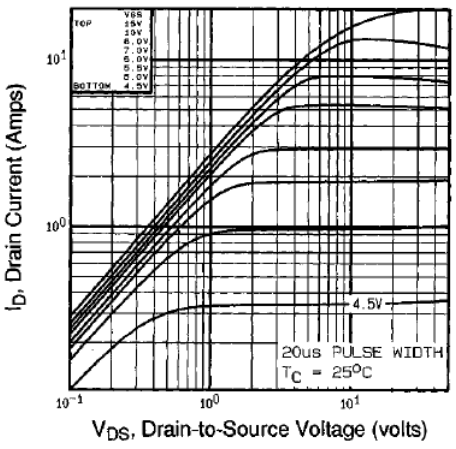


Fig 1. Typical Output Characteristics, T_C=25°C

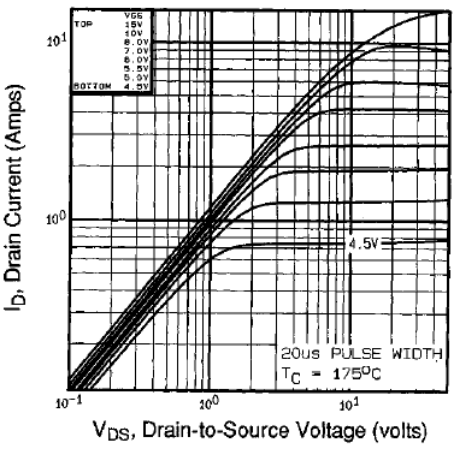


Fig 2. Typical Output Characteristics, T_C=175°C

REFERENCES

[1.1] T. Starner and D. Ashbrook, “Augmenting a pH medical study with wearable video for treatment of GERD,” IEEE Eighth International Symposium on Wearable Computers ISWC 2004, Vol.1, pp.194-195, 2004.

[1.2] Nihal Fatma Guler and Elif Derya Ubeyli, “Theory and Applications of Biotelemetry,” Journal of Medical Systems, Vol. 26, No. 2, April 2002.

[1.3] Y. Lee and P. Sorrells, Passive RFID basics, Application Note AN680, Microchip Technology Inc., 2001.

[1.4] M. Ghovanloo and K. Najafi, “A fully digital frequency shift keying demodulator chip for wireless biomedical implants,” IEEE Southwest Symposium on Mixed-Signal Design 2003, pp.223–227, 2003.

[1.5] Thermpon Ativanichayaphong, "Wireless Devices For Medical Applications", Phd Dissertion, The University Of Texas At Arlington, December 2007

[1.5] M. Ghovanloo and K. Najafi, “A fully digital frequency shift keying demodulator chip for wireless biomedical implants,” IEEE Southwest Symposium on Mixed-Signal Design 2003, pp.223–227, 2003.

[1.6] Thermpon Ativanichayaphong, Wen-Ding Huang, Jianqun Wang, Smitha M.N. Rao, H.F. Tibbals, Shou-Jiang Tang, Stuart Spechler, H. Stephanou and J.-C. Chiao, “An Implantable Wireless Impedance Sensor Capable of Distinguishing Air, Water and Acid in Gastroesophageal Reflux,” Digestive Disease Week 2007, Washington DC, May 19-24, 2007.

[1.7] Thermpon Ativanichayaphong, Wen-Ding Huang, Jianqun Wang, Smitha M.N. Rao, H.F. Tibbals, Shou-Jiang Tang, Stuart Spechler, H. Stephanou and J.-C. Chiao, “A Wireless Sensor for Detecting Gastroesophageal Reflux,” SPIE International Smart Materials, Nano- & Micro-

Smart Systems Symposium, Biomedical Applications of Micro- and Nanoengineering Conference, Adelaide, Australia, Dec.10-13 2006.

[1.8] Thermpon Ativanichayaphong, Shou Jiang Tang, Jianqun Wang, Wen-Ding Huang, Harry F. Tibbals, Stuart J. Spechler, J.-C. Chiao, "An Implantable, Wireless and Batteryless Impedance Sensor Capsule for Detecting Acidic and Non-Acidic Reflux," Gastroenterology, Vol. 134, No. 4, pp. A-63, 2008.

[1.10] Y. Yao, S. Yin and F. Dai, "A novel low-power input-independent MOS AC/DC charge pump," IEEE International Symposium on Circuits and Systems, Vol.1, pp.380-383, 2005.

[1.10] Sample, A.P.; Yeager, D.J.; Powledge, P.S.; Smith, J.R.;"Design of a Passively-Powered, Programmable Sensing Platform for UHF RFID Systems," IEEE International Conference on 26-28 March 2007, Page(s):149 - 156.

[1.11] Wen-Ding Huang, Jianqun Wang, Thermpon Ativanichayaphong, Lun-Chen Hsu, Sanchali Deb, Mu Chiao and J.C. Chiao,"Investigation of Repeatability of Sol-Gel Iridium Oxide pH Sensor on Flexible Substrate," SPIE Proceedings Vol. 7269, Smart Materials, Nano+Micro-Smart Systems Symposium, Micro- and Nanotechnology: Materials, Processes, Packaging, and Systems Conference, Melbourne, Australia, Dec. 9–12, 2008.

[1.12] E. Haile and J. Lepkowski, Oscillator Circuits for RTD Temperature Sensors, Application note AN895, Microchip Technology Inc., 2004.

[1.13] Lun-Chen Hsu, Wen-Ding Huang, Hung Cao, Sanchali Deb, J-C. Chiao, "Integrated Dual Sensors in a Batteryless Wireless Capsule", BMES Biomedical Engineering Society Annual Fall Scientific Meeting, Pittsburgh, PA, October 7-10, 2009.

[1.14] Wen-Ding Huang, Jiquan Wang, Thermpon Ativanichayaphong, Lun-Chen Hsu, Mu Chiao and J.-C. Chiao, "Progress Report on Flexible Metal-Oxide pH Sensors," BMES 2008, Biomedical Engineering Society Annual Meeting, St. Louis, Oct. 1–4 2008.

[1.15] K. Finkenzeller, RFID handbook: fundamentals and applications in contactless smart cards and identification, Chichester, England, New York: Wiley; 2003.

[2.1] Y. Lee, Antenna Circuit Design for RFID Applications, Application Note AN710, Microchip Technology Inc., 2001.

[2.2] N. O. Sokal and A. D. Sokal, “Class E—A New Class of High-Efficiency Tuned Single-Ended Switching Power Amplifiers,” IEEE Journal of Solid-State Circuits, Vol SC-10, No. 3, pp 168-176, June 1975.

[2.3] Tiaotiao Xie,"Design and Development of the Class E RF Power Amplifier Prototype by Using a Power MOSFET,"Technical Report, CReSIS TR 129.

[2.4] Nathan O. Sokal, “Class-E High-Efficiency Power Amplifiers, from HF to Microwave,” Proceedings of the IEEE International Microwave Symposium, June 1998, Baltimore

[2.5] Nathan O. Sokal, “Class- E Switching-Mode High-Efficiency Tuned RF Microwave Power Amplifier: Improved Design Equations, ”Proceedings of the IEEE International Microwave Symposium, June 2000, Boston.

[2.6] W. A. Davis and K. K. Agarwal, Text Book: Radio Frequency Circuit Design John Wiley, New York, 2001.

[2.7] W.Aerts, E.De Mulder,B.Preneel, Vandenbosch, G.; Verbauwhede, I.; Dependence of RFID Reader Antenna Design on Read Out Distance, IEEE Transactions On Antennas And Propagation, Vol. 56, No. 12, December 2008.

[2.8] microID® 13.56MHz RFID System Design Guide, Microchip Technology Inc.,2004.

[2.9] Gao Tongqiang, Zhang Chun, Chi Baoyong, and Wang Zhihua, “Design and analysis of a highly-integrated CMOS power amplifier for RFID readers,” Journal of Semiconductors, Vol. 30, No. 6, pp. 065008-1-5, June 2009.

[6.1] S. Castillo, N. K. Samala, K. Manwaring, B. Izadi and D. Radhakrishnan,” Experimental Analysis of Batteries under Continuous and Intermittent Operations,” Proceedings of the International Conference on Embedded Systems and Applications, pp. 18 – 24, June 2004.

BIOGRAPHICAL INFORMATION

Sandeep Battula was born in Andhra Pradesh, India. He received his Bachelors of Engineering in Electrical Engineering from Osmania University, Hyderabad, India in May 2007. He received his Master of Science in Electrical Engineering from The University of Texas at Arlington in August 2007. He worked as IEEE Mentor for Power Systems and Electronics at The University of Texas at Arlington from Sept. 2008 to Sept. 2009. His research interests include Circuit Designing for Micro-Medical Devices and Biotelemetry.