DESIGN AND IMPLEMENTATION OF A SOFTWARE DEFINED RADIO

RECEIVER FOR AM BAND

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

_____

Kalpesh Anil Shetye

Certificate of Approval:

_____	_____
Bogdan M. Wilamowski Professor Electrical and Computer Engineering	Richard C. Jaeger, Chair Distinguished University Professor Electrical and Computer Engineering
_____	_____
Fa Foster Dai Associate Professor Electrical and Computer Engineering	Ramesh Ramadoss Assistant Professor Electrical and Computer Engineering

_____

Joe F. Pittman
Interim Dean
Graduate School

DESIGN AND IMPLEMENTATION OF A SOFTWARE DEFINED RADIO RECEIVER FOR AM BAND

Kalpesh Anil Shetye

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Masters of Science

Auburn, Alabama
August 4, 2007

DESIGN AND IMPLEMENTATION OF A SOFTWARE DEFINED RADIO

RECEIVER FOR AM BAND

Kalpesh Anil Shetye

_____

Signature of Author

_____

Date of Graduation

VITA

Kalpesh Shetye was born in Mumbai, India, on June 14, 1980. He is the eldest son of Anil and Maya Shetye. He completed his Bachelor of Engineering in Electronics degree from University of Mumbai, India in 2002. After completing his Bachelors he pursued his Masters program at Auburn University in Spring 2003, where he started working on Analog / Mixed-signal circuit design and subsequently Software-Defined Radio concepts. His primary areas of interest are Analog / Mixed-signal circuit design, RF IC design and DSP for Software Defined Radio. He is currently employed with Siemens VDO Automotive, Huntsville as Design Engineer – Electrical R&D.

THESIS ABSTRACT

DESIGN AND IMPLEMENTATION OF A SOFTWARE DEFINED RADIO RECEIVER FOR AM BAND

Kalpesh Shetye

Master of Science, August 4, 2007
(Bachelor of Engineering, University of Mumbai, 2002)

75 Typed Pages

Directed by Richard C. Jaeger

Since the mid-1990s, the radio industry has actively focused on implementing more and more radio functions in the digital domain. This has been furthered by availability of high speed, high performance data converters and faster digital processors. In 1993, Joe Mitola, III coined the term 'Software Radio (SR)' for a radio system that uses DSP primitives to perform signal manipulation instead of the traditional analog hardware. Such a system is more robust, compact, power-efficient and highly reconfigurable. An ideal Software Radio system consists of a transmitting/receiving antenna, high speed data converter and a powerful digital processor. However, the state of current technology is such that this can only be partially achieved. Due to speed and performance limitations of existing data converters and digital processors, it is customary to use an RF front-end between the antenna and the data converter. Such a system is then termed as a Software-Defined Radio (SDR).

This thesis deals with the design and implementation of a low-cost SDR receiver which bandpass samples AM Intermediate Frequency (IF) and demodulates it in real-time using quadrature demodulation. The system uses an AM/FM trainer kit to obtain an AM IF, a high speed PCI-based data acquisition (DAQ) card for analog-to-digital (A/D) conversion, MATLAB to perform signal processing in the digital domain and a sound card to produce the demodulated analog signal. A Graphical User Interface (GUI) is developed which allows the user to start/stop the program, select a suitable bandpass sampling frequency and view the time and frequency domain representation of the demodulated signal. This work also discusses bandpass sampling and quadrature demodulation followed by a rigid mathematical analysis to point out advantages and disadvantages of the two techniques.

ACKNOWLEDGEMENTS

I am grateful to my parents Anil and Maya, my brother Akshay and my close relatives for providing me financial and emotional support to study in United States. I thank my friends Anjani, Gautham, Pradeep, Kashi, Shaman, Santosh and Nitin for providing me accommodation during my weekend trips to Auburn.

My deepest respect and appreciation goes out to my advisor, Dr. Richard C. Jaeger, without whose support this work could not be completed. He strongly encouraged me to accept the job offer at Siemens VDO Automotive and continue studies while working. I greatly acknowledge his technical guidance and financial support which was necessary to complete this work. I am also grateful to Dr. Fa Foster Dai, Dr. Bogdan Wilamowski and Dr. Ramesh Ramadoss for having served on my committee and for their valuable inputs and suggestions. I wish to thank Mr. Joe Haggerty for his advice and help on electronics equipment. Last but not the least, I express my gratitude to Siemens VDO mentor Mr. Paul Evans and manager Mr. Blane McCoy who motivated me to complete my Masters.

I would like to dedicate this thesis to my spiritual guru, Paramahansa Yogananda, whose life and teachings have had a significant impact on me during my stay in United States.

Format of body: Auburn University Graduate School: Guide to preparation and submission of theses and dissertations.

Computer software used: Microsoft Office Professional 2003

MATLAB Release 14

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Prior to the infusion of digital signal processing technology, most of the functions in a radio system were implemented using analog circuitry. This had several limitations. First of all, such a system was not reconfigurable. Any modification was possible only through physical intervention. Secondly, complex communication algorithms were difficult to implement in the analog domain due to the size of the components, associated costs and power consumption. Also, performance of analog radio was dependent on external parameters like noise, temperature, etc. With increase in speed of data converters and signal processors, it became possible to implement analog functions in the digital domain. The ultimate goal was to directly digitize the RF signal at the output of the receiving antenna and implement all receiver functions in either digital hardware or software. This gave birth to the software-defined radio (SDR) concept. An SDR system is a radio communication system which uses software for modulation and demodulation of radio signals [3].

This thesis presents the design of a low-cost SDR receiver which bandpass samples an AM Intermediate Frequency (IF) and demodulates it in digital domain using quadrature demodulation. This work can form the foundation of an undergraduate

wireless education or a graduate wireless research laboratory. The thesis is organized as follows:

i. Chapter 1 is a primer on the SDR concept, its advantages and potential applications. An overview of the SDR system designed and implemented for this project is also presented.

ii. Chapter 2 is a literature review of different RF front-end architectures, sampling techniques and signal processing options.

iii. Chapter 3 explains the system design of the SDR receiver. The components used for building the system are explained in detail.

iv. Chapter 4 deals with the algorithms used in this project and the mathematics behind them. Bandpass sampling and quadrature demodulation are discussed in detail here.

v. Chapter 5 explains the implementation of the algorithms in MATLAB using its Data Acquisition Toolbox. It also explains the Graphical User Interface (GUI) created for the end user.

vi. Chapter 6 presents the results and suggests future improvements.

vii. Appendix contains the entire MATLAB code along with appropriate comments.

1.1 Software Defined Radio (SDR)

As suggested in [4], radio systems can be classified into 5 tiers depending upon their capability and flexibility. Tier 0 includes strictly a Hardware Radio (HR) which can be modified through physical intervention only. All traditional analog radio systems with no software element are included in this group. Tier 1 includes a Software Controlled

Radio (SCR) which has limited functions changeable using software. Tier 2 includes Software Defined Radio (SDR) which uses software for the modulation and demodulation of radio signals. Some RF front-end processing is still necessary in such a system. Tier 3 includes Ideal Software Radio (ISR) which eliminates the RF front-end processing completely. The antenna is directly connected to the data converter in this system. Tier 4 includes Ultimate Software Radio (USR) which is a fully programmable radio which can support broad range of frequencies and multiple air-interfaces.

Figure 1.1.1 illustrates an ISR. Here, the DSP does the modulation and demodulation in addition to baseband signal processing, thus eliminating the need of RF front end. The user can alter the functionality of the radio simply by reprogramming the DSP. However in practicality, it is not possible to attach the antenna directly to the data converter due to a variety of reasons (discussed in Chapter 2). Use of RF front end therefore becomes necessary converting the radio from ISR to SDR.

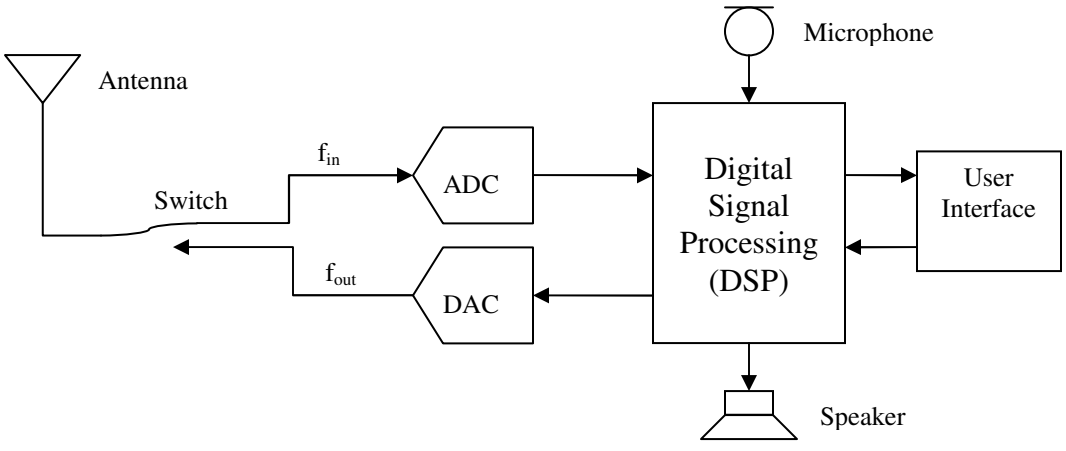

Figure 1.1.1: An Ideal Software Radio (ISR)

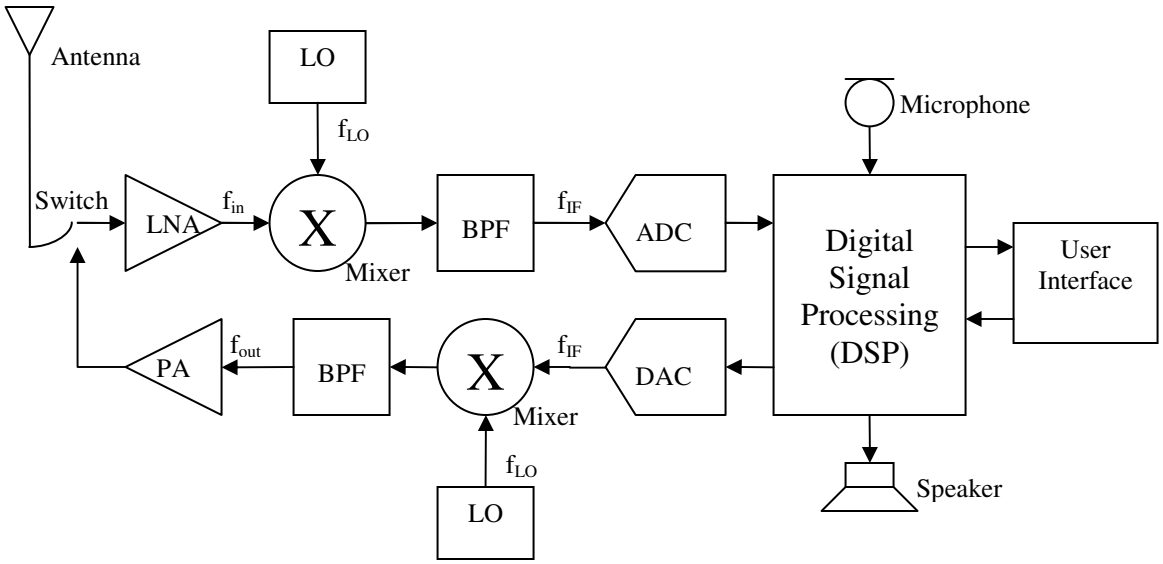

Figure 1.1.2: A Software-Define Radio (SDR)

Figure 1.1.2 illustrates a practical SDR architecture. In the receive path, the antenna signal is amplified by the Low Noise Amplifier (LNA). It is then mixed and bandpass filtered (BPF) to generate the IF signal. This IF signal is then digitized by a high speed ADC. The DSP downconverts the IF signal to baseband and subsequently demodulates it in digital domain. The demodulated signal is played on a speaker.

Likewise, in the transmit path, the DAC outputs the modulated signal at the IF frequency. It is then upconverted and bandpass filtered to generate the RF signal. The RF signal is further amplified by the Power Amplifier (PA) and fed to the antenna.

Radios built using SDR concept have the following advantages:

1) Increased system performance, flexibility and cost efficiency as the digitization is done at an early stage.
2) A standard architecture can be used for a wide range of communication products [4]. Hence, interoperability is possible.

3) Increased adaptability. The radio can be reprogrammed to improve performance or add more functionality.
4) Software modifications can be done at a fraction of the time of hardware modifications. This can drastically reduce time to market and life-cycle costs.

1.2 Applications of SDR

Software defined radios have significant use in military and wireless industry because both of them have a variety of changing radio protocols in real time. One of the first software defined radios was a US military project called SPEAKeasy [3]. The goal of the project was to develop a radio for US military that could operate from 2 MHz to 2 GHz. Its architecture was identical to Fig. 1.1.2. It was one of the first projects to use Field Programmable Gate Arrays (FPGA) for digital signal processing of radio data.

Another project, called Joint Tactical Radio Systems (JTRS), is a US and allied program to make radios which provide flexible and interoperable communications. It is based on the Software Communications Architecture (SCA).

A potential application of SDR is in the automotive industry. Many OEM manufacturers, including Siemens VDO Automotive, are researching the option of eliminating bulky RF tuners in car radio and digitizing entire AM/FM bands. Multiple AM/FM channels can be demodulated simultaneously in digital domain. This will allow playing of one radio channel on the main radio and another on Rear Seat Entertainment (RSE).

In academia, research in SDR field is being in pursued in top universities like Georgia Tech, MIT and UCLA. MIT is investigating the use of SDR in Radio Frequency

Identification (RFID) where devices use various communication protocols to operate on various frequencies. Recently, UCLA introduced a practical SDR receiver which can tune and detect any desired RF signal in the 800MHz to 5GHz band [12]. Key blocks for the receiver are wideband LNA, highly linear low-flicker mixer, wide tuning range synthesizer, and programmable anti-aliasing filters.

SDR has also crept in the amateur radio field. In [13], a PC-based SDR is described that downconverts RF to low-IF in the audio frequency range. It then uses PC sound card to sample and demodulate the signal. The FlexRadio SDR-1000 [14], shown in Fig. 1.2.1, is based on this concept. It can demodulate desired RF signal from 12 kHz to 60 MHz.

Figure 1.2.1: FlexRadio SDR-1000

The basic block diagram of the low-cost SDR receiver designed for this project is shown in Fig. 1.2.2. The AM/FM trainer kit is used to convert the AM signal to amplified AM IF signal. It is then undersampled using the high speed ADC of Measurement Computing's PCI-based DAS4020/12 data acquisition board. Quadrature demodulation is used to demodulate the signal in and play it in real-time on PC speakers. DSP algorithms are written in MATLAB which is a simulation and mathematical software from The Mathworks Inc.

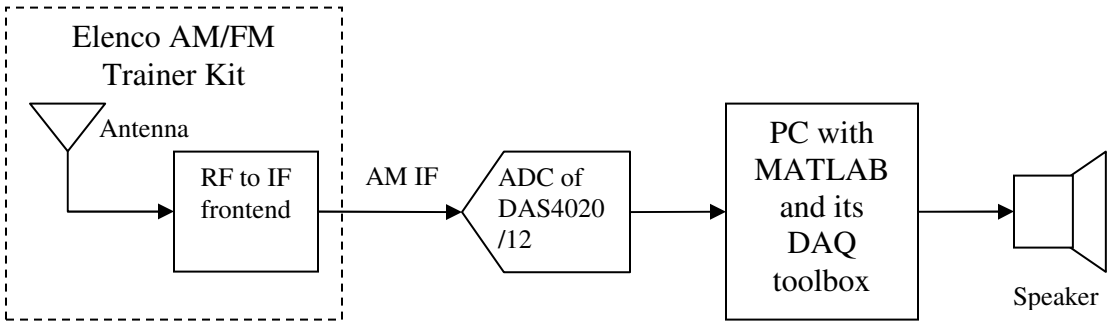

Figure 1.2.2: Software Defined Radio Receiver for AM Band

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

This chapter discusses common RF front-end architectures and uniform sampling techniques used in radio receivers. It also talks about the data conversion challenges for software-defined radio implementation. Finally it concludes with digital signal processing alternatives for SDR.

2.1 RF Front-end Architectures

The primary criteria in selecting any RF front-end architecture are complexity, cost, power distribution and number of external components. There are three RF front-end architectures in popular use today. They are heterodyne (or superhetrodyne), homodyne (or direct conversion or zero-IF) and low-IF (or digital-IF) architecture.

2.1.1 Heterodyne Architecture

In heterodyne architecture, the RF signal is translated to lower IF frequencies in multiple stages by mixing it with a local oscillator signal. Figure 2.1.1 depicts such a design. The RF signal is passed through the BPF and amplified by the LNA. Before the signal is mixed with first local oscillator (f_{LO1}) to generate first IF (f_{IF1}), it is passed through the image reject filter (IRF). The IRF rejects the image frequency located at the sum of the LO and IF frequencies ($f_{LO} + f_{IF1}$). If the image is not rejected then it will fall

directly on the IF after mixing and corrupt the signal information. The channel select filter rejects adjacent channels and improves channel selectivity. The first IF signal is mixed with second local oscillator (f_{LO2}) to obtain the second IF signal (f_{IF2}).

The major disadvantage of heterodyne topology is the number of required components. For example, a two stage heterodyne receiver employs two mixers, two local oscillators, one image reject filter and two channel select filters. The choice of IF also depends on trade-offs among three parameters: the amount of image noise present, the spacing between the desired band and the image and the loss of the image-reject filter [5]. A low IF allows great suppression of nearby interferers whereas a high IF leads to better image rejection. Thus heterodyne topology exhibits tradeoff between selectivity and sensitivity. Another problem which exits is the half IF effect due to the second order non-linearity in the RF and IF paths. Assume that there is a strong interferer at half of the IF from the desired band towards the LO ($(f_{in} + f_{LO})/2$) and it undergoes second order distortion in the RF path. If the LO signal contains its second harmonic then the interferer falls on the IF ($|2f_{LO} - (f_{in} - f_{LO})| = f_{IF}$) after mixing. Another possibility is that the interferer gets translated to $(f_{in} - f_{LO})/2 = f_{IF}/2$. If the IF path exhibits second order non-linearity then the interferer will still fall on the IF.

Most AM/FM radios use heterodyne architecture with two stages of downconversion. The second IF frequency for AM and FM band is 455 kHz and 10.1 MHz respectively.

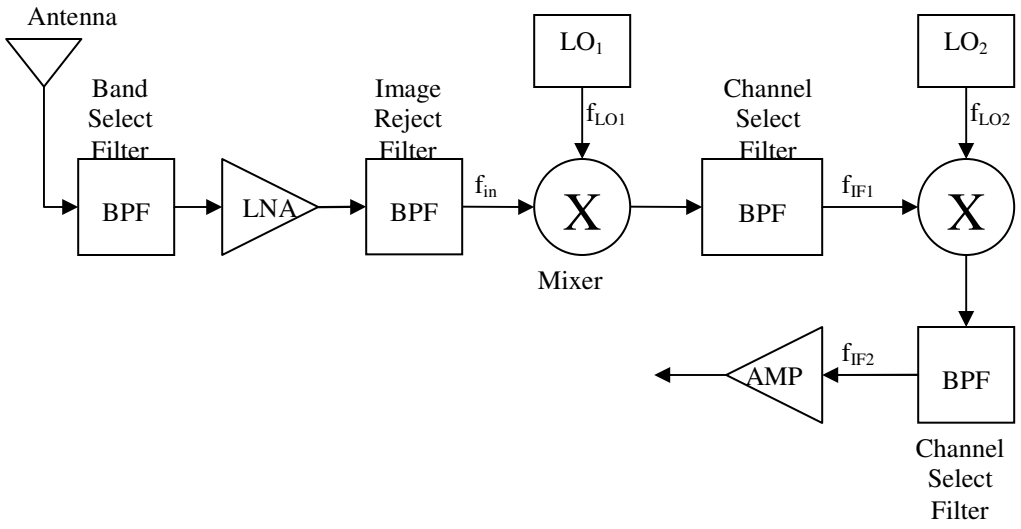

Figure 2.1.1: Heterodyne (or superheterodyne) architecture

2.1.2 Homodyne Architecture

In homodyne architecture, the RF signal is directly translated to baseband by mixing it with LO signal whose frequency is same as the carrier frequency. This is illustrated in Fig. 2.1.2 (a). For double-sided AM signal, this technique works well because it overlaps positive and negative parts of the input spectrum. However for FM and phase-modulated signals, quadrature downconversion (In-phase 'I' and Quadrature 'Q') is necessary to avoid loss of information. This is shown in Fig. 2.1.2 (b).

The simplicity of the homodyne topology has its own advantages and disadvantages. No image reject filter is required. Also, the LPF and amplifier operate at lower frequencies and their monolithic integration is possible. The channel filtering can be done in digital domain provided the ADC has high linearity and sufficient spurious-free dynamic range (SFDR). A serious disadvantage of homodyne topology is self-mixing of the LO signal. The isolation between the LO and inputs of the LNA and mixer

is not infinite. There is a finite amount of feedthrough from the LO to these inputs called as LO leakage [5]. The leakage signal mixes with the LO signal to produce a DC offset. This can cause corruption of the baseband signal and can saturate the following stages of a receiver. A similar effect is seen when a strong interferer leaks from the LNA or mixer input to the LO and mixes with itself. In phase and frequency modulation schemes, where quadrature downconversion is employed, amplitude and phase mismatch between I and Q can corrupt the downconverted signal. Also if the architecture is implemented using MOS devices, then flicker noise (*1/f* noise) can be a potential source of corruption.

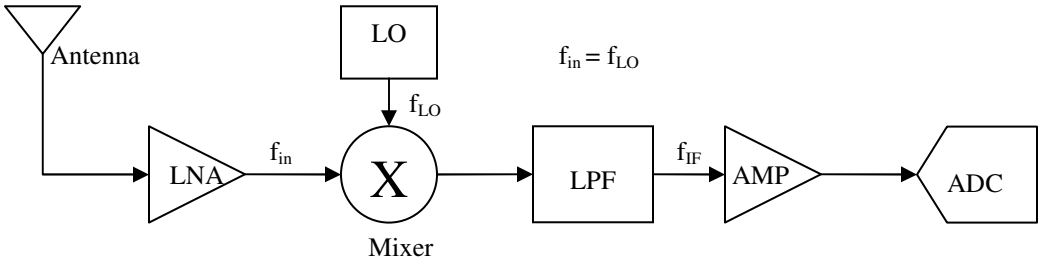

Figure 2.1.2: (a) Simple homodyne (or direct conversion or zero-IF) architecture

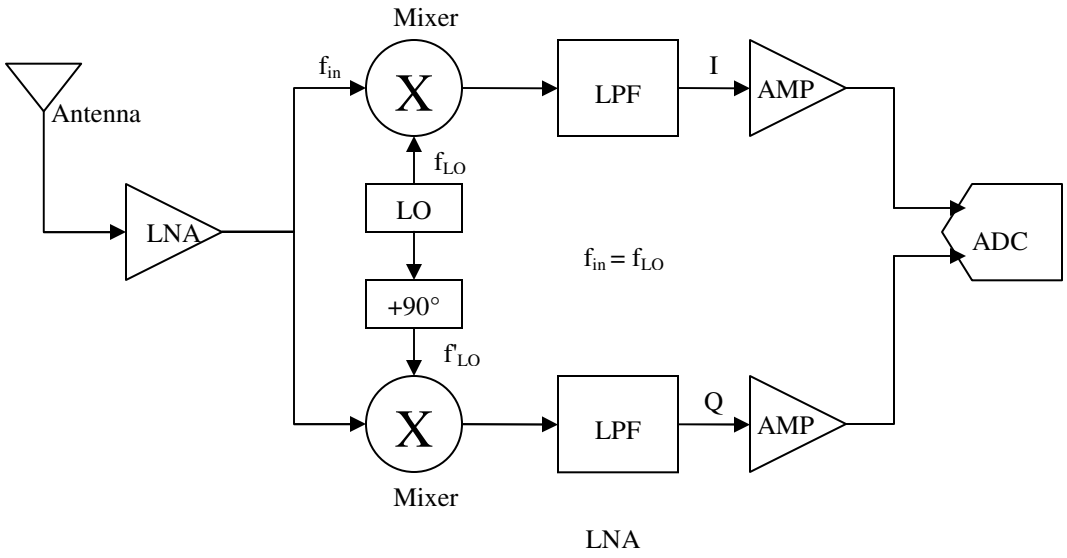

Figure 2.1.2: (b) Homodyne architecture with quadrature downmixing

2.1.3 Low-IF Architecture

The low-IF architecture is hybrid of heterodyne and homodyne architectures. Here, the RF signal is converted into a low-IF signal and then digitized. Downconversion from low-IF to baseband takes place in the digital domain. This is shown in Fig. 2.1.3. Unlike the homodyne topology, problem of self-mixing does not exist. Also I and Q mismatch can be minimized due to digitization of the IF signal. Mixing and filtering can be done efficiently in the digital domain. However, the ADC requirements are more stringent. The ADC should have sufficient input analog bandwidth. The dynamic range of the ADC must be wide enough to accommodate variations in the signal level due to path loss and multipath fading. Also, the SFDR should be sufficiently high to keep the baseband signal from getting corrupted. The BPF filter before the ADC should have sharp cut-off frequencies to attenuate out-of-band signals. Any signal with frequency

more than half the sampling frequency will fold over in the digital domain. This can cause baseband corruption.

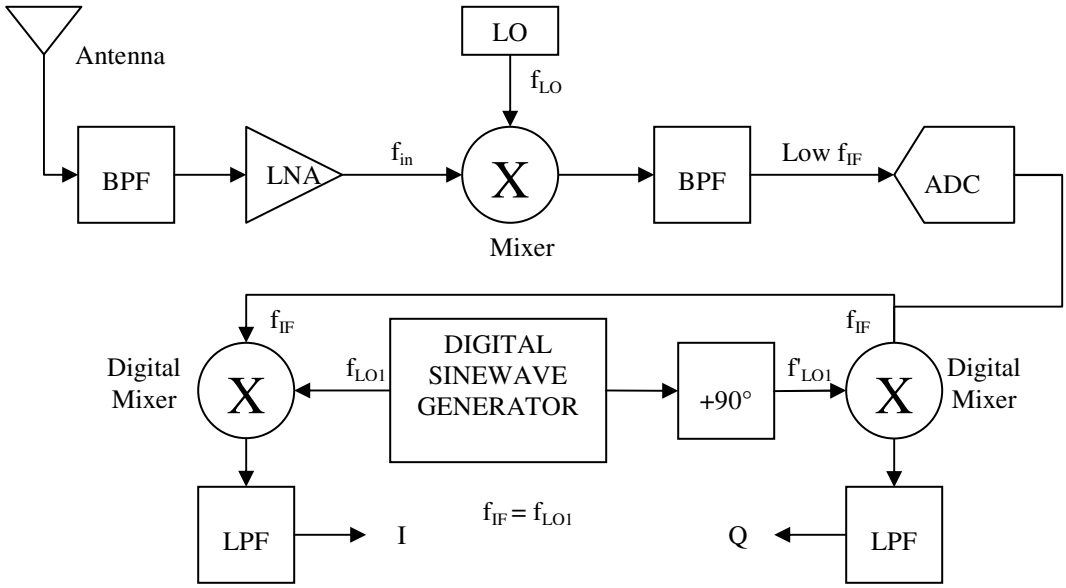

Figure 2.1.3: Low-IF (or digital-IF) architecture

Because of the advantages listed above, the low-IF architecture is the most commonly used architecture in SDRs like in [13], [14] and in this project.

2.2 Sampling Techniques

The sampling process is very important in radio receivers using digitization at the RF or IF. The content of the sampled signal is mainly dependent upon the sampling rate and the minimum and maximum frequency components of the analog input signal [7]. When a continuous time signal is uniformly sampled, the spectrum of the original signal

F (f) is repeated at integral multiples of the sampling frequency, f_s. In other words, *F (f)* becomes periodic. This is shown in Fig. 2.2.1 (a), (b).

The four commonly used uniform sampling techniques are – Nyquist sampling, over-sampling, quadrature sampling and bandpass sampling.

2.2.1 Nyquist sampling

The Nyquist sampling theorem says that exact reconstruction of a continuous time analog signal from its samples is possible if the signal is bandlimited and the sampling frequency is greater than twice the signal bandwidth. The sampling frequency at twice the signal bandwidth is called as Nyquist frequency or Critical frequency. If f_{max} is the maximum frequency component of an analog signal then spectrum of the signal sampled at the Nyquist frequency is shown in Fig. 2.2.1 (b).

If the signal is sampled at less than the Nyquist frequency (called undersampling), the spectral replicas overlap causing aliasing. The sampled signal gets corrupted and cannot be exactly recovered. Figure 2.2.1 (d) depicts aliasing due to undersampling. In order to avoid aliasing, an anti-aliasing filter is used before the ADC. The cut-off frequency of the anti-aliasing filter is one half of the sampling frequency. Nyquist sampling demands an extremely sharp cut-off anti-aliasing filter. Unfortunately, practical realizable filters cannot provide this type of 'brickwall' response.

Even in Nyquist sampling, if an undesired (i.e. out-of-band) signal is present along with the analog signal, it folds over and causes spectral overlap thus corrupting the signal of interest. This is shown in Figure 2.2.2. The anti-aliasing filter serves the purpose of attenuating the undesired signal too.

2.2.2 Oversampling

In oversampling, the signal is sampled at much more than twice the Nyquist rate. As depicted in Fig. 2.2.1 (c), the main advantage of this technique is that the spectral replicas of the sampled signal are spaced further apart from each. This relaxes the steep cut-off frequency requirements of the anti-aliasing filter.

2.2.3 Quadrature sampling

As explained in section 2.1.2, the homodyne architecture for phase and frequency modulated systems use quadrature downconversion to generate the 'I' and 'Q' quadrature components. These are complex valued signals and contain twice the information as the real valued signal. Hence, they can be sampled at one half the sampling rate of the real valued signal. This type of sampling is called as quadrature sampling. The only disadvantage is that ADC needs to have two input channels for digitizing the two components.

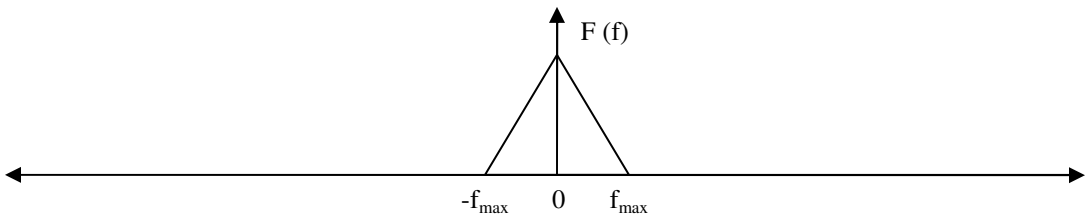

Figure 2.2.1: (a) Spectrum of a bandlimited continuous time analog signal

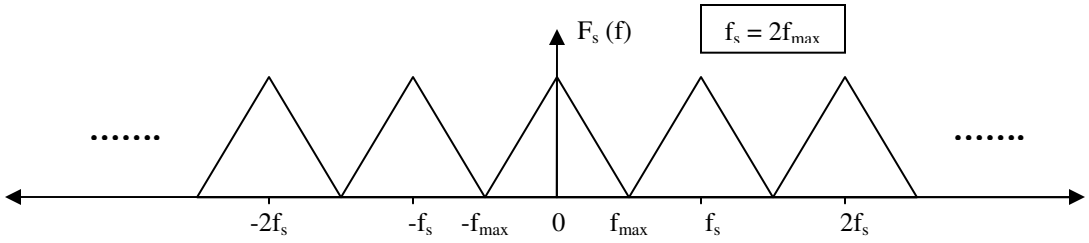

Figure 2.2.1: (b) Spectrum of the signal sampled at $f_s = 2f_{max}$

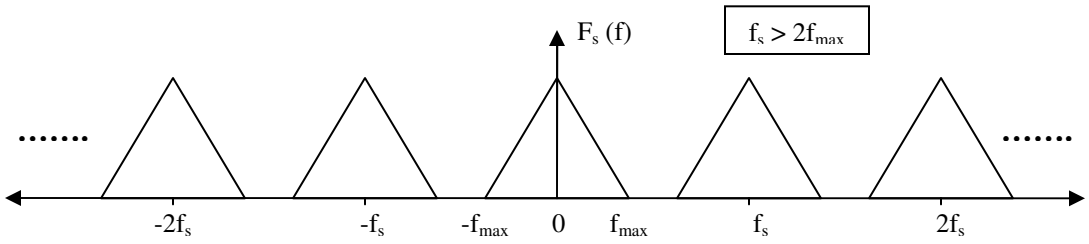

Figure 2.2.1: (c) Spectrum of the signal sampled at $f_s > 2f_{max}$

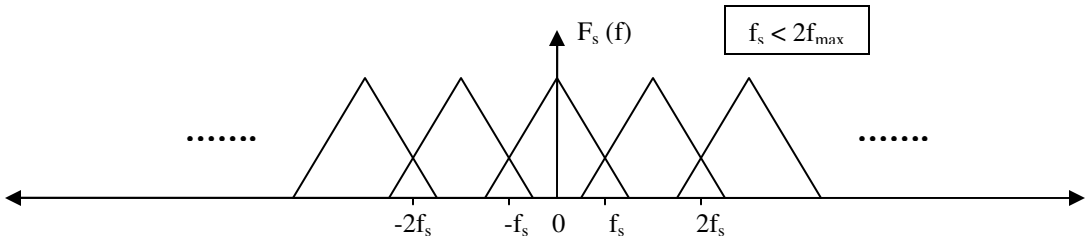

Figure 2.2.1: (d) Spectrum of the signal sampled at $f_s < 2f_{max}$

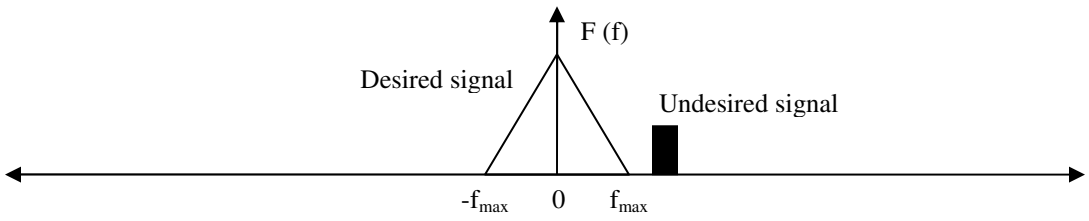

Figure 2.2.2: (a) Spectrum of a bandlimited analog signal with undesired component

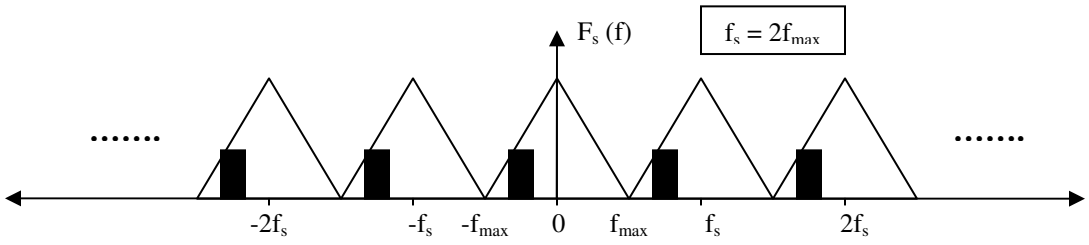

Figure 2.2.2: (b) Spectrum of the signal sampled at $f_s = 2f_{max}$

2.2.4 Bandpass Sampling

RF signals are typically bandpass signals. The information bandwidth of an RF signal is much less than it's RF or IF frequency. Bandpass sampling is the technique of undersampling a modulated signal to achieve frequency translation by intentional aliasing [6]. Here, the sampling frequency is based on the information bandwidth of the RF signal and not on the carrier or IF. Radio receivers that digitize at RF or IF usually use bandpass sampling. The concept is graphically depicted in Fig. 2.2.3. The mathematical relationship between the sampling frequency f_s and the translated RF or IF frequency $f_{IF,trans}$ is explained in detail in Chapter 5. Since aliasing takes place, it is necessary to make sure that no portion of the information bandwidth of the signal folds on top if itself, creating interference.

For bandpass sampling to work effectively, a very steep roll-off bandpass filter is required to attenuate undesired signals outside the band of interest. Another severe limitation is that the ADC should be able to effectively operate on the highest frequency component in the RF signal.

Like most other SDR projects, this project also uses bandpass sampling to sample the AM IF signal.

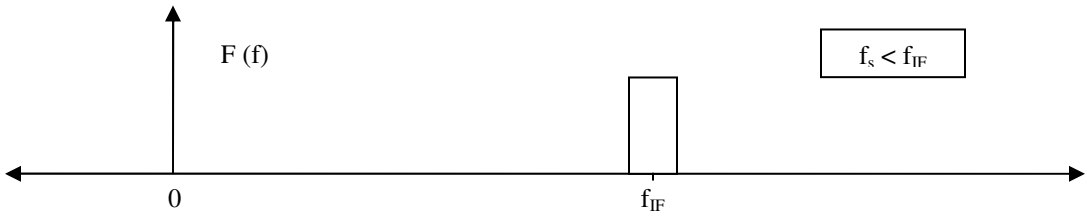

Figure 2.2.3: (a) Spectrum of a bandpass signal centered at IF

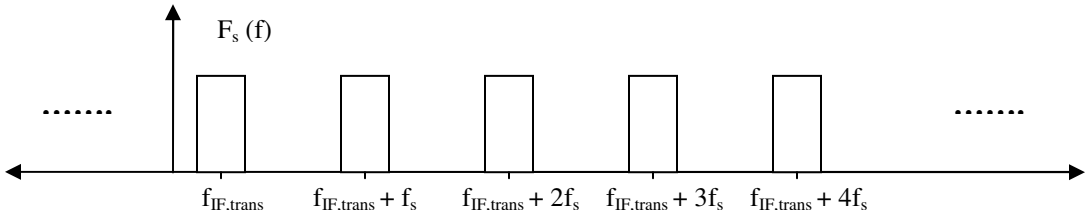

Figure 2.2.3: (b) Spectrum of the signal bandpass sampled at $f_s < f_{IF}$

2.3 Data Conversion Challenges

The critical limiting factor in software radio implementation is the sluggish ADC technology. The antenna cannot be hooked directly to the ADC because it doesn't have sufficient analog input bandwidth and dynamic range to digitize the RF signal directly.

Another important limitation is the power consumption of the ADC. A common Figure of Merit (*FoM*) used for ADCs is:

$$FoM = \frac{2^{2 \bullet ENOB} \bullet BW}{P_{ADC}} \tag{2.3.1}$$

where ENOB = Effective Number of Bits

BW = Full–power Analog Input Bandwidth of the ADC

P_{ADC} = Power Consumption of the ADC

ENOB is directly proportional to the signal-to-noise and distortion ratio (SINAD) of the ADC. Full-power analog input bandwidth is the range from DC to the frequency where, for a full-scale input, the amplitude of the output of the ADC falls to 3 dB below the maximum output level [7]. Equation (2.3.1) indicates that for a given *FoM*, the power consumption of the ADC increases as ENOB and BW increase.

There is a strong trade-off between ADC sampling frequency and its performance. Normally, as the sampling frequency of the ADC increases its performance decreases. In radio receivers using IF or RF digitization, the ADC should have high linearity, signal-to-noise ratio (SNR) and spurious free dynamic range (SFDR). SFDR is defined as the ratio of the signal power to the peak power of the largest spurious product. Whereas SNR indicates ADC's sensitivity to small signals, SFDR is a measure of the ADC's capability to reject undesired interferers. SNR is inversely proportional to the aperture jitter and sampling frequency of the ADC. Aperture jitter is the variation in time of the exact sampling instant.

In comparison, DAC technology is not much of a limiting factor in software radio development. The operating frequencies of the current DACs surpass the ability of current lower-power signal processors.

2.4 Digital Signal Processing Alternatives

The alternatives available for digital signal processing are - Application Specific Integrated Circuits (ASIC), Digital Signal Processors (DSP) and field programmable gate arrays (FPGA).

ASICs are integrated circuits customized to accomplish specific tasks at high performance levels. They are unrivaled in speed, power efficiency and computational density. Because of their high design and production cost, they are usually used in high volume designs. The major disadvantage is that they are not reconfigurable. They can be used for implementing limited to standard static functions. Because of these limitations, their use in SDRs is limited.

A DSP is a specialized microprocessor designed specifically for digital signal processing, generally in real-time computing. It has an optimized signal processing instruction set and can be programmed using high level programming languages like C and C++. On a performance matrix, DSPs fall in between ASICs and FPGAs. They have moderate costs but very short times to market. They are the backbone of most SDR systems today. Many leading semiconductor companies are currently developing SDR specific DSPs. Texas Instruments (TI) is one of them.

An FPGA is a semiconductor device containing programmable logic components and programmable interconnects. The functionality of basic logic gates (AND, OR, XOR,

NOT) or more complex combinational functions such as decoders and simple math functions can be duplicated by programming logic components of the FPGA. Most FPGAs today include memory blocks too. Different logic components are connected together by a hierarchy of programmable interconnects. Hardware descriptive languages like VHDL or VERILOG are used to program FPGAs. They are generally slower than the ASICs and more power hungry. Many SDRs today use FPGAs for signal processing.

CHAPTER 3

SYSTEM DESIGN

This chapter gives a brief system level overview of the SDR receiver. The four major components that make the system are: AM/FM Trainer kit, PCI-DAS4020/12 DAQ card, MATLAB with DAQ toolbox and PC with soundcard.

3.1 AM/FM Trainer Kit

This trainer kit, manufactured by Elenco Electronics, Inc., is a low cost receiver kit used in undergraduate laboratories for demonstrating principles of communication. It is a superheterodyne receiver of the standard AM and FM frequencies. Figure 3.1.1 (a) shows the assembled AM/FM Trainer kit and Fig. 3.1.1 (b) shows its schematic [8].

Figure 3.1.1 (a): Assembled AM/FM Trainer kit

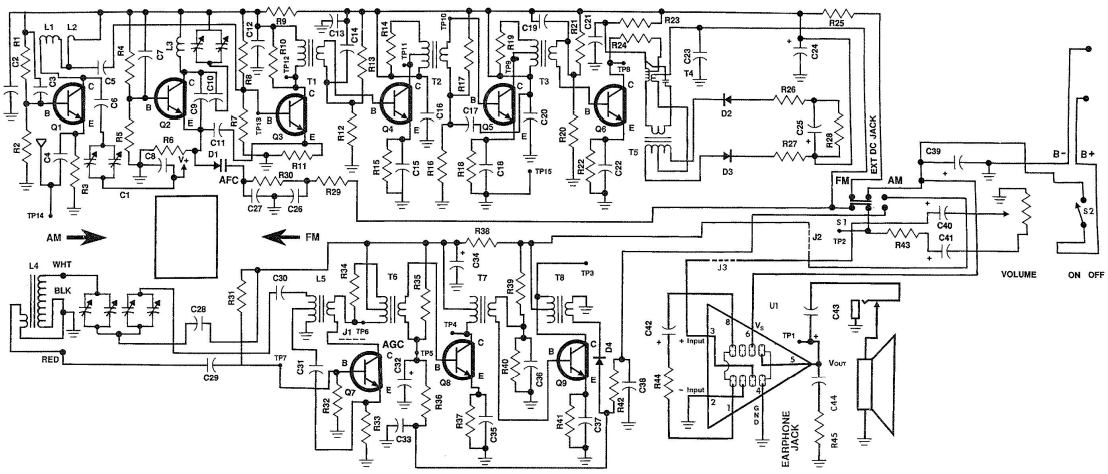

Figure 3.1.1 (b): Schematic of AM/FM Trainer kit

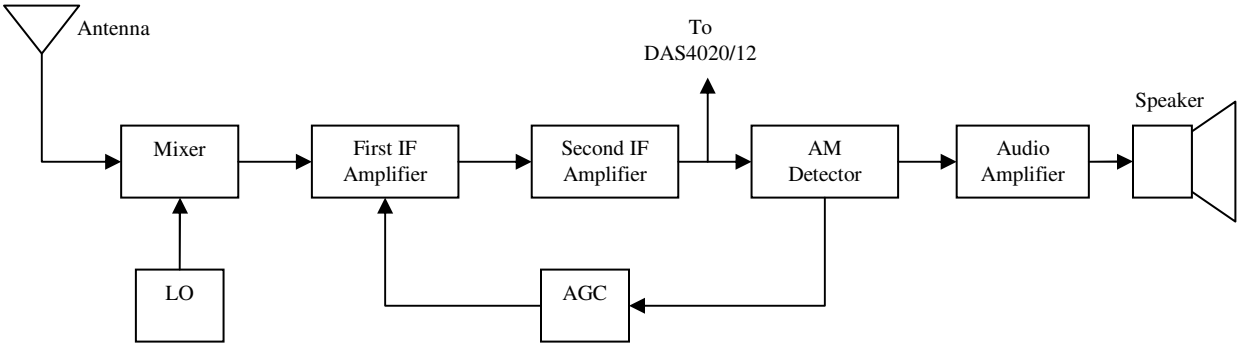

Figure 3.1.2: Block diagram of AM section of the kit

Figure 3.1.2 shows the block diagram of the AM section of the receiver. The antenna signal is fed to the mixer which downconverts the RF signal to an IF of 455 kHz. This is accomplished by heterodyning the RF signal with the Local Oscillator (LO) signal. The weak IF signal from the mixer is amplified by the first IF amplifier which is tuned to 455 kHz. The first IF amplifier has a variable gain which depends upon the

voltage of the AGC (Automatic Gain Control) stage. The AGC stage feeds back a DC voltage to the first AM IF amplifier in order to maintain a near constant level of audio at the detector. The second IF amplifier is also tuned to 455 kHz and has a fixed gain of about 50. It selectively amplifies the IF signal and feeds it to the AM Detector. The AM Detector converts the IF signal to a low level audio signal. The Audio Amplifier stage increases the power of the demodulated audio signal received from the AM Detector to a power level capable of driving the speaker.

For this project, the IF signal is taped out at the output of the second IF amplifier. The gain is set so that the IF signal at the output of second IF amplifier is in the range of +/- 1V.

3.2 DAS4020/12 PCI-based DAQ Card

The AM IF signal from the AM/FM trainer kit is converted into digital domain by Measurement Computing (MMC)'s DAS4020/12 PCI-based DAQ card. Figure 3.2.1 (a) and Figure 3.2.1 (b) depict the card and its block diagram respectively [9].

The PCI-DAS4020/12 is a high speed, analog data acquisition board for PCI bus computers. Its features are:

1. Wide analog BW - 20MHz total throughput rate
2. 12-bit A/D resolution
3. 4 analog input channels
4. Software selectable input ranges
5. One A/D Converter per Channel
6. Dual 12-bit D/A Converter

7. Fully Plug-and-Play
8. Fully Autocalibrating
9. Data acquisition through MATLAB, LabVIEW, Visual Studio.net

The main advantage of using this card is its seamless operation with standard engineering softwares like MATLAB and LabVIEW. Most undergraduate / graduate schools use these softwares in their laboratories. Table 3.2.1 lists some electrical specifications that are of relevance to this project.

Figure 3.2.1 (a): MCC's PCI-DAS4020/12 DAQ card

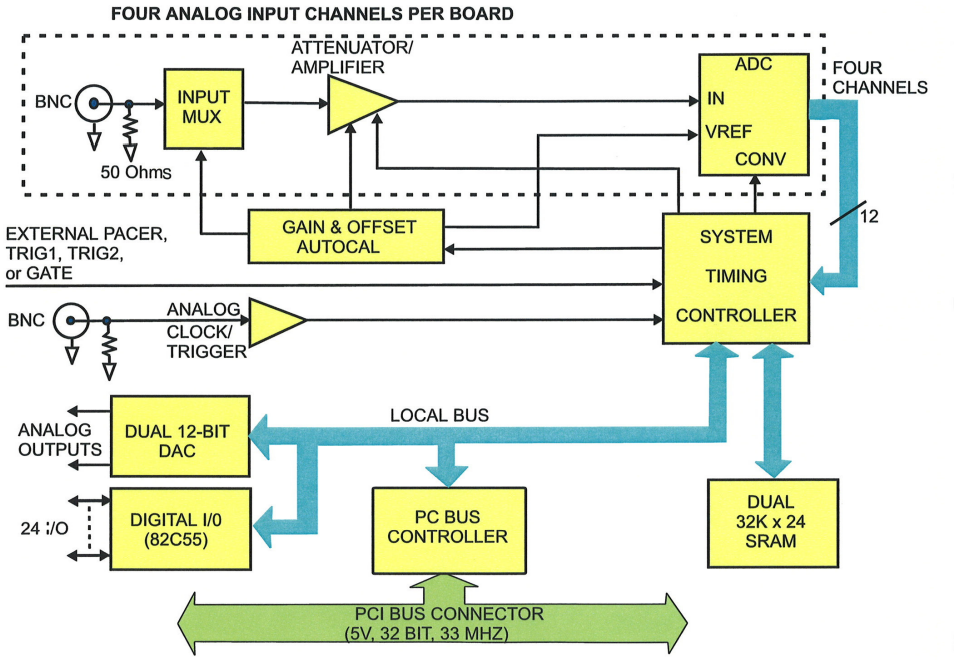

Figure 3.2.2 (b): Block diagram of PCI-DAS4020/12

Sr No	Parameter	Value
1	Minimum sampling frequency	1 kHz
2	Input programmable range	±5 V, ±1 V (software configurable)
3	Input impedance	1.5MOhm (default), 50Ohm
4	Typical Accuracy	±3.0 LSB error (either range)
5	SNR (Signal-to-Noise Ratio)	66.6dB
6	SFDR (Spurious Free Dynamic Range)	80dB
7	THD (Total Harmonic Distortion)	80dB
8	ENOB (Effective Number Of Bits)	10

Table 3.2.1: Electrical specifications of the DAQ card

3.3 MATLAB with Data Acquisition Toolbox (DAQ)

MathWorks' MATLAB is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation [10]. It has command line, scripting, modular, and graphical programming modes. In this project modular and graphical programming is employed. Add-on toolboxes like DAQ Toolbox, Signal Processing Toolbox, Communications Toolbox, etc extend the MATLAB environment to solve problems pertaining to the areas of signal/image processing, communications and control design.

For this project we use MATLAB's DAQ Toolbox to control and communicate with the DAQ card and PC soundcard. The DAQ Toolbox provides a complete set of tools for analog input, analog output and digital I/O from a variety of PC-compatible data acquisition hardware including those from Measurement Computing [11]. The toolbox allows configuring external hardware devices, reading data into MATLAB for immediate analysis, and sending out data. Together, MATLAB and DAQ Toolbox offer a single, integrated environment to support the entire data acquisition and analysis process.

MATLAB's Graphical User Interface Development Environment (GUIDE) is used to build the Graphical User Interface (GUI). GUIDE provides a set of tools that simplify the process of laying out and programming GUIs.

3.4 PC with Speakers

A PC is required to run MATLAB to acquire data, perform signal processing and output the processed data to the speakers through the soundcard. For this work, the PC

used is a standard 2.4 GHz Pentium desktop with 512 MB RAM, standard soundcard and options from Dell Computer Corporation.

CHAPTER 4

DSP ALGORITHMS

This chapter discusses the DSP algorithms used to sample and demodulate the AM IF signal. Bandpass sampling and quadrature demodulation are discussed along with mathematical analysis.

4.1 Bandpass Sampling

Chapter 2, discussed different sampling techniques that can be used for directly digitizing RF / IF signal. The usual method of sampling at twice the Nyquist rate is not practically feasible due to system limitations. The MCC DAQ card has a very wide bandwidth and can easily sample AM RF or IF signal at more than twice the Nyquist rate. For example, in the AM IF case this sampling rate could 1 MHz. But the amount of data that MATLAB would have to handle and process will put it out of sync with the sound card. In other words, a real time system implementation will not be possible.

Data per sample = 1.5 Bytes (12 bits)

Samples / second = 1 M

Total data / second = 1.5 Bytes * 1 M = 1.5 MB

Soundcard Output rate = 8 kS / second

This means that MATLAB + DAQ will have to process 188 kB data in less than 125μs and output it to the sound card. Given the current CPU and MATLAB speed, this is not feasible. Hence bandpass sampling is used in this project.

Bandpass sampling is the technique of undersampling a modulated signal to achieve frequency translation by intentional aliasing. As stated in [6], the mathematical relationship describing the translation of the actual IF f_{IF} frequency and translated IF frequency $f_{IF,trans}$ is:

$$fix\left(\frac{f_{IF}}{\frac{f_s}{2}}\right) = even, f_{IF,trans} = rem(f_{IF}, f_s) \\ = odd, f_{IF,trans} = f_s - rem(f_{IF}, f_s) \tag{4.1.1}$$

Here, *fix (a)* is the truncated portion of argument *a* and *rem (a, b)* is the remainder after dividsion of *a* by *b*. Associated with this translated IF are the corresponding modulation sidelobes that contain information bandwidth of interest. It is important to make sure that no portion of the information bandwidth of the signal folds on top if itself, creating interference. Hence the following two constraints should be met.

$$f_{IF,trans} > \frac{BW}{2} \tag{4.1.2}$$

$$f_{IF,trans} < \frac{f_s}{2} - \frac{BW}{2} \tag{4.1.3}$$

For AM frequencies, f_{IF} = 455 kHz and BW = 10 kHz. Based on equation (4.1.1), Appendix A.1 lists the translated AM IF frequencies for the sampling frequencies from 1

kHz to 100 kHz. It also specifies whether constraints (4.1.2) and (4.1.3) are satisfied or not. The lower limit of 1 kHz is set by the minimum sampling rate of the DAQ card. The higher limit of 100 kHz is decided by maximum processing speed that would make the system real time. It is dependent on DSP algorithm and processor speed.

Table 4.1.1 lists the useful bandpass sampling frequencies in the 1 kHz - 100 kHz range and their corresponding translated frequencies for 455 kHz AM IF. These frequencies are used in the GUI of SDR. User can select any of these sampling frequencies to demodulate the incoming signal.

Actual IF, f_{IF} = 455 kHz BW = 10 kHz	
Useful Bandpass Sampling Frequency, f_s (kHz)	**Translated IF frequency, $f_{IF,trans}$ (kHz)**
28	7
29	9
31	10
32	7
33	7
37	11
39	13
42	7
44	15
47	15
49	14
52	13
55	15
56	7
58	9
59	17
62	21
63	14

64	7
66	7
67	14
68	21
69	28
71	29
72	23
73	17
74	11
77	7
78	13
79	19
80	25
81	31
84	35
85	30
86	25
87	20
88	15
89	10
93	10
94	15
95	20
96	25
97	30
98	35
99	40

Table 4.1.1: Useful bandpass sampling frequencies and the corresponding translated frequencies for 455 kHz AM IF

4.2 Quadrature Demodulation

In this project, the digitized AM IF signal is demodulated using quadrature demodulation. Quadrature demodulation has some interesting properties when used for

AM demodulation. To appreciate these properties, it is necessary to understand the scheme mathematically.

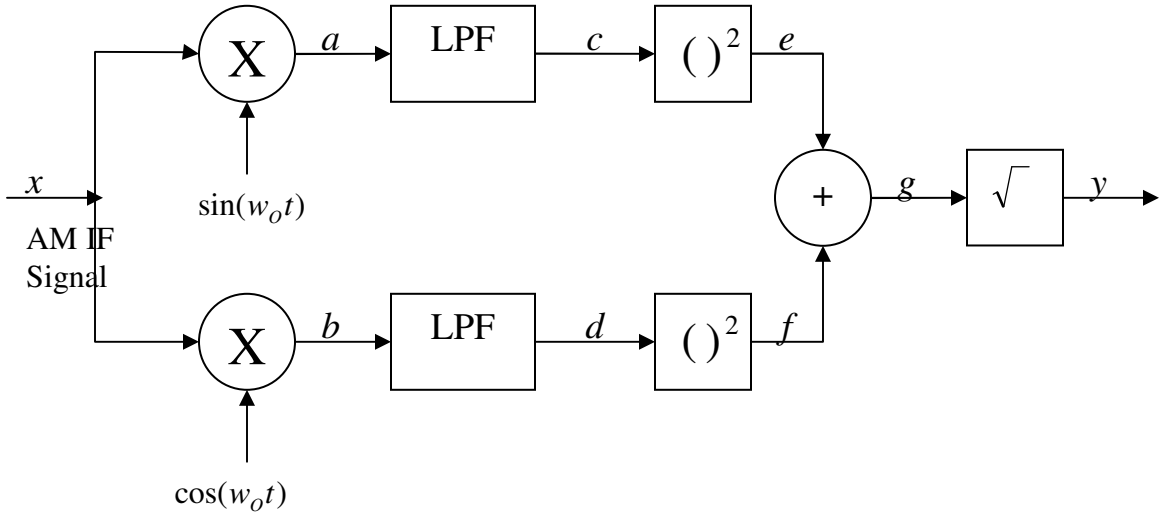

Figure 4.2.1: Quadrature demodulation architecture for AM band

Figure 4.2.1 depicts the quadrature demodulation scheme. The AM IF signal is mixed with the local oscillator to directly convert to baseband. The output is passed through a low-pass filter (LPF) to reject all the high frequency components. It is then squared and fed to the adder. Likewise the AM IF signal is also mixed with the quadrature component of the local oscillator. The output is then passed through LPF, squared and fed to the adder. The adder outputs the summation of the two inputs which is then square-rooted to produce the demodulated signal. The reason for squaring, adding and then finally square-rooting will become clear soon.

For the sake of argument, consider that the AM IF signal and local oscillator signal are continuous and have unit amplitude. The input AM IF signal can then be ideally represented as:

$$x = \cos(w_{IF}t + \Phi) + \frac{1}{2}\cos[(w_{IF} + w_m)t + \Phi] + \frac{1}{2}\cos[(w_{IF} - w_m)t + \Phi] \qquad (4.2.1)$$

where w_{IF} = AM carrier frequency

w_m = modulating signal

Φ = phase difference between the carrier frequency and the local oscillator frequency

This signal mixes with the local oscillator signal to give,

$$\begin{aligned} a &= \cos(w_{IF}t + \Phi)\sin(\mathrm{w_O}t) + \frac{1}{2}\cos[(w_{IF} + w_m)t + \Phi]\sin(\mathrm{w_O}t) + \frac{1}{2}\cos[(w_{IF} - w_m)t + \Phi]\sin(\mathrm{w_O}t) \\ &= \frac{1}{2}\sin[(w_{IF} + w_O)t + \Phi] - \frac{1}{2}\sin[(w_{IF} - w_O)t + \Phi] \\ &\quad + \frac{1}{4}\sin[(w_{IF} + w_O + w_m)t + \Phi] - \frac{1}{4}\sin[(w_{IF} - w_O + w_m)t + \Phi] \\ &\quad + \frac{1}{4}\sin[(w_{IF} + w_O - w_m)t + \Phi] - \frac{1}{4}\sin[(w_{IF} - w_O - w_m)t + \Phi] \end{aligned} \qquad (4.2.2)$$

Similarly, at point *b* we have,

$$\begin{aligned} b &= \cos(w_{IF}t + \Phi)\cos(\mathrm{w_O}t) + \frac{1}{2}\cos[(w_{IF} + w_m)t + \Phi]\cos(\mathrm{w_O}t) + \frac{1}{2}\cos[(w_{IF} - w_m)t + \Phi]\cos(\mathrm{w_O}t) \\ &= \frac{1}{2}\cos[(w_{IF} - w_O)t + \Phi] - \frac{1}{2}\cos[(w_{IF} + w_O)t + \Phi] \\ &\quad + \frac{1}{4}\cos[(w_{IF} - w_O + w_m)t + \Phi] + \frac{1}{4}\cos[(w_{IF} + w_O + w_m)t + \Phi] \\ &\quad + \frac{1}{4}\cos[(w_{IF} - w_O - w_m)t + \Phi] + \frac{1}{4}\cos[(w_{IF} + w_O - w_m)t + \Phi] \end{aligned} \qquad (4.2.3)$$

The output of the mixer is passed through the LPF whose cut-off frequency is at least ($|w_{IF} - w_O| + w_m$). Thus, at point *c* we have,

$$c = LPF(a) = -\frac{1}{2}\sin[(w_{IF} - w_O)t + \Phi] - \frac{1}{4}\sin[(w_{IF} - w_O + w_m)t + \Phi] - \frac{1}{4}\sin[(w_{IF} - w_O - w_m)t + \Phi]$$
$$= -\frac{1}{2}\sin[(w_{IF} - w_O)t + \Phi]\ [1 + \cos(w_m t)] \qquad (4.2.4)$$

Similarly, at point *d* we have,

$$d = LPF(b) = \frac{1}{2}\cos[(w_{IF} - w_O)t + \Phi] + \frac{1}{4}\cos[(w_{IF} - w_O + w_m)t + \Phi] + \frac{1}{4}\cos[(w_{IF} - w_O - w_m)t + \Phi]$$
$$= \frac{1}{2}\cos[(w_{IF} - w_O)t + \Phi]\ [1 + \cos(w_m t)] \qquad (4.2.5)$$

Squaring and adding we get, at point *g*,

$$g = e + f = \frac{1}{4}\ [1 + \cos(w_m t)]^2 \qquad (4.2.6)$$

Taking square-root results in demodulated AM signal with dc offset which can be easily removed,

$$y = \sqrt{g} = \frac{1}{2}\ [1 + \cos(w_m t)] \qquad (4.2.7)$$

An identical mathematical analysis proves that this demodulation scheme works for single sideband (SSB) as well as double sideband suppressed carrier (DSB-SC) transmissions.

From the above analysis, one can conclude that so long as ($|w_{IF} - w_O| + w_m$) is passed by the LPF, the signal can be demodulated using quadrature demodulation. When this scheme is implemented in digital domain, the LO is an accurate digitally generated sine wave. For AM signal, maximum f_m is 5 kHz. If the LPF cut-off is at 10 kHz, then so

long as the offset between the IF and local oscillator frequency is less than 5 kHz, signal can be successfully demodulated. In other words, the AM IF at input of the ADC need not be exactly 455 kHz. This is a very useful property of quadrature demodulation as it relaxes the tight requirements on the LO in the RF front-end.

Also, if the above architecture is implemented in digital domain, self-mixing can be avoided and I / Q mismatch can be minimized.

CHAPTER 5

MATLAB IMPLEMENTATION

This chapter deals with the MATLAB implementation of the SDR. It explains the MATLAB code used to initialize and run the DAQ card, digitally process the acquired data using the DSP algorithms discussed earlier and output the processed data through the soundcard. There is also a brief explanation of creating GUI using GUIDE. Only relevant MATLAB code is shown here in italics. The entire MATLAB code is included in the Appendix A.2.

5.1 DAQ Devices Hardware Setup

As discussed in Chapter 3, the DAQ card and soundcard can be configured using MATLAB's DAQ toolbox. Before doing that, it is necessary to reset all the data acquisition hardware present.

daqreset;

Since the system has to operate in real time, both the DAQ devices need to be initialized and they should work in tandem. Data acquisition objects for these devices are created by issuing the following commands.

ai=analoginput('mcc',2);

ao=analogoutput('winsound');

First command creates an analog data input object called 'ai' that communicates with card #2 from *mcc*. *mcc* is the hardware vendor that MATLAB has assigned for Measurement Computing boards. *ai* configures and controls various parameters of the DAQ card. Likewise, the second command creates an analog data output object called *ao* that communicates with the sound card.

Since DAQ card has four input channels, it is necessary to tell MATLAB which ones to use for acquisition. In this project, Channel 2 is used to acquire data. Similarly, for soundcard Channel 1 is used.

addchannel(ai,2);

addchannel(ao,1);

Next, the input range of the DAQ card is set to $\pm$ 1V using the following command.

ai.Channel.InputRange=[-1 1];

The input and output sampling rates are set by using the following functions.

set(ai,'SampleRate',fs);

set(ao,'SampleRate',fs_out);

Here *fs* and *fs_out* are MATLAB variables which are initialized to 80k and 8k respectively. The GUI allows the user to change the value of *fs* but not the value of *fs_out*.

The type of trigger for the data acquisition objects is decided by the *TriggerType* property. When set to *Manual*, the trigger occurs immediately after the trigger function is issued.

set([ai ao],'TriggerType','Manual');

The number of input samples to be acquired per trigger is set by using the command:

set(ai,'SamplesPerTrigger',inf);

Since the *SamplesPerTrigger* is set to infinity, the DAQ card acquires samples for 1 second and transfers them from its hardware FIFO to PC memory. This is repeated infinitely until the device is stopped.

For soundcard, the *SamplesOutputFcn* property decides which function to call after outputting # *Output_samples* samples. When *SamplesOutputFcnCount* equals *Output_samples*, function *qmoredatanew* is called with *hObject* passed as a parameter. *hObject* is handle to the figure of the GUI.

set(ao,'SamplesOutputFcn',{'qmoredatanew', hObject})

set(ao,'SamplesOutputFcnCount',Output_samples);

The card is set to Direct Memory Access (DMA) transfer mode. This allows the DAQ card to access system memory independently of the CPU. The CPU can therefore concentrate on DSP related tasks. The size of the contiguous memory allocated for DMA transfer is decided by the software provided by MCC. This allocation is performed during the PC bootup sequence.

set(ai,'TransferMode','DMA');

After setting all the parameters, the DAQ devices can be started and triggered to start acquiring data and logging it to memory.

start([ai ao]);

trigger([ai ao]);

The start command will inform the devices to acquire data. DAQ card's internal clock will start. The trigger command will start the process of data acquisition.

The data logged into memory is retrieved by the DAQ Toolbox using *getdata()* function. It returns data and absolute time at which each sample was taken in a matrix format. The processed data is written to sound card using *putdata()* function.

```
[y t]=getdata(ai,fs);
putdata(ao,y)
```

The data acquisition can be halted by using the *stop()* function.

```
stop([ai ao])
```

5.2 Finding Translated IF frequency

Though the translated IF frequency, $f_{IF,trans}$, can be known from Table 4.1.1, it can also be determined from the Fast Fourier Transform (FFT) of the bandpass sampled signal. The peak of the FFT will occur at the translated IF frequency, $f_{IF,trans}$. The following code takes the FFT of the sampled signal, finds the peak of the FFT and the corresponding frequency associated with the peak.

```
fft_y=fft(y);                          % Find FFT of the bandpass sampled signal
[m,imax]=max(abs(fft_y(1:end/2)));     % Index of max peak
freq_vec=fs*(1:length(y))/length(y);   % Generate frequency vector
freq_carrier=freq_vec(imax);           % Find IF frequency
```

5.3 Demodulation

As discussed in Chapter 4, Quadrature demodulation scheme is used in this project. First the in-phase and out-phase components of the local oscillator are created to downconvert the translated IF to baseband.

*fo=sin(2*pi*freq_carrier.*t); % In-phase component*

*fo_90=cos(2*pi*freq_carrier.*t); % Out-phase component*

The input sampled signal is averaged out to filter the dc component and then normalized.

y=y-mean(y);

y=y/max(abs(y));

The normalized signal is then mixed with the in-phase and out-phase component of local oscillator and the product is passed through a low pass filter (LPF). The LPF is a 50^{th} order FIR filter with a linear phase and cut-off frequency, f_c, of 10 kHz.

b1=fir1(50,10e3/fs);

The magnitude and phase response of the filter for a sampling frequency of 80 kHz is shown in Fig. 5.3.1.

*x1=filter(b1,1,fo.*y); % Multiplication with in-phase and subsequent LPF*

*x2=filter(b1,1,fo_90.*y); % Multiplication with out-phase and subsequent LPF*

The two outputs are then squared, summed up and square rooted to get the demodulated output.

x=sqrt(x1.^2+x2.^2);

5.4 Downsampling and Normalization

The demodulated output is downsampled from the bandpass sampling rate to the output sample rate. It is then averaged out and normalized to remove the dc component.

z=x(1:(fs/fs_out):length(x));

z=z-mean(z);

z=z/max(abs(z));

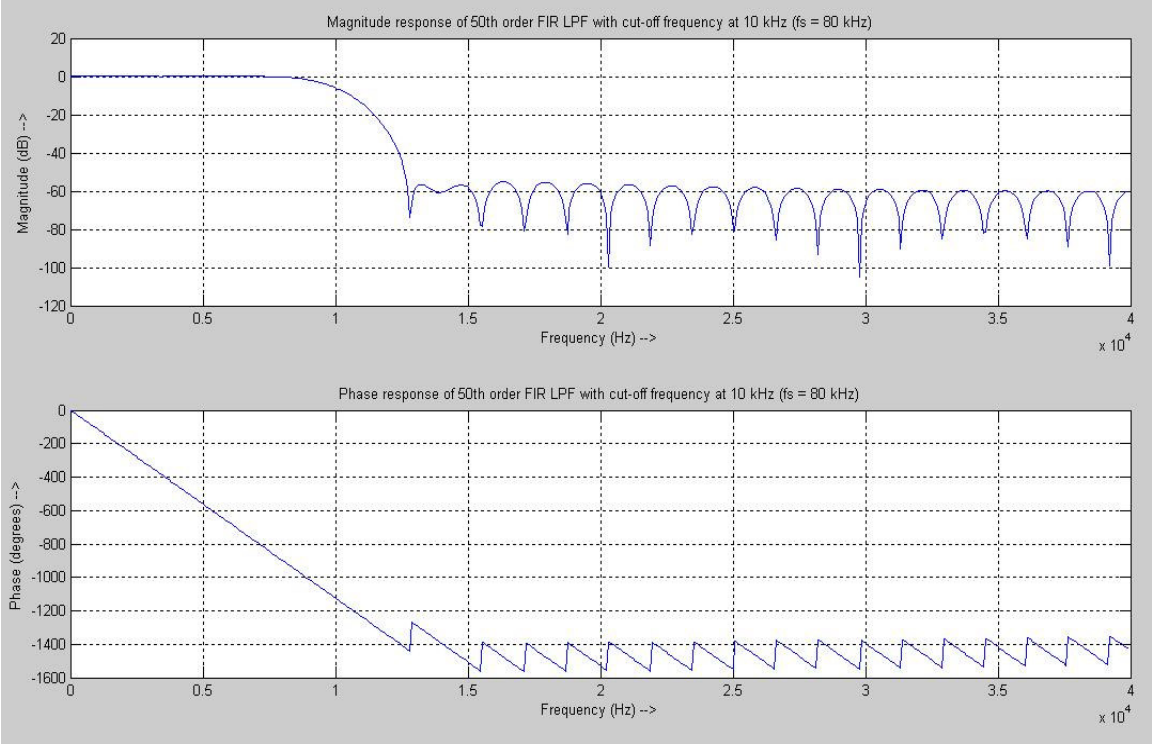

Figure 5.3.1: Magnitude and Phase response of 50th order FIR filter with f_c = 5 kHz

5.5 Graphical User Interface (GUI)

The GUI for SDR is shown in Fig. 5.51. The pop menu at the left-hand side allows the user to select one of the appropriate bandpass sampling frequencies listed in table 4.1.1. If nothing is selected then sampling is done at the default rate of 80 kS/s. After selecting the frequency, the user has to press the ON/OFF toggle button to start the DAQ devices. The time domain representation of the demodulated signal is displayed in the top plot. The X-axis range is for 1 second. The bottom plot shows the frequency domain representation. The Y-axis range is from -4 kHz to 4 kHz as the demodulated data is sent to the soundcard at the rate of 8 kHz. To stop the devices, the user has to

depress the ON/OFF button. The following GUIDE components are used to build the GUI:

1. Popup Menu
2. Axes
3. Toggle Button
4. Static Text

Their details can be found in [10] and are not discussed here.

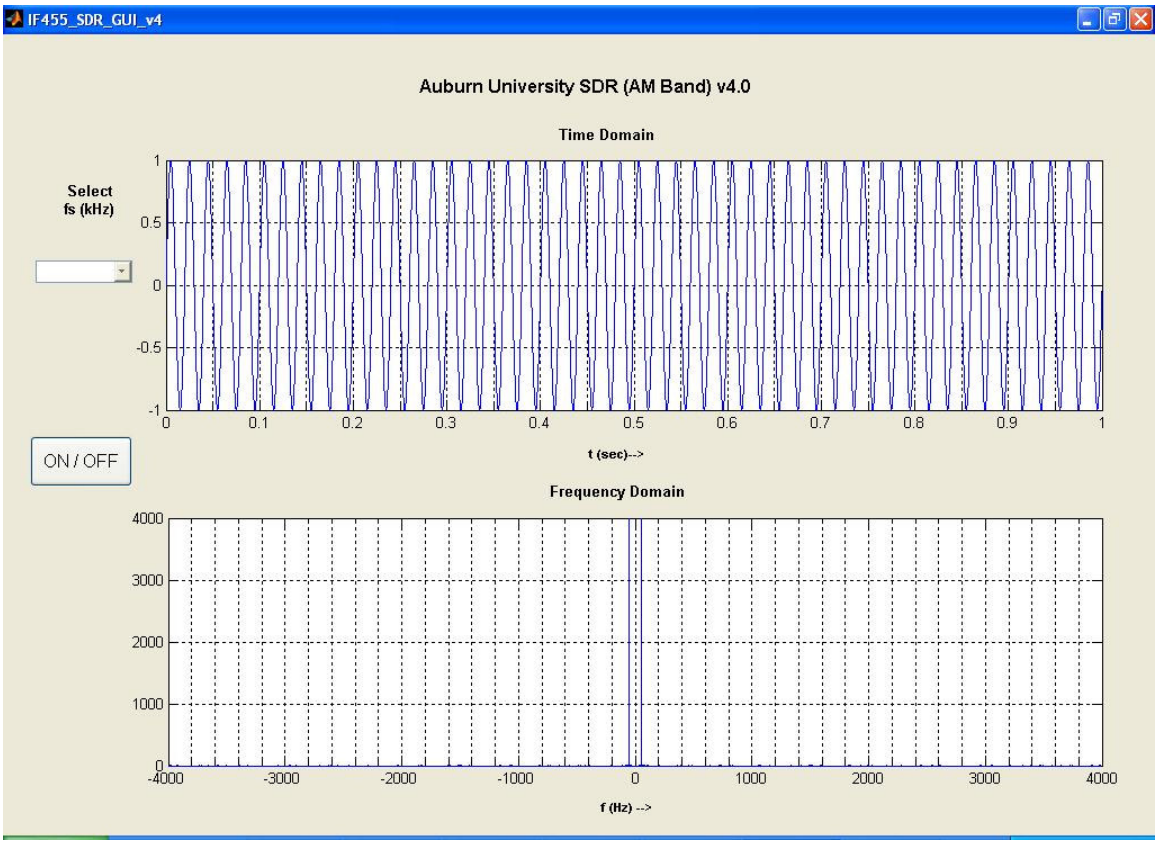

Figure 5.5.1: SDR GUI

CHAPTER 6

RESULTS AND FUTURE IMPROVEMENTS

The results of the project are presented in this chapter. Qualitative aspect of the SDR performance is also discussed. Though this project is fully functional, by no way it is a complete one. There is plenty of room for improvement. Continuous improvement is needed in the areas of RF downconversion, sampling and efficient DSP algorithms.

6.1 Results

The AU SDR (AM Band) v4.0 is used to demodulate WAUD 1230 AM which is Auburn's local station. Figure 6.1.1 shows the GUI display for a sampling frequency of 80 kHz.

The demodulated signal appears a little noisy as can also be seen from its frequency spectrum. This is because the signal reception is not very good in the laboratory where this test was run. Also the receiver is a low-cost radio and hence has a relatively poor performance for weak signals. There is no noise reducing DSP algorithm implemented in this project.

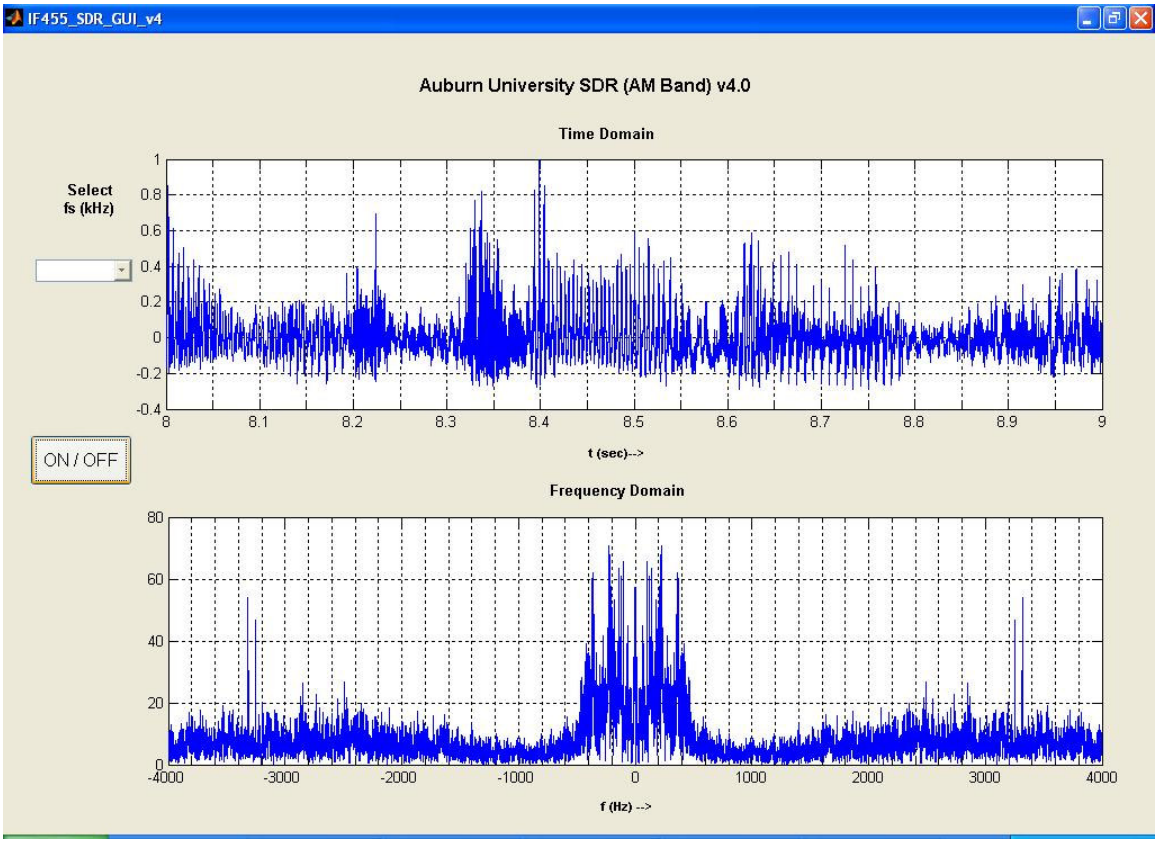

Figure 6.1.1: Demodulated WAUD 1230 AM station with f_s = 80 kHz

Figure 6.1.2 shows the demodulated WAUD 1230 AM station for a sampling frequency of 32 kHz. Here it can also be seen that that there is a peak at about 4 kHz which causes a whistling sound in audio output. The source of this peak is still unknown but it is a matter of further investigation.

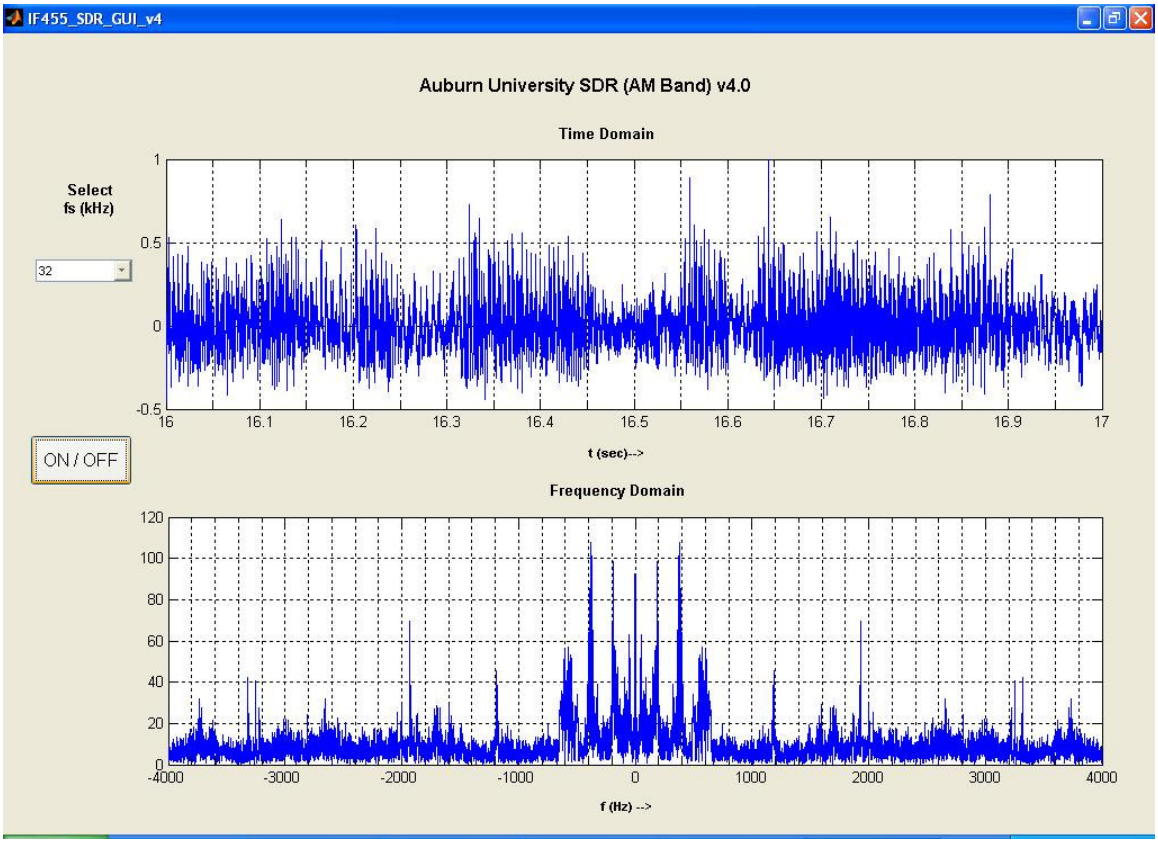

Figure 6.1.2: Demodulated WAUD 1230 AM station with f_s = 32 kHz

6.2 Future Improvements

Instead of converting RF to IF, the whole AM band could be bandpass sampled. This will allow demodulation of multiple AM channels at the same time. As per the equations given in Chapter 4, the minimum bandpass sampling frequency for AM band (530 kHz - 1710 kHz) would be 3421 kHz (assuming that the sampling can be adjusted in steps of 1 kHz). However this will require an amplifier with a sharp bandpass response to reject all out-of-band signals. This amplifier will be placed between the antenna and the DAQ card.

The project could be extended to include other modulation schemes as well like FM, SW, CB, etc. Quadrature demodulation can be used for FM if two analog input channels are available. Noise canceling / squelch algorithms can also be implemented to further improve signal to noise ratio.

MATLAB is a powerful computing tool but it is very resource hungry. It is not as efficient as C/C++ for performing DSP tasks like FFT. Also the use of GUI in MATLAB slows down signal processing further. The efficiency can be significantly improved if the coding is done entirely in C/C++. The code will then be portable to other operating systems. To make the system even more portable, USB based data-acquisition can be used.

REFERENCES

[1] J. Mitola III, “Software Radios Survey, Critical Evaluation and Future Directions,” *IEEE AES Systems Magazine*, pp. 25-35, April 1993

[2] J. Mitola III, “Software Radio Architecture,” *IEEE Communications Magazine*, pp. 26-36, May 1995

[3] Wikipedia, http://en.wikipedia.org/wiki/Software_defined_radio

[4] SDR Forum, http://www.sdrforum.org

[5] B. Razavi, “RF Microelectronics,” Upper Saddle River, NJ: Prentice Hall PTR, 1998, Chapter 5

[6] Dennis M. Akos, Michael Stockmaster, James B. Y. Tsui, Joe Caschera, “Direct Bandpass Sampling of Multiple Distinct RF Signals,” *IEEE Trans. on Communications*, vol. 47, pp. 983-988, July 1999

[7] Jeffery A. Wepman, “Analog-to-Digital Convereters and Their Application in Radio Receivers,” *IEEE Communications Magazine*, pp. 39-45, May 1995

[8] Elenco Electronics, Inc., “AM/FM Radio Kit Assembly and Instruction,” 150 Carpenter Ave, Wheeling, IL 60090

[9] Measurement Computing Corporation, "PCI-DAS4020 User's Guide," 10 Commerce Way, Norton, MA 02766

[10] The Mathworks, Inc., http://www.mathworks.com/access/helpdesk/help/techdoc/

[11] The Mathworks, Inc., "Data Acquisition Toolbox for Use with MATLAB User’s Guide Version 2," The Mathworks Inc., 2001

[12] Rahim Bagheri, Ahmad Mirzaei, Mohammad E. Heidari, Saeed Chehrazi, Minjae Lee, Mohyee Mikhemar, Wai K. Tang, and Asad A. Abidi, “Software-Defined Radio Receiver: Dream to Reality,” *IEEE Communications Magazine*, pp. 111-118, August 2006

[13] Flex Radio Corporation, http://www.flex-radio.com, 12100 Technology Blvd, Austin, TX 78727
[14] Fraidun Akhi, “Design and implementation of a software radio testset for research and laboratory instruction.” MS Thesis, Auburn University, 2003

APPENDIX

A.1 Bandpass Sampling Frequencies and their Translated IFs for AM IF

Actual IF, f_{IF} = 455 kHz BW = 10 kHz				
Sampling Frequency, f_s (kHz)	**Translated IF frequency, $f_{IF,trans}$ (kHz)**	**Equation (4.1.2) satisfied**	**Equation (4.1.3) satisfied**	**Equations (4.1.2) and (4.1.3) satisfied**
1	0	FALSE	FALSE	FALSE
2	1	FALSE	FALSE	FALSE
3	1	FALSE	FALSE	FALSE
4	1	FALSE	FALSE	FALSE
5	0	FALSE	FALSE	FALSE
6	1	FALSE	FALSE	FALSE
7	0	FALSE	FALSE	FALSE
8	1	FALSE	FALSE	FALSE
9	4	FALSE	FALSE	FALSE
10	5	FALSE	FALSE	FALSE
11	4	FALSE	FALSE	FALSE
12	1	FALSE	FALSE	FALSE
13	0	FALSE	TRUE	FALSE
14	7	TRUE	FALSE	FALSE
15	5	FALSE	FALSE	FALSE
16	7	TRUE	FALSE	FALSE
17	4	FALSE	FALSE	FALSE
18	5	FALSE	FALSE	FALSE
19	1	FALSE	TRUE	FALSE
20	5	FALSE	FALSE	FALSE
21	7	TRUE	FALSE	FALSE
22	7	TRUE	FALSE	FALSE
23	5	FALSE	TRUE	FALSE
24	1	FALSE	TRUE	FALSE

25	5	FALSE	TRUE	FALSE
26	13	TRUE	FALSE	FALSE
27	4	FALSE	TRUE	FALSE
28	7	TRUE	TRUE	TRUE
29	9	TRUE	TRUE	TRUE
30	5	FALSE	TRUE	FALSE
31	10	TRUE	TRUE	TRUE
32	7	TRUE	TRUE	TRUE
33	7	TRUE	TRUE	TRUE
34	13	TRUE	FALSE	FALSE
35	0	FALSE	TRUE	FALSE
36	13	TRUE	FALSE	FALSE
37	11	TRUE	TRUE	TRUE
38	1	FALSE	TRUE	FALSE
39	13	TRUE	TRUE	TRUE
40	15	TRUE	FALSE	FALSE
41	4	FALSE	TRUE	FALSE
42	7	TRUE	TRUE	TRUE
43	18	TRUE	FALSE	FALSE
44	15	TRUE	TRUE	TRUE
45	5	FALSE	TRUE	FALSE
46	5	FALSE	TRUE	FALSE
47	15	TRUE	TRUE	TRUE
48	23	TRUE	FALSE	FALSE
49	14	TRUE	TRUE	TRUE
50	5	FALSE	TRUE	FALSE
51	4	FALSE	TRUE	FALSE
52	13	TRUE	TRUE	TRUE
53	22	TRUE	FALSE	FALSE
54	23	TRUE	FALSE	FALSE
55	15	TRUE	TRUE	TRUE
56	7	TRUE	TRUE	TRUE
57	1	FALSE	TRUE	FALSE
58	9	TRUE	TRUE	TRUE
59	17	TRUE	TRUE	TRUE
60	25	TRUE	FALSE	FALSE
61	28	TRUE	FALSE	FALSE
62	21	TRUE	TRUE	TRUE
63	14	TRUE	TRUE	TRUE
64	7	TRUE	TRUE	TRUE

65	0	FALSE	TRUE	FALSE
66	7	TRUE	TRUE	TRUE
67	14	TRUE	TRUE	TRUE
68	21	TRUE	TRUE	TRUE
69	28	TRUE	TRUE	TRUE
70	35	TRUE	FALSE	FALSE
71	29	TRUE	TRUE	TRUE
72	23	TRUE	TRUE	TRUE
73	17	TRUE	TRUE	TRUE
74	11	TRUE	TRUE	TRUE
75	5	FALSE	TRUE	FALSE
76	1	FALSE	TRUE	FALSE
77	7	TRUE	TRUE	TRUE
78	13	TRUE	TRUE	TRUE
79	19	TRUE	TRUE	TRUE
80	25	TRUE	TRUE	TRUE
81	31	TRUE	TRUE	TRUE
82	37	TRUE	FALSE	FALSE
83	40	TRUE	FALSE	FALSE
84	35	TRUE	TRUE	TRUE
85	30	TRUE	TRUE	TRUE
86	25	TRUE	TRUE	TRUE
87	20	TRUE	TRUE	TRUE
88	15	TRUE	TRUE	TRUE
89	10	TRUE	TRUE	TRUE
90	5	FALSE	TRUE	FALSE
91	0	FALSE	TRUE	FALSE
92	5	FALSE	TRUE	FALSE
93	10	TRUE	TRUE	TRUE
94	15	TRUE	TRUE	TRUE
95	20	TRUE	TRUE	TRUE
96	25	TRUE	TRUE	TRUE
97	30	TRUE	TRUE	TRUE
98	35	TRUE	TRUE	TRUE
99	40	TRUE	TRUE	TRUE
100	45	TRUE	FALSE	FALSE

A.2 MATLAB Software Code

This is the main file (IF455_SDR_GUI_v4.m) which creates and controls the GUI, takes user input, calls different signal processing functions and displays messages on the MATLAB command window.

```
function varargout = IF455_SDR_GUI_v4(varargin)
% IF455_SDR_GUI_V4 M-file for IF455_SDR_GUI_v4.fig
%      IF455_SDR_GUI_V4, by itself, creates a new IF455_SDR_GUI_V4 or raises the
%      existing  singleton*.
%      H = IF455_SDR_GUI_V4 returns the handle to a new IF455_SDR_GUI_V4 or
%      the handle to the existing singleton*.
%      IF455_SDR_GUI_V4('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in IF455_SDR_GUI_V4.M with the given input
%      arguments.
%      IF455_SDR_GUI_V4('Property','Value',...) creates a new IF455_SDR_GUI_V4 or
%     raises the existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before IF455_SDR_GUI_v4_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to IF455_SDR_GUI_v4_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
```

```
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help IF455_SDR_GUI_v4
% Last Modified by GUIDE v2.5 17-Sep-2006 13:58:24
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @IF455_SDR_GUI_v4_OpeningFcn, ...
                   'gui_OutputFcn',  @IF455_SDR_GUI_v4_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before IF455_SDR_GUI_v4 is made visible.


function IF455_SDR_GUI_v4_OpeningFcn(hObject, eventdata, handles, varargin)
```

```
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to IF455_SDR_GUI_v4 (see VARARGIN)
clc;
% Declare parameters shared between different functions as global
global fs fs_out Output_samples flag;
warning off;
% Display welcome message on MATLAB command prompt
fprintf('Welcome to Auburn University SDR (AM Band) v4.0 program.');
fprintf('\nSelect the appropriate sampling frequency and then press ON/OFF...
button to start.\n');
% Set default sampling rate as 80 kHz. This can be changed by end user using GUI
fs=80e3;
% Set output sample rate as 8 kHz. This is fixed and cant be changed by end  user
fs_out=8e3;
Output_samples=8e3;
% Assign a flag to determine ON/OFF. flag = 1 means ON, flag = 0 means OFF
flag=1;
% Generate a 50 Hz sine wave and its FFT
t = 0:1.25e-4:(1-1.25e-4);
x=sin(2*pi*50*t);
```

```
N=length(x);
Ts=length(t);
% Frequency vector
ssf=(-N/2:N/2-1)/(N/fs_out);
% Do FFT
fx=fft(x(1:N));
% Shift it for plotting
fxs=fftshift(fx);
% Get structure of handles.
handles = guihandles(hObject);
%Plot the sine wave in time and frequency domain.
axes(handles.axes1)
plot(t,x)
set(handles.axes1,'XMinorTick','on')
set(handles.axes1,'XMinorGrid','on')
grid on
axes(handles.axes2)
plot(ssf,abs(fxs));
set(handles.axes2,'XMinorTick','on')
set(handles.axes2,'XMinorGrid','on')
grid on
% Choose default command line output for IF455_SDR_GUI_v4
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes IF455_SDR_GUI_v4 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = IF455_SDR_GUI_v4_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

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

% --- Executes on button press in togglebutton1.
function togglebutton1_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles=guihandles(hObject);
global ai ao fs fs_out Output_samples freq_carrier b1 flag;
% Get toggle state of ON/OFF button
```

```
button_state = get(hObject,'Value');
if button_state == get(hObject,'Max')
    % Disable popup menu which selects sampling frequency
    set(handles.popupmenu1,'Enable','off');
    % Reset DAQ devices
    daqreset;
    % Configure the DAQ devices
    ai=analoginput('mcc',2);
    ao=analogoutput('winsound');
    addchannel(ai,2);
    addchannel(ao,1);
    ai.Channel.InputRange=[-1 1];
    % Determine the translated carrier frequency;
    freq_carrier=findcarrier(fs,Output_samples);
    % Determine corfficients of 50th order FIR LPF with cut-off frequency at 5 kHz
    b1=fir1(50,10e3/fs);
    set(ai,'SampleRate',fs);
    set(ao,'SampleRate',fs_out);  %setting the soundcard to 8k out
    set([ai ao],'TriggerType','Manual');
    set(ai,'ManualTriggerHwOn','Trigger');
    set(ai,'SamplesPerTrigger',inf);
    set(ao,'SamplesOutputFcn',{'qmoredatanew',hObject})
    set(ao,'SamplesOutputFcnCount',Output_samples);
```

```
    set([ai ao],'StopFcn',@daqstopped);
    set(ai,'TransferMode','DMA');

    % Workaround to get the DAQ devices running in sync
    y=zeros(Output_samples,1);
    putdata(ao,y);
    start([ai ao]);
    trigger([ai ao]);
    pause(5);
    stop([ai ao]);

    % Start the actual Data acquistion
    putdata(ao,y);
    start([ai ao]);
    fprintf('Program started. Press ON/OFF button to stop.\n');
    trigger([ai ao]);

elseif button_state == get(hObject,'Min')
    flag = 0;
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
```

% hObject   handle to axes1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

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

% Hint: place code in OpeningFcn to populate axes1

% --- Executes on mouse press over axes background.

function axes1_ButtonDownFcn(hObject, eventdata, handles)

% hObject   handle to axes1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles   structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.

function axes2_CreateFcn(hObject, eventdata, handles)

% hObject   handle to axes1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

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

% Hint: place code in OpeningFcn to populate axes2

% --- Executes on mouse press over axes background.

function axes2_ButtonDownFcn(hObject, eventdata, handles)

% hObject   handle to axes1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

```
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1
global fs fs_out Output_samples;
% Contents
contents = get(hObject,'String');
index_selected = get(hObject,'Value');
fs = 1e3*str2double(contents(index_selected));


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


% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```

The main file calls the following function (findcarrier.m) to compute FFT on 'N' samples and find out the exact translated carrier frequency after downsampling the input at 'fs' kHz.

```
function freq_carrier=findcarrier(fs,N)
global ai;
duration=N/fs;
set(ai,'SampleRate',fs);
set(ai,'SamplesPerTrigger',N);
set(ai,'TriggerType','immediate');
start(ai);
y=getdata(ai);
fft_y=fft(y);
[m,imax]=max(abs(fft_y(1:end/2)));      % Index of max peak
freq_vec=fs*(1:length(y))/length(y);    % Generate frequency vector
freq_carrier=freq_vec(imax);            % Find IF frequency
stop(ai);
```

The following function (qmoredatanew.m) is called by the main file to perform quadrature demodulation of the AM signal if 'flag' is set to 1 and to stop the DAQ devices if 'flag' is set to 0.

```
function qmoredatanew(obj,event,hObject)
global ai ao fs fs_out Output_samples freq_carrier b1 flag;
handles=guidata(hObject);
if (flag==0)
   stop([ai ao]);
   set(handles.popupmenu1,'Enable','on');
   fprintf('Program stopped. Press ON/OFF button to start again.\n');
   flag=1;
else
    [y t]=getdata(ai,fs);
   y=y-mean(y);
   y=y/max(abs(y));
   fo=sin(2*pi*freq_carrier.*t);        % In-phase component
   fo_90=cos(2*pi*freq_carrier.*t); % Out-phase component
   x1=filter(b1,1,fo.*y);                % Multiplication with in-phase and subsequent LPF
   x2=filter(b1,1,fo_90.*y);            % Multiplication with out-phase and subsequent LPF
   x=sqrt(x1.^2+x2.^2);
   z=x(1:(fs/fs_out):length(x));
   z(1)=z(4); z(2)=z(4); z(3)=z(4);
```

```
    z=z-mean(z);
    z=z/max(abs(z));
    N=length(z);                    % length of the signal z
    t1=t(1:(fs/fs_out):length(x));
    ssf=(-N/2:N/2-1)/(N/fs);        % frequency vector
    fz=fft(z(1:N));                 % Perform DFT/FFT
    fzs=fftshift(fz);               % shift it for plotting
    plot(handles.axes1,t1,z)
    set(handles.axes1,'XMinorTick','on')
    set(handles.axes1,'XMinorGrid','on')
    set(handles.axes1,'YGrid','on')
    plot(handles.axes2,ssf,abs(fzs))
    set(handles.axes2,'XMinorTick','on')
    set(handles.axes2,'XMinorGrid','on')
    set(handles.axes2,'YGrid','on')
    putdata(obj,z);
end
```