APPLICATION, ANALYSIS AND REALIZATION OF ELECTRONIC ARTICLE SURVEILLANCE (EAS) SYSTEMS

by

Mustafa Orhan Dirik

B.S., Electrical and Electronics Engineering, Boğaziçi University, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University
2012

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisors, Prof. Emin Anarım and Assoc. Prof. M. Kıvanç Mıhçak for their continuous support, guidance and encouragement they provided in all stages of this thesis. Without their assistance and suggestions, this thesis could not have been completed.

I would also like to thank to my dearest friend Gürkan Sönmez and to my family for their support and motivation during the period of this study.

ABSTRACT

APPLICATION, ANALYSIS AND REALIZATION OF ELECTRONIC ARTICLE SURVEILLANCE (EAS) SYSTEMS

The aim of this work is to design, implement and verify a low cost microcontroller based electronic article surveillance (EAS) system prototype and is accomplished in great success. An EAS system is basically used for preventing the shoplifters. The system basically consists of three parts, namely, the tag, the transceiver and the antenna. The tag is a series LC circuit that consists of a wire wound coil and a capacitor. The transceiver circuit is responsible for detecting the presence of the tag in the environment and finally the antenna is basically a large coil that is used for energizing the tag in the environment. Our study basically consists of two parts, namely the simulation results based on acquired data and the hardware implementation. The physical data is captured via a commercially available EAS system from Checkpoint Company. The transceiver circuit and the antenna is used to energize the tag in the environment. A separate loop antenna that is tuned for 8.2 MHz is used for capturing data. This loop antenna is called "sniffer" throughout the rest of this work. The captured data is saved via Tektronix DPO4104 oscilloscope as "csv" files. MATLAB is used for processing the captured data and for developing the algorithm. The hardware implementation part is the practical realization of our work. There are mainly five blocks used in a typical EAS system which are frequency synthesizer, amplifier, receiver, power unit and the processing unit. Each of them are carefully designed and tested as they will be explained in great detail within the relevant chapter.

ÖZET

ELEKTRONİK NESNE KONTROL (EAS) SİSTEMLERİ UYGULAMA, ANALİZ VE GERÇEKLENMESİ

Bu çalışmanın amacı mikro denetleyici tabanlı, düşük maliyetli bir EAS sisteminin tasarlanması, tasarımın uygulanması ve doğrulanmasıdır. Proje kapsamında hedeflenen sonuca başarı ile ulaşılmıştır. EAS sistem esasen hırsızlığın (shoplifters) engellenmesi amacıyla kullanılmaktadır. Sistem temel olarak etiket, alıcı-verici ve anten olmak üzere üç bileşenden oluşur. Etiket, bir kapasitör ve tel bobini içeren bir dizi LC devresinden oluşur. Alıcı-verici devre ortamda bir etiketin varlığının tespit edilmesinden sorumludur. Anten ise ortamdaki etikete enerji yüklenmesini sağlamak amacıyla kullanılan bir bobinden ibarettir. Bu çalışma elde edilen ölçüm değerlerinin simülasyonu ve donanım uygulaması olmak üzere 2 kısımdan oluşmaktadır. Ölçüm değerleri Checkpoint firmasının ticari bir EAS sistemi kullanılarak elde edilmiştir. Alıcı-verici devre ve anten, ortamdaki etiketlerin indüklenmesi amacıyla kullanılmış; ayrıca 8.2 Mhz frekansına ayarlanmış ayrı bir sarmal anten ölçüm değerlerinin toplanması için kullanılmıştır. Bu anten çalışmanın geri kalan kısımlarında "sezici" olarak isimlendirilmiştir. Elde edilen değerler Tektronix DPO4104 osiloskop ile "csv" dosyası olarak kaydedilmiş ve MATLAB ile algoritma geliştirmek için işlenmiştir. Donanım uygulama kısmı teorik çalışmanın pratik uygulaması şeklinde yapılmıştır. Temel olarak bir EAS devresinde yer alan frekans sentezleyici, güçlendirici, alıcı, güç ünitesi ve işlemci ünitesi dikkatli bir şekilde tasarlanmış ve test edilmiştir. Tasarımla ilgili detaylar ilerleyen kısımlarda detaylı bir şekilde anlatılmaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

C	Capacitor
L	Inductor
R	Resistor
Ω	Ohm

LIST OF ACRONYMS/ABBREVIATIONS

ADC	Analog-To-Digital Converter
COHO	Coherent Oscillator
CPU	Central Processing Unit
DC	Direct Current
DDS	Direct Digital Synthesizer
EAS	Electronic Article Surveillance
I	In-Phase
PSRR	Power Supply Rejection Ratio
Q	Quadrature
RF	Radio Frequency
SF	Step Frequency
SFS	Stepped Frequency Synthesizer
STALO	Stable Local Oscillator
SNR	Signal-To-Noise Ratio

1. INTRODUCTION

Electronic article surveillance (EAS) system is the name of technology, which is used to prevent unauthorized removal of products from shops, libraries or other kind of public exhibition sites. In these environments it is not preferred to keep products in locked cases to avoid shoplifting. Merchandises must be protected while customers are interacting with them. EAS systems offer a solution for this problem. With this technology, merchandises are protected by tags attached to them. If someone tries to go away with a product which has an unpacified tag on it, an alarm is triggered while going through the gate. Therefore, it provides security without disturbing product-costumer interaction.

System contains three main parts: the tag, which is attached on the book or product; the passivator, which is for authorized deactivation of the tag and the detector, which is for detection of any kind of shoplifting attempt. Detector is placed at the exit of hall, thus theft attempt can be precluded at the gate. The tag can be made of magnetic material or electronic resonant circuit. The passivator either removes the tag from product or deactivates it and lets it remain attached. The detector is actually a radar. It transmits signals and receives possible answers from tag. If the tag is between arms of the detector, it produces a signal which can be perceived by the detector and the alarm is triggered.

2. APPROACH AND METHODOLOGY

2.1. EAS Basics

As mentioned above, an EAS system consists of three basic parts; the tag, the passivator and the detector. The tag is attached on the product and activated. The detector is placed at the exit point and emits signals with defined frequencies from its transmitter antenna. These signals are received by the receiver and define the no-tag-in-between situation. If the tag is between the arms of the detector system, tag produces a unique signal as an answer to transmitter signal and it is captured by the receiver. By the microprocessor of the detector, it is decided if the received signal is coming from the tag or it is natural environmental noise. If tag signal is detected, then the alarm procedure is performed.

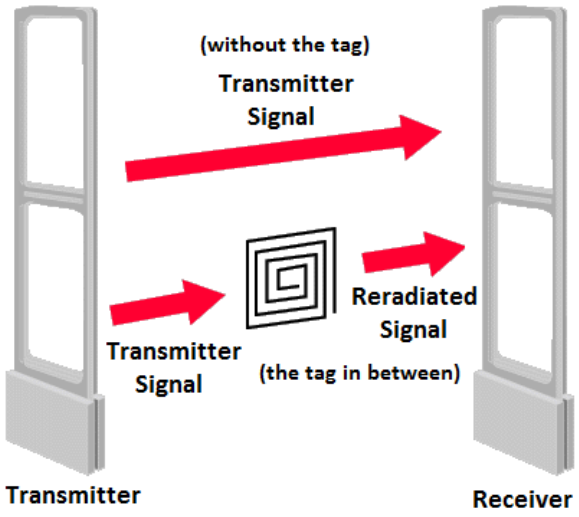

Figure 2.1. Schematic of the EAS system.

2.2. Tag System

There are various tag systems as magnetic stripes, microwave antennas or electronic resonant circuitries which are used in EAS systems. In this work, the tag is an RLC circuit whose resonance frequency is adjusted to the signal of the EAS transmit-

ter. Tag includes a helical antenna and an RC part to produce the answer signal. Tags for EAS systems are produced as thin papers, so they are easily being attached on products. They can be attached during the production process by the manufacturer or added by the seller separately. When the tag is exposed to transmitter's signals, it starts to resonate and emits signals at the same frequency. In the absence of transmitter signals, tag signal decays gradually, however it can be detected by the receiver. The deactivation process of the tag includes destroying the capacitor of the circuit which is achieved by applying high electromagnetic field.

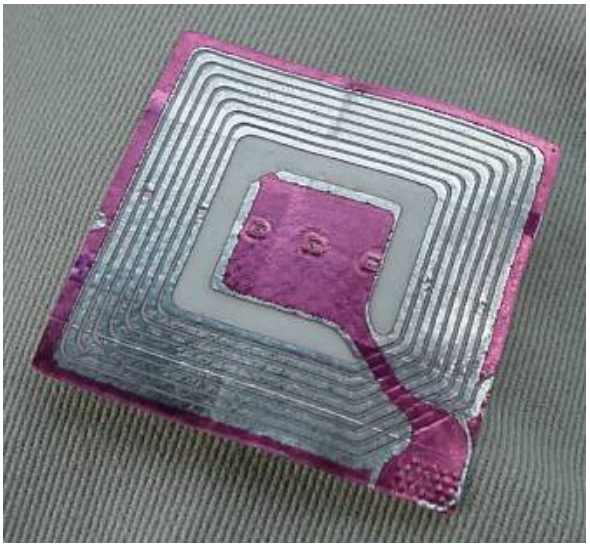

Figure 2.2. The tag with helical antenna.

2.3. Detector System

There are several ways of object detection using RF signal. The main idea is sending the RF signal and capturing the reflection of the signal that was sent. Based on the incoming signal, signal processing techniques are applied. Most of these techniques are based on their frequency analysis. There is limited work which is carried on time domain. On of the techniques for time domain analysis is done via counting the number of zero crossings and by observing the time passed between an upper and a lower threshold in the falling envelope [1].

Most of the work done for object detection is based on frequency analysis of the

incoming signal and hence spectrum analyzers are used for rapid frequency analysis. It is obviously not a practical and cost effective way of detection [2,3]. There are also some work done previously which models and analyzes the effects of the environmental conditions on the detection performance [4]. There is also relevant information about the effect of the type of environment like sand, soil, jungle, etc on the performance of the presence detection [5].

One of the most popular ways of object detection via RF signals is the use of step frequency radars. Step frequency radar technique is widely used for buried object detection [6, 7]. These systems are basically used for through wall object detection, buried object finding and for military identification purposes. The aims of these systems are basically localize the object of the body. For this reason there are lots of works for efficiently detecting the distance of the object to be found to the system [8,9]. There are lots of works that suggest methods like ARMA and MUSIC but obviously these methods strongly depend on the available computing power [10,11]. On the other hand, maximum likelihood [12, 13] and correlation methods are also popular. Correlation is widely used for detecting object as it was used in several previous works. Basically the decaying exponential is estimated and is correlated with the actual acquired signal envelope. The presence of the object is decided based on the correlation output of those two [14].

Step frequency radar technique is also used for moving object detection widely [15–20]. On the other hand there are also some works in order to find the number of targets within range [21, 22] and there are algorithms for detecting the velocity of the target [23]. The studies are mostly based on the pointing the target that is to be found via step frequency radar technique or by some other radar techniques. Since the main objective is detecting the exact location of the object of the target, 2-D imaging or signal capturing would result in better results than 1-D systems provide. However, this method basically increases the computing power needed and hence the cost of the system [24, 25].

Optimizing the step size of the detection system is beyond our study. There are

methods which can be used for optimizing the increment in the frequency of the pulse that is applied in each step [26, 27].

Detector system includes a radar which has transmitter and receiver parts. Many radar systems are utilized in EAS systems, however in this work the step frequency (SF) radar is used. SF radar systems use discrete frequency steps in one sweep. In each step, frequency of the signal is increased linearly. The reason for transmitting in changing frequencies is production-based deviations in tags. Every tag has some uncertainty in their resonance frequency values; therefore frequencies in this range must be swept to be able to catch these tags. The resolution gets better when increments between successive steps are smaller.

In SF radar, instantaneous bandwidth is narrow; hence it requires a less complicated hardware, such as lower speed analog to digital converters or slower processors. Moreover, bandwidth at the receiver side is also smaller, which provides a better signal-to-noise ratio (SNR) value.

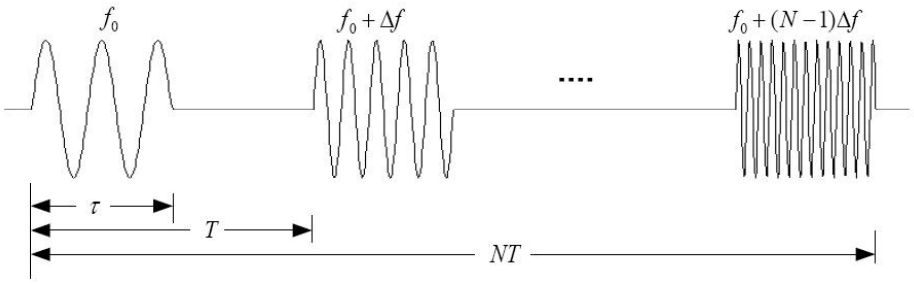

Figure 2.3. SF radar signal. τ is the transmit time and T is the period.

Block diagram of the SF radar is shown in Figure 2.4. In transmitter, baseband signal is produced by coherent oscillator (COHO). Increment amount is calculated and produced by the stepped frequency synthesizer (SFS) and added to COHO signal. After that, this intermediate signal is upconverted for transmitting purposes by stable local oscillator (STALO). Then it is amplified by power amplifier and transmitted via transmitter antenna. Received signal is first downconverted by STALO and increment

amount for transmitted step is subtracted. Then, resultant signal is mixed with COHO signal and its 90° phase shifted version to obtain in-phase (I) and quadrature (Q) outputs. These values are used by the microprocessor to determine if the received signal is tag-oriented. If tag signal is detected, the alarm is triggered.

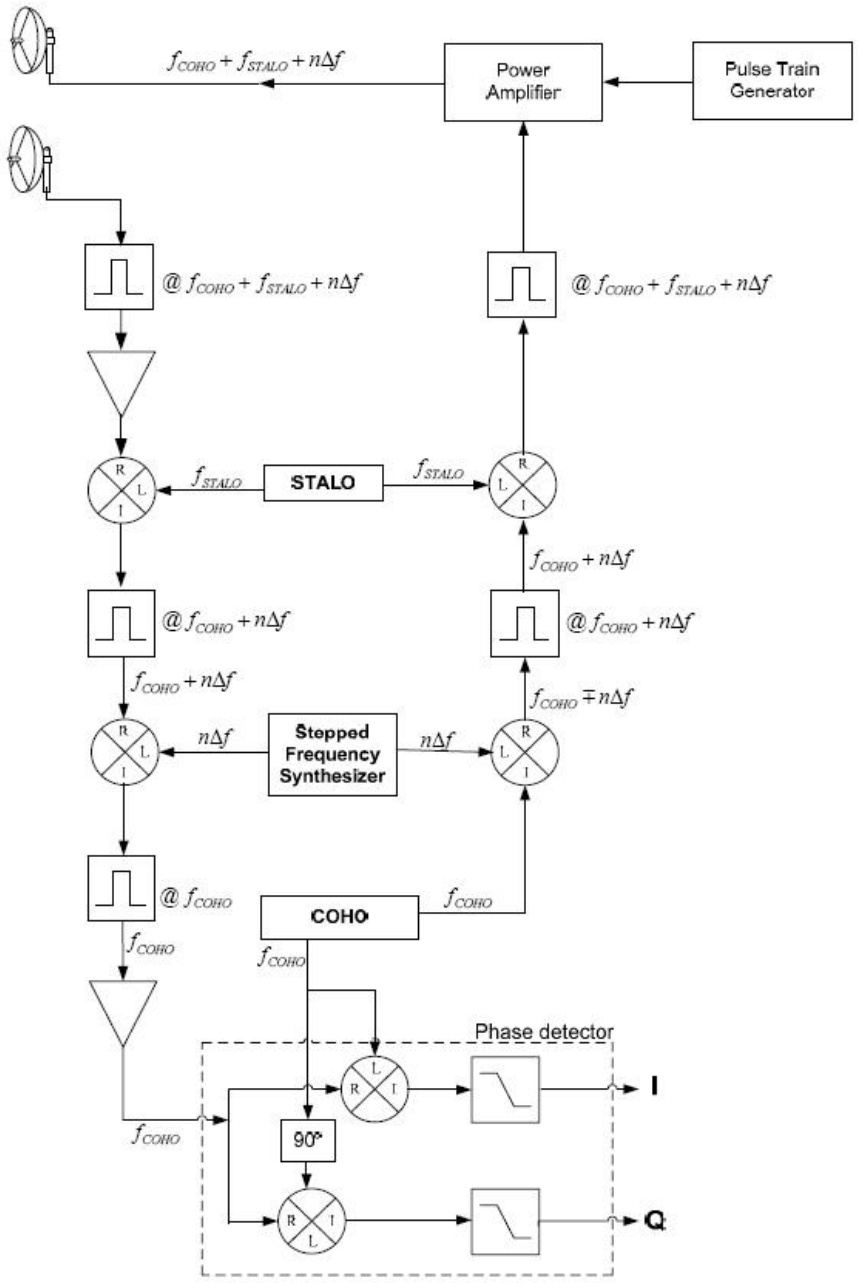

Figure 2.4. Block diagram of SF radar system.

Suppose the transmitted signal is [28]:

$$\begin{aligned} s_T(t) &= A_1 cos(2\pi f_n t) \\ &= A_1 cos[2\pi(f_{COHO} + f_{STALO} + n\Delta f)t], \end{aligned}$$

and the time between Rx and Tx is $2R/c$

$$\begin{aligned} s_R(t) &= A_n cos[2\pi(f_{COHO} + f_{STALO} + n\Delta f)(t - 2R/c)] \\ &= A_n cos[2\pi(f_{COHO}t + f_{STALO}t + n\Delta f t) - 2\pi(f_{COHO} + f_{STALO} + n\Delta f)(2R/c)]. \end{aligned}$$

The output signal of the mixer with $s_{STALO}(t) = 2A_s cos(2\pi f_{STALO}t)$ is

$$\begin{aligned} s_R(t)s_{STALO}(t) &= 2A_S A_n cos[2\pi(f_{COHO} + f_{STALO} + n\Delta f)(t - 2R/c)]cos(2\pi f_{<STALO}t) \\ &= A_S A_n cos[2\pi f_{COHO}t + 2\pi n\Delta f t - 2\pi(f_{COHO} + f_{STALO} + n\Delta f)(2R/c)] \\ &\quad + A_S A_n cos[2\pi(f_{COHO}t + 4\pi f_{STALO}t + 2\pi n\Delta f t) \\ &\quad - 2\pi(f_{COHO} + f_{STALO} + n\Delta f)(2R/c)]. \end{aligned}$$

After applying bandpass filter to the IF signal we have:

$$s_{IF1}(t) = A_S A_n cos[2\pi f_{COHO}t + 2\pi n\Delta f t - 2\pi(f_{COHO} + f_{STALO} + n\Delta f)(2R/c)].$$

To get the baseband signal output of the first mixer is mixed with the SFS signal:

$$\begin{aligned} s_{IF1}(t)s_{SFS}(t) &= 2A_{SFS}A_S A_n cos[2\pi f_{COHO}t + 2\pi n\Delta f t - 2\pi(f_0 + n\Delta f)(2R/c)]cos(2\pi n\Delta f t) \\ &= A_{SFS}A_S A_n cos[2\pi f_{COHO}t - 2\pi(f_0 + n\Delta f)(2R/c)] \\ &\quad + A_{SFS}A_S A_n cos[2\pi f_{COHO}t + 4\pi n\Delta f t - 2\pi(f_0 + n\Delta f)(2R/c)]. \end{aligned}$$

Again a bandpass filter is used for getting rid of the high frequency component:

$$s_{IF2}(t) = A_{SFS}A_S A_n cos[2\pi f_{COHO}t - 2\pi(f_0 + n\Delta f)(2R/c)].$$

To get the I and Q components of the baseband signal, the output of the second mixer is mixed by two signals which have 90 degrees phase difference:

$$\begin{aligned} s_{IF2}(t)s_{COHO,I}(t) &= 2\pi A_{SFS}A_S A_n A_C cos[2\pi f_{COHO}t - 2\pi(f_0 + n\Delta f)(2R/c)]cos(2\pi f_{COHO}t) \\ &= A_{SFS}A_S A_n A_C cos[-2\pi(f_0 + n\Delta f)(2R/c)] \\ &\quad + A_{SFS}A_S A_n A_C cos\{4\pi f_{COHO}t - [2\pi(f_0 + n\Delta f)(2R/c)]\}, \end{aligned}$$

$$\begin{aligned} s_{IF2}(t)s_{COHO,I}(t) &= 2\pi A_{SFS}A_S A_n A_C cos[2\pi f_{COHO}t - 2\pi(f_0 + n\Delta f)(2R/c)]sin(2\pi f_{COHO}t) \\ &= A_{SFS}A_S A_n A_C sin[-2\pi(f_0 + n\Delta f)(2R/c)] \\ &\quad + A_{SFS}A_S A_n A_C sin\{4\pi f_{COHO}t - [2\pi(f_0 + n\Delta f)(2R/c)]\}. \end{aligned}$$

After passing the output from a low pass signal, we have:

$$\begin{aligned} I &= A_{SFS}A_S A_n A_C cos[-2\pi(f_0 + n\Delta f)(2R/c)] \\ &= Acos[-2\pi(f_0 + n\Delta f)(2R/c)], \end{aligned}$$

and

$$\begin{aligned} Q &= A_{SFS}A_S A_n A_C sin[-2\pi(f_0 + n\Delta f)(2R/c)] \\ &= Asin[-2\pi(f_0 + n\Delta f)(2R/c)]. \end{aligned}$$

3. SIMULATION RESULTS

3.1. Data

Before developing the algorithm, we acquired data from a commercially available EAS system from Checkpoint Company. The model of the device that we used for data acquisition purposes is TR4024. The system uses one antenna both for transmission and reception. An internal RF switch is used for switching the RF signal to and from the antenna. We tied an EAS tag to a loop antenna that is tuned for 8.2 MHz and captured the signal via DPO4104. The loop antenna that is tied to the EAS tag will be called as the "sniffer" through this chapter.

DPO4104 is a high speed, four channels digital oscilloscope from Tektronix that can acquire data at 5 GS/s. the sniffer output is captured as and saved as a csv file to a flash memory via oscilloscope. The csv files that are recorded are processed with MATLAB and the results are discussed.

The algorithm first reads the csv file and creates the signal in workspace. The plot of the captured signal from the sniffer is given in Figure 3.1.

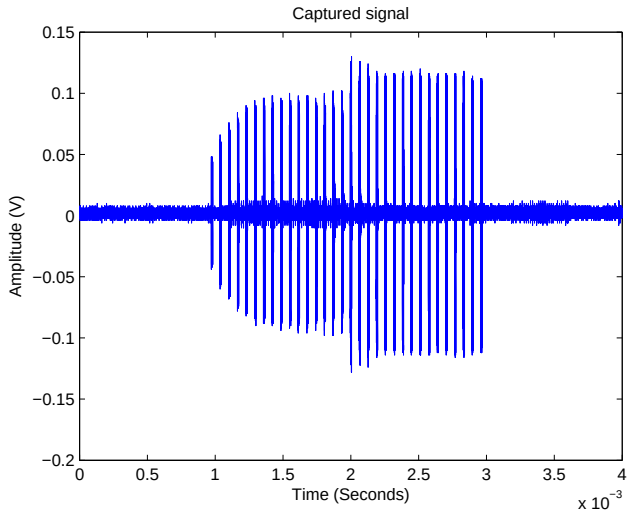

Figure 3.1. Captured signal when sniffer is at 40 cm and tag is at 30 cm.

There are 32 pulses with different frequencies in the plot above. Each pulse lasts for 10 μs and there is a 54 μs silence period between the pulses. The frequencies of the pulses change from pulse to pulse in order to sweep the frequency band that an EAS tag may resonate. The above signal is captured with the sniffer with a distance 40 cm from the antenna and the distance between the EAS tag and the antenna is 30 cm.

The pulse that creates resonance on the EAS tag is given in Figure 3.2.

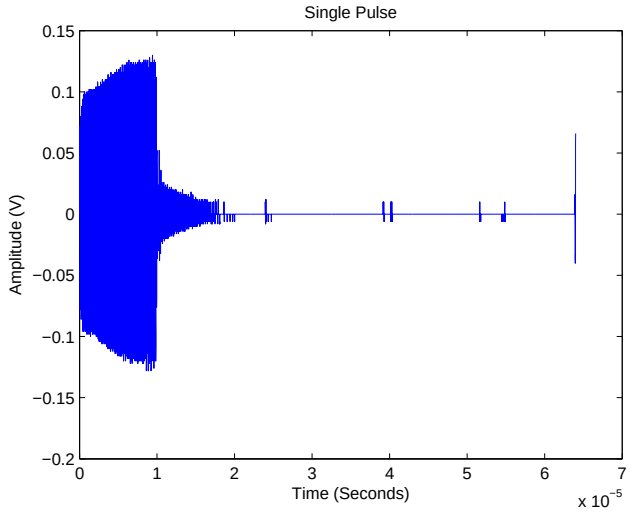

Figure 3.2. Single pulse when sniffer is at 40 cm and tag is at 30 cm.

As seen in Figure 3.2, the transmit signal lasts for 10 μs. After transmission the transmission stops and the antenna is switched to receive mode. Since the signal above is captured with the sniffer the tag response which is a damping sinusoidal waveform that starts after the 10^{th} microsecond can also be observed.

The algorithm then carries out I/Q demodulation on the signal. Cosine and sine waveforms are created, multiplied by the signal and $\sqrt{I^2 + Q^2}$ is calculated. After applying a low pass filter in the frequency domain the envelope of the signal is acquired and is given in Figure 3.3.

As seen in Figure 3.3, the transmission stops at 10 μs and the tag response is observed afterwards.

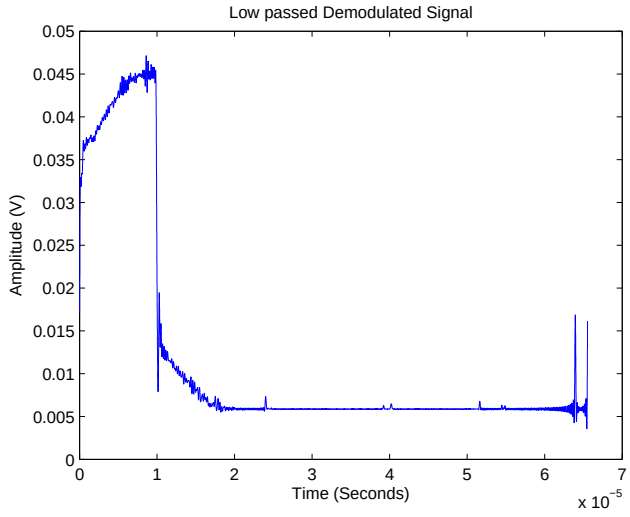

Figure 3.3. Demodulated signal when sniffer is at 40 cm and tag is at 30 cm.

The algorithm detects the start of the tag response and measures the time of the response. It is done by comparing the sum of the energy level with an empirically detected threshold. When a zero value is detected the summation of the energy levels is started and continued for a predetermined amount of time. If the total energy level is smaller than the threshold it is recorded as the end of tag response. The resulting waveform is given in Figure 3.4.

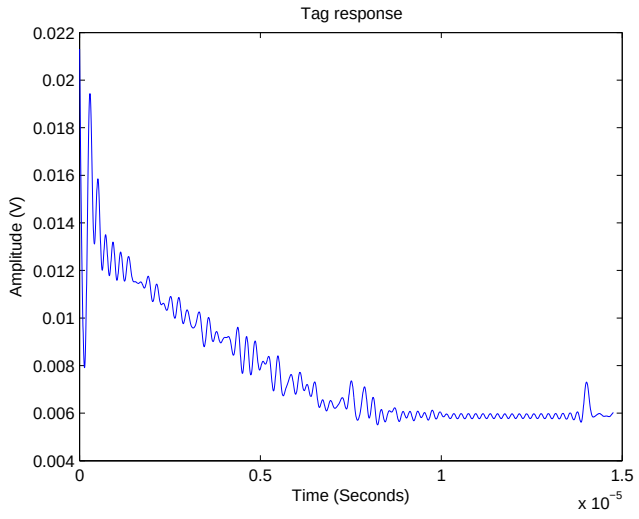

Figure 3.4. Tag response when sniffer is at 40 cm and tag is at 30 cm.

As it is seen in Figure 3.4, tag response lasts for about 8 μs. The duration of the

tag response is compared with a predetermined value in order to detect the presence of an EAS tag in the environment.

The following four plots are acquired by running the algorithm on the signal which is captured from 30 cm apart from antenna and there is no EAS tag in the environment.

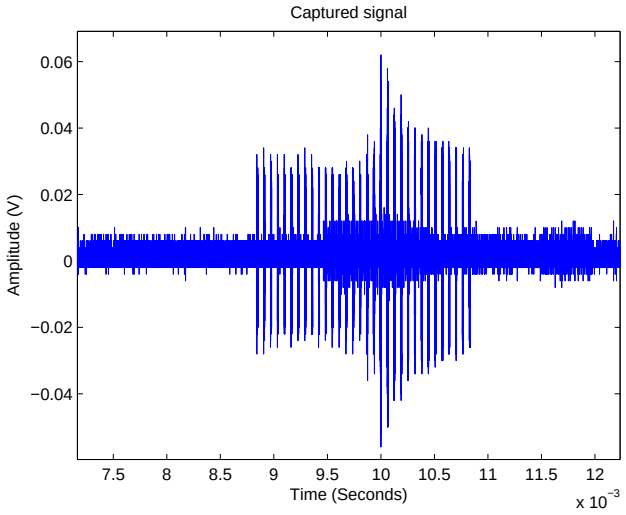

Figure 3.5. Captured signal when sniffer is at 30 cm and there is no tag.

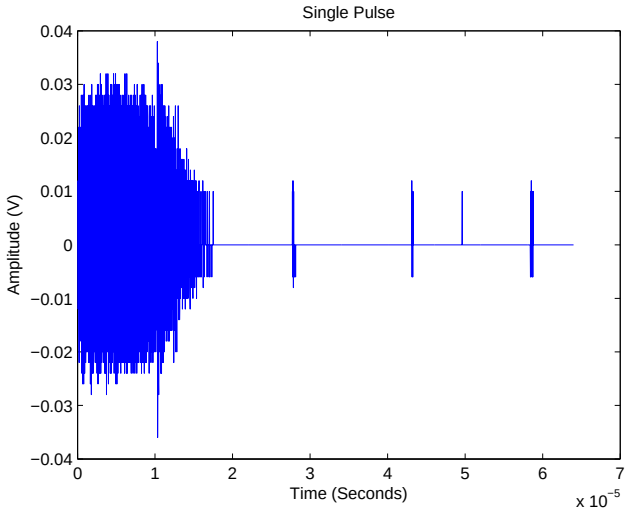

Figure 3.6. Single pulse when sniffer is at 30 cm and there is no tag.

The following four plots are acquired by running the algorithm on the signal which is captured from 30 cm apart from antenna and the EAS tag is 25 cm apart

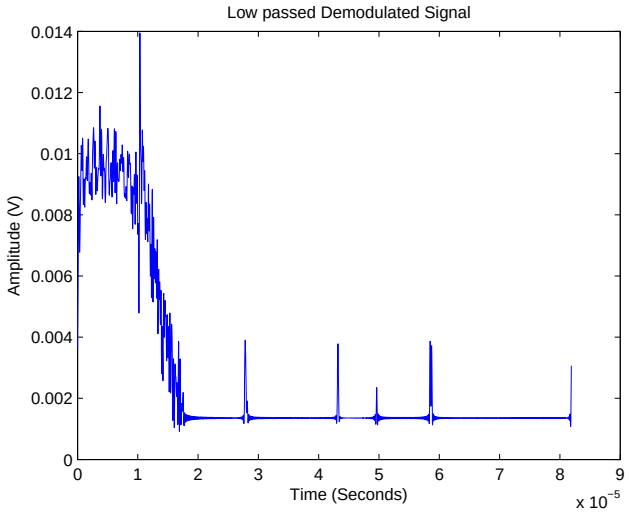

Figure 3.7. Demodulated signal when sniffer is at 30 cm and there is no tag.

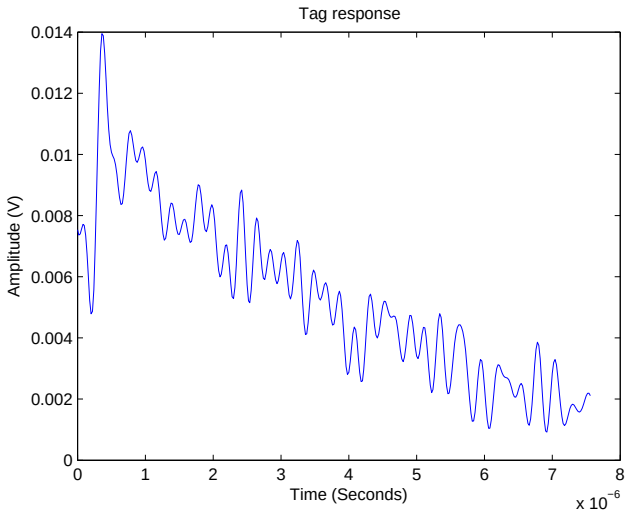

Figure 3.8. Tag response when sniffer is at 30 cm and there is no tag.

from the antenna.

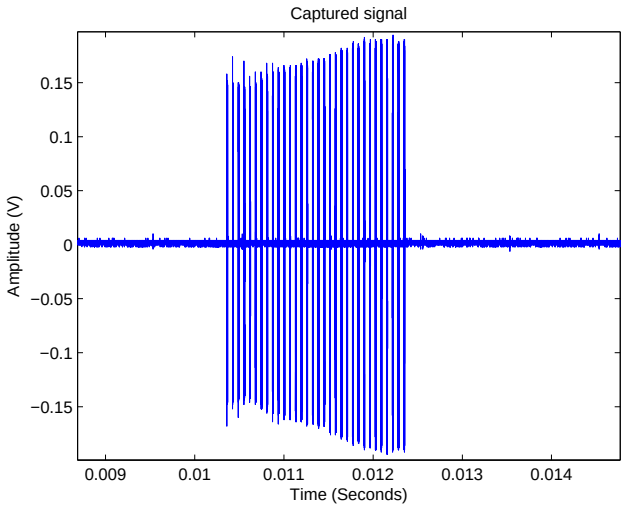

Figure 3.9. Captured signal when sniffer is at 30 cm and tag is at 25 cm.

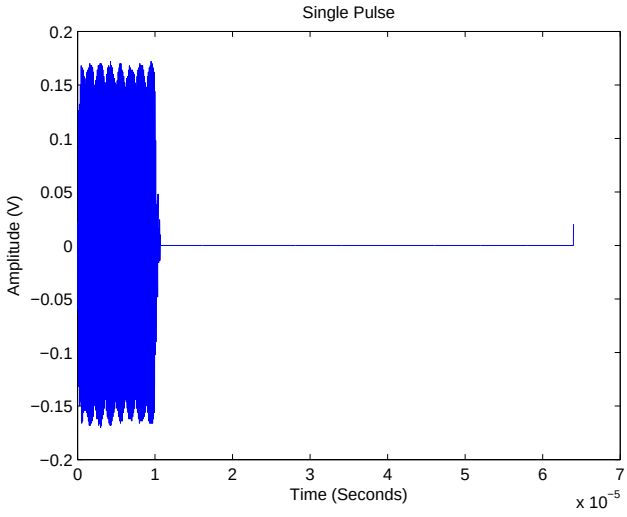

Figure 3.10. Single pulse when sniffer is at 30 cm and tag is at 25 cm.

The following four plots are acquired by running the algorithm on the signal which is captured from 50 cm apart from antenna and there is no EAS tag in the environment.

The following four plots are acquired by running the algorithm on the signal which is captured from 50 cm apart from antenna and the EAS tag is 45 cm apart from the antenna.

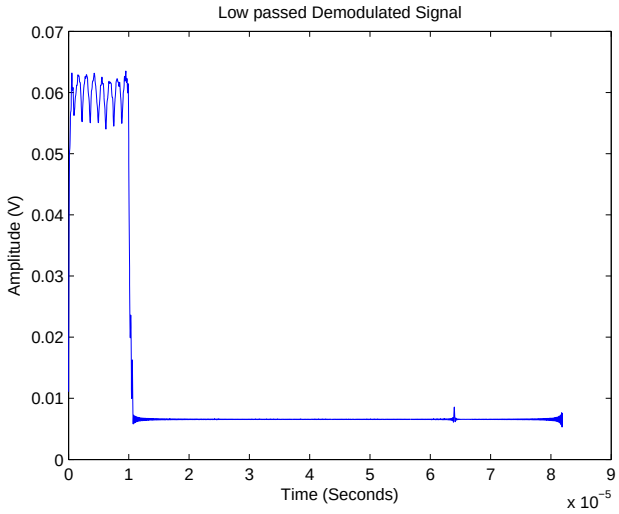

Figure 3.11. Demodulated signal when sniffer is at 30 cm and tag is at 25 cm.

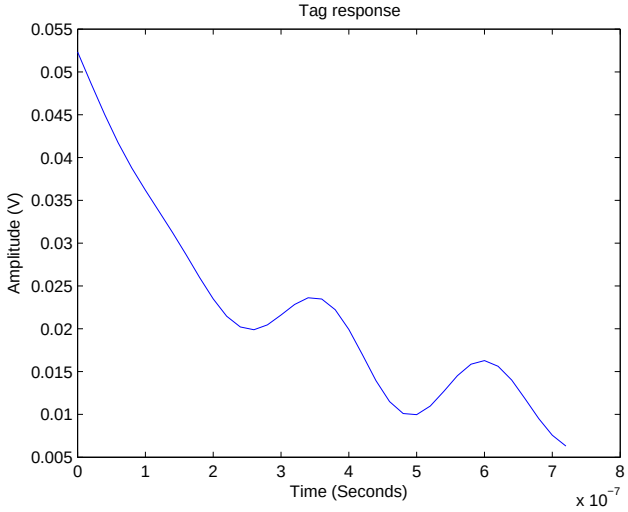

Figure 3.12. Tag response when sniffer is at 30 cm and tag is at 25 cm.

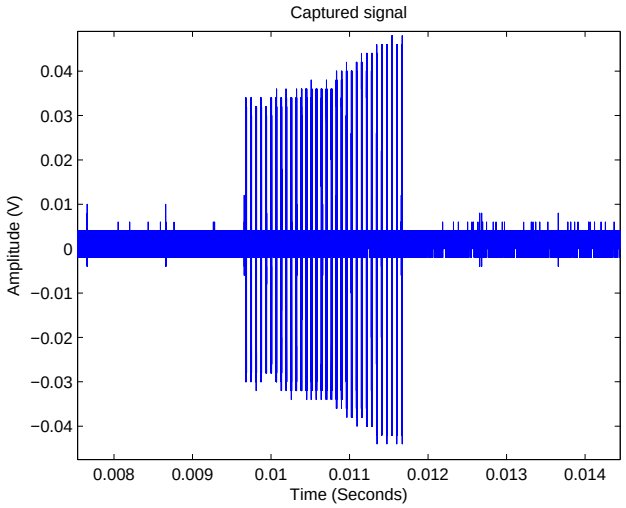

Figure 3.13. Captured signal when sniffer is at 50 cm and there is no tag.

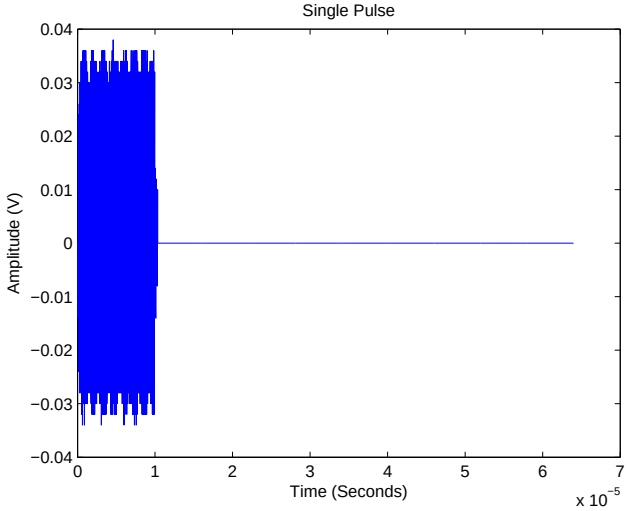

Figure 3.14. Single pulse when sniffer is at 50 cm and there is no tag.

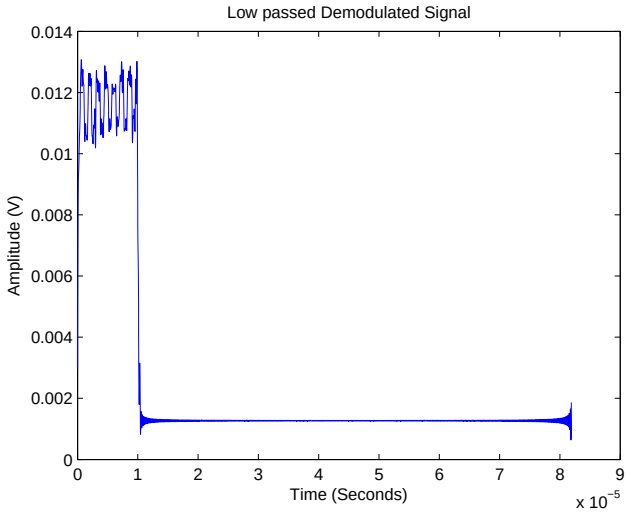

Figure 3.15. Demodulated signal when sniffer is at 50 cm and there is no tag.

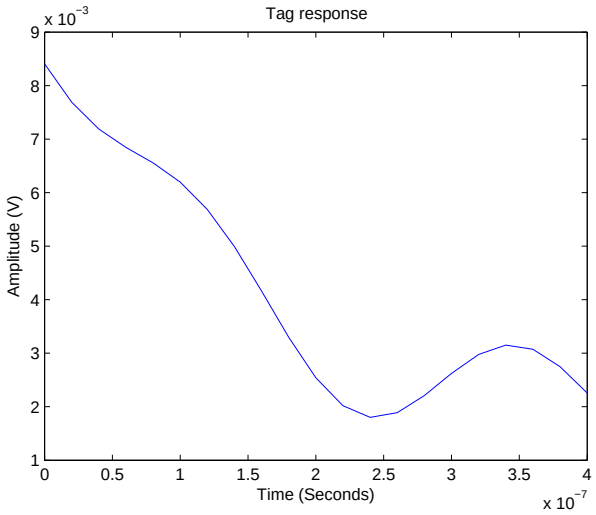

Figure 3.16. Tag response when sniffer is at 50 cm and there is no tag.

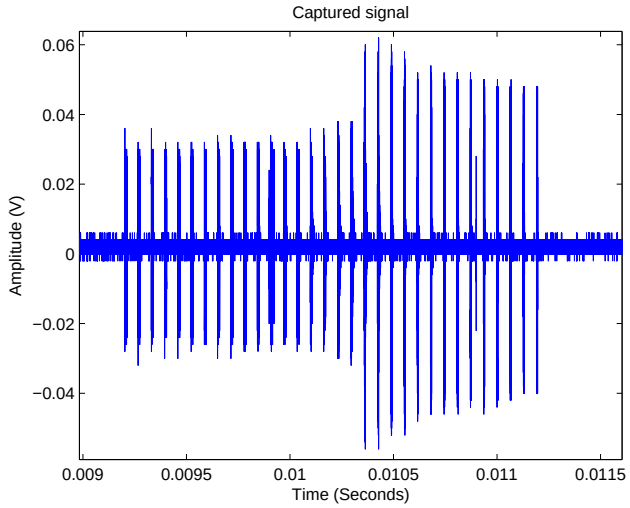

Figure 3.17. Captured signal when sniffer is at 50 cm and tag is at 45 cm.

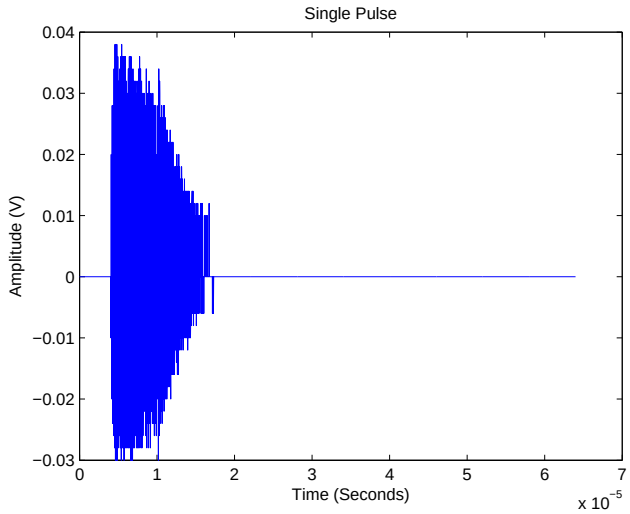

Figure 3.18. Single pulse when sniffer is at 50 cm and tag is at 45 cm.

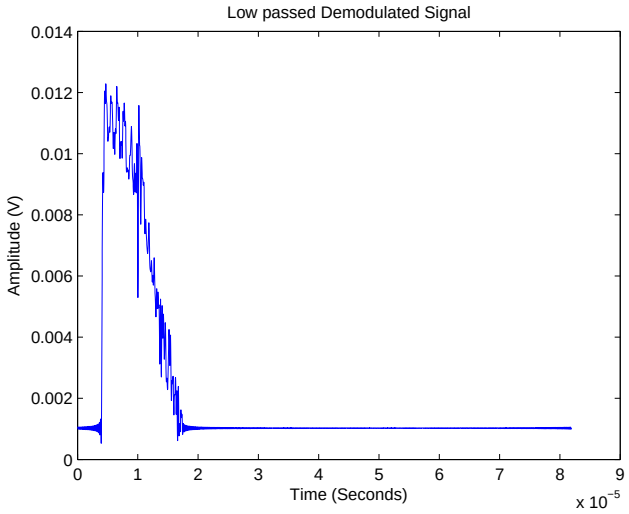

Figure 3.19. Demodulated signal when sniffer is at 50 cm and tag is at 45 cm.

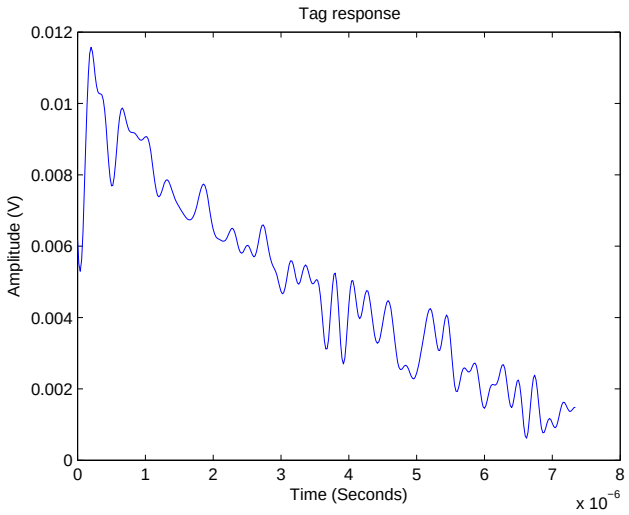

Figure 3.20. Tag response when sniffer is at 50 cm and tag is at 45 cm.

In Figure 3.21, there is a comparison of the effect of the presence of the tag in the environment while all other parameters are the same.

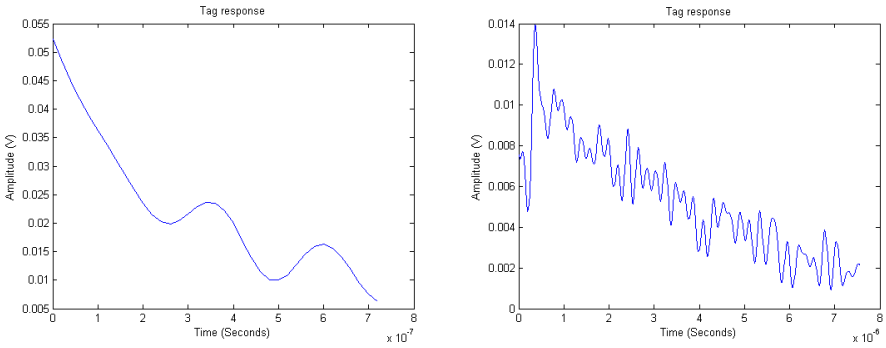

Figure 3.21. Tag response when sniffer is at 30 cm and there is no tag (left) and tag is at 25 cm (right).

As seen in Figure 3.21, the signals that are captured after stopping the transmission are totally different. We obverse an after-transmission signal which lasts for 0.7 μs when there is no EAS tag in the environment where the duration of the signal after stopping the transmission is 10 times longer, i.e. 7 μs when there is an EAS tag in the environment.

Similar situation is observed when we increased the distance from 30 cm to 50 cm.

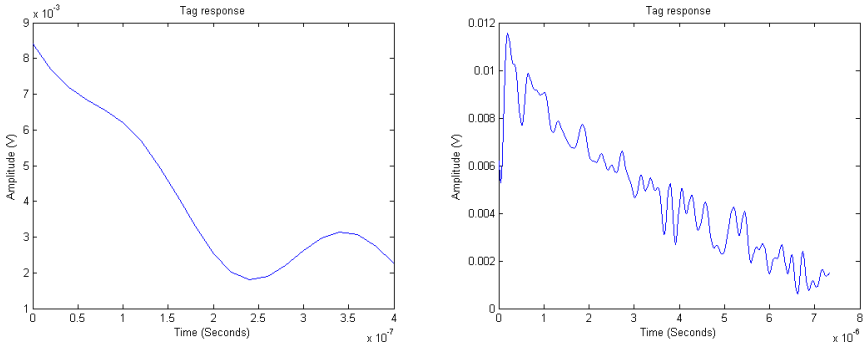

Figure 3.22. Tag response when sniffer is at 50 cm and there is no tag (left) and tag is at 45 cm (right).

As seen in Figure 3.22, the signals that are captured after stopping the transmission are totally different. We obverse an after-transmission signal which lasts for 0.4 μs when there is no EAS tag in the environment where the duration of the signal after stopping the transmission is almost 20 times longer, i.e. 7.5 μs when there is an EAS tag in the environment.

3.2. Performance Metrics

The captured data is deeply analyzed in MATLAB and optimum threshold for energy level detection is calculated based on false rejection and false acceptance curves for different setups.

First of all, the sniffer data for a specific distance is captured. This data is examined in terms of energy levels and corresponding histograms are created. After creating the relevant histograms, a Gaussian distribution is fit on the histogram via using MATLAB.

Another set of data for the same distance is also captured when there is no tag in the environment. The same procedure is also carried on and the corresponding histograms are calculated. The probability distribution for no tag condition is estimated via histograms.

After calculating the histograms for no tag response and tag response, false reject and false acceptance curves are created. The threshold is decided as the point where false reject and false acceptance probabilities are equal to each other for the maximum distance.

False reject and false acceptance curves are calculated using the $erfc()$ function of MATLAB. The $erfc(x)$ function is defined as follows:

$$erfc(x) = \frac{2}{\sqrt{\pi}} \int_{x}^{\infty} e^{-t^2} dt. \tag{3.1}$$

Since we have a non-normal distribution, the argument at the $erfc()$ function needs to be modified:

$$p = \int_a^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx. \tag{3.2}$$

Let $\frac{x-\mu}{\sqrt{2}\sigma} = t$. Then $\frac{1}{\sqrt{2}\sigma} dx = dt$ and for $x = a$, $t = \frac{a-\mu}{\sqrt{2}\sigma}$. Therefore, for $x = \infty$ and $t = \infty$;

$$\begin{aligned} p &= \int_{\frac{a-\mu}{\sqrt{2}\sigma}}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-t^2} \sqrt{2}\sigma dt \\ &= \frac{1}{\sqrt{\pi}} \int_{\frac{a-\mu}{\sqrt{2}\sigma}}^{\infty} e^{-t^2} dt, \end{aligned}$$

where, by definition

$$p = \frac{1}{2} erfc\left(\frac{a-\mu}{\sqrt{2}\sigma}\right). \tag{3.3}$$

The FA&FR curves are calculated based on the data which is captured from 50 cm apart from the antenna and are shown in Figure 3.23 and Figure 3.24.

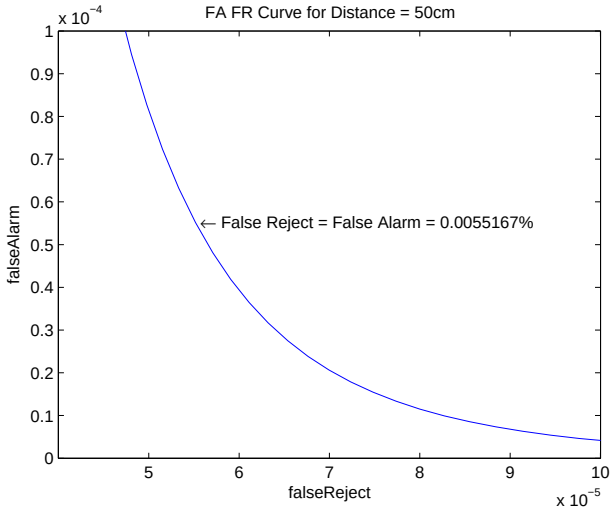

Figure 3.23. False reject vs. false acceptance curve for 50 cm.

In Figure 3.23, the curve is created for probability of miss equals to probability of false alarm. The resulting threshold for detecting the existence of the tag is shown in Figure 3.24.

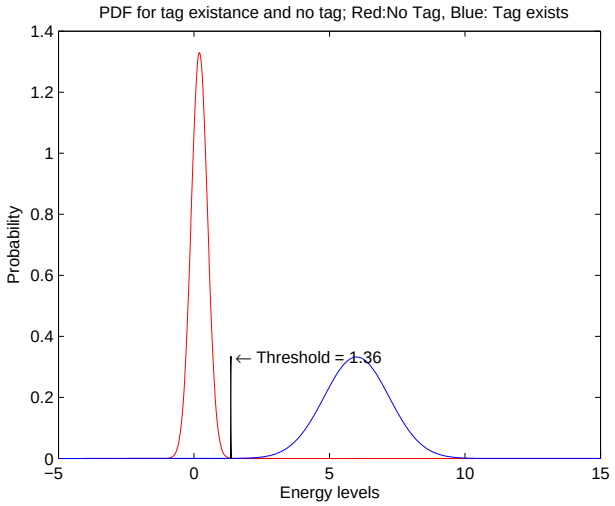

Figure 3.24. Probability distribution at 50 cm.

3.3. Conclusion

Based on the simulations that are carried on the acquired real system data, it is obvious that we can use the portion of the captured data after a 1 μs delay when transmission is over. In other words, waiting for about 1 μs after the transmission is completed and then starting to capture data from the environment will definitely let us detect the presence of a tag in the environment. It is clear that when there is no EAS tags in the environment the residual magnetic field do not persist longer than 0.8 μs.

4. HARDWARE IMPLEMENTATION

4.1. Overview

A typical EAS system consists of an EAS tag, an EAS transceiver circuit and one or more antennas. The system basically works by first applying a magnetic field at a known frequency in order to induce some potential energy on the EAS tag. We can call this phase as "transmission". After inducing some potential energy on EAS tag, the power transfer is turned off and the energy stored on the EAS tag starts to decrease due to emission. The system captures the emission that is coming from the EAS tag and processes in order to detect the presence of the EAS tag in the environment. We may call this phase as "reception".

Some of the EAS systems use only one antenna for both transmission and reception, where some of the systems use two antennas, one for transmission and the other for reception. In this work, we implemented a system that is having two different antennas, one is used for transmission and the other one is used for reception.

4.1.1. EAS Tag

The tag which is detected by an EAS system is basically a passive circuit that is composed of an inductor and a capacitor. Since it creates a LC circuit, it has a resonant frequency of $1/\sqrt{LC}$. External magnetic field that has a frequency of the resonant frequency of the tag will result in inducing potential difference between the terminals of the inductor.

In order to reduce the production costs of the EAS tags, the components that are used during the manufacturing process are chosen to be quite cheap. In other words, the capacitance and the inductance values of the capacitors and the inductors may greatly vary from the target value. This fact corresponds to different resonance frequencies of the manufactured EAS tags. In order to detect the EAS tags which have

different resonance frequencies, a typical EAS system needs to sweep the frequency range that covers all possible resonance frequencies of the manufactured tags.

4.1.2. EAS Circuit

A typical EAS circuit contains at least one block for frequency synthesis, one block for amplification, one block for reception, one block for power management and one block for processing.

The frequency synthesizer block is mainly used for generating sinusoidal waveforms that are used for inducing potential energy of the tag circuit. As stated previously, the resonance frequency of an EAS tag may vary from tag to tag. In order to detect the presence of any EAS tag in the environment, we need a frequency synthesizer that is capable of generating sinusoidal waveform at different frequencies. The state of art technology for creating sinusoidal waveforms at different frequencies is the Direct Digital Synthesizer (DDS). Direct Digital Synthesizer is a kind of frequency synthesizer used for realizing local oscillators, waveforms generators, mixers and modulators.

The amplifier block is needed to amplify the signal that is generated by the frequency synthesizer block. Power amplifiers are used for this purpose. The amplified signal is then filtered and transferred to the antenna. Impedance matching is very important since a mismatch will result in loss of output power and degrade the performance of the systems. A mismatch will basically reduce the detection range of the EAS system. In our work we used surface mount on board pre-amplifiers and external power amplifiers in order to boost the signal. Impedance matching is done via Smith chart.

The receiver block is used for capturing the signal that is emitted from the EAS tag during discharge. The EAS tag will emit electromagnetic waves at its resonant frequency. The signal is captured by the receiver antenna and some numeric value is generated from the received signal. This numeric value may be the duration, the received signal strength or some other value that is used to recognize the tag response.

In our work, we used a logarithmic amplifier in order the measure the power level of the incoming signal which the tag response itself.

The power management block is basically used for generating the supply voltage of the block in the circuit. Since this is an RF system one of the most important problems of the system is the noise. Low noise voltage generators, DC-DC converters or linear regulators are needed in order to reduce the amount of noise that is generated within the circuit. In our work, we used low noise and high power supply rejection ratio (PSRR) linear regulators for generating the supply voltages.

Processing block is used for controlling the frequency synthesizer and also for processing the data that is coming from the receiver block. It runs the decision algorithm in order to detect the presence of the EAS tag in the environment. Several technologies including microcontrollers, microprocessors, digital signal processors and field programmable gate arrays can be used for the processing block. In our work, we used a microcontroller that has a 16-bit RISC CPU.

The overall schematics of our work is given in Figure A.1.

4.2. Description Of The Implementation

4.2.1. Frequency Synthesis

Frequency synthesis block is mainly used for generating sinusoidal waveforms that are going to be amplified and then sent to the transmitter antenna. In our work, we used AD9834 Direct Digital Synthesizer IC from Analog Devices. AD9834 is a low cost, low power DDS IC that is used for generating sinusoidal, triangular and square waveforms at the desired frequency. When using with a 75 MHz clock generator it can reach a resolution of 0.28 Hz. When using AD9834 with a 1 MHz clock source it can generate waveforms at the desired frequency with a resolution of 0.004 Hz.

The control of AD9834 is carried out via serial transmission of the configuration

words from the processing block. The internal registers are updated via 3-pin serial port. The functional block diagram is given in Figure 4.1 .

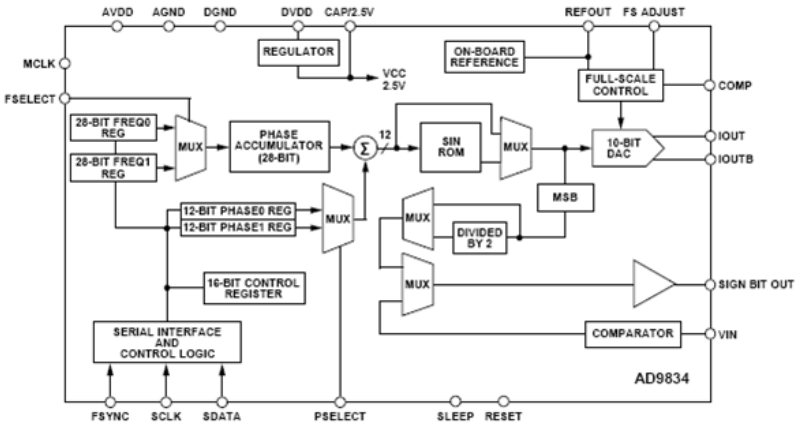

Figure 4.1. Functional block diagram of AD9834. After [29].

As seen in the block diagram two frequency control registers are used in order to determine the frequency of the output signal. For standard operation using only one of the control registers is enough. Furthermore, it is also possible to use only the 14 most significant bits of the control registers, if coarse adjustment is enough for the application. The least significant bits of the frequency registers can be used for fine tuning of the resultant output. In our work, we only used the most significant 14 bits of the frequency register. The phase registers are use to introduce phase shift into the resultant output signal. These registers are basically used for phase modulation. Since it is not relevant in our work we did not use phase control registers.

SIN ROM is a built-in lookup table that is used for generating the output. The address of the necessary entry is calculated based of the frequency and phase control registers. The value is fetched from the SIN ROM and is converted into an analog signal through the 10-bit ADC.

The frequency of the resultant signal of AD9834 is given as

$$f = f_{MCLK}/2^{28} \times FREQREG \tag{4.1}$$

where f_{MCLK} is the frequency of the main clock that is generated by an external source. It is a 75 MHz oscillator from "CTS Frequency controls" in our work.

Data is loaded into the control registers via FSYNC, SCLK and SDATA pins of AD9834. FSYNC pin is used to start the communication and act as the chip select pin in standard 3-SPI serial communication. A low value will initiate the communication. After enabling the communication data that is available on SDATA pin is loaded into the selected register at every falling edge of the SCLK pin. The data on the SDATA pin should be 16-bits long in order to be evaluated as valid data. Register selection is done the first two bits of 16-bit data that is sent via SDATA pin.

The implementation of AD9834 in our work is given in Figure 4.2.

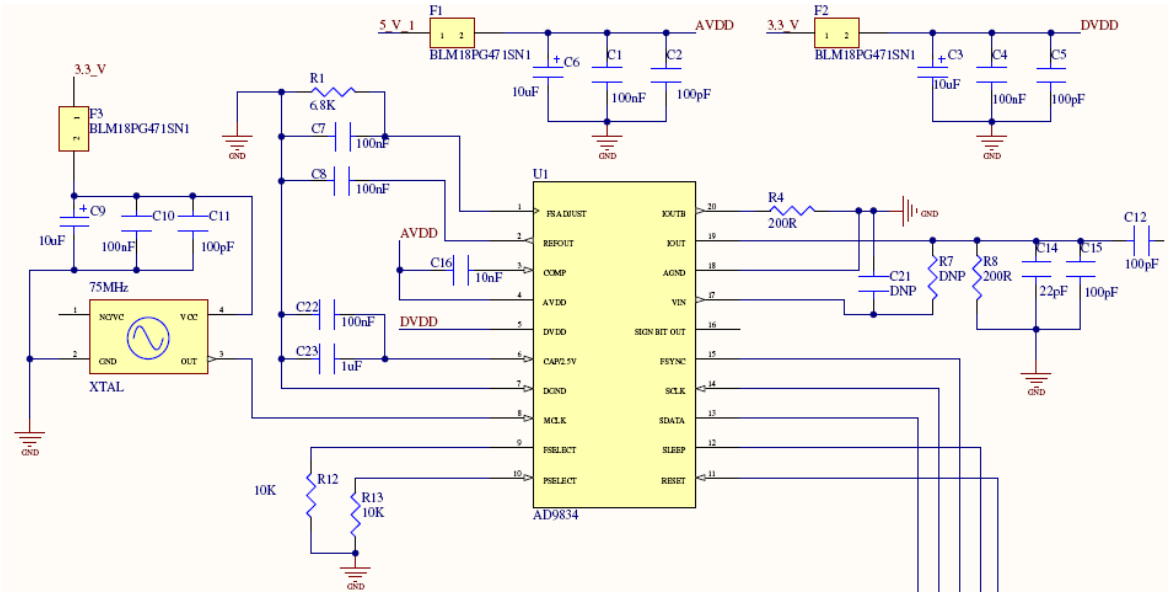

Figure 4.2. Implementation of AD9834.

The output of AD9834 is passed through two pre amplifiers stages in order to get power amplification. In our work we used HMC589ST89 from Hittite Microwave. The resultant screenshots are given in Figure 4.3 and Figure 4.4.

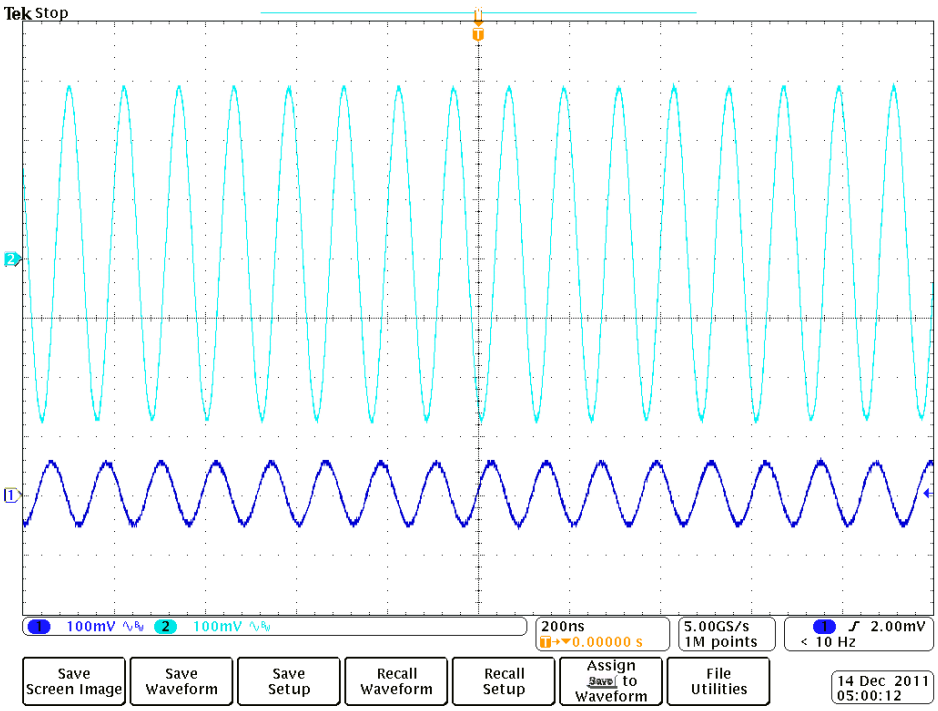

Figure 4.3. Pre-amplifier 1 input (blue) and output (green).

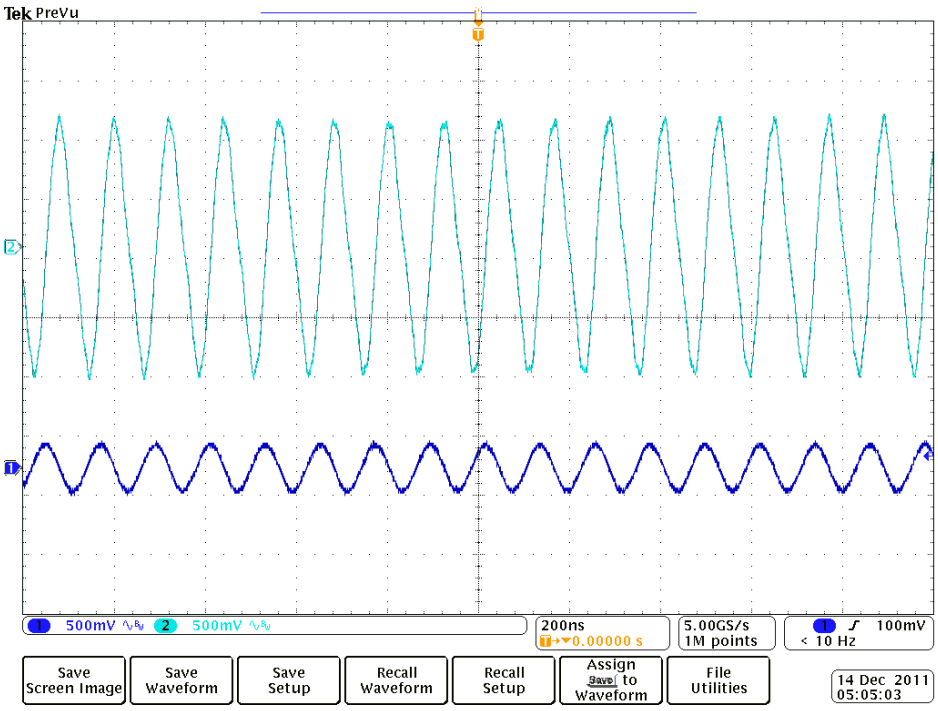

Figure 4.4. Pre-amplifier 2 input (blue) and output (green).

4.2.2. Power Detection

Power detection block is mainly used for detecting the presence of the EAS tag in the environment. Due to the physical nature of the EAS tag, it starts to emit signals when external magnetic field is stopped. The emission is captured by the receive antenna and the power level is detected via a logarithmic converter IC. In our work, we used AD8310 logarithmic amplifier from Analog Devices.

AD8310 is basically a logarithmic converter rather than a logarithmic amplifier as its name states. It converts the input voltage to an analog output in terms of decibels. As seen in the clock diagram of AD8310, the input passes through six amplifier states which are having a gain of 14.3 dB. The levels are detected via nine detector cells which have current outputs. The resultant currents added summed up and converted to voltage on 3 kΩ resistor. The signal is multiplied by three at the output stage and is observed at the output pin.

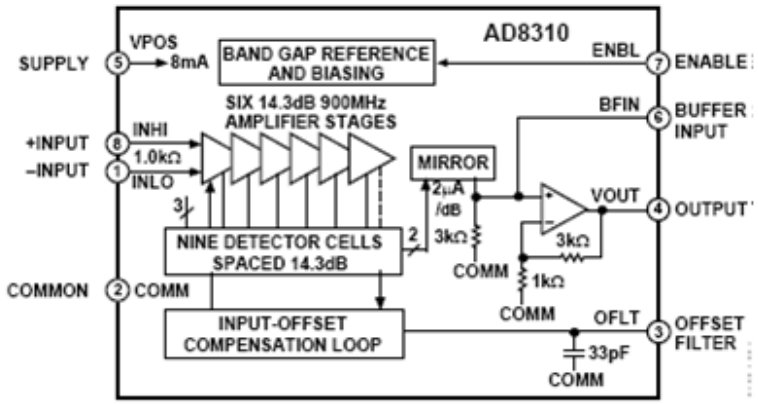

Figure 4.5. Block diagram of AD8310. After [30].

The input-output relationship of AD8310 is given as

$$V_{OUT} = V_{SLOPE}(P_{IN} - P_0) \tag{4.2}$$

where V_{SLOPE} is 24 mV/dB and P_0 is the intercept point which is -108 dBV or equiva-

lently -95 dBm for AD8310. The logarithmic scale called dBV is decibels with respect to a 1 V RMS sine wave.

For example, an input signal of 1 mV RMS sinusoidal waveform (-60 dBV) will correspond to $0.024 \times (20log(0.001) + 108) = 1.15\ V$ output

The input level vs. output voltage plot is given in Figure 4.6.

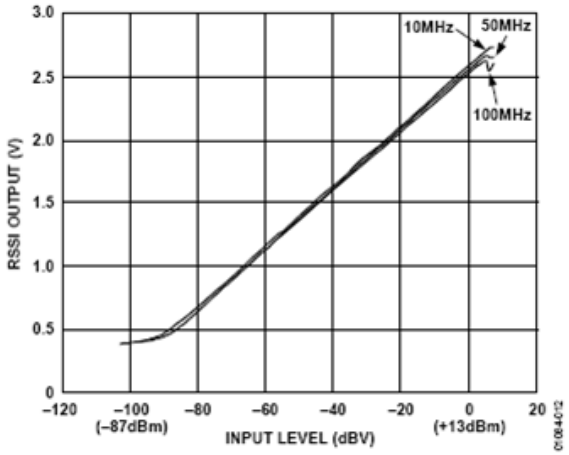

Figure 4.6. RSSI output vs. input level at T_A=25°C for frequencies of 10 MHz, 50 MHz, and 100 MHz.

Based on the description above, we used AD8310 to detect the power of the received signal. The output of AD8310 is digitized by the ADC block of the microcontroller and presence of the tag is detected. The tag detection algorithm will be described in the microcontroller section. The implementation of AD8310 in our circuit is shown in Figure 4.7.

From Figure 4.8 to Figure 4.11, oscilloscope results are given.

4.2.3. Microcontroller

A microcontroller or a processer block is used for doing the initial settings of the system, configuring AD9834 DDS IC and AD8310 logarithmic amplifier. It also runs

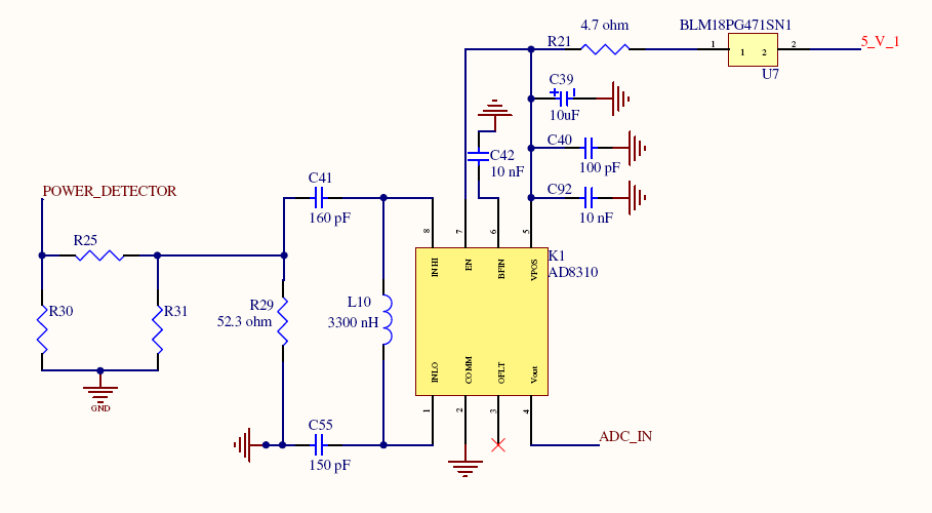

Figure 4.7. Implementation of AD8310.

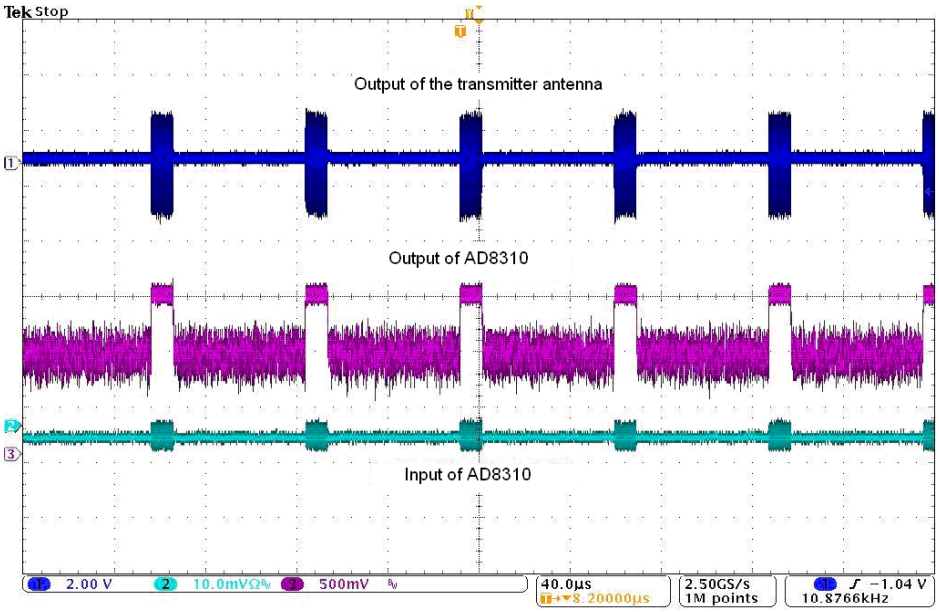

Figure 4.8. No tag (blue: output of the transmitter antenna, purple: output of AD8310, green: input of AD8310).

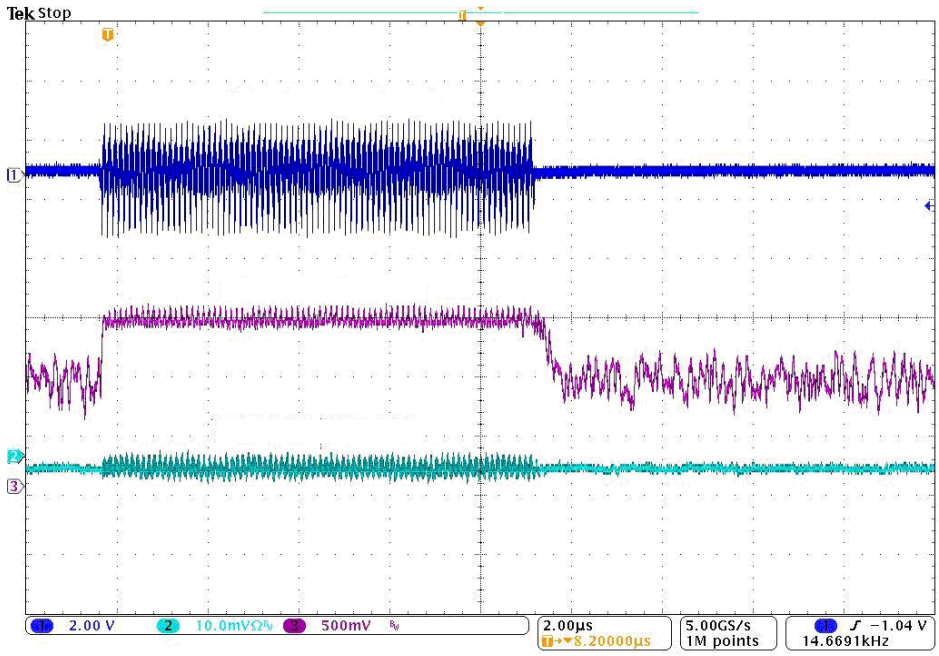

Figure 4.9. No tag, close up(blue: output of the transmitter antenna, purple: output of AD8310, green: input of AD8310).

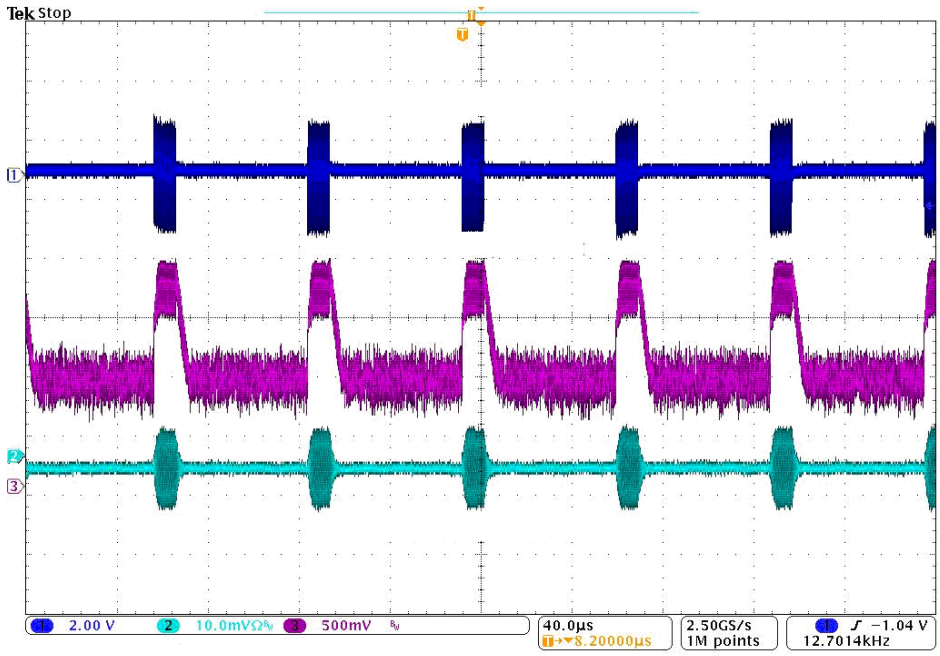

Figure 4.10. Tag between antennas (blue: output of the transmitter antenna, purple: output of AD8310, green: input of AD8310).

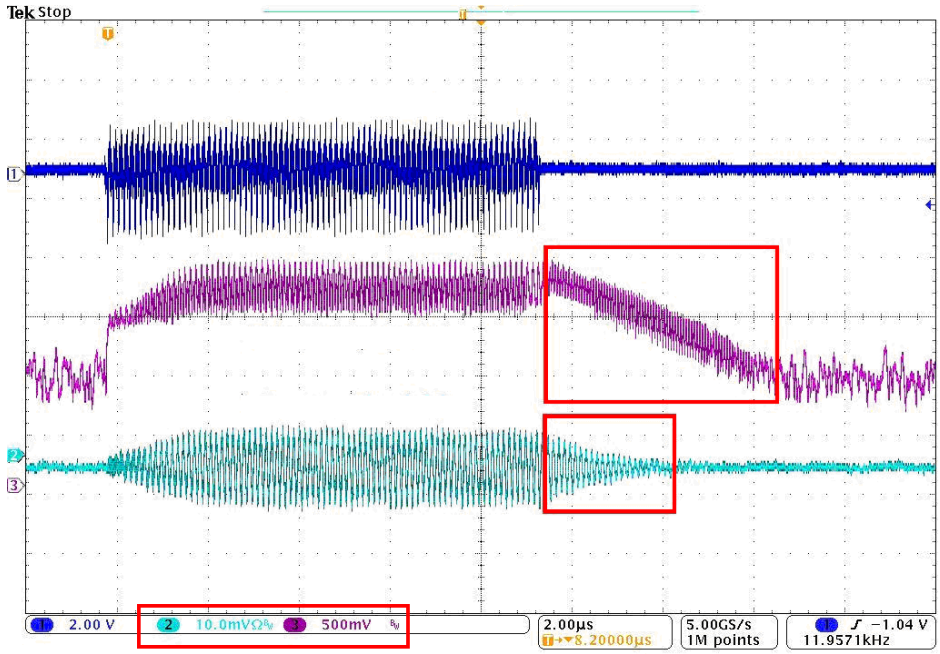

Figure 4.11. Tag between antennas, close-up (blue: output of the transmitter antenna, purple: output of AD8310, green: input of AD8310).

the detection algorithm and arranges the frequency sweeping. In our work, we used a 16-bit microcontroller which has a RISC CPU from Texas Instruments [31].

As stated previously, EAS tags do not have a specific resonance frequency due to their manufacturing process. In order to detect the presence of a tag in the environment, the frequency band is swept with 32 pulses in the range of 7.6 MHz and 9.4 MHz. Each pulse is a magnetic burst which is sent from the transmitter antenna and they are around 40 kHz apart from each other. In Table 4.1 there is a list of pulse frequencies.

The program running on the microcontroller starts with the initial setting of the systems. First of all, watchdog timer is stopped. Watchdog timer is a free running counter that raises a system reset when overflows. In order to prevent system locks and infinite dead loops, watchdog timer is started and its value is reset at different points in the program. This prevents the watchdog timer to overflow and the program flow is not disturbed. If a system lock or an unwanted infinite loop happens watchdog timer overflow occurs and the system is rebooted to start from the beginning. Since

Table 4.1. Pulse frequencies

Pulse No.	Frequency (MHz)	Pulse No.	Frequency (MHz)
1	7.600	17	8.250
2	7.641	18	8.291
3	7.681	19	8.331
4	7.722	20	8.372
5	7.763	21	8.413
6	7.803	22	8.453
7	7.844	23	8.494
8	7.884	24	8.534
9	7.925	25	8.575
10	7.966	26	8.616
11	8.006	27	8.656
12	8.047	28	8.697
13	8.088	29	8.738
14	8.128	30	8.778
15	8.169	31	8.819
16	8.209	32	8.859

the algorithm and the main flow is not very complex in behavior, we do not need the watchdog timer, so it is disabled in order not to generate a systems reboot interrupt during the flow.

The program then carries out the basic clock module settings of the microcontroller. Since timing values are very important during transmission and reception, a precise clock source is needed while running the program. MSP430 microcontroller has a built in clock source which is called digitally controller oscillator (DCO). However, DCO produces a clock output which can be affected by temperature changes and is not a very accurate clock source. In order to have a low tolerance and precise clock source we used a 16 MHz crystal externally connected to the microcontroller which results in a 62.5 ns clock period. This clock speed is precise and also fast enough to implement the detection algorithm that we use.

After setting the clock registers, DDS initialization is carried out. Basic SPI interface registers are handled according to the datasheet of MSP430. As previously stated the communication interface between AD9834 and MSP430 is not a standard 3-wire SPI connection. In standard 3-wire SPI connection, chip select output is reset before each byte sent and set after each byte sent. However, AD9834 accepts data in words which means 16 bits should be transferred before the ship select bit is set again. In order to implement a workaround for this problem, chip select control is disabled in SPI control registers of the microcontroller. FSYNC pin which is used to tell AD9834 that the data transmission is started is controller via a standard I/O pin instead of built-in chip select pin of the SPI controller.

After setting the SPI registers and the relevant I/O settings, ADC block of the microcontroller is set. The sampling rate is controller via ADC control registers. Like most other peripherals of MSP430, the inputs pins of the ADC block should be configured for a special purpose which is different from a standard I/O pin. Each I/O pin is set as output pin initially. So we need to set the corresponding pin as the analog input pin in order to direct the analog data to the ADC block.

After setting the ADC control registers and the corresponding I/O settings 32 frequency values are computed and recorded in an array. These values are equally spaced from each other and computation is done assuming that a 75 MHz oscillator is used as the main clock source for AD9834. The start and the stop frequencies are given in two variables in "main.c".

After calculating the frequency values the loop starts. For each frequency value in the array, the configuration word is sent to AD9834 in order to make the DDS IC to produce the desired sinusoidal waveform. When the configuration word is sent to the DDS IC, the transmit antenna is enabled and the pre-amplifiers that are on the path from DDS to the transmit antenna output are enabled. Transmission lasts for 10 μs. If there is an EAS tag in the environment, this amount of time would be enough to make the EAS tag resonating. After sending the signal form the transmit antenna for 10 μs, the transmit antenna is disabled ant ADC block is activated immediately. Activating

the ADC block results in acquiring the samples from the output of the logarithmic amplifier (AD8310) which is connected to the receive antenna. The analog signal is digitized and is stored in an array. This process is repeated 32 times since there 32 discrete frequency values to try in the calculated frequency array.

The resultant digitized data is then averages and the resulting value is compared to a threshold value. This threshold value is detected experimentally and is tuned for maximum detection range. If the result is greater than the threshold, it means at least one EAS tag is found in the environment and an audio visual alarm is raised. The main algorithm is restarted from the beginning after finishing the steps that are explained above. The flowchart of the detection algorithm and the main loop is given in Figure B.1.

4.2.4. Impedance Matching

Impedance matching is very important in RF systems since it greatly affect the overall system performance. Impedance matching is basically matching the output impedance of the source to the load impedance. This practice protects the system from return loss and reflection of the electromagnetic wave. If the impedance of the load is not matched, in other words if it is not the complex conjugate of the output impedance of the source, reflections will occur at the load side. This means the power that is intended to be transferred to the load will be totally delivered to the load. Instead some of the power will be reflected back to the source according to the reflection coefficient. This will reduce the output power of the system and more importantly, the reflected power may harm the source itself if the reflected power is high enough. In order to maximize the output power delivered to the antenna and hence to maximize the detection range, impedance matching should be carried out. In most RF systems the characteristic impedance is 50 Ω and the input and output impedances are generally matched to this common impedance.

In our work, we used a loop antenna layout that is provided by Texas Instruments for access control purposes. According to the application note provided by Texas

Instruments, the antenna is modeled as a series combination of an inductor which has a value of 1.2608 μH and a resistor which has a value of 0.4648 Ω [32]. The corresponding impedance for 8.2 MHz is (0.4648 + j64.96) Ω. In order to match this impedance to 50 Ω, Smith chart is used. A 176 pF shunt capacitor, 470 Ω shunt resistor and 138 pF series capacitor are used in the matching network in order to have an impedance of 50 Ω. The work done on Smith Chart is given if Figure 4.12.

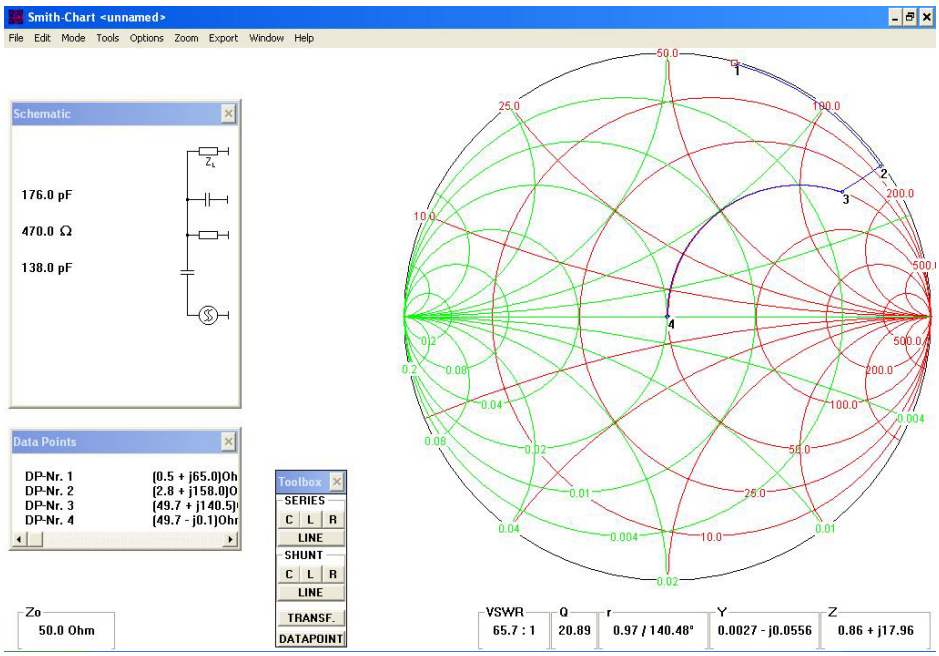

Figure 4.12. Smith chart calculations.

5. CONCLUSION

As previously stated, the aim of this work was to design, implement and verify a low cost microcontroller based EAS system prototype. In order to develop a working prototype of an EAS system, the physical characteristics of the EAS tag were observed. A sniffer and a high speed oscilloscope were used for data acquisition from the tag which was energized by an external EAS system. Simulations were carried out on the acquired data in order to determine a robust algorithm for detection.

Based on the simulation results on the acquired data, the necessities of the hardware blocks were determined. For frequency synthesizer, a direct digital synthesizer IC, AD9834 from Analog devices was used. Surface mount amplifiers, HMC589ST89 from Hittite Microwave were used to amplify the generated excitation signal. A logarithmic amplifier, AD8310 from Analog devices was used for detecting the power level of the incoming signal from the tag. This analog power level was digitized by the analog to digital converter block of our micro controller MSP430F2617 from Texas Instruments. The same micro controller is also used for running the detection algorithm and the main loop.

APPENDIX A: OVERALL SCHEMATICS

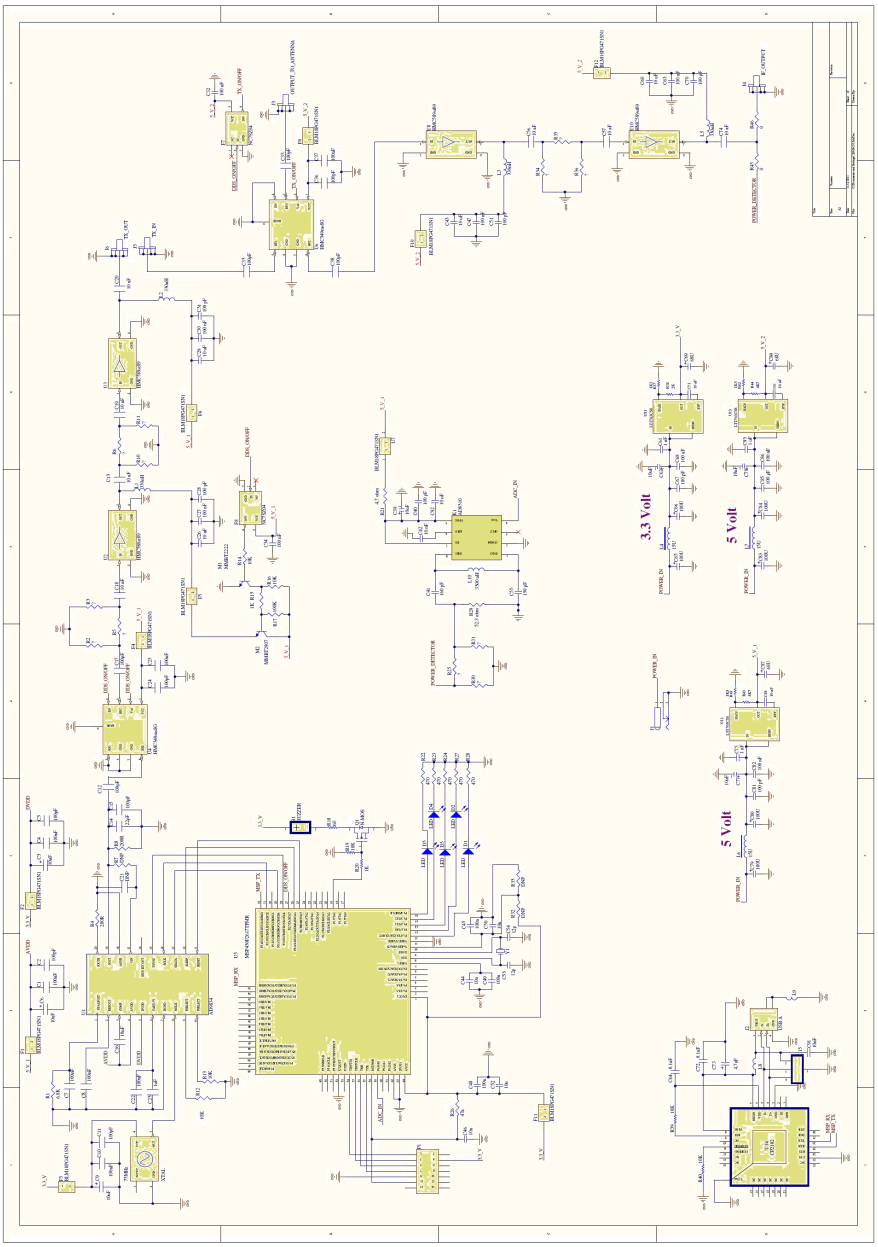

Figure A.1. Overall schematics of the work.

APPENDIX B: EAS FLOWCHART

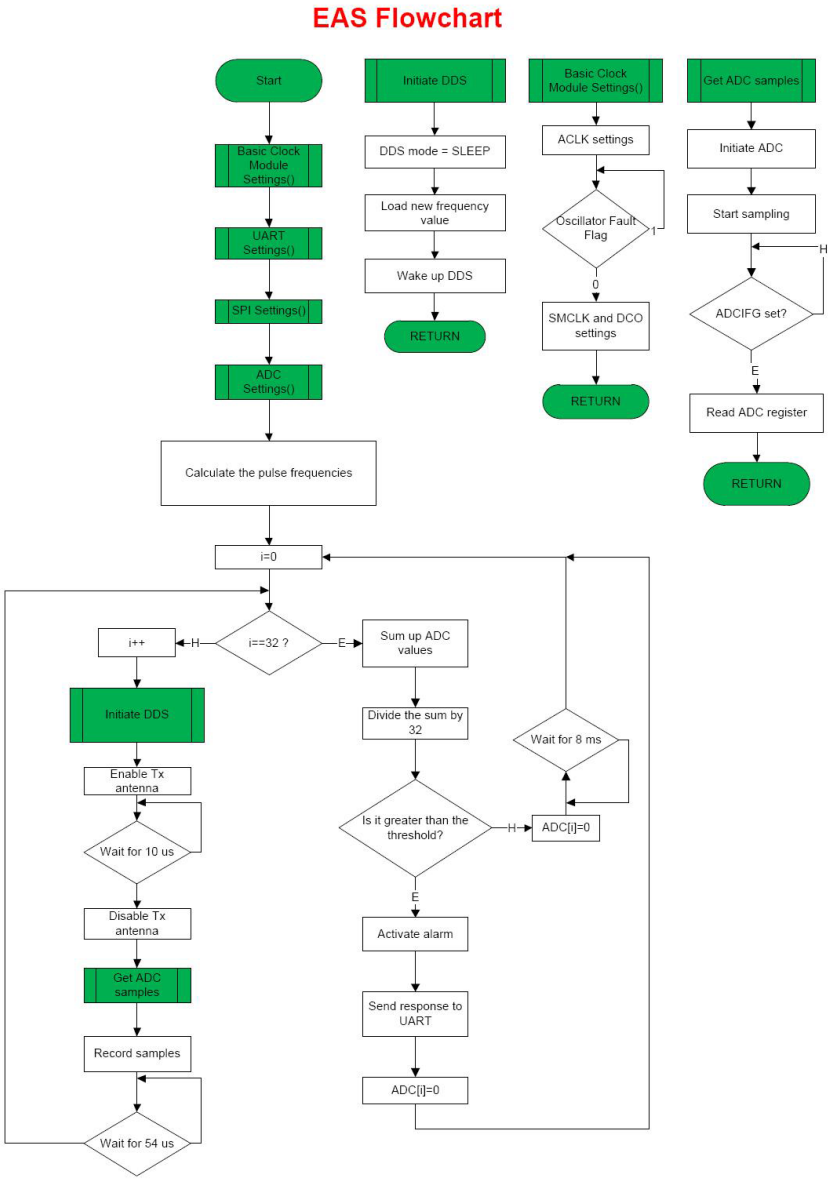

Figure B.1. EAS flowchart.

APPENDIX C: FIRMWARE

C.1. Main.c

```
#include ``globals.h"
#include ``hardware.h"
#include ``DDS.h"
#include ``uart.h"
#include ``ADC.h"

unsigned char sleep[2];
unsigned char set[2];
unsigned char reset[2];
unsigned char farray[32][2][2];
unsigned char TimerOverflowFlag;
unsigned char i,ii;
unsigned int temp[32];

void main(void)
{
      System_Init();
        P1DIR=0xFF;
        P2DIR=0xFF;
        P3DIR=0xFF;
        P4DIR=0xFF;
        P5DIR=0xFF;
      DDS_Initial_Set();


      DDS_ON;
```

```
        Led2OFF;
        Led3OFF;
        Led4OFF;
        Led5OFF;
        unsigned int first, last;
        first=7600;
        last=8900;
    //    first=8200;
    //    last=8232;
        DDS_Write(reset);
        DDS_Set_Array(first, last);
        ADC_Init();

        TimerBstart(37);

int sonuc;

BuzzerOFF;



        while(1)
        {
          Led3ON;

          for(i=0;i<32;i++)        //2ms surecek olan transmit
              dongu
          {
            DDS_Write(sleep);       //DDS cikisinda midscale
                cikis ver.
            DDS_AWrite(i);          //yeni frekans degerini
                yazdir.
```

```
    TimerOverflowFlag=0;
    TBR = 0x0000;
    startCounterB;              //bitis (yaklasik 19us)
    while(!TimerOverflowFlag) //timer dolmasini bekle
    {

    }
    DDS_Write(set);             //DDS cikisini ac
    DDS_ON;                     //DDS cikisini RF katina
        gonder
    delay_us(10);
    ADC12CTL0 |= ADC12SC;                   // Sampling
        open
    //delay_us(1);
    DDS_OFF;                    //DDS cikisi ile RF kati
        arasindaki baglantiyi kes.
    while (!(ADC12IFG & BIT0));
    temp[i]=ADC12MEM0;

}
sonuc=0;
Led3OFF;

//8ms surecek olan transmit kapali dongu.
for(i=0;i<32;i++)
{
    temp[i]=temp[i]>>1;
    sonuc=temp[i]+sonuc;
}
      if(sonuc>25000)
      {
```

```
BuzzerON;
Led5OFF;
Led2ON;
delay_ms(70);
BuzzerOFF;
Led5ON;
Led2OFF;
delay_ms(50);

BuzzerON;
Led5OFF;
Led2ON;
delay_ms(70);
BuzzerOFF;
Led5ON;
Led2OFF;
delay_ms(50);

BuzzerON;
Led5OFF;
Led2ON;
delay_ms(70);
BuzzerOFF;
Led5ON;
Led2OFF;
delay_ms(150);

BuzzerON;
Led5OFF;
Led2ON;
```

```
delay_ms(300);
BuzzerOFF;
Led5ON;
Led2OFF;
delay_ms(50);

BuzzerON;
Led5OFF;
Led2ON;
delay_ms(300);
BuzzerOFF;
Led5ON;
Led2OFF;
delay_ms(150);

BuzzerON;
Led5OFF;
Led2ON;
delay_ms(70);
BuzzerOFF;
Led5ON;
Led2OFF;
delay_ms(50);

BuzzerON;
Led5OFF;
Led2ON;
delay_ms(70);
BuzzerOFF;
Led5ON;
Led2OFF;
```

```
                delay_ms(50);

                BuzzerON;
                Led5OFF;
                Led2ON;
                delay_ms(70);
                BuzzerOFF;
                Led5ON;
                Led2OFF;
                delay_ms(150);
            }
            else
            {
                BuzzerOFF;
            }
            delay_ms(8);
        }
}
```

C.2. UART.c

```
#include ``uart.h"
#include ``hardware.h"

unsigned char RXdone = 0;


void UARTset(void)                          //Uses USCI_A1
{
        UCA1CTL1 |= UCSWRST;                //disable UART
        UCA1CTL1 |= UCSSEL_1;               //ACLK ile senkronize
            (16Mhz)
```

```
        UCA1CTL0 = 0x00;
        UCA1BR0 = BAUD0;                    //sets the baudrate
           according to the calculated values of BAUD0 and
           BAUD1;
        //for detailed information *#* see MSP user guide
        UCA1BR1 = BAUD1;                                    //BAUD
            = 128000
        UCA1MCTL = 0;                       //Modulation UCBRSx =
           5
        P3SEL |= BIT4 + BIT5;               //P3.4, P3.5 UART mode
        P3DIR |= BIT4;                                      //P3.4
            - output
        UCA1CTL1 &= ~UCSWRST;               //Initialize USCI
           state machine
        UC1IFG &= ~UCA1RXIE;                  //Disable USCI_A1
           RX interrupt
}

void Put_byte(unsigned char abyte)
{
        //This Function Sends a byte to the Computer via UART
           module .
        while(!(UC1IFG & UCA1TXIFG)); /* wait for TX register
           to be empty */
        UCA1TXBUF = abyte;
}

//////////////////////////////////////
//Common Interrupt RX Vector for both USCIA - UART & USCIB -
   SPI
#pragma vector = USCIAB1RX_VECTOR
```

```
__interrupt void RXhandler (void)
{
        //When a byte is observed in the communication bus,
           this Interrupt arises.
        TimerBstart(3000);        //timer B 3 msec sonra
           interrupt vermesi icin kurulur.
        //UART'tan herhangi bir byte geldiginde timer
           baslatilir. Bu islem her byte geldiginde
           tekrarlanir
        //iki frame arasindaki sure, frameleri olusturan byte'
           lar arasindaki sureden cok cok daha uzundur.
        //Bu timer'in overflow etmesi demek, butun frame'in
           aliminin tamamlanmasi demektir.
        //Timer ISR'inda ise bir flag set edilir ve ardindan
           delay_ms_UART() fonksiyonu bunu detect edip
        //gelen datayi isler.
        //*#* bkz flowchart
        RXdone = 1;
}


void ExecuteUART()
{
        // UART uzerinden data aliminin tamamlandigi UART-
           Timer Interruptlari ve delay_ms_UART() fonksiyonu
           ile tamamlandiginda
        // Datananin frame yapisina uyup uymadigi, komutun
           readerin kendisine gelip gelmedigi, geldiyse hangi
           komutun geldigi
        // hangi komuta nasil cevap verecegi, cevabin
           verilmesi gibi islemler bu fonksiyonda yapilir.
```

```
        //*#* bkz flowchart
        DisableUART;            //butun bu islemler sirasinda
           baska bir UART data alimi yapilmasi istenmedigi
           icin UART Interrupt disable edilir.
        RXdone = 0;             //data alimi sirasinda
           degistirilen degerler sifirlanir.

}

void EnableUART()
{
        TimerBstart(4000);
    AGAIN1:
        while(UC1IFG & UCA1RXIFG)
        {
                //rxdata = UCA0RXBUF;
                if(TimerOverflowFlag)
                {
                        TimerOverflowFlag = 0;
                        RXdone = 0;
                        disableTBInt;
                        return;
                }
        }
        delay_us(410);
        if(!(UC1IFG & UCA1RXIFG))
        {
                stopCounterB;
                UC1IE |= UCA1RXIE;
        }
        else goto AGAIN1;
```

```
}
```

C.3. Hardware.c

```
#include ``hardware.h"
#include ``DDS.h"
#include ``uart.h"
/*
==========================================
==========================================
*/
void delay_ms(unsigned int n_ms)
{
//Bu fonksiyon milisaniye cinsinden aldigi parametre kadar
   bekleme yapar...
        unsigned int ii1, ii0;
        for(ii0=n_ms; ii0>0; ii0--)
        {
          ii1 = 0x0DFF;                    // Delay
          while (ii1 != 0)
          {
            --ii1;
          }
        }
}
// end of delay_ms


/*
==========================================
==========================================
*/
```

```
void delay_us(unsigned int n_us)
{
        //Bu fonksiyon milisaniye cinsinden aldigi parametre
           kadar bekleme yapar...
 // unsigned char i=0xFF;
  while(n_us)
  {
    n_us--;
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();
  }
}
// end of delay_ms


/*
==========================================
==========================================
*/

void TimerAstart()
{
        TACTL |= TACLR;
```

```
        TACTL &= ~TACLR;                        //reset the
            timerA
        TACTL |= TASSEL1;       //ACLK, div 8, interrupt
            enable, timer stoped
        TAR = 0x0000;
        TACCTL0 |= CCIE;
}

void TimerBstart(int microsec)
{
        if(microsec>4096) microsec = 4096;      // IMPORTANT :
            input cannot be bigger than 77, otherwise it is
            set to 77 miliseconds.
        TBCTL |= TBCLR;
        TBCTL &= ~TBCLR;                //reset the timerB
        TBCTL |= TBSSEL0;               //ACLK, div 1,
            interrupt enable, timer stoped
        TBR = 0x0000;   //that is the line eq of the milisec
            vs TBR graph.
        TBCCR0=16*microsec;
        TBCCTL0 |= CCIE;                //enable interrupt
}

void TimerBint(void)
{
        TBCTL |= TBCLR;
        TBCTL &= ~TBCLR;                                //
            reset the timerB
        TBCTL |= TBSSEL0;       //ACLK, div 8, interrupt
            enable, timer stoped
        TBR = 0x0000;
```

```
        TBCCTL0 |= CCIE;
}
/*
==========================================
==========================================
*/

void OSCsel()
{
        unsigned int ii1;
        DCOCTL |= DCO0+DCO1;
        delay_ms(5);
        //BCSCTL1 |= CALBC1_16MHZ_;
        BCSCTL1 |= XTS + XT2OFF + RSEL0 + RSEL1 + RSEL2 +
           RSEL3;                    // ACLK = LFXT1 HF XTAL
        delay_ms(5);
        BCSCTL3 |= LFXT1S1;                                  //
           EXTERNAL 3-16 MHz clock
        delay_ms(5);
        // Note : if MCU is getting its clk from a crystal osc
           . directly (Not from TRF), we must add some delays
           between and/or after - before
        // registry BCSCTL1 - BCSCTL3 settings. Emprically, a
           4ms delay is found to be enough.
        // the aim of do-while block is to wait until the
           oscillator works perfectly.
        do
        {
                IFG1 &= ~OFIFG;                              //
                   Clear OSCFault flag
```

```
                for (ii1 = 0xFF; ii1 > 0; ii1--);           //
                    Time delay for flag to set
        }
        while ((IFG1 & OFIFG) == OFIFG);          // OSCFault
            flag still set?
        delay_ms(5);
        Led1ON;
        BCSCTL2 |= SELM1 + SELM0 + SELS;          //SMCLK = DCO
            (1.1MHz)/8
        //BCSCTL2 |= SELM1 + SELM0 + DIVS0 + SELS;
        //SMCLK = SYS_CLK(13.56MHz)/2
        // IMPORTANT NOTE : in user guide, 'present' means
            wheter the MCU has XT2 pin or not.
        // It does NOT mean that whether an oscillator is
            connected to that pin.
        delay_ms(5);
        return;
}

/*
================================================================
================================================================
*/

#pragma vector=TIMERA0_VECTOR
__interrupt void TimerAhandler(void)
{
        __low_power_mode_off_on_exit();
                //interrupt geldiginde LPM'den cik
}//TimerAhandler
```

```
#pragma vector=TIMERB0_VECTOR
__interrupt void TimerBhandler(void)
{
        // Timer B is used in UART communication.
        TimerBint();
        TimerOverflowFlag = 1;
}



void System_Init ()
{
        WDTCTL = WDTPW + WDTHOLD;              //Stop WDT
        Led1OFF;
        OSCsel();                              //switch from DCO
           to crystal 16 MHz oscilator, see basic clk module
           app note.
        delay_ms(10);
        EnableInterrupts;                     //General enable
           interrupts
        TimerOverflowFlag = 0;                //initially zero,
           it is set when timer is overflew, which is the case
            for a frame to be completely received
        UARTset();
}

void SPI_Set()
{
```

```
    //1.Set UCSWRST (BIS.B #UCSWRST, &UCxCTL1)

        UCB0CTL1 |= UCSWRST;                          //
            reseti bas once

    //2.Initialize all USCI registers with UCSWRST=1(including
       UCxCTL1)

        UCB0CTL0 |= UCMSB + UCMST + UCSYNC + UCCKPL+UCCKPH ;
            // UCMSB=MSB first , UCMST=Master mode, UCSYNC=
            Synchronous mode, UCCKPL=High is inactive state (3-
            pin, 8-bit SPI master)
        UCB0CTL1 |= UCSSEL_1;                         // ACLK
        UCB0BR0 = 0x01;                               //
            bitrate control reg. (divider=1)
        UCB0BR1 = 0x00;

     //3.Configure ports

        P3SEL |= BIT1 + BIT3;                         // P3
            .1,3.3 UCB0SIMO, UCB0OCLK option select

     //4.Clear UCSWRST via software (BIC.B #UCSWRST,&UCxCTL1)

        UCB0CTL1 &= ~UCSWRST;                         // **
            Initialize USCI state machine**

    //5.Enable interrupts (optional) via UCxRXIE and/or UCxTXIE
        (veri almadigimiz icin spi interrupta gerek yok)
}
```

C.4. DDS.c

```
#include ``DDS.h"
#include ``hardware.h"
#include ``globals.h"
unsigned long FREQ;
unsigned char LSword[2], MSword[2];
unsigned char j,k;

/*******************************/
// DDS iletisimi icin gerekli SPI ve PIN ayarlamalarini yapar
/*************************************/
/*****************************/
/*!* Buradaki reset ve set bytelari FREQ0 ve PHASE0
   registerlari kullanilacak sekilde ayarlandi.
bunlarda degisiklik olacaksa byte degeri de degisecek UNUTMA!
   (set kisminda gecerli bu, reset bolumunde
midscale cikis vereceginden -FREQ kismi ignored-..)*/
/*****************************/
void DDS_Initial_Set()
{

        SPI_Set();                  //SPI modulunu ayarlayan
           fonksiyon
        FsyncHIGH;              //STE pini aslinda ayni isi
           goruyor. fakat SPI modulu 8 bitlik iletisimi
           destekliyor.
                                    //DDS cipi tek pakette 16
                                       biti anlamlandirdigi icin
                                       burada kullanamadik

        reset[0]=0x31;          //Reset dizisi
```

```
        reset[1]=0x18;

        DDS_Write(reset);

        //burada PHASE0 registeri 0 lanacak. simdilik
           kullanmadigimiz icin boyle
        set[0] = 0xD0;          //basina command code yollamadik,
            cunku yukarda gonderdigimiz reset komutu ayni
           zamanda command code oldu.
        set[1] = 0x01;          //Bu kullanilirsa OGKG kodu
           aktiflenebilir. BYTOK da olasi.
        DDS_Write(set);             //bunun icinde function yazman
            gerekir, falan filan...

        set[0]=0x30;          //Set dizisi
        set[1]=0x18;

        sleep[0]=0x98;
        sleep[1]=0x20;
}

/************************/
//bu fonksiyon sadece 16 bitlik tek bir frame gondermek icin
   cagiriliyor.
//bunu yapabilmek icin de 8 bitten olusan 2 bytelik bir dizi
   gonderiyor.
//gonderirken MSB den LSB ye gonderiyor.
//eger 2 frame den olusan (toplam 4 byte) bir frekans verisi
   gonderilecekse once LSB frame sonra MSB frame gonderilecek
//bununla karistirma sakin!
```

```
/* Buraya giren dizi her zaman 2 byte olacak, bunu UNUTMA!
   byte[1] MSB yi tutacak, byte[0] LSB yi tutacak. */
/************************/
/*void DDS_Write(unsigned char byte[])
{
        k=0x80;
        FsyncLOW;
        for(j=0;j<8;j++)
        {
          if(byte[1] & k)
          {
            SdataHIGH;
          }
          else
          {
            SdataLOW;
          }
          SclkHIGH;
          SclkLOW;
          k = k >> 1;
        }
        k=0x80;
        for(j=0;j<8;j++)
        {
          if(byte[0] & k)
          {
            SdataHIGH;
          }
          else
          {
            SdataLOW;
```

```
            }
            SclkHIGH;
            SclkLOW;
            k = k >> 1;
          }
          FsyncHIGH;
}*/

void DDS_Write(unsigned char byte[])
{
          FsyncLOW;

          while (!(IFG2 & UCB0TXIFG));  //SPI TX flagini kontrol
              et. register bosalmissa yeni veriyi yazabilirsin.
          UCB0TXBUF = byte[0];
          while (!(IFG2 & UCB0TXIFG));
          UCB0TXBUF = byte[1];
          while (!(IFG2 & UCB0TXIFG));  //son kez kontrol
             edilmesinin sebebi yeni data yazilacagindan degil.
                                        //FSYNC high yapmadan
                                           once gonderimin
                                           bitmesi gerekiyor.
                                           yoksa DDS datayi
                                           anlamayacak.
          FsyncHIGH;
}


/*****************************/
/* 75 MHz MCLK varken, 1KHz hassasiyette frekans
   yazdirabiliyon. gonderdigin degeri KHz olarak gonderdigini
```

```
   unutma. */
/*****************************/

void DDS_AWrite(unsigned char x)
{
        FsyncLOW;

        while (!(IFG2 & UCB0TXIFG));  //SPI TX flagini kontrol
            et. register bosalmissa yeni veriyi yazabilirsin.
        UCB0TXBUF = farray[x][0][1];
        while (!(IFG2 & UCB0TXIFG));
        UCB0TXBUF = farray[x][0][0];
        while (!(IFG2 & UCB0TXIFG));
        UCB0TXBUF = farray[x][1][1];
        while (!(IFG2 & UCB0TXIFG));
        UCB0TXBUF = farray[x][1][0];
        while (!(IFG2 & UCB0TXIFG));  //son kez kontrol
           edilmesinin sebebi yeni data yazilacagindan degil.
                                      //FSYNC high yapmadan
                                         once gonderimin
                                         bitmesi gerekiyor.
                                         yoksa DDS datayi
                                         anlamayacak.
        FsyncHIGH;
}
/***********************************/
/* 75 MHz MCLK varken, 1KHz hassasiyette frekans
   yazdirabiliyon. gonderdigin degeri KHz olarak gonderdigini
   unutma. */
/***********************************/
void DDS_Set_Freq(unsigned long f)
```

```
{
        FREQ=3579*f;  //75MHz kristal icin hesaplandi.
           degisirse tekrar hesapla.

        LSword[1] = FREQ;                    //1. sekiz biti al.
        // LS Byte part of FREQ0 LSB

        FREQ = FREQ >> 8;                    //sondaki 8 bitten
           kurtul.
        LSword[0] = FREQ;                    //2. sekiz biti al.
        LSword[0] = LSword[0] | 0x40;        //7. biti 1 yap
        if(LSword[0]>0x80)
        {
          LSword[0] = LSword[0] + 0x80;      // 8. biti 0 yap.
        }
        // MS Byte part of FREQ0 LSB


        FREQ = FREQ >> 6;                    //sondaki 6 bitten
           kurtul (LSB kisminda 8+6=14 bit gonderdik.)
        MSword[1] = FREQ;                    //3. sekiz biti al
        // LS Byte part of FREQ0 MSB

        FREQ = FREQ >> 8;                    //sondaki 8 bitten
           kurtul.
        MSword[0] = FREQ;                    //4. sekiz biti al
        MSword[0] = MSword[0] | 0x40;        //7. biti 1 yap
        if(MSword[0]>0x80)
        {
          MSword[0] = MSword[0]+0x80;        //8.biti 0 yap.
        }
```

```
        //MS Byte part of FREQ0 MSB

        DDS_Write(reset);
        DDS_Write(LSword);
        DDS_Write(MSword);

}

/*******************************/
/* Bu kisimda baslangic bitis ve toplam sayisi verilen bir
   frekans bandi icin dizi olusturulur */
/*******************************/

void DDS_Set_Array(unsigned int first, unsigned int last)
{
  unsigned char i;
  unsigned char incr;
  last=last-first;
  incr=last/32;
  unsigned long temp;
  temp=first;
  /* farray [x][y][z]  y kismi FREQ registerinin 16 bitlik
     kisminin MSB/LSB sini belirtir. 0 LSB, 1 MSB dir.
                       z kismi FREQ registerinin  MSB veya LSB
                          sinin 8 bitlik parcalarini belirtir
                          . 0 LSB, 1 MSB dir.
                       x kismi FREQ register paketlerinin
                          tutuldugu array dizinidir.

  (Bu sekilde siralamanin nedeni; frekans yazilirken ilk once
     LSB 16 bit sonra MSB 16 bit yollanir.
```

```
Bu gonderilen 16 bitlik paketler de MSB den LSB ye seklinde
   gonderilir.)*/


for(i=0;i<32;i++)
{
      FREQ=3579*temp;  //75MHz kristal icin hesaplandi.
         degisirse tekrar hesapla.

      farray[i][0][0] = FREQ;                    //1. sekiz
         biti al.
      // LS Byte part of FREQ0 LSB

      FREQ = FREQ >> 8;                  //sondaki 8 bitten
         kurtul.
      farray[i][0][1] = FREQ;                    //2. sekiz
         biti al.
      farray[i][0][1] = farray[i][0][1] | 0x40;      //7.
         biti 1 yap
      if(farray[i][0][1]>0x80)
      {
        farray[i][0][1] = farray[i][0][1] + 0x80;   // 8.
           biti 0 yap.
      }
      // MS Byte part of FREQ0 LSB


      FREQ = FREQ >> 6;                  //sondaki 6 bitten
         kurtul (LSB kisminda 8+6=14 bit gonderdik.)
      farray[i][1][0] = FREQ;                    //3. sekiz
         biti al
```

```
        // LS Byte part of FREQ0 MSB

        FREQ = FREQ >> 8;                    //sondaki 8 bitten
           kurtul.
        farray[i][1][1] = FREQ;                    //4. sekiz
           biti al
        farray[i][1][1] = farray[i][1][1] | 0x40;      //7.
           biti 1 yap
        if(farray[i][1][1]>0x80)
        {
          farray[i][1][1] = farray[i][1][1]+0x80;      //8.biti
              0 yap.
        }
        //MS Byte part of FREQ0 MSB

        temp=temp+incr;
  }
}

/*************************/
void Set_TX_Att(unsigned char mode)
{/*
  switch (mode)
  {
  case 0:
    P3OUT |= 0x74;
    P4OUT |= 0x04; break;
  case 1:
    P3OUT |= 0x74;
    P4OUT |= 0x04;
    TX5_Clr;  break;
```

```
  case 2:
    P3OUT |= 0x74;
    P4OUT |= 0x04;
    TX4_Clr;  break;
  case 4:
    P3OUT |= 0x74;
    P4OUT |= 0x04;
    TX3_Clr;  break;
  case 8:
    P3OUT |= 0x74;
    P4OUT |= 0x04;
    TX2_Clr;  break;
  case 16:
    P3OUT |= 0x74;
    P4OUT |= 0x04;
    TX1_Clr;  break;
  }*/
}

/*******************************/
void Set_RX_Att(unsigned char mode)
{/*
  switch (mode)
  {
  case 0:
    P2OUT |= 0xE0;
    P3OUT |= 0x01;
    P4OUT |= 0x01; break;
  case 1:
    P2OUT |= 0xE0;
    P3OUT |= 0x01;
```

```
    P4OUT |= 0x01;
    RX5_Clr;  break;
  case 2:
    P2OUT |= 0xE0;
    P3OUT |= 0x01;
    P4OUT |= 0x01;
    RX4_Clr;  break;
  case 4:
    P2OUT |= 0xE0;
    P3OUT |= 0x01;
    P4OUT |= 0x01;
    RX3_Clr;  break;
  case 8:
    P2OUT |= 0xE0;
    P3OUT |= 0x01;
    P4OUT |= 0x01;
    RX2_Clr;  break;
  case 16:
    P2OUT |= 0xE0;
    P3OUT |= 0x01;
    P4OUT |= 0x01;
    RX1_Clr;  break;
  }*/
}

/*******************************/
void Set_LO_Att(unsigned char mode)
{/*
  switch (mode)
  {
  case 0:
```

```
    P5OUT |= 0xF8; break;
  case 1:
    P5OUT |= 0xF8;
    Lo5_Clr;  break;
  case 2:
    P5OUT |= 0xF8;
    Lo4_Clr;  break;
  case 4:
    P5OUT |= 0xF8;
    Lo3_Clr;  break;
  case 8:
    P5OUT |= 0xF8;
    Lo2_Clr;  break;
  case 16:
    P5OUT |= 0xF8;
    Lo1_Clr;  break;
  }*/
}
```

C.5. ADC.c

```
#include ``ADC.h"
#include ``hardware.h"
#include ``uart.h"
unsigned int sonuc;
unsigned char bas;

void ADC_Init()
{
  ADC12CTL0 |= SHT0_1 + ADC12ON + REFON + REF2_5V;
                  // Set sampling time, turn on ADC12
  ADC12CTL1 |= SHP /*+ ADC12SSEL1+CONSEQ_2*/;
```

```
  //ADC12IE |= 0x01;                          // Enable
     interrupt
  ADC12MCTL0 = 0x00 ;
  P6SEL |= 0x01 ;                           // P6.0 ADC option
     select
  ADC12CTL0 |= ENC ;                         // Conversion
     enabled
}

#pragma vector=ADC12_VECTOR
__interrupt void ADC12_ISR (void)
{



//if (ADC12MEM0 < 0x7FF)
   //    Led5OFF;                          // Clear P6.2 - LED off
// else
 //       Led5ON;                          // Set P6.2 - LED on
  //   _BIC_SR_IRQ(CPUOFF);                       // Clear CPUOFF
     bit from 0(SR)
              sonuc = ADC12MEM0;                              //
                 RXBUF0 to TXBUF0
              if(sonuc>2000)
                BuzzerON;
              else
                BuzzerOFF;


  //Led1OFF;
}
```

APPENDIX D: PCB LAYERS

D.1. Top Layer

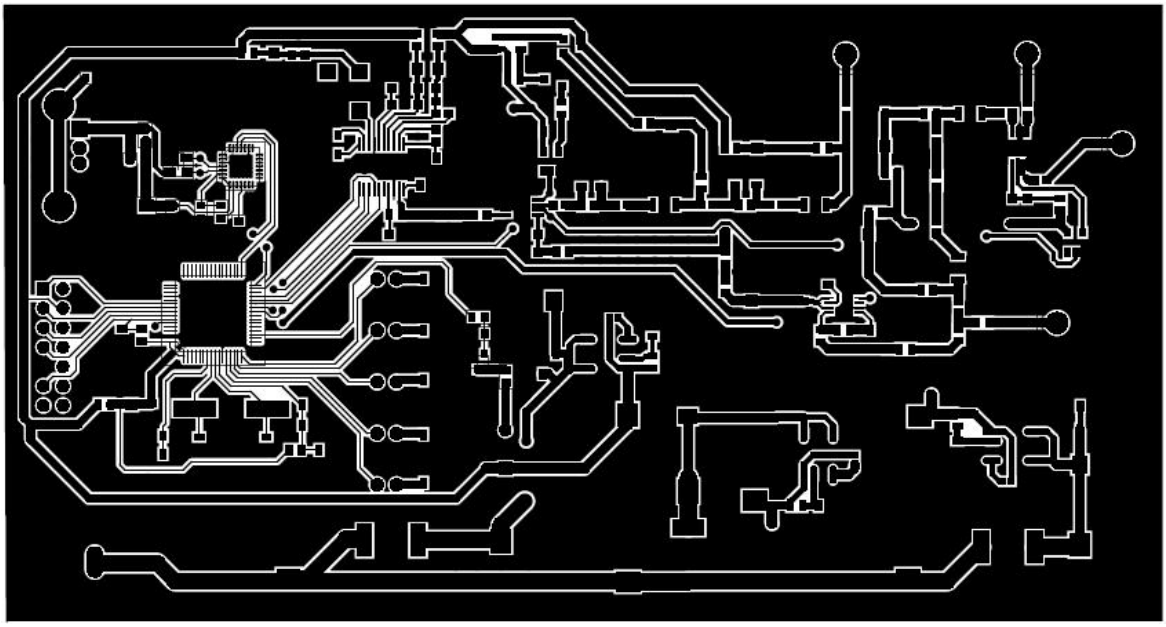

Figure D.1. Top layer.

D.2. Bottom Layer

D.3. Top Silk Screen

D.4. Solder Mask

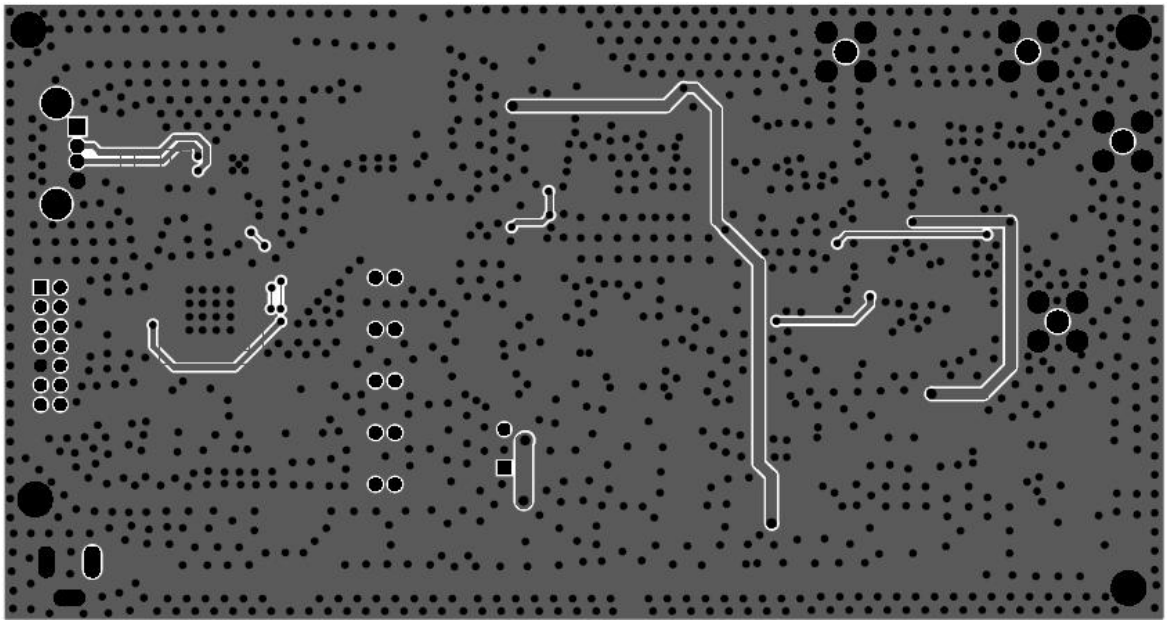

Figure D.2. Bottom layer.

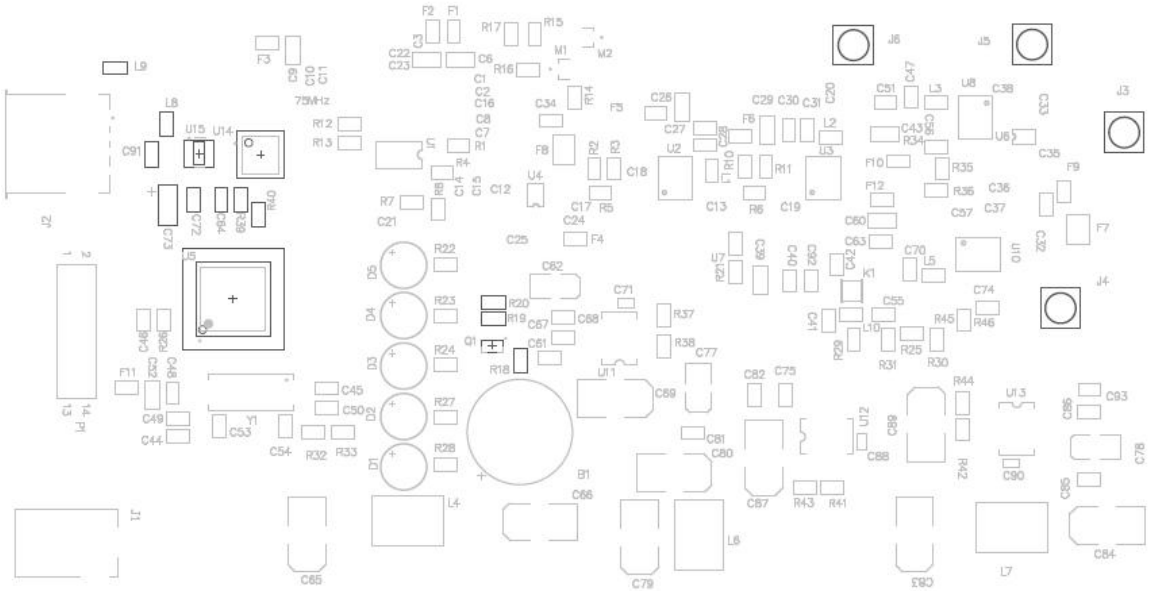

Figure D.3. Top silk screen.

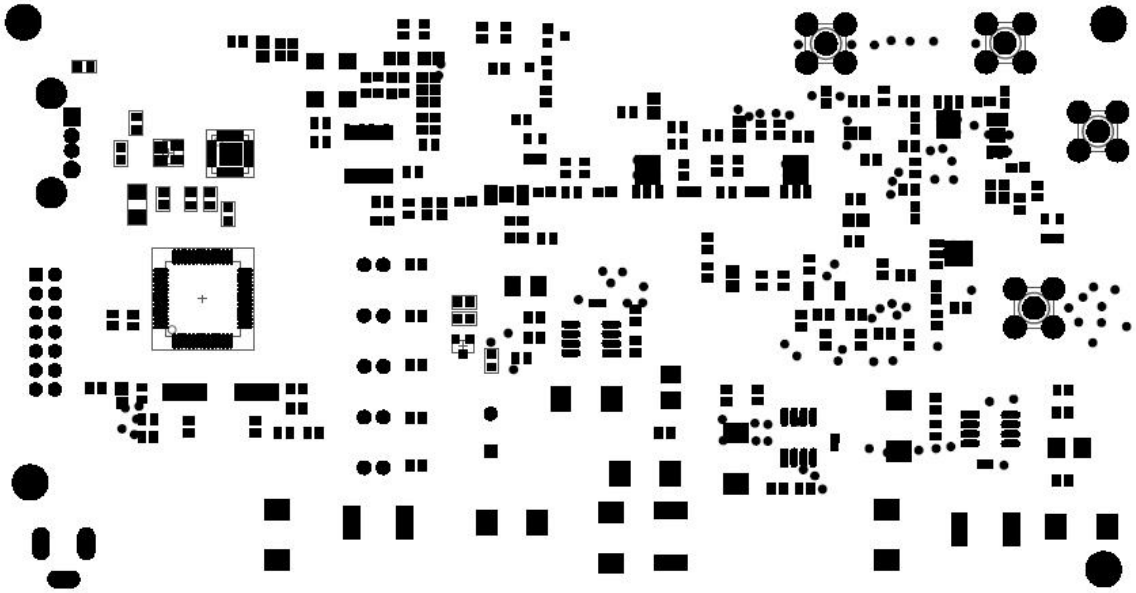

Figure D.4. Top solder mask.

APPENDIX E: MATLAB CODES

E.1. Main Script

```
% Packet: A single one frequency signal
    % Group: Entire sweep signal covering 32 packets

    % Threshold values are chosen considering the distance
       between sniffer
    % and the transmitter: 30cm
    clear
    close all

    samplingFrequency = 50000000; %Hz
    totalNumOfSamples = 1000000;
    upperThres = 0.009;
    lowerThres = -0.005;
    packetPeriod = 64 * samplingFrequency /1000000; % number
       of samples that corresponds to 64 us (time length
       between starts of two consecutive packets)
    % wrt the sampling frequency
    TxSignalLength = 10 * samplingFrequency /1000000; % number
        of samples that corresponds to 10 us (Tx signal length
       ) wrt the sampling frequency
    energyThres = 2; % Has to be crosschecked
    gapBetweenPackets = (54 * 10 ^ (-6)) / (1 /
       samplingFrequency);
% number of samples that corresponds to 54 us (silence between
    two packets) wrt the

% sampling frequency
```

```
tagExistanceThreshold = 2 * 10^(-6); %Threshold damping
   time which is used to decide whether tag exists or not

filePath ='C:\Documents and Settings\TechS6\Desktop\Matlab
   \Oscilloscope outputs\';
fileName ='Sniffer50cm_Tag40cmParallel.csv';
M= csvread(['' filePath fileName ''], 15, 1, [15, 1,
   1000014, 1]);
t = (0:totalNumOfSamples-1)*(1/samplingFrequency);
receivedSignal = M(:,1);

figure
plot(t,receivedSignal)
title('Captured signal')
xlabel('Time (Seconds)')
ylabel('Amplitude (V)')
samplesUnderThres = 0;
groupSearchStartIndexFound = 0;
groupSearchStartIndex = 0;
startOfFirstPacket = 0;
processedPacketIndex = 17;

for i = 1 : totalNumOfSamples
    if ((receivedSignal(i) < upperThres) && (
       receivedSignal(i) > lowerThres))
        receivedSignal(i) = 0;
        if(groupSearchStartIndexFound == 0)
            samplesUnderThres = samplesUnderThres + 1;
            if(samplesUnderThres > gapBetweenPackets * 3)
% Multiplication with 3 is to guarantee that the
   samplesUnderThres is greater than the silence
```

```
% between two consecutive packets
                    groupSearchStartIndex = i;
                    groupSearchStartIndexFound = 1;
                end
            end
        else
            samplesUnderThres = 0;
        end
    end

%     figure
%     plot(t,receivedSignal)
%     title('Captured Signal after Noise Removed')
%     xlabel('Time (Seconds)')
%     ylabel('Amplitude (V)')
%
    for i = groupSearchStartIndex : totalNumOfSamples - 300
        if(receivedSignal(i) > upperThres)
            sum = 0;
            for k = i : i + 300
                sum = sum + abs(receivedSignal(k));
            end
            if(sum > energyThres) % These 300 samples belong
               to a pulse stream
                startOfFirstPacket = i;
                break;
            end
        end
    end
```

```
    processedPacket = receivedSignal((startOfFirstPacket + (
        processedPacketIndex - 1) * packetPeriod): (
        startOfFirstPacket -1 + (processedPacketIndex) *
        packetPeriod));
    t = (0:packetPeriod-1)*(1/samplingFrequency);
    figure
    plot(t,processedPacket)
    title('Single Pulse')
    xlabel('Time (Seconds)')
    ylabel('Amplitude (V)')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%       Quadrature_Demodulator(receivedSignal(startOfFirstPacket
    + processedPacketIndex * packetPeriod: startOfFirstPacket
   + (processedPacketIndex + 1) * packetPeriod), packetPeriod,
    8200000, samplingFrequency);
%       FrequencySpectrum(receivedSignal(startOfFirstPacket +
   processedPacketIndex * packetPeriod: startOfFirstPacket + (
   processedPacketIndex + 1) * packetPeriod), packetPeriod,
   samplingFrequency);
    lowPassed_DemodulatedSignal = Quadrature_Demodulator(
        processedPacket, packetPeriod, 8200000,
        samplingFrequency);
    FrequencySpectrum(processedPacket, packetPeriod,
        samplingFrequency);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if(FindTimeConstant(processedPacket(TxSignalLength : size(
        processedPacket, 1)), samplingFrequency) >
        tagExistanceThreshold)
         msgbox('ETIKET BULUNDU','Sonuc');
```

```
    else
        msgbox('ETIKET YOK','Sonuc');
    end
    dampingDuration=FindTimeConstant(processedPacket(
        TxSignalLength : size(processedPacket, 1)),
        samplingFrequency);
    tagResponse = lowPassed_DemodulatedSignal(TxSignalLength:
        TxSignalLength+dampingDuration*samplingFrequency);
    t = (0:size(tagResponse,2)-1)*(1/samplingFrequency);
    figure
    plot (t,tagResponse)
    title('Tag response')
    xlabel('Time (Seconds)')
    ylabel ('Amplitude (V)')
```

E.2. Quadrature Demodulator

```
function [lowPassed_DemodulatedSignal] =
    Quadrature_Demodulator (signal, signalLength,
    centerFrequency, samplingFrequency)

%       figure
%       plot(signal)
%       title('QDM input')

    t = (0:signalLength-1)*(1/samplingFrequency);

    FrequencySpectrum(signal, signalLength, samplingFrequency)
        ;
    title('Spectrum of the Signal')

    Qmixer = sin(2*pi*centerFrequency*t);
```

```
    Imixer = cos(2*pi*centerFrequency*t);

    Qsignal = zeros(signalLength, 1);
    Isignal = zeros(signalLength, 1);
    signal = transpose(signal);
    Qsignal = Qmixer .* signal;
    Isignal = Imixer .* signal;

    DemodulatedSignal = sqrt(Qsignal.*Qsignal + Isignal.*
        Isignal);
    DemodulatedSignal = transpose(DemodulatedSignal);
    FT_signal = FrequencySpectrum(DemodulatedSignal,
        signalLength, samplingFrequency);
    title('Spectrum of Demodulated Signal')
    lowPassedFTSignal = LowPassFilter(FT_signal, 5000000,
        samplingFrequency);
    title('Low passed filtered frequency spectrum Demodulated
        Signal')
%
%     FT_Isignal = FrequencySpectrum(Isignal, signalLength,
   samplingFrequency);
%I component in Fourier domain.
%     title('Spectrum of I component')
%     FT_Qsignal = FrequencySpectrum(Qsignal, signalLength,
   samplingFrequency);
 %Q component in Fourier domain.
%     title('Spectrum of Q component')
%
%     lowPassed_FT_Isignal = LowPassFilter(FT_Isignal,
   5000000, samplingFrequency);
%Low pass filtered I component in Fourier domain.
```

```
%     title('Low passed filtered frequency spectrum of I
    component')
%

%     lowPassed_Isignal = real(ifft(lowPassed_FT_Isignal));



    if(samplingFrequency == 250000000)
        nfft = 16384;
    end
    if(samplingFrequency == 50000000)
        nfft = 4096;
    end
%     lowPassed_Isignal = lowPassed_Isignal * Maximum(signal,
    signalLength) / Maximum(lowPassed_Isignal, nfft);
%
%     figure
%     plot(lowPassed_Isignal)
%     title('Low passed I component')

    lowPassed_DemodulatedSignal = real(ifft(lowPassedFTSignal)
        );
    lowPassed_DemodulatedSignal = lowPassed_DemodulatedSignal
        * nfft;
%     figure
%     plot(lowPassed_DemodulatedSignal)
%     title('Low passed Demodulated Signal')
%
```

```
%       lowPassed_DemodulatedSignal =
    lowPassed_DemodulatedSignal * Maximum(signal, signalLength)
     / Maximum(lowPassed_DemodulatedSignal, nfft);
     t = (0:nfft-1)*(1/samplingFrequency);
     figure
     plot(t,lowPassed_DemodulatedSignal)
     title('Low passed Demodulated Signal')
     xlabel('Time (Seconds)')
     ylabel('Amplitude (V)')


end
```

E.3. Low Pass Filter

```
function [lowPassedFT] = LowPassFilter(FT, cutoffFrequency,
    samplingFrequency)

     lowPassedFT = zeros(1, size(FT, 1));
     for n = 1 : size(FT, 1)*(cutoffFrequency /
        samplingFrequency)
         lowPassedFT(n) = FT(n);
     end
     f = linspace(0, samplingFrequency/2, size(FT, 1)/2+1); %
        0'dan Fs/2 frekansina kadar esit araliklarla NFFT/2
        adet frekansin genliklerini gosterir
     figure
     plot(f, 2*abs(lowPassedFT(1:size(FT, 1)/2+1)))

end
```

E.4. Frequency Spectrum

```
function [FT] = FrequencySpectrum(signal, length,
    samplingFrequency)

    NFFT = 2^nextpow2(length);
    FT = fft(signal, NFFT) / NFFT;
    f = linspace(0, samplingFrequency/2, NFFT/2+1); % 0'dan Fs
        /2 frekansina kadar esit araliklarla NFFT/2 adet
        frekansin genliklerini gosterir

    figure
    plot(f, abs(FT(1:NFFT/2+1)))
    title('Single-Sided Amplitude Spectrum')
    xlabel('Frequency (Hz)')
    ylabel('Amplitude')

end
```

E.5. Find Time Constant

```
function [dampingDuration] = FindTimeConstant(dampingSignal,
    samplingFrequency)

    % dampingSignal: Islenecek olan paketin 10.
        mikrosaniyesindeki ornegi ile  bir
    % sonraki paketin baslangici arasinda kalan orneklerin
        dizisi. 10.
    % mikrosaniye paketin transmit edildi halinin sonuna denk
        geliyor. Bu
    % andan itibaren EAS etiketinden gelen sonumlenen sinyal
        basliyor.

    figure
```

```
plot(dampingSignal,'r')

dampingDuration = 0; % Sonumlenme sinyaline ait ornek
    sayisini temsil etmekte kullanilacak.
minimumSilence = (4 * 10^(-6)) / (1 / samplingFrequency);
% 4us'ye denk gelecek ornek sayisi: 4us'lik bir sessizlik (0V
    array'i) sonumlenmenin bittigini garanti etmek icin yeterli
    .



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Asagidaki blokta Incelenen paketin sonundaki sonumlenme
    suresi
% hesaplaniyor. dampingSignal icerisinde 0V'luk bir ornege
% rastlandiginda bu ornege ardisik 'minimumSilence' sayida
    0V'luk ornek
% bulunursa o ornek sonumlenmenin sonu olarak kabul
    ediliyor ve
% dampingSignal'in basindan o ornege kadar kac ornek
    olduguna ve
% ornekleme frekansina bakilarak sonumlenme suresi
    hesaplaniyor.

for i = 1 : size(dampingSignal, 1)
    if(dampingSignal(i) == 0) % Bu ornek sonumlenme sonu
        olabilir.
        endFound = 1;
        for k = i : i + minimumSilence % Gercekten
            sonumlenme sonuysa bu ornekten sonraki '
```

```
                minimumSilence' kadar ornegin tamami 0V olmali.
                if(abs(dampingSignal(k)) > 0) % Ornegin
                    sonumlenme sonu olmadigi goruluyor.
                    endFound = 0;
                    break;
                end
            end
            if(endFound == 1) % 0V'luk bu ornegin pesisira
                gelen 'minimumSilence' kadar ornek de 0V oldugu
                icin 'endFound' degiskeni'nin degeri 1 olarak
                kaldi. Dolayisiyla bu ornek sonumlenme sonunu
                temsil ediyor.
                dampingDuration = i * (1 / samplingFrequency);
% Bu ornegin index'i kullanilarak sonumlenme suresi
    hesaplaniyor.
                break;
            end
        end
    end
    tau = dampingDuration / 5; % exp(-t/tau) = exp(-5),
        because exp(-5) can be treated as zero

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




    timeAxis = (1 / samplingFrequency) * linspace(1,
        dampingDuration * samplingFrequency, dampingDuration *
        samplingFrequency);
```

```
    t = (1 : dampingDuration * samplingFrequency) * (1 /
        samplingFrequency);
    figure
    fx = exp(-t / tau);
    plot(timeAxis, fx,'r')
    title('Damping Envelope')

end
```

REFERENCES

1. Armstrong, B. and N. Ahmed, "Detection of Broadband Dispersive Signals", *IEEE International Conference on Systems Engineering*, Vol. 89, No. 2767, pp. 265–268, 1989.

2. Bredow, J., S. P. Gogineni and K. Jezek, "Polarimetric Radars for In-Situ Studies: Design, Calibration and Applications", *Antennas and Propagation Society International Symposium Digest*, Vol. 2, No. 45, pp. 1014–1017, 1989.

3. Ybarra, G. A., G. L. Bilbro, S. M. Wu, S. H. Ardalan, C. P. Hearn and R. T. Neece, "High Resolution Target Range Estimation Using Frequency-Stepped CW Radar", *IEEE International Microwave Symposium Digest*, Vol. 1, pp. 249–252, 1993.

4. Anderson, K. D., "Radar Detection of Low-Altitude Targets in a Maritime Environment", *IEEE Transactions on Antennas and Propagation*, Vol. 43, No. 6, pp. 609–613, 1995.

5. Martinez-Madrid, J. J., J. R. C. Corredera and G. de Miguel-Vela, "High Resolution Imaging Techniques in Step-Frequency Subsurface Radars", *International Geoscience and Remote Sensing Symposium*, Vol. 1, pp. 769–771, 1996.

6. Cattin, V., J. J. Chaillout and R. Blanpain, "Detection and Localisation with a Step Frequency Radar", *Second International Conference on the Detection of Abandoned Land Mines*, pp. 86–90, 1998.

7. Zych, M., D. Gromek and A. Gromek, "Focused Algorithms for Ground Penetrating Radar Imaging", *11th International Radar Symposium*, pp. 1–5, 2010.

8. Wilkinson, A. J., R. T. Lord and M. R. Inggs, "Stepped-Frequency Processing by Reconstruction of Target Reflectivity Spectrum", *Proceedings of the 1998 South African Symposium on Communications and Signal Processing, 1998. COMSIG '98*, pp. 101–104, 1998.

9. Park, J. and C. Nguyen, "A New Milimeter-Wave Step-Frequency Radar Sensor for Distance Measurement", *IEEE Microwave and Wireless Components Letters*, Vol. 12, No. 6, pp. 221–222, 2002.

10. Chakrabarti, S., P. Kannagaratnam and P. Gogineni, "Model-Based Technique for Super Resolution and Enhanced Target Characterization Using a Step-Frequency Radar: A Simulation Study", *International Geoscience and Remote Sensing Symposium*, Vol. 3, pp. 1867–1869, 1996.

11. Yuehua, L., L. Xingguo and W. Min, "An Analysis of Signal of MMW Step Frequency High Resolution Radar with MUSIC Algorithms", *International Conference on Microwave and Millimeter Wave Technology Proceedings, 1998. ICMMT '98*, pp. 443–447, 1998.

12. Chen, J. C., R. E. Hudson and K. Yao, "Maximum-Likelihood Source Localization and Unknown Sensor Location Estimation for Wideband Signals in the Near-Field", *IEEE Transactions on Signal Processing*, Vol. 50, No. 8, pp. 1843–1854, 2002.

13. Kelly, E. J., "An Adaptive Detection Algorithm", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 22, No. 1, pp. 115–127, 1986.

14. Dana, R. A., "Effects of Two-Way Decorrelation on Radar Detection Scintillation", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 31, No. 2, pp. 795–804, 1995.

15. Hua, Y., F. Baqai, Y. Zhu and D. Heilbronn, "Imaging of Point Scatterers from Step-Frequency ISAR Data", *IEEE Transactions on Aerospace and Electronic*

Systems, Vol. 29, No. 1, pp. 195–205, 1993.

16. Gill, G. S., "Step Frequency Waveform Desing and Processing for Detection of Moving Targets in Clutter", *Record of IEEE International Radar Conference*, pp. 573–578, 1995.

17. Pettersson, M. I., "Four-Dimensional Discretization for Detection of Moving Objects in Wide Band SAR", *Proceedings of 4th European Radar Conference, 2007. EuRAD 2007*, pp. 79–82, 2007.

18. Vu, V. T., T. K. Sjogren, M. I. Pettersson, H. J. Zepernick and A. Gustavsson, "Experimental Results on Moving Target Detection by Focusing in UWB Low Frequency SAR", *IET International Conference on Radar Systems*, pp. 1–5, 2007.

19. Haibo, L. and L. Teng, "A Target Velocity Measurement Method in Submilimeter Step Frequency Radar", *IET International Radar Conference*, pp. 1–4, 2009.

20. Perlovsky, L., R. Ilin, R. Deming, R. Linnehan and F. Lin, "Moving Target Detection and Characterization with Circular SAR", *IEEE Radar Conference*, pp. 661–666, 2010.

21. Du, L. and G. Su, "Target Number Detection Based on a Order Choi-Williams Distribution", *Proceedings of Seventh International Symposium on Signal Processing and Its Applications*, Vol. 1, pp. 317–320, 2003.

22. Gini, F., F. Bordoni and A. Farina, "Multiple Radar Targets Detection by Exploiting Induced Amplitude Modulation", *IEEE Transactions on Signal Processing*, Vol. 52, No. 4, pp. 903–913, 2004.

23. Isoda, K. and T. Hara, "Doppler Velocity Measurement Method with a Second-Time-Around Echoes Suppression for Synthetic Bandwidth Radars", *Proceedings of the 8th European Radar Conference*, pp. 134–137, 2011.

24. Karoubi, B., Y. M. Zhu, G. Gimenez and J. Bigun, "Detection of Objects in RF Ultrasonic Images Using 2-D Spatial Phase Techniques", *Proceedings of International Conference on Image Processing*, Vol. 1, No. 2, pp. 641–644, 1996.

25. Li, S., X. Lv, H. Sun, W. Hu and Q. Hu, "Scattering Centers Measurements by a 2-D ESPRIT Type Method", *7th International Symposium on Antennas, Propagation EM Theory, 2006. ISAPE '06*, pp. 1–5, 2006.

26. Jill, G. S. and J. C. Huang, "The Ambiguity Function of the Step Frequency Radar Signal Processor", *Proceedings of CIE International Conference of Radar*, pp. 375–380, 1996.

27. Gillespie, B. W. and L. E. Atlas, "Optimization of Time and Frequency Resolution for Radar Transmitter Identification", *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 3, pp. 1341–1344, 1999.

28. Ozdemir, E., *Super Resolution Spectral Estimation Methods for Buried and Through-The-Wall Object Detection*, Ph.D. Thesis, Middle East Technical University, 2005.

29. Devices, A., *AD9834 - 20 mW Power, 2.3 V to 5.5 V, 75 MHz Complete DDS*, 2011, `http://www.analog.com/static/imported-files/Data_Sheets/AD9834.pdf`, accessed at January 2012.

30. Devices, A., *AD8310 - Fast, Voltage-Out, DC to 440 MHz, 95 dB Logarithmic Amplifier*, 2010, `http://www.analog.com/static/imported-files/Data_Sheets/AD8310.pdf`, accessed at January 2012.

31. Instruments, T., *MSP430f2617 - Mixed Signal Controller*, 2011, `http://www.ti.com/lit/gpn/msp430f2617`, accessed at January 2012.

32. Schillinger, J., *Antenna Matching For The TRF7960 RFID Reader*, 2009, `http://www.ti.com/lit/an/sloa135/sloa135.pdf`, accessed at January 2012.