2000

Voice Command Controller

Hoang Nghia Nguyen
Edith Cowan University

Edith Cowan University

Copyright Warning

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

VOICE COMMAND CONTROLLER

Hoang Nghia Nguyen

Faculty of Communications, Heath and Science
Edith Cowan University
Western Australia

Date of submission:

Acknowledgments

I would like to take this opportunity to express my sincere thanks and gratitude to my supervisor Dr. Daryoush Habibi for his endless support and guidance throughout the course of this project and my undergraduate study. I particularly thank him for his kindness, encouragement, without that this project will not be possible.

Many thanks to Mr. Lam S. Phung for his suggestions in designing of this project and giving time, effort to review this thesis.

I wish to thank Utrak surveillant company for providing related technical documents, equipment and has assisted me in making PCBs for this project.

Finally, I would like to all lectures in Engineering faculty, whom has been giving me generously of their time, help and expertise throughout my undergraduate study.

Abstract

Signal processing technology has been strongly developed and it has attracted interest from scientists and engineers around the world from the last decade. Speech synthesis and speech recognition are particular topic in the field that have been widely used and developed in many different area such as business, controlling, education and entertainment.

The project's main objective is to study and develop an application program with the Speech SDK through design and implementation of Tele-Control system based on the commercial product of National Semiconductor: Carrier-Current Transceiver (LM 1893) and Speech development kit (Speech SDK4.0) from Microsoft Corporation.

The project is suitable to be used in restricted areas where space, wiring, decoration and signal interference are issues of concerned.

Speech SDK is an interesting and useful tool in helping develop a Voice application programs. In this project, the user can use voice command interact with the control program to control a remote device.

In conjunction with hardware modification, extra function can be added to the program such as controlling camera, video capture and position control buttons on the environment map, the project will be suitable for security purposes.

DECLARATION

I certify that this thesis does not, to the best of my knowledge and belief:

(i) incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education;

(ii) contain any material previously published or written by another person except where due reference is made in the text; or

(iii) contain any defamatory material.

Signature_

Date 05/9/2000

Table of contents

Part 1 General introduction

1.1 Overview

Signal processing technology has been strongly developed and it has attracted interest from scientists and engineers around the world from the last decade. Speech synthesis and speech recognition are particular topics in the field that have been widely used and developed in many different areas such as business, controlling and entertainment.

There have been various techniques of using Voice applications in controlling like using Speech processor, PCs based control, or pre-record and access voice data from EEPROM. In this project, we design and implement a PC-based Voice Command Control, which allows electrical devices to be controlled remotely. The project employs the speech development kit (Speech SDK) developed by Microsoft and the Carrier Current Transceiver LM 1893 designed by National Semiconductor.

1.2 Background and Motivation

I have worked in electronics industry in repairing and installation of various audio communication and controlling products. However, the lack of knowledge in Communication engineering and Computer programming have restricted my understanding of the products in ways such as: How they are designed? How the hardware and software interact with each other to make such smart products? How to define which functions should be designed and in what way to implement them? This project gives an opportunity to apply my knowledge in designing a particular application and to overcome the above limits.

Thousands of control circuits have been designed and implemented using technologies such as radio frequency, infrared, optical cable. Each method has its strengths and weaknesses in term of transmission range, cost, complexity, and signal-interference. One of the techniques is sending control signals to the main line, which is attractive because of its low cost and its suitability in areas where signal interference, wiring, space and decoration are important issues. In this project, we will use this technique to implement a remote control circuit that uses the electronic device Carrier Current Transceiver LM1893.

1.3 Aims

The principal aim of this project is to design and implement a Tele-Control system using the Carrier-Current Transceiver (LM 1893) from National Semiconductor and the speech development kit (Speech SDK 4.0) from Microsoft.

The main tasks of the project are:

- Firstly, study the architecture, characteristics of the LM 1893 and its operation.
- Secondly, we'll explore the features of the Speech SDK 4.0 and learn how to use these features in this project.
- Thirdly, design and implement a simple Tele-Control network using existing main power line as the communication medium.
- Then, design a computer-based control interfaces using Visual Basic and implement with voice synthesis and voice recognition methods studied in tasks 1 and 2
- Finally, implement the designed circuit and test it.

1.4 Thesis outline

This thesis is organised into four parts:

Part 1: General introduction. This part provides an introduction to the project, and describes the main tasks in design and development of the project.

Part 2: The LM 1893 (current transceiver) and Speech SDK. This part explores the characteristics and features of the electronics devices LM 1893 and Speech synthesis/ recognition technique from Microsoft Speech Development kit, and from that shows how those features can be implemented in this project.

Part 3: Circuit design and program control interface. It contains the project's requirement and analysis, project block module functions, circuit design and the program flow-chart, program modules.

Part 4: Implementing and testing. This part describes the implementation of the circuit, program simulation and testing the complete project. In addition a description is provided on the problems arising throughout the project.

Part 5: summarises what has been achieved in this project. Concluding remarks, recommend applications of the project and further research directions be also proposed in this part.

Part 2 Introduction to the LM1893 and Speech SDK4.0 technologies

This project involves both hardware and software implementation and its characteristic are defined by the two main components: the LM 1893 Carrier current Transceiver for hardware, and the Speech SDK.4.0 for the software. In this part we will study the characteristic, operations and features of these two components, and discuss how they can be applied in this project.

2.1 The LM-1893 Current Transceiver

Designed by the National Semiconductor, the LM-1893 Carrier Current Transceiver main purpose is to allow analogy data to be transmitted and received between remote location over the main power line as a transmission medium. Thus this device serves as a power line interface for half-duplex communication of any digital data stream generated from any digital source such as computer or micro-controller regardless of how those data are encoded.

The following discusses in detail the features and characteristics of this device.

2.1.1 Theory of operation

According National Semiconductor Technical data (1989), the LM-1893 can be set to operate at two different modes: Transmit (TX) or Receive (RX) through a simple hardware configuration.

In the TX mode, the binary data stream from the source is fed to the input of the LM-1893, and is then modulated using FSK (Frequency Shift Keying) at a carrier frequency from 50 Hz to 300 kHz on the line. Whilst in the transmit mode, the baseband data generates a signal to drive the current control oscillator and the differential attenuator which deliver a current sinusoid though the Automatic level Control (ALC) current output amplifier with the gain of 200.

The high current modulated signal at the carrier I/O develops a voltage swing on the resonance tank circuit proportional to the line impedance and then passed through the coupling transform to the main power line. If large line impedance that causes an excessive output voltage level, the ALC will shunt the current away from the output amplifier hence holding the output signal constantly within the amplifier compliance limit. The amplifier is stable with a load of any magnitude or phase angle.

When operating in the receiver mode, the TX sections are disabled. From the coupling circuit, the modulated signal produces a voltage swing in the tank circuit, swinging about the positive supply to drive the Carrier I/O receiver input. The balanced input limiter amplifier removes the DC offset. The limiter also performs as the line attenuation.

The differential demodulated output signal from the phase detector, containing AC and DC data signal, noise, large carrier frequency component, passes through a low-pass filter to drive the offset cancel circuit differentially. The data signal is then delivered by the comparator, passing through the impulse noise filter to remove noise spikes that might be in the data signal before sending it to the external decoder circuit.

The offset cancelling circuit works by insuring that the fixed $\pm 50mV$ signal delivered to the comparator is centred around 0mV-comparator switch point. Whenever the comparator signal plus the DC offset and noise moves outside the matched $\pm 50mV$ voltage "window" of the offset cancel circuit, it adjusts its DC correction voltage in series with the differential signal to force the signal back into the "window".

2.1.2 Main Features

From the above operation, the LM 1893 has a number of features and advantage as follows:

- It provides a very good noise rejection. Noise is filtered out along the demodulation process.
- It allows the user to select impulse noise filtering by changing the relevant component value according to the requirement.
- The LM 1893 can be set to work at different modulation frequencies ranging from 50 Hz to 300kHz. However, the range of the signal that can propagate along the transmission medium is dependant on the line attenuation, which implies that it is also dependant on the signal frequency. For the highest range of propagation, the recommend frequency is about 125 kHz.
- The data signal can be sent at a rate up to 4.8 kBaud or 2,400 Hz. From the point of view of the performance of the LM 1893, the lower the maximum data

transmission rate, the better it will be for the modulation and demodulation processes.

- The LM 1893 can be used to drive any available conventional power line as the transmission medium without extra external wiring.

2.1.3 LM 1893 application design considerations

a) Power line impedance and attenuation

An application designed using LM 1893 requires an estimate of the lowest expected line impedance Z_L encountered for the most efficient transmitter-to-line coupling, line impedance should be measured and Z_L limits fixed to a given confidence level. (National Semiconductor)

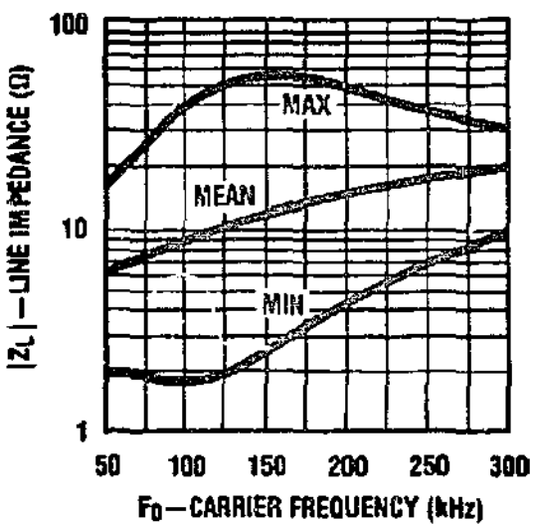


Figure 2-1 Measured of line impedance range for residential and commercial power line

The above figure is the data drawn from the research of Nicholson and Malack, a limited sampling of Z_L during the LM 1893 design on the residential and commercial power line.

From the Data sheet of the LM 1893, the total line attenuation that allows from full signal to limiting sensitivity is about 70 dB. A test on the cable has 100 Ohms characteristic impedance, 125 kHz carrier frequency gives a measured result of 7dB of attenuation for 50 m of signal propagation with 10 Ohms termination. Hence, in order to increase the communication range, matching line impedance and filter

network should be taken into account to reduce signal power loss due to attenuation and prevents noise from polluting the power line.

b) Thermal Considerations

To get maximum range of signal propagation, the signal placed on the line must be as large as possible. The chip power dissipation, and the maximum junction temperature (T_j.) limit this. The falling output power at elevated T_j allows a more optimal output power: a high power at low T_j and lower power at high T_j for the chip self-protection. However, the worst-case condition may occur within the ambient temperature limit $T_a = 85^0 C$, the value of T_j may exceed the maximum Junction temperature (T_j max $= 150^0 C$). The proper design and measurement of T_j will keep the operation normally under worse case condition

c) Data Encoding

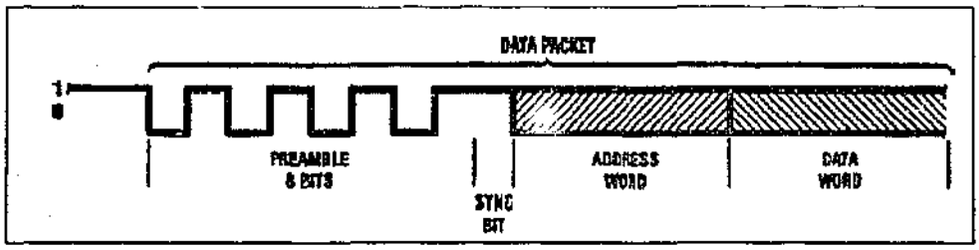


Figure 2-2. A simple encoded data packet, generated by the transmit controller (National Semiconductor, LM1893 Data sheet)

In the LM1893 reference data book has discussed the Data Encoding scheme as follows:

> The first 0 to 2 bits of the data transmitted may be lost whilst the receiver settles to the DC bias point required for the given transmitter/ receiver pair carrier frequency. Hence, with proper data encoding will tolerate the lost bits and correct communication.
>
> A system consists of many transceivers that normally listen to the line at all times, waiting for a transmission that directs one or more of the receivers to operate. If any receiver finds its address in the transmitted data packet, further action such as handshaking with the transmitter is initiated. The receiver may communicate with the transmitter, via re-transmission, that it has received the

> data, waiting for acknowledgment before acting according to the received command. Error detection and correcting codes may be employed throughout. The transmitter must have the capability to re-transmit after the time if no response from the receiver is heard (under the assumption that the receiver does not detect its address due to noise or line collision). (National Semiconductor, Linear Circuit, p 5-154)

Fig. 2-2 is an example of a simple transmission data packet. The 8 bits preamble allows the receiver biasing with enough bits left over, which allows the receiver controller to detect the synchronous bit that signal the start of a transmission. If there is no transmission for sometime, the receiver simply needs to note that a data transition has occurred and begin its watch for a synchronous bit.

The receiver uses the synchronous bit to signal that the address and data immediately following. The address data is then tested against itself. If the address is correct, the received data will be loaded. In case the address is not correct, the controller returns to the state of watching the incoming data for its address. The signal detection, address load and checking mechanism should be fast enough to minimise the time spend in loops after being false-triggering. If the controller detects an error, it should resume watching the LM 1893's data out for transmission, the next bit will be shifted in and the process repeated.

2.2 ActiveX control: Speech SDK4.0

2.2.1 Introduction

Today, adding speech to the application is no longer need special hardware and software. Writing a Voice command-Enable Win32 or Web application is a fairy routine task. The key change to this is the fast development in technology and effort made in recent years for signal processing such as speech synthesis and voice recognition. Several leading vendors have promoted third party development for speech-base application, moving away from vertical integration to a licensing model. Many vendors offer their own Software Development Kit (SDK), which requires the developer and end user have the appropriate engine software present on their platform.

Speech recognition engines have features varying amongst systems and the decision to which engine to work with depends on the needs of the application as well as its environment.

2.2.2 Overview of Speech engines

a) Speech recognition

A Speech recognition engine is a software program that converts a digital audio signal into recognition speech. The engine allows the programmer to write applications that can treat the voice as an input device.

First, the sound is captured by the sound card through a microphone and then is converted into a more manageable format. The converter translates the stream of amplitudes that form the digital sound wave into its frequency components. The basic challenge is to extract the meaningful sound information from the raw audio data. The next stage is the identification of phonemes, the elementary sounds that are the building blocks of words. Each frequency component of the sound is mapped to a specific phoneme. This process actually finishes the conversion from sounds to sentences. The final step is to analyse the string. A grammar, the list of words known to the program, lets the engine associate sets of phonemes with particular words.

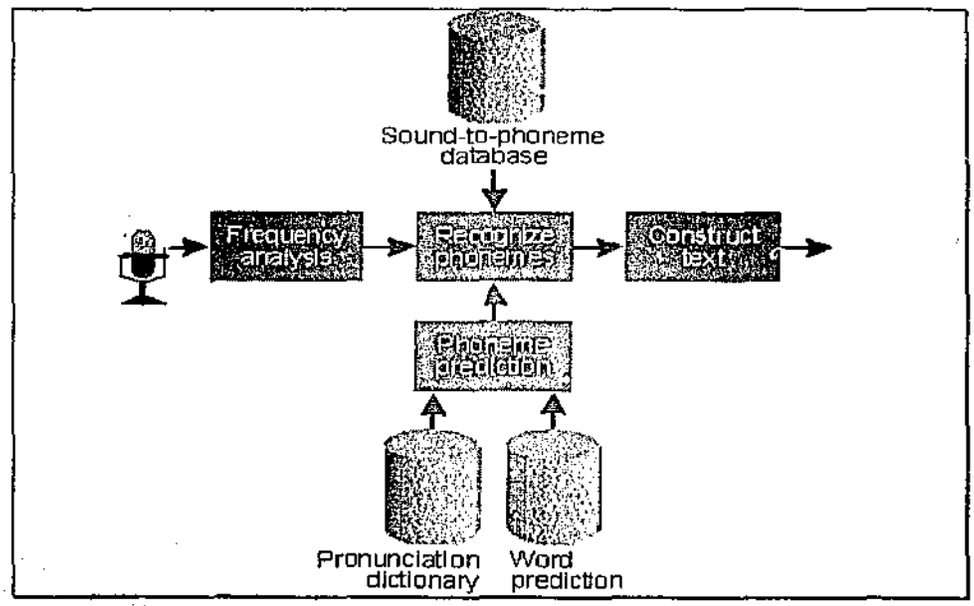


Figure 2-3. Speech Recognition engine

Chaint Inc (1999, p3) has described that most speech-recognition engines can be categorised by the way in which they perform the following basic tasks:

1. Word separation or Speaking mode.
2. Enrollment or Speaker dependence.
3. Matching technique
4. Vocabulary type and size.

- **Word separation**: is the degree of isolation between words required for the engine to recognise a word.

Speech-recognition engines typically require one of the following types of verbal input to detect words:

Discrete speech: Every word must be isolated by a pause before and after the word, usually about a quarter of a second for the engine to recognise it. Discrete speech recognition requires much less processing than word-spotting or continuous speech, but it is less user-friendly.

Word-spotting. A series of words may be spoken in a continuous utterance, but the engine recognises only one word or phrase. For example, if a word-spotting engine listens for the word "time" and the user says: "Tell me the time" or "Time to go," the engine recognises only the word "time."

Word spotting is used when a limited number of commands or answers are expected from the user and the way that the user speaks the commands is either unpredictable or unimportant.

Continuous speech: The engine encounters a continuous utterance with no pauses between words, but it can recognise the spoken words.

Continuous speech recognition is the best technology from a useability standpoint, because it is the most natural speaking style for human beings. However, it is the most computational intensive because identifying the beginning and ending of words is difficult, much like reading printed text without spaces or punctuation.

• **Enrollment / Speaker dependence**: The degree to which the engine is restricted to a particular speaker.

Speech-recognition engines may require training to recognise speech well for a particular speaker, or they may be able to adapt to a greater or lesser extent. Engines can be grouped into these categories:

Speaker-dependent: The engine requires the user to train it to recognise his or her voice. A speaker-dependent system require providing the user's speech sample and is trained to recognise the speech patterns of a particular user so it can only be reliable by recognising commands spoken by the person who trained it.

Training usually involves speaking a series of pre-selected phrases. Each new speaker must perform the same training.

Speaker-dependent engines work without training, but their accuracy usually starts below 95 percent and does not improve until the user completes the training. This technique takes the least amount of processing, but it is frustrating for most users because the training is tedious, taking anywhere from five minutes to several hours.

Speaker-adaptive: The engine trains itself to recognise the user's voice whilst the user performs ordinary tasks. Accuracy usually starts at about 90 percent, but rises to more acceptable levels after a few hours of use.

Two considerations must be taken into account with speaker-adaptive technology. First, the user must somehow inform the engine when it makes a mistake so that it does not learn based on the mistake. Second, even though recognition improves for the individual user, other people who try to use the system will get higher error rates until they have used the system for a whilst.

Speaker-independent: Speaker- independent system in contrast does not require information about the speaker, so that anyone can speak to the application. With the myriad variations between speakers and so their accent, the speech engine can be trained to improve performance.

The engine starts with accuracy above 95 percent for most users (those who speak without accents). Almost all speaker-independent engines have training or adaptive abilities that improve their accuracy by a few more percentage points, but they do not require the use of such training.

• **Matching technique**: is the method that speech engine uses to match a detected word to known words in its vocabulary.

Speech-recognition engines match a detected word to a known word using one of these techniques:

> **Whole-word matching**: The engine compares the incoming digital-audio signal against a pre-recorded template of the word. This technique takes much less processing than sub-word matching, but it requires that the user (or someone) prerecord every word that will be recognised — sometimes several hundred thousand words. Whole-word templates also require large amounts of storage (between 50 and 512 bytes per word) and are practical only if the recognition vocabulary is known when the application is developed.
>
> **Sub-word matching**: The engine looks for sub-words, usually phonemes and then performs further pattern recognition on those. This technique takes more processing than whole-word matching, but it requires much less storage (between 5 and 20 bytes per word). In addition, the pronunciation of the word can be guessed from English text without requiring the user to speak the word beforehand.

• **Vocabulary type and size:** is the number of words that the engine searches to match a word.

Vocabulary type and size is which allows the developer to create application specific vocabulary and language model. Small vocabularies consist of 50 words or less, and are applicable for controlling computer functions using spoken command. Large vocabularies are used for dictation purpose.

Speech-recognition engines typically support several different sizes of vocabulary. Vocabulary size does not represent the total number of words that a given engine can recognise. Instead, it determines the number of words that the engine can recognise accurately in a given state, which is defined, by the word or words that are spoken before the current point in time. For example, if an engine is listening for "Tell me the time," "Tell me the day," and "What year is it?" and the user has already said "tell," "me," and "the," the current state has these two words: "time" and "day."

Speech-recognition engines support these vocabulary sizes:

Small vocabulary: The engine can accurately recognise about 50 different words in a given state. Although many small-vocabulary engines can exceed this number, larger numbers of words significantly reduce the recognition accuracy and increase the computation load. Small-vocabulary engines are acceptable for command and control, but not for dictation.

Medium vocabulary: The engine can accurately recognise about 1000 different words in a given state. Medium-vocabulary engines are good for natural language command and control or data entry.

Large vocabulary: The engine can recognise several thousand words in a given state. A large-vocabulary engine requires much more computation than small-vocabulary engine, but the large vocabulary is necessary for dictation.

b) Text to Speech (TTS)

A speech synthesis engine is a software program that converts word into a digital audio signal (Microsoft MSDN library, 1999). The speech synthesis engine does that by converting words to phonetic and prosodic symbols. The prosodic symbols are codes that the synthesis engine uses to control the speech, emphasis of the synthesised speech by making them louder or longer, or giving them a higher pitch. Other words may be de-emphasised. Without word emphasis, or "prosody", the result is a monotone voice that sounds like a robotic.

The TTS engine does basically the reverse of what the SR and the DSR engines do. Its input is plain ASCII text, and its output is a mono, 8-bit, 11 kHz audio formats and is described as Pulse Code Modulation (PCM). PCM is a commonly used method to obtain a digital representation of an analog voice signal. A PCM bit stream is composed of a sequence of numbers that are samples of the voice amplitude. As mentioned earlier, a basic element in any speech- related engine is the phoneme. A phoneme is an atomic unit of sound that can be used to form words. The programmer can identify a spoken language by looking at the set of its phonemes. The TTS engine provides a way to convert from a string to its phoneme-based representation and then to an audible sound. The sound then can be played through a computer's speakers or saved to disk as a WAV file.

A text-to-speech engine translates text or phonemes into audible sound in one of several ways, either by synthesis or by di-phone concatenation.

Concatenated Word. Although Concatenated Word systems are really synthesisers, these are one of the most commonly used text-to-speech systems around. In a concatenated word engine, the application designer provides recordings for phrases and individual words. The engine pastes the recordings together to speak out a sentence or phrase. If the programmer use voice-mail then he will hear one of these engines speaking, "[You have] [three] [new messages]." The engine has recordings for "You have", all of the digits, and "new messages".

Synthesis: A text-to-speech engine that uses synthesis generates sounds like those created by the human vocal cords and applies various filters to simulate throat length, mouth cavity, lip shape, and tongue position. The voice produced by current synthesis technology tends to sound less human than a voice produced by diphone concatenation, but according to Microsoft, it is possible to obtain different qualities of voice by changing a few parameters.

Diphone Concatenation: A text-to-speech engine that uses diphone concatenation links short digital-audio segments together and performs inter-segment smoothing to produce a continuous sound. Each diphone consists of two phonemes, one that leads into the sound and one that finishes the sound. For example, the word "hello" consists of these phonemes: *h eh l œ*. The corresponding diphones are *silence-h h-eh eh-l l-œ œ-silence*.

Diphones are acquired by recording many hours of a human voice and identifying the beginning, ending of phonemes. Although this technique can produce a more realistic voice, it takes a considerable amount of work to create a new voice and the voice is not localisable because the phonemes are specific to the speaker's language.

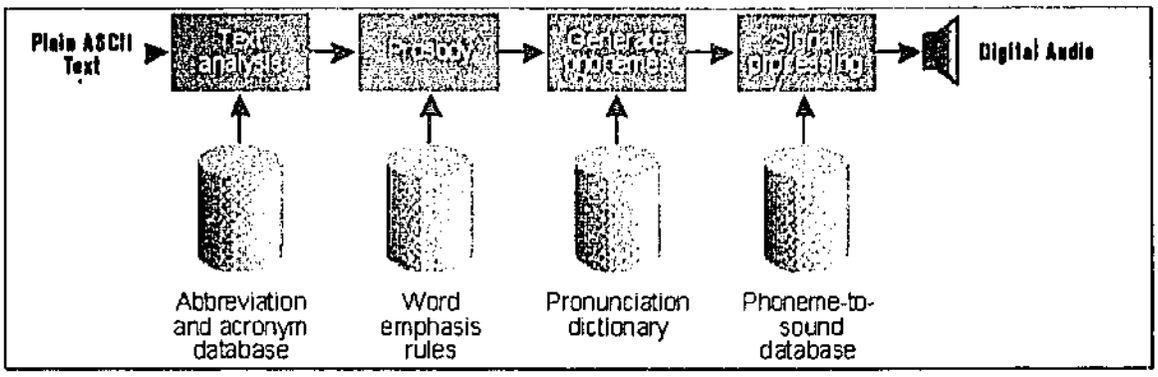


Figure 2-4. Text-to-Speech Engine

2.2.3 Introduction to Microsoft Speech development Kit (Speech SDK4.0)

As speech has become more feasible on average PCs, vendors have been developing and promoting their speech engines. Even though they all support similar functionality, each speech engine has its own specific features and proprietary API. In designing a speech application, the developer has to decide which engine to use and probably have to rewrite the program substantially to use the other API.

The Microsoft Speech API is an attempt to correct this problem. By promoting an industry-standard programming interface for speech.

The Speech API allows writing Win32-based applications (for Windows 95 or Windows NT) that use speech recognition and text-to-speech. The API is specified as a collection of OLE Component Object Model (COM) objects. Using OLE makes speech readily available to developers writing in Visual Basic, C/C++, or any other programming language that can access OLE objects directly or through automation.

With other Windows Open Services Architecture (WOSA) services, the Speech API is intended as a standard interface that application developers and engine vendors alike can code to. Programmers can write applications without worrying about which engine to use, engine vendors can get instant compatibility with all speech applications, and users gain the freedom to choose whichever speech engine meets their budget and performance requirements

Microsoft has discussed in the Speech SDK reference manual, the Speech API offers two levels of access: high-level objects designed to make implementation easy, and

low-level objects that offer total control but requires more work. In the high-level objects, they just call the low-level objects to do the work (the low-level objects are provided by the speech engine vendor). When the application uses the low-level API, it's talking directly to the third-party code, bypassing Microsoft code completely.

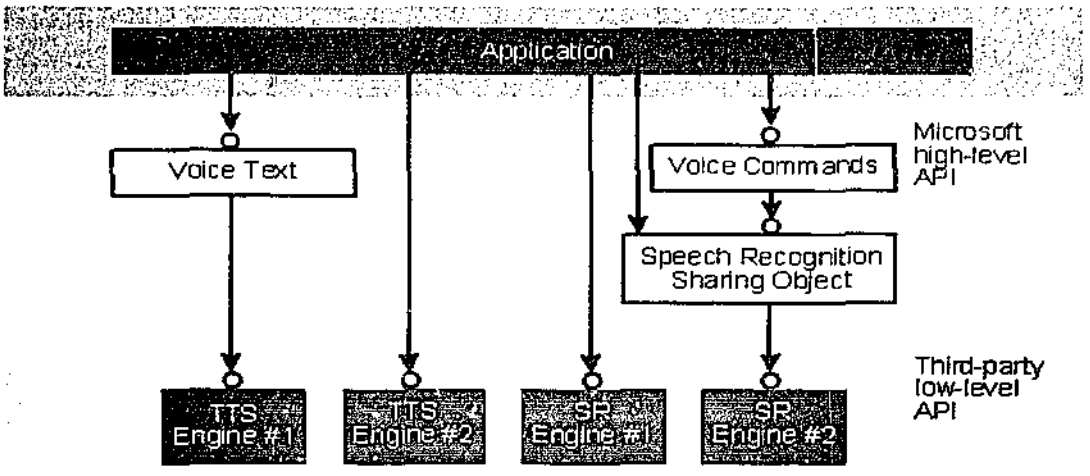


Figure 2-5 . relationship between high and low level API

The Speech API consists of four basics engines: Speech recognition (SR), Dictation Speech recognition (DSR), Text-to Speech (TTS) and Telephony.

The Speech API is implemented as a series of Component Object Model (COM) interfaces.

As discussed before, the SAPI model is divided into two distinct levels:

- *High-level SAPI*-This level provides basic speech services in the form of command-and-control speech recognition and text-to-speech output.
- *Low-level SAPI*-This level provides detailed access to all speech services, including direct interfaces to control dialogs and manipulation of both speech recognition (SR) and text-to-speech (TTS) behaviour attributes.

Michael C. Amundsen (SAPI, TAPI developer's, 1999) and Microsoft (Speech SDK manual) have shown that each of these two levels of SAPI services has its own set of objects and methods. In the following section, we will discuss the high-level SAPI as it is used to implement the software part of the project. The high-level SAPI is chosen because of the project' s time constrain (The *High-level SAPI allows less developing*

time compare to the Low-level SAPI), however it provided enough methods and properties that satisfied the requirement of this type of project.

a) High-Level SAPI

The high-level SAPI provides access to basic forms of speech recognition and text-to-speech services. This is ideal for providing voice-activated menus, command-buttons, and so on. It is also sufficient for basic rendering of text into speech.

The high-level SAPI interface has two top-level objects, one for voice command services: Speech Recognition (SR), and one for voice text services Text-To-Speech (TTS).

Voice Commands and Voice Text provide access to the Speech API's high-level shared interfaces. These interfaces are somewhat more limited, and perhaps slower, than the Direct APIs, but provide automatic resource and memory (out-of-process) sharing between voice applications.

Voice Command

The Voice Command object is used to provide speech recognition services. It is useful for providing simple command, and control speech services such as implementing menu options, activating command buttons, and issuing other simple operating system commands.

The Voice Command object has one child object and one collection object. The child object is the Voice Menu object and the collection object is a collection of enumerated menu objects.

The Voice Command object supports three interfaces:

- The Voice `Command` interface
- The Attributes interface
- The Dialogs interface

The Voice Command interface is used to enumerate, create, and delete voice menu objects. This interface is also used to register an application to use the SR engine. An additional method defined for the Voice Command interface is the `Mimic` method. It

supplies a voice-aware application with the equivalent of a spoken voice command, causes the command engine to act as if the recogniser has heard the command.

The Attributes interface is used to set and retrieve a number of basic parameters that control the behaviour of the voice command system. The programmer can enable or disable voice commands, detect wave-in-audio device, adjust input gain, establish the SR mode.

The Voice Menu object is the only child object of the Voice Command object. It is used to allow applications to define, add, and delete voice commands in a menu. The programmer can also use the Voice Menu object to activate and deactivate menus and, optionally, to provide a training dialog box for the menu.

The voice menu collection object contains a set of all menu objects defined in the voice command database. Microsoft SAPI defines functions to select and copy menu collections for use by the voice command speech engine.

Voice Text

The SAPI model defines a basic text-to-speech service called *voice text*. This service has only one object-the Voice Text object. The Voice Text object supports three interfaces:

- The Voice Text interface
- The Attributes interface
- The Dialogs interface

The Voice Text interface is the primary interface of the TTS portion of the high-level SAPI model. The Voice Text interface provides a set method to start, pause, resume, fast-forward, rewind, and stop the TTS engine whilst it is speaking text.

The Attribute interface provides access to settings that control the basic behaviour of the TTS engine. For example, the programmer can use the Attributes interface to set the audio device to be used, set the playback speed (in words per minute), and turn the speech services on and off. If the TTS engine supports it, the programmer can also use the Attributes interface to select the TTS speaking mode. The TTS speaking mode

usually refers to a predefined set of voices, each having its own character or style (for example, male, female, child, adult, and so on).

The Dialogs interface can be used to allow users the ability to set and retrieve information regarding the TTS engine. Microsoft does not determine the exact contents and layout of the dialog boxes but by the TTS engine developer.

Dialog Name	***Description***
About Box	Use to display the dialog box that identifies the TTS engine and shows its copyright information.
Lexicon Dialog	Can be used to offer the speaker the opportunity to alter the pronunciation lexicon, including altering the phonetic spelling of troublesome words, or adding or deleting personal vocabulary files.
General Dialog	Can be used to display general information about the TTS engine. Examples might be controlling the speed at which the text will be read, the character of the voice that will be used for playback, and other user preferences as supported by the TTS engine.
Translate Dialog	Can be used to offer the user the ability to alter the pronunciation of key words in the lexicon. For example, the TTS engine that ships with Microsoft Voice has a special entry that forces the speech engine to express all occurrences of "TTS" as "text to speech," instead of just reciting the letters "T-T-S."

Table 1 The Voice Text dialog boxes.

b) Low level SAPI

DSR

Direct Speech Recognition (DSR) is the process that converts sounds into strings whilst the user is speaking. DSR needs a large vocabulary that exists in context. A context-free grammar defines a specific set of words, whilst an in-context grammar involves a virtually endless list of words. Therefore, DSR needs more processing power than SR. Recognising speech involves many variables, from the speaker's voice and accent to the type application. There are two types of dictation-based recognition: discrete and continuous. The Speech SDK 4.0 supports continuous dictation, the more sophisticated form. In discrete dictation, the application has a defined vocabulary of recognisable words, and the user must pause for a few milliseconds between words when speaking. This limitation doesn't exist in continuous dictation, where the end of a word need not be clearly indicated by a pause. This means much more work for the engine, necessitating high optimisation and computing power for good results.

TEL

Telephony applications are applications that are accessed via the telephone rather than locally over the PC. A GUI application may also support telephony features although the user interface deigns for the two interaction mechanisms are significantly different. Many GUI applications support telephony because of the flexibility that a long-distance connection to the PC provides. Telephony applications use the same speech recognition engines used for Command and Control speech recognition, and the same text-to-speech engines used on the PC.

TEL applications use objects called "telephony controls" that work like the constituent controls of a dialog box. A telephony application is composed of questions and answers exchanged between the application and its users. Each of these fragments of conversation occurs under the supervision of a telephony control.

The Telephony control implements an interface to allow voice synthesis, voice recognition, wave synthesis, and DTMF on single or multi-line voice telephone devices.

Part 3 Circuits design and software development

This part will focuses on the design of the project in both hardware and software. Firstly, the details project's requirements, and the system blocks functions will be defined. Secondly, for each corresponding block function, design the details of electronics circuits for each function, and they then are added together to complete the system. Finally, in this part we will construct the program flowchart, which is used for software implementation.

3.1 Project specification and analysis

As introduced in the first part, the purposes of the project are to control electrical devices remotely using the LM 1893 Carrier-Current transceiver, which takes the main power line as its communication medium, and it will let the user to control the devices directly or indirectly through telephone network.

For any communication or control system, error may occur in transmission as the result of noise, in this project data encoder/ decoder will be used to detect the corrupted data at the receiver.

The program interface sends the control/ data signal through the parallel port, as the serial port will be used for interface with voice modem. Besides that, the unused pins of the parallel port can be used for future implementation.

The project should complement with the Australia electrical standard AS 3000.

The program interface should provide sufficient functions and basic help system.

The project specifications can be summarised as follows:

- Use main power line for transmission medium.
- Provide error protection for data transmission.
- Data and control signals are send through the PCs parallel port.
- Voice telephony command can be accessed through modem.
- Provide sufficient control functions, program user interface and help system.

3.2 Hardware design

There are large number of remote control circuits that have been designed and developed using different techniques and methods of transmitting control signal such as RF, twister wire, friber optic, IF, etc. Transmitting the control signal or data through the main power line is not a new idea. The Passenger Intercom installed on the train is an example, this project is just one amongst various ways of implementing this idea.

The fundamental requirement for any communication system is involved at least two parties, which are referred as a transmitter unit and one or more receiver unit. There are no exceptions for this project, it must have a transmitter to send the control signal and a receiver to get and executes the received command. In this case, the process at the receiver is just simply to perform a switching function to control the power to the controlled devices.

3.3 Main control transmitter

According to the project specifications, the main control transmitter consists of a transmitting unit for data, a computer interface unit to receive the signal from the PC and a Data encoder unit for error protection. In here, a telephone interface unit is added to provide flexibility in telephony control such we can activate the control through modem or the built in telephone interface module. In addition, a power supply unit is also necessary for all the electronic devices in the whole circuit. Thus, in total the transmitter has five main parts: a computer interface, a telephone interface, a data encoder, and a main transmitting unit. The operation of the transmitter is as follows:

The computer interface unit is responsible for receiving the incoming serial data stream from the program control. The data signal is then converted into a suitable format and sent to the encoder unit.

At the encoder unit, the information signal will be encoded and serially sent to the transmitting unit.

The transmitting unit is represented by the LM 1893 Carrier current transceiver. Encoded information from the encoder is used to generate FSK modulated signal

about 125kHz carrier. After be amplified, the modulated information is then passed through the coupling transform on to the power line.

The telephone interface converts the analog signal of the incoming ring to the digital form suitable for recognised by the program. The program control constantly monitors the incoming ring signal from the telephone interface. If the signal is detected and after a certain predefined number of ring, the line will be connected by the off-hook signal sending from the control program. It then can detect voice command via telephone line and is done by extracting and transferring the audio signal between the sound card and the telephone. Fig. 3.1 summarises the transmitter basic block functions and the position of each block in sequence.

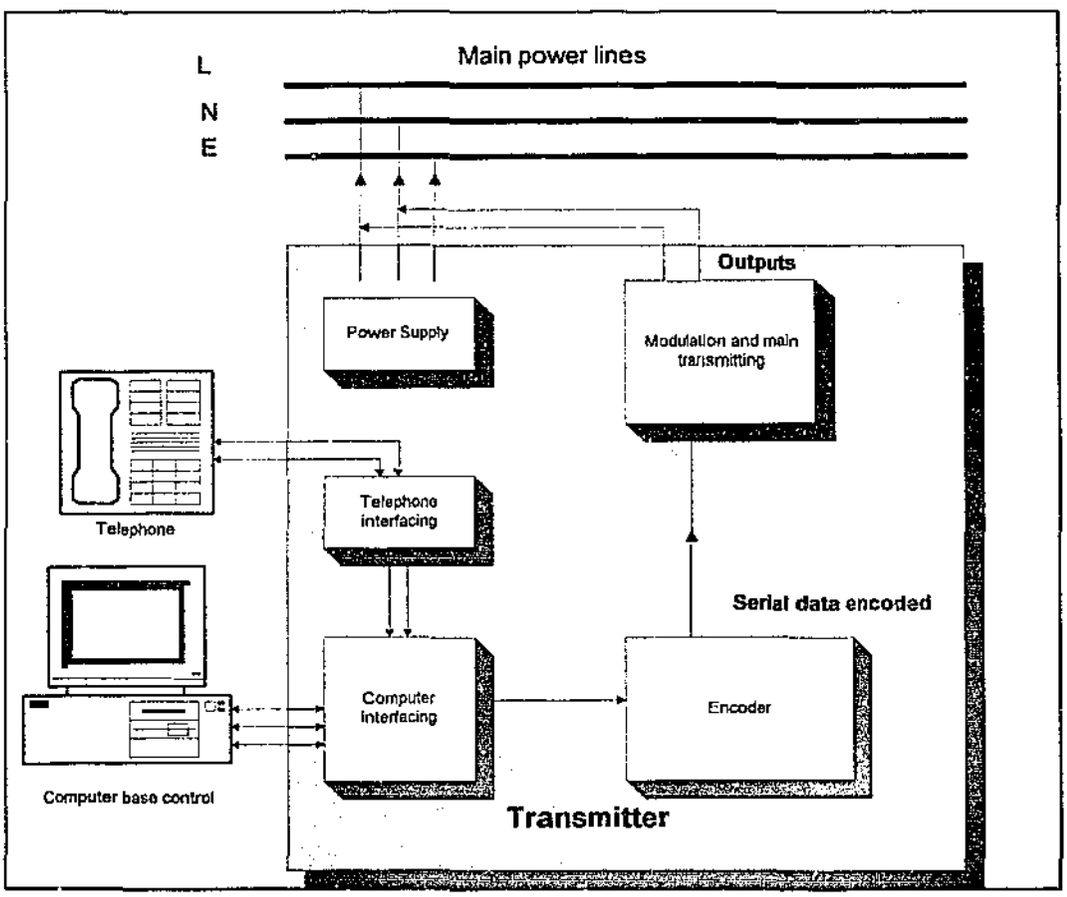


Figure 3-1. Transmitter block functions

3.3.1 Transmitter Circuit

a) Modulation and Transmitter section

The module is designed base on the LM 1893 CCT, with the central modulation frequency is Fo = 125 KHz set by the capacitor C3 (560pF) and resistor R10 (5.6 KΩ). From the technical data sheet, the LM 1893 can be operated with the frequencies ranging from 50 KHz to 300 KHz. I have decided to select 125 KHz as the carrier frequency for two reasons: First, it is safer to work with a middle range rather than at the boundaries. Second, recommendations and application testing designed provided by the manufacture are done based on the above selected frequency.

R9 (10KΩ) and C4 (100nF) are used to control the dynamic characteristics of the transmitter output envelope. Their value are not critical and given by the manufacture. These two components act as an Automatic level Control (ACL) and are functions of loaded coupling transform tank Q, R18 (5.6KΩ), line impulse noise.

Capacitor C8 (33nF) and the coupling transformer (T2) together construct a tank circuit. The tank resonant frequency Fq must be correct to allow passage of transmitter signal to the line. The value of C8 can be obtained from the following equation:

$$C8 = \frac{1}{(2\pi Fo)^2 L1}$$ (L1 is the inductance of the transformer' primary)

C2 (10nF) and R8 (3.3KΩ) are components of the Phase Lock Loop (PLL) loop filter, which remove some of the noise and most of 2Fo components present in the demodulated differential output voltage signal from the phase detector. Their values affect the PLL capture range, loop bandwidth, damping and capture time. The data sheet supplied the graphs of the relationship of they value versus the carrier frequencies, hence these values can be obtained graphically.

R15 (4.7Ω) acts as a voltage divider with D1, absorbing transient energy that attempts to pull the Carrier input above 44Volts.

C9 and C10 both have value of 220 nF/220V, and they primary function is to block the power line voltage from the coupling T2's line side winding. In addition, together with the T2's line side winding, they comprise a LC highpass filter.

The tank transform (T1) serves as isolation and matches the power line impedance and so produces maximum carrier signal coupled to the main line. For 125 KHz carrier frequency, a transformer with turn ratio N=7.07, L1 = 49uF is used as required from technical data of the LM 1893.

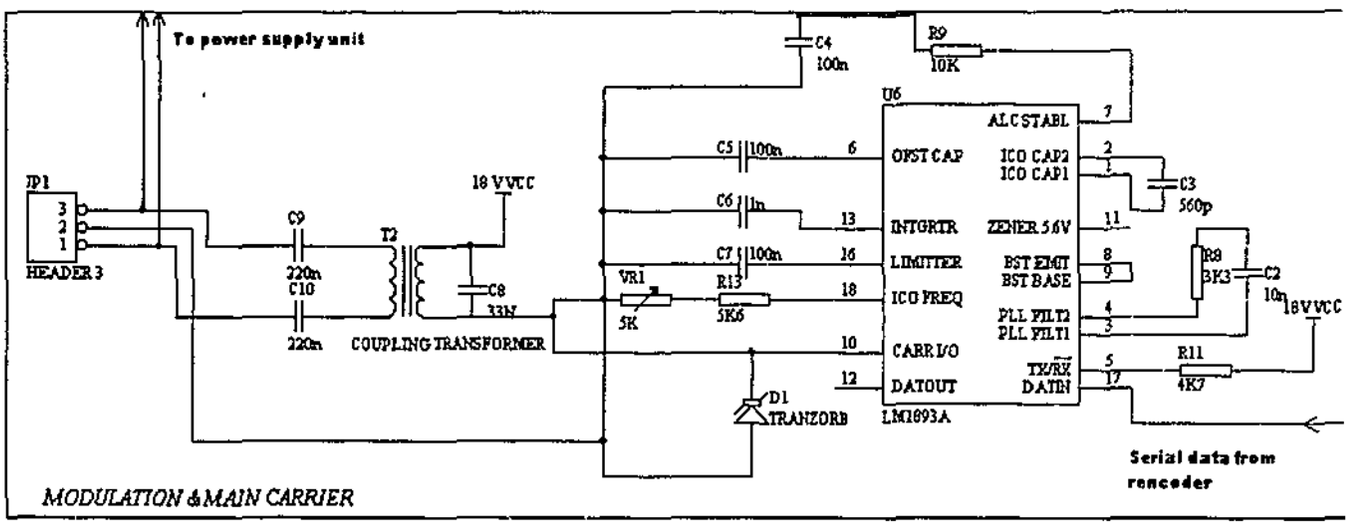


Figure 3-2 Main Carrier

b) The Encoder section

The serial binary data from the computer interface is converted to parallel form by the shift register MC 74164 (8 bit serial to parallel converter). The parallel data at the output of the shift register is then fed to the Encoder MC 145026 (Form of Encoder and Decoder Pair CMOS MC 145026/ MC 145027. Motorola Semiconductor Technical data)

The MC 145026 encodes nine lines of information and serially sends this information upon receipt of transmit enable signal ($\overline{TE}$ pin). Pin A9/D9 of the device is grounded as there are only 8 bits data coming from the shift register. In the circuit, the $\overline{TE}$ input is grounded to make the encoder transmit data continuously.

Each data bit is encoded into two data pulses. A logic zero (0) is encoded as two consecutive short pulses, a logic one (1) as two consecutive long pluses. During

transmission, the MC 145026 will output two identical data words, and between them, no signal is sent for three data periods.

The capacitor C1 (5.1 nF) and resistors R5 (50KΩ), R6 (100KΩ) define the timing for the device with the oscillation frequency of 1.71 KHz. We can use other value of frequency of oscillation, but a good match between the Encoder and Decoder must be ensured.

The data encoded from the MC 145026 at its output pin (pin 15) is then sent to the Modulation/ transmitter section (pin 17 of LM 1893). A pulse up resistor R7 (4.7 KΩ) is used to ensure the data in its digital level. Fig. 3-3 below is the circuit diagram of the Encoder section.

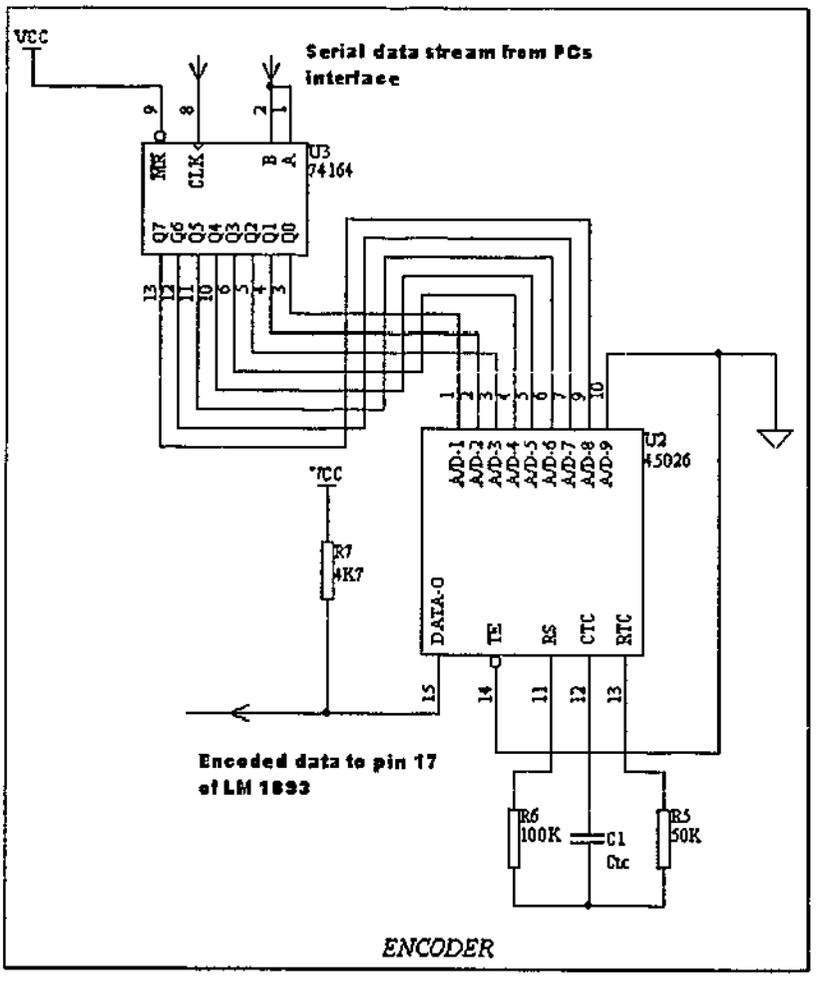


Figure 3-3 The Encoder circuit

c) Computer interface section

Provided isolation between the computer-based controller and the transmitter, this section consists of two opto-couplers H11G1 (High Voltage Darlington Output), which get the data signals and clock from the program via the parallel interface 25-pin, D-shell connector. The values of resistors R1, R2, R3 and R4 serve as current limiting to the devices. The current transfer of H11G1 is typically 10mA

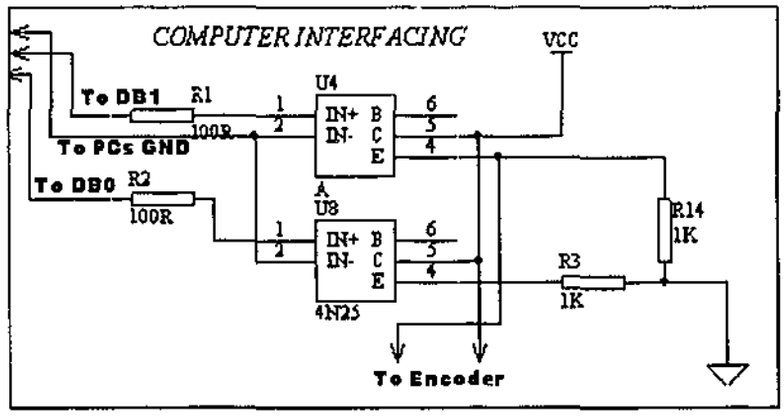


Figure 3-4 Computer interface circuits

d) Telephone interface section

The telephone interface circuit consists of a bridge rectifier diode and a 5V zener diode D6 1N-4748, which convert 50AC Volts of the Ring signal from telephone line (Ring and Tip) to low level 5 Volts DC. The zener diode is also using to protect the circuit from voltage spikes. An opto-coupler 4N35 (U1) will then detect the DC level, converts it to digital form, and sends to the computer monitor program via parallel cable DB-25.

The circuit is protected against voltage spikes or transients by a metal oxide varistor. A low pass filter is form by a 1 μF capacitor C13, which attenuates any substantial AC voice signals to prevent varistor from placing an excessive load on speech signal. The 6.8 KΩ resistor R4 helped to dissipate power away from the rest of the circuit.

When the program detects the incoming ring signal and after a certain number of rings defined by the user, the computer sends an off-hook signal to the U7, which is also an opto-coupler. The output U7 in turn drives the transistor, hence provides a complete path for the relay RL1 and the telephone is brought to the answer mode.

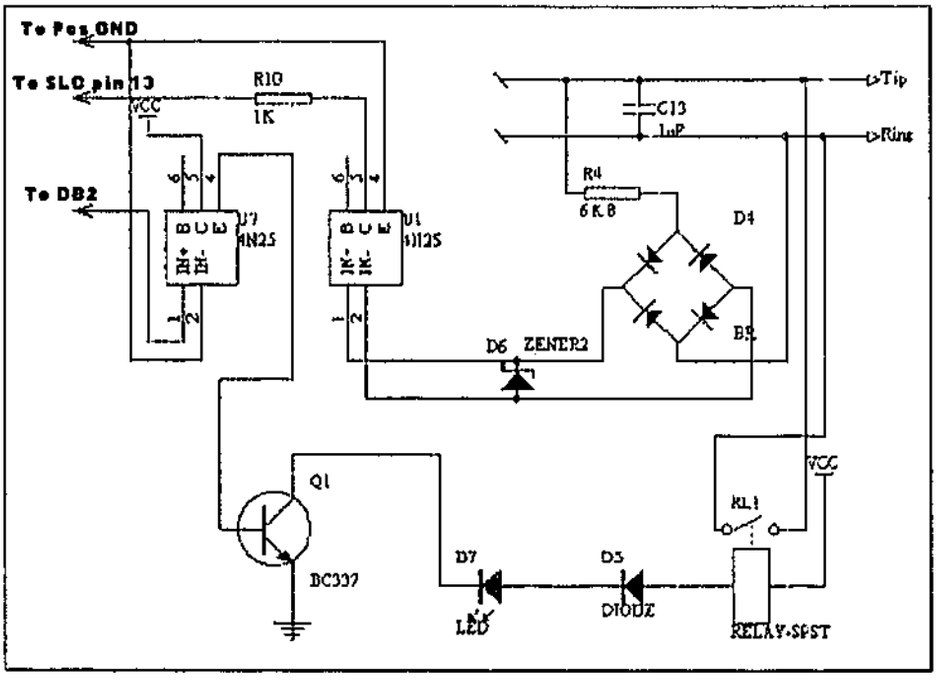


Figure 3-5 Telephone interface circuits.

In the speech circuit, a 600 Ohms: 600 Ohms isolation line transformer (T3) is used (Stephen J.Bigelow, 1983) to provide audio transfer whilst still maintain the impedance matching level between the telephone line and the sound card. . In addition, two 10V zener diodes 1N4733 are connected in a back-to-back configuration across the line for transient protection purpose.

Fig. 3-6 below is the simple speech circuit. It should be noted that, the speech circuit is required to get Austel permitted before it can connect to the phone line according to Australia Telecommunication law.

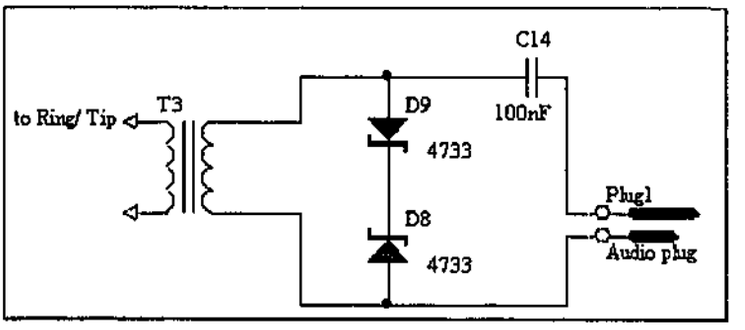


Figure 3-6 Speech circuits

3.4 The Receiver

The receiver has a carrier detect and demodulation unit in order to pick the designed signal, and demodulates the signal to its original form. A decoder section is then required for translating the encoded signal fed from the demodulation unit. The correct and suitable command signal from the decoder will then drive the control circuitry, in this case is a control-switching unit. Similar to the transmitter circuits, the receiver also needs a power supply to provide power necessary to operate the electronic and electrical devices in the whole circuit. All together, there are four main parts, which exist in the receiver: A main carrier detection and demodulation unit, a decoder unit, a control switching and finally a power supply unit.

The receiver circuit can be designed quickly by re-configured the modulation and transmitting part of the transmitter (pin 5 of the LM 1893) then modify the rest with suitable devices.

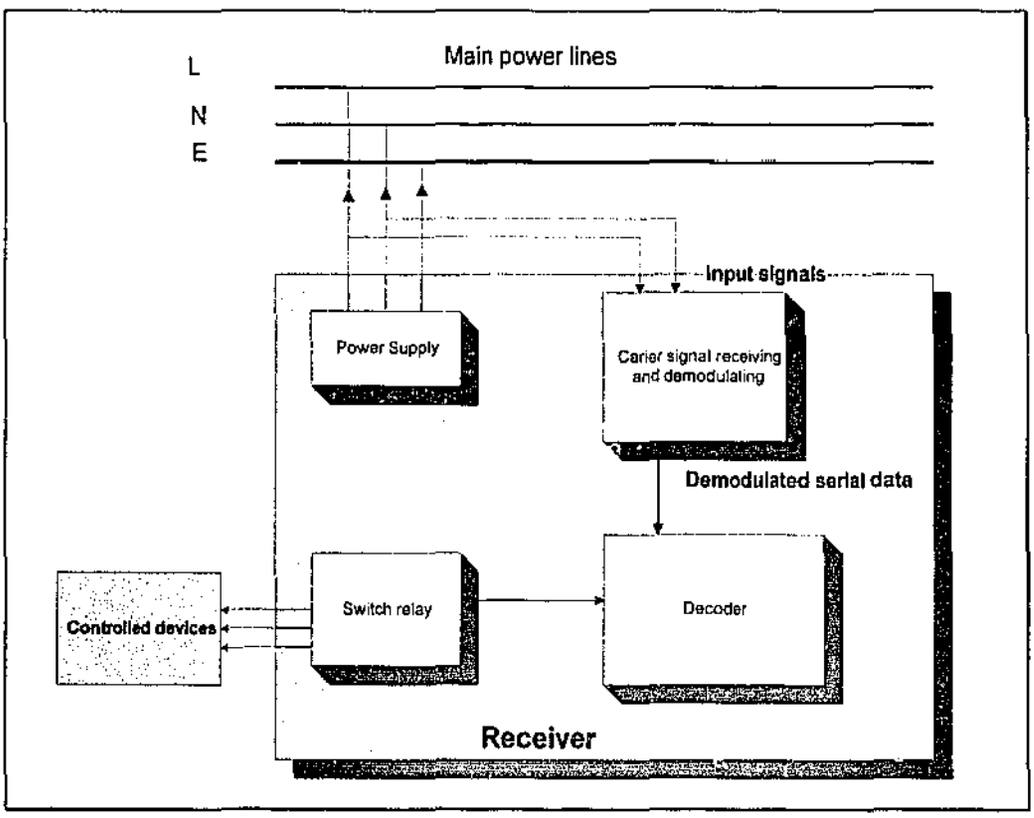


Figure 3-7 The Receiver and control circuitry

3.4.1 Receiver circuit

a) Receiver and Demodulation

This circuit is the same as the modulation circuit discussed in the transmitter, the differences are the LM 1893 is now set in receiving mode, and its operation is reverse to that of the transmitter.

In receiving mode, the Tx /Rx pin (Pin 5) has been held low, so that the transmitting section of the chip is being disabled.

The high pass filter selects the signal from the controller. The filter consisting of the two capacitors C9 and C10, the transformer T1 and the bandpass filter that make up of C8 and T1. As the result, only the carrier signal and the band-limited noise are allowed to pass. These filters also attenuate heavily the 240 V AC and the transient spike energy. The signal is fed into the Carrier I/O receiver input (Pin 10). In side the LM1893, the balanced Norton-limiter amplifier removes DC offsets, attenuates line frequency, performs as a bandpass filter and limits the signal to drive the Phase Lock loop (PLL) phase detector.

The output signal from the phase detector containing AC and DC data signal, noise, system DC offset and other frequency components passes through a RC lowpass filter and finally through an impulse noise filter to produce serial data at the open collector output data out (Pin 12).

The capacitor C2 (10 nF) and resistor R8 (3.3 KΩ) have been chosen as components of the (PLL) loop filter that removes some of the noise and the most of the 2FO components present in the demodulated differential output voltage signal from the phase detector. According to the Technical data of the LM 1893, they affect the PLL capture range, loop bandwidth, damping, and capture time. Because the PLL has an inherent loop pole due to the integrator action of the ICO (via C3), the loop pole set by C2 and the zero set by R8 gives the loop filter a classical 2nd-order response.

Capacitor C5 (100 nF) stores a voltage corresponding to a correction factor required to cancel the phase detector differential output DC offsets.

The resistor R7 (4.7 KΩ) is the pull-up resistor is sized to supply adequate pull-up voltage whilst preserving adequate output low current drive.

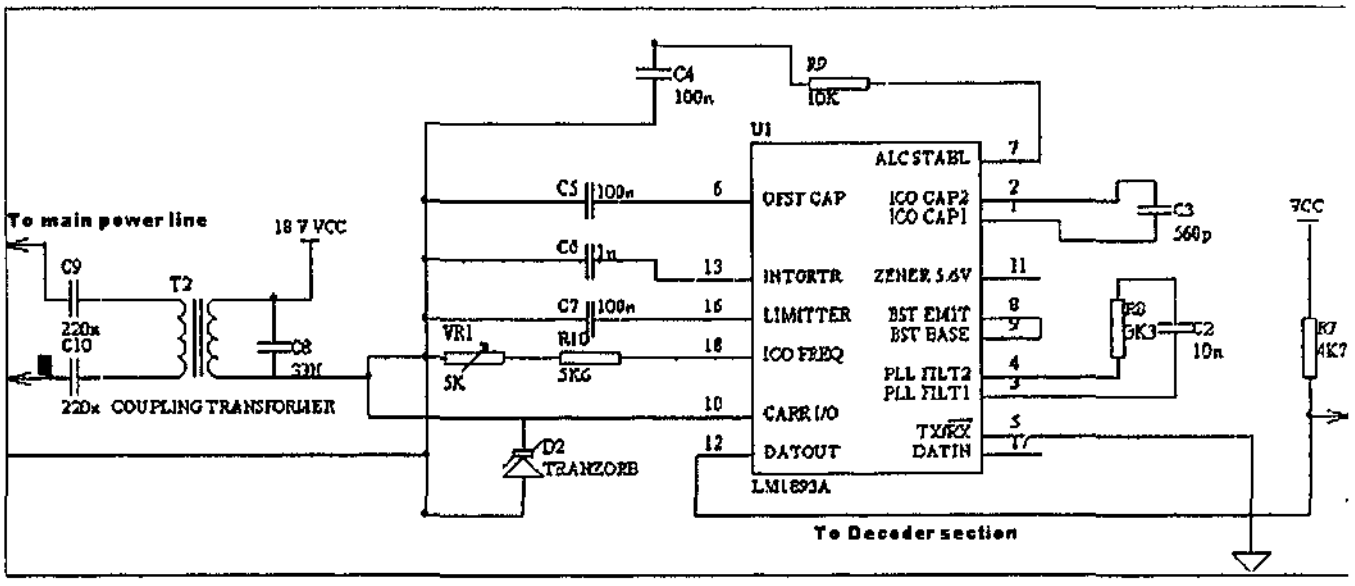


Figure 3-8 Receive and Demodulation circuits

b) Decoder circuits

The main component in the decoding unit is a MC145027 decoder IC, which is in pairs with the MC145026 encoder used in the transmitter.

The operation of the MC145027 decoder is as follows: the MC145027 receives the serial data form the output of the de-modulator (LM 1893), which are encoded by the MC145026, outputs the received data if it is valid.

Motorola (Communication Devices, 1993) has discussed that the transmitted data (consisting of two identical data words) are examined bit by bit during reception by the decoder.

The first five digits are assumed to be address bits, and if the received address matches the local address (set by Dipswitch SW1) A1 through A5 (Pin 1 - 5) of the decoder. The next four data bits are stored in an internal register but not transferred to the output data latch.

As the second encoded word is received, the address must again match. If the match occurs, the new data bits are now checked against the previous stored data in the register. If the four bits data match, the data is transferred to the output data latch by Valid Transmission signal (VT at pin11) and remains until new incoming data replaces it. In the mean time, the VT output (pin 11) is set to high, and remains high until an error is received or no signal is received for four data periods.

The resistor pack (4.7 K, type Plastic DIP) is for current limiting purpose between the Local Address (SW1) and the decoder.

The values of components attached to the MC 145027 are available from the technical data of the device in form of look-up table, with different of oscillation frequencies. The resistor R5 (50 K) and C1 (0.02 uF) together determine whether the narrow pulse or wide pulse has been received. The resistor R6 (200K) and C13 (100nF) are for detecting both ends of a received word and end of a transmission.

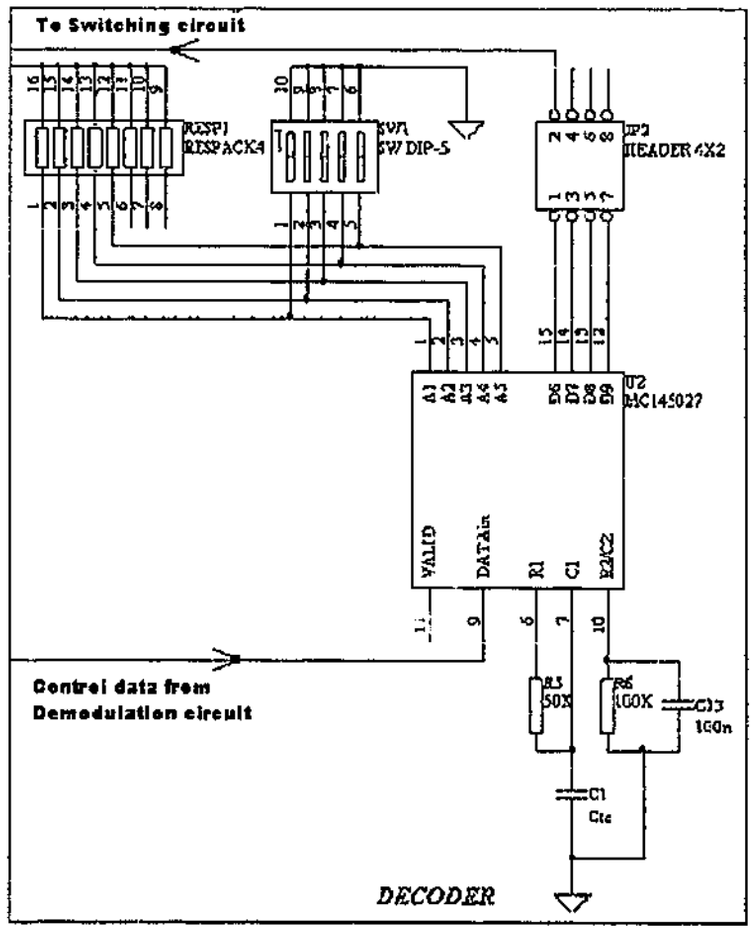


Figure 3-9 The Decoder circuits

c) Switching circuit

The first bit of the data output from the Decoder (pin 15 of MC 145027) is a control bit, (bit 6 in the information data stream) and connected to the relay unit via switch SW2 to control the operation of the transistor Q1 (BC 337). The input Neutral line of the main plug is connected directly to the Neutral of the output main socket. The input main Livewire is connected to the Livewire output of the main socket via the Main Relay.

Logic 1 or 0 of the control bit will trigger Q1, which in turn controls the Close /Open state of the main relay. A Led (D4) attached to the circuit indicates the status On /Off of the switching unit. The present of the diode D5 (1N4848 small signal diode) at the collector of Q1 is used to claim back the EMF generated by the main relay that may cause oscillation in the control circuit.

The switch SW2 can be set to Override mode, and in this mode the Livewire line of the main plug is directly connected to the Live output of the main socket, bypass the control of the controlling program.

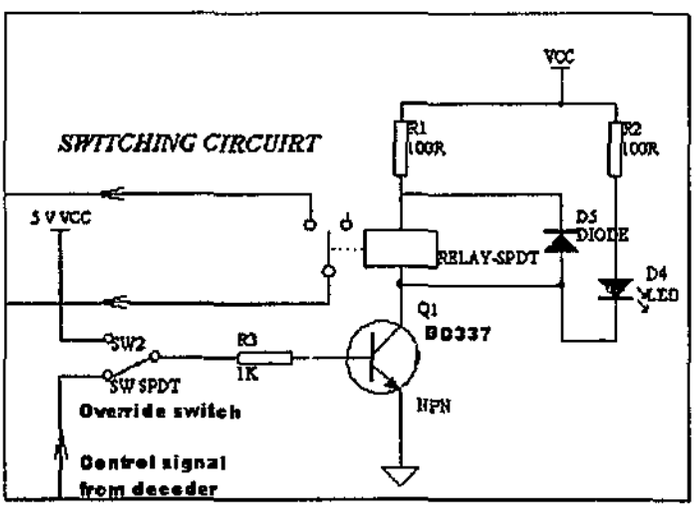


Figure 3-10. Switching circuits

d) Power Supply circuit

The power supply unit consists of a thermally protected main transformer T1 that converted 240V AC into 12V AC (rms value). A bridge rectifier BR1 and capacitor C11 convert the AC into 18 VDC used to power up the LM 1893. Capacitor C12 (100nF) is used to improve the filtering and to eliminate possible noise signals that may present in the conductor. Other electronic devices in the circuit are powered by a 5V DC derived from a voltage regulator U5 (LM7805).

Wirings and cables add inductance to the power source. This means that the power supply cannot respond quickly to high, transient power requirement. To solve this problem, decoupling capacitors are added at the key points in the circuits so that transient power can be drawn from them rather than directly from the power supply. The decoupling capacitor also reduces high frequency noise present in the power line. The value of the decoupling capacitor widely chosen is 100 nF and normally placed between the power and ground pins of all integrate-circuit devices.

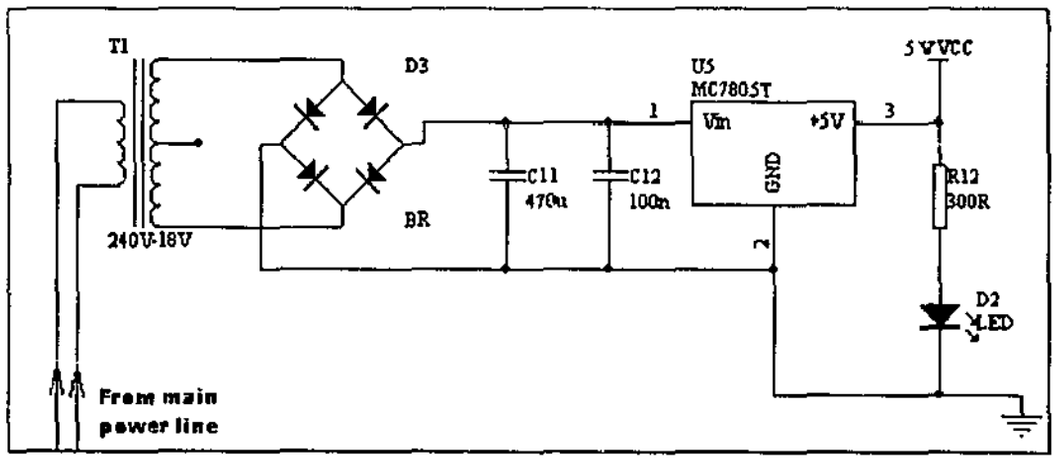


Figure 3-11. Power Supply circuits

3.5 Software design

The planning stage associated with any problem is probably the most important part of the solution. The program algorithm is a planning the solution, consists of series step-by-step instructions that produce results to solve problem, it also easily to translated in any different programming language. In this part, we will discuss and develop of the program algorithm for the project control interface and graphically represent it in the form of flow-charts.

a) Main program structure

The program consists of two levels in the module structure chart. The first level has four sub-programs namely: User login, Main Control, Configuration, and Help system.

For security purpose, the User login sub is designed so that it allows only authorised person can gain access into the program. Furthermore, only person who has master key can manipulate the user-access authorisation database.

The Configuration module contains a number second level sub-programs, in which they will let the user to edit the Devices database and setting up options such as: select different voice response, set volume, number of ring before answer a phone call and change control mod between Direct-control and Voice telephony-control.

The Help-system is designed and compiled separately with the program but is called by the main program. The purpose of the help system is that it provides information about the Tele-Control program likes how to do the configuration, setting up the controller, etc. The Help system contains information about the Speech SDK, some basic step in programming with Speech SDK, example programs, and the Tele-Control hardware details.

The Main control is the heart of the program. It controls device by sending control data to the controller via parallel port, performs speech synthesis / recognition and responses to the command from the user.

In the case of telephony control mode is select, the main control will constantly monitoring the telephone line and responses to the caller after a certain number of ring pre-defined by the user. An answering machine will be activated if the caller is unauthorised to access the program, and in this case the message from the caller will be recorded in separate directory. The program can contain a large number of messages, and it is depended on the system capacity.

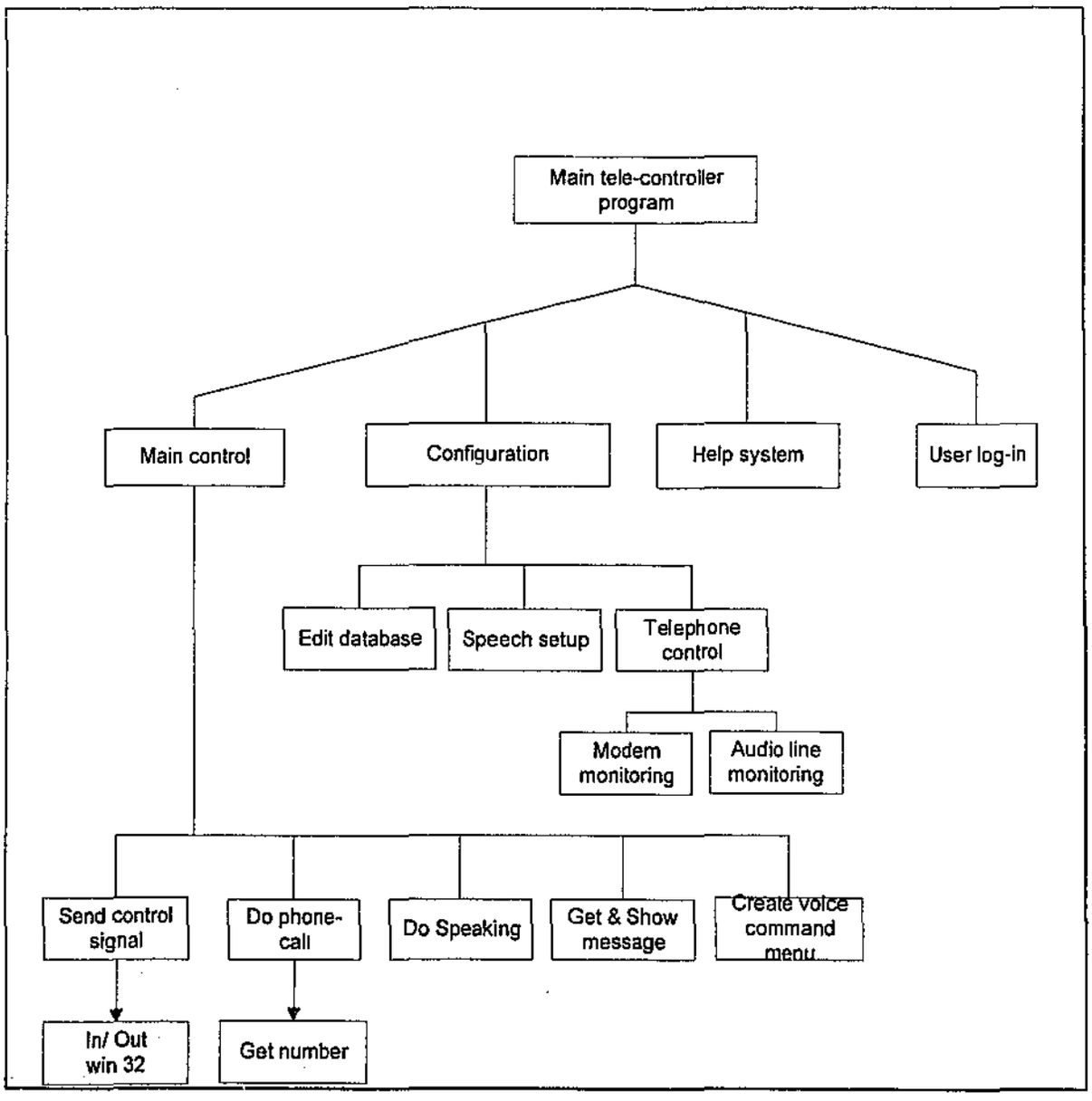


Figure 3-12. Main program structure chart

b) The Set-up Options sub routine

This sub-routine when called will display a user interface in form of Tab strips panel. It allows the user to select 4 difference control dialogs such as:

- Edit Device database: This lets the user to change the Device's name dynamically to the database.
- Select program voice response/ volume: Through this control dialog, the user can select difference program's voice responses and volume of the speaker.
- Control Mode dialog: In this dialog, telephony-control mode or direct control mode is either selected

  If the telephony-control mode is chosen, the communication and control will be done through voice modem. In case of the Direct-control mode, the user can gives command control via microphone or mouse click. Telephony-control can also be set in this mode, but in this case, the audio card is used instead of Voice modem.

Fig. 3-13 is the flowchart of the Set-up Options sub routine. The program will response to the new setting, which is done by the user right after exiting the set-up dialogs. Even if the devices' names are changed, the new name of the corresponding device should be recognised by the program when the device's name is heard. This is done by re-created, and loads a new command menu into memory used by the voice recognition engine.

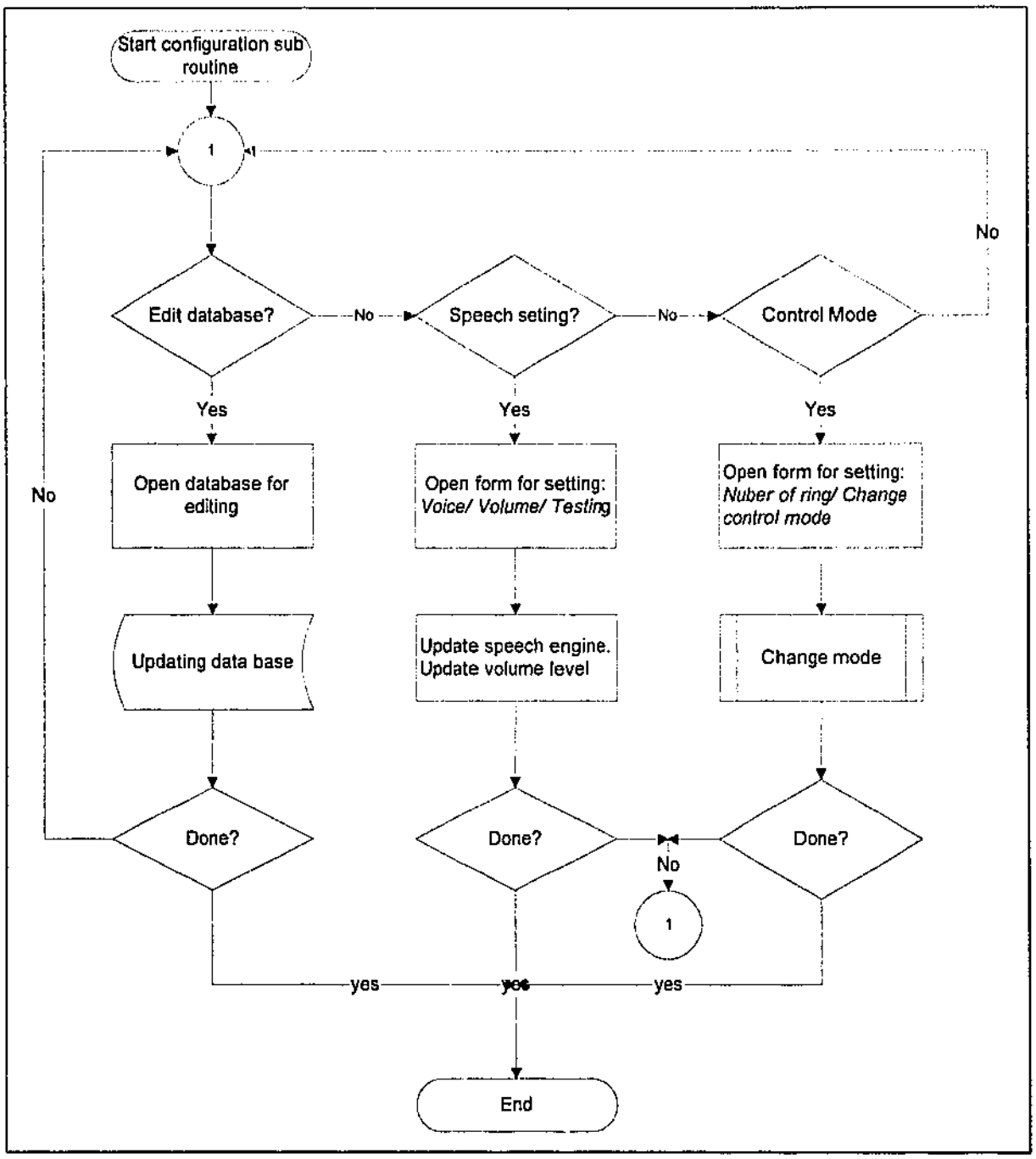


Figure 3-13 Set-up Options sub routine

c) Main Control interface sub-routine

The Main-control interface only allows controlling 16 devices, for practical purposes it can be designed to control up to 32 devices by setting 5 bits address of the receivers.

On the program control-interface, devices to be controlled are represented by their nametags with the On/ Off control buttons and its status such as time-on, time-off. The user can interact directly with the program via the mouse click event as well as using voice command.

The status of each device shall be updated and displayed when command is executed. This is done by extracting information from memory and placed in the corresponding device's position. The program continuously monitors and a response to the command events as long as it is executed. Before returns to the system, the program releases the resources used from memory, specially the speech engine. (Fig.0-14)

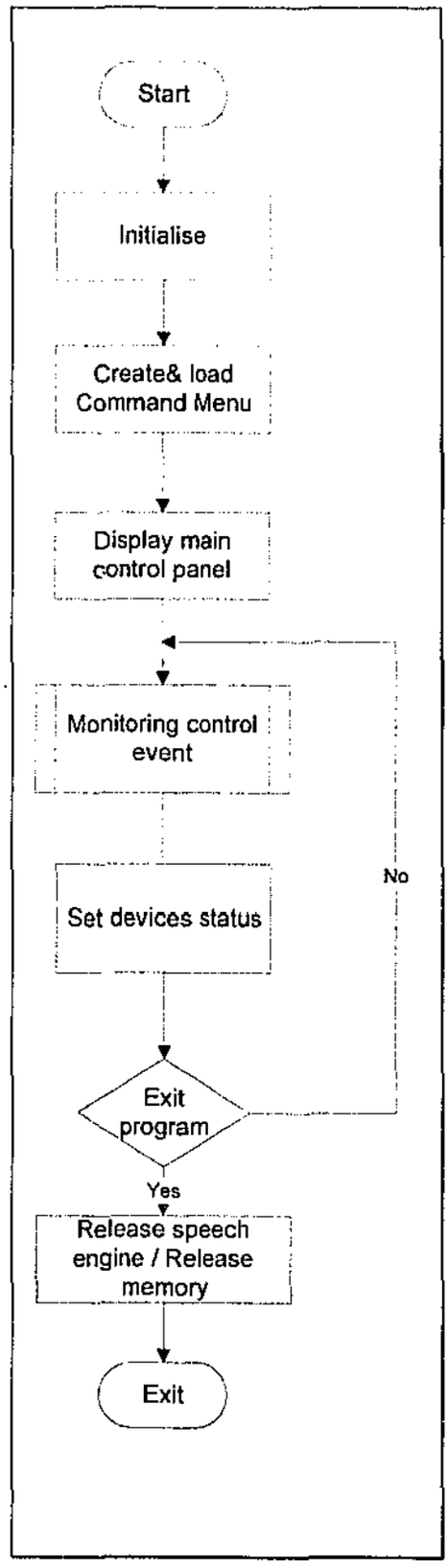


Figure 3-14 Main Control interface sub-routine

d) Monitoring Control Event Sub-routine

The main functions of this sub-routine are monitoring and responses to the control events occur during program execution such as: the On/ Off buttons are clicked or the Voice Recognition engine has heard commands.

In the case of the control button is clicked, a serial bit data stream will be send through the LPT1 port without confirming with the user. But for the case of using voice command, when the program hears the command, before executes the command this sub-routine will verify it with the user. The previous command heard will be discard if the user said "No" or another command has heard by the engine. Because of the limiting of the speech recognition technique, quality of hardware support and users speaking in manner that cause speech recognition makes mistakes, so that confirming command is necessary for any application which using voice command, particularly in the case for destructive or irreversible command such as "Format disk"

After executes the control evens, status of the device is also updated and displayed as applied bold formatting on the text of the last control event and the time of occurrent will be displayed in 'red' if the corresponding device is set active.

The flowchart of the 'Monitoring Control Event ' Sub-routine is shown in Fig. 3-15 below.

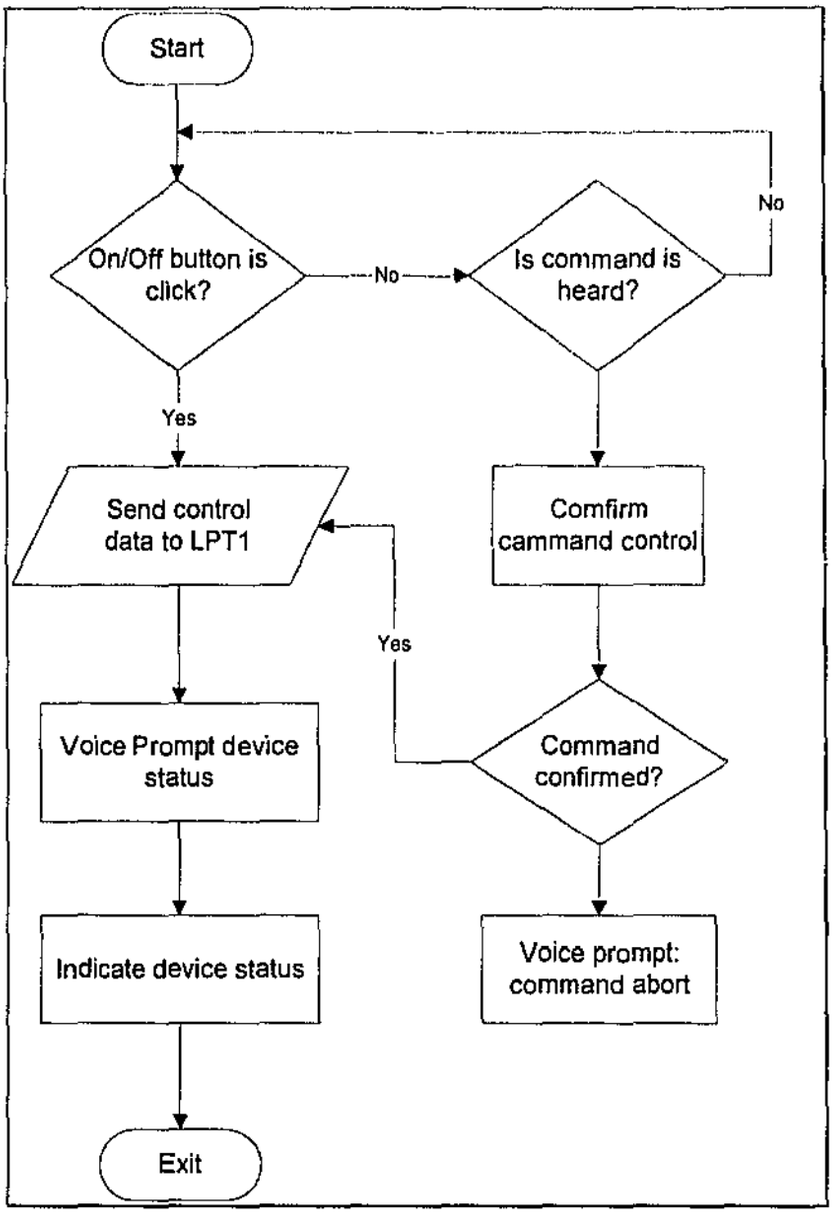


Figure 3-15 Monitoring Control Event sub-routine

e) Send Control signal sub-routine

This sub-routine will be executed when called from the Control-Even sub-routine. When executes, first it gets the index of the device to be controlled and translates to 8 bits binary number represent for the controlled device. Depends on the control event, bit 6 in the data stream, which is the control bit is set to either zero (0) or one (1) according to the Off or On demand of the Control event.

Although, the parallel port is used for communicating, but only three (3) bits data and one (1) status bit of the port are actually used for monitoring and controlling. The

signals are sent serially through the port form the program at DB0 with the clock signal at DB1 to shift register (74LS164) of the hardware interface.

For telephony control mode, pin 13 of the port is used to indicate the select condition; here it is used to detect the ring signal on the telephone line. The voltage level on this pin will be pulled low for every incoming ring signal from the phone line. After a certain ring, the Off-hook signal is send from the program to the controller via pin DB2 of the port. The phone line will be on-line as long as the voltage level at DB2 is in high state. The status of the line now is monitored by another sub-routine called 'Monitoring Telephone line'.

Both two control-modes use the same function to send the control signal. The difference here is in the telephony-control mode, the program uses of pin DB2 and SLCT (pin 13) of the port to monitors the telephone's line and get the voice commands. The flowchart of this sub-routine is shown in fig. 3.8

Digital signal can be easily send out through the Parallel port with the help of a dll function called In-Out32.dll.

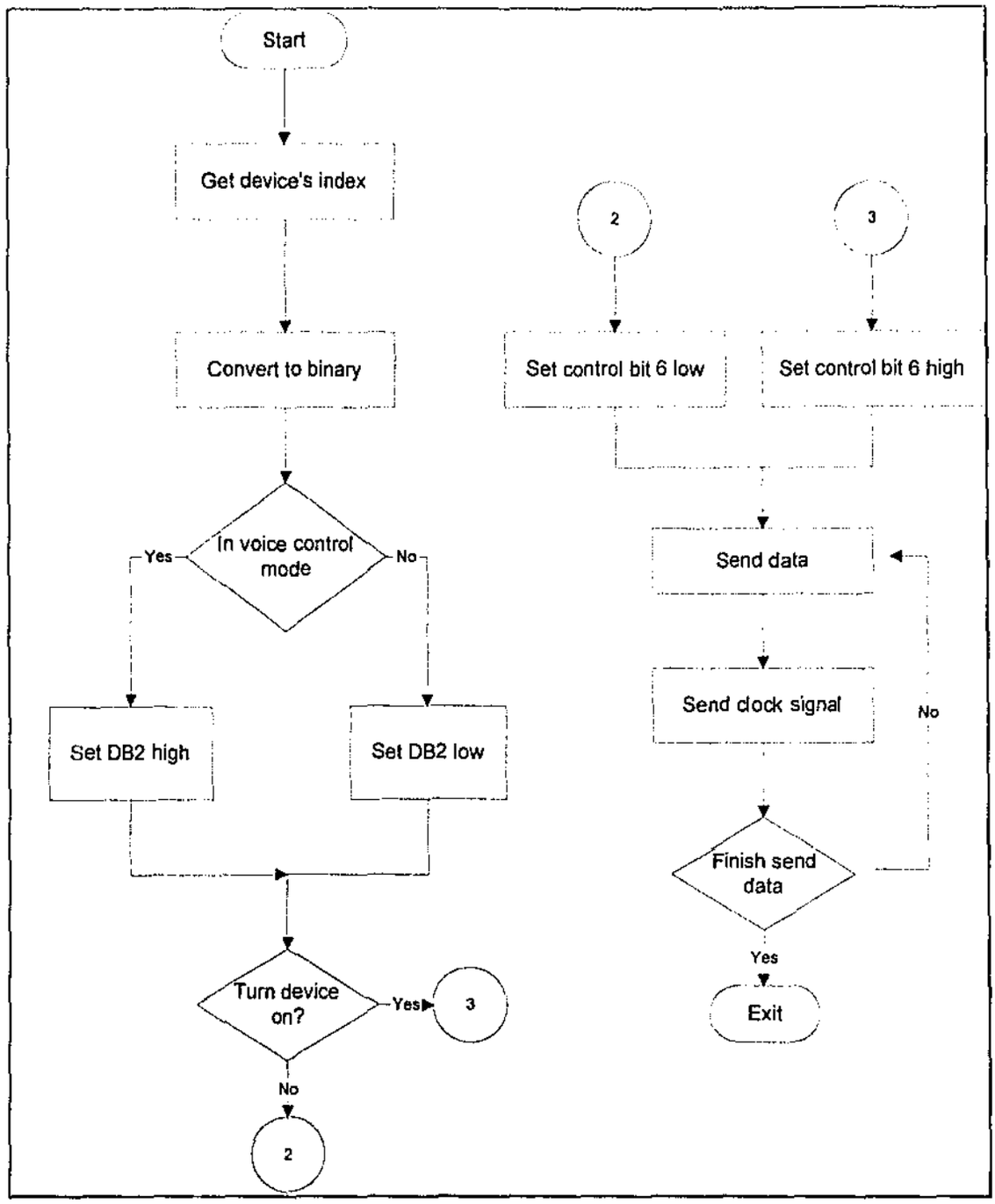


Figure 3-16 Send Control signal sub-routine

f) Telephony Control sub-routine (through modem)

There are two difference telephony control methods in this project, one operates based on the built-in telephone interface circuit and the other uses voice modem. Fig. 3-16 is the illustration of the Telephony control sub-routine using modem.

By using modem as the telephone interface it allows the program performs more functions, which are provided by Win32 API (Win32 Application Programmer Interface) and TAPI (Telephony Application Programmer Interface).

When this type of control mode is selected, the program will invoke an API function *CallDialog* and starts to listening to the attached phone line through communication port via modem. A dialog box also brings up for the user to stop the application or monitor the line is in used. This dialog box does not return until the user decides to stop monitoring by pressing the associate button.

After a certain number of rings, the Off-hook signal will be sent out and the line is connected. As illustrated in fig. 3-17, a message is sent to the caller and requires providing password.

If the program detects an incorrect password or a silent message after thirty seconds, another voice prompt asking the caller to leave message. The program now works just like a normal answering machine. The user can play back the message from the pop up directory. This is automatically created and display when selects this type of control mode.

In case the control program detects a correct password, first the program will report all the device status and then waiting for the command from the caller. The caller will be asked to verify correct command recognised by the program before execution. The phone line will be disconnected if the program detects thirty seconds idle time. The idle time is defined as the silent duration starting from the end of utterance.

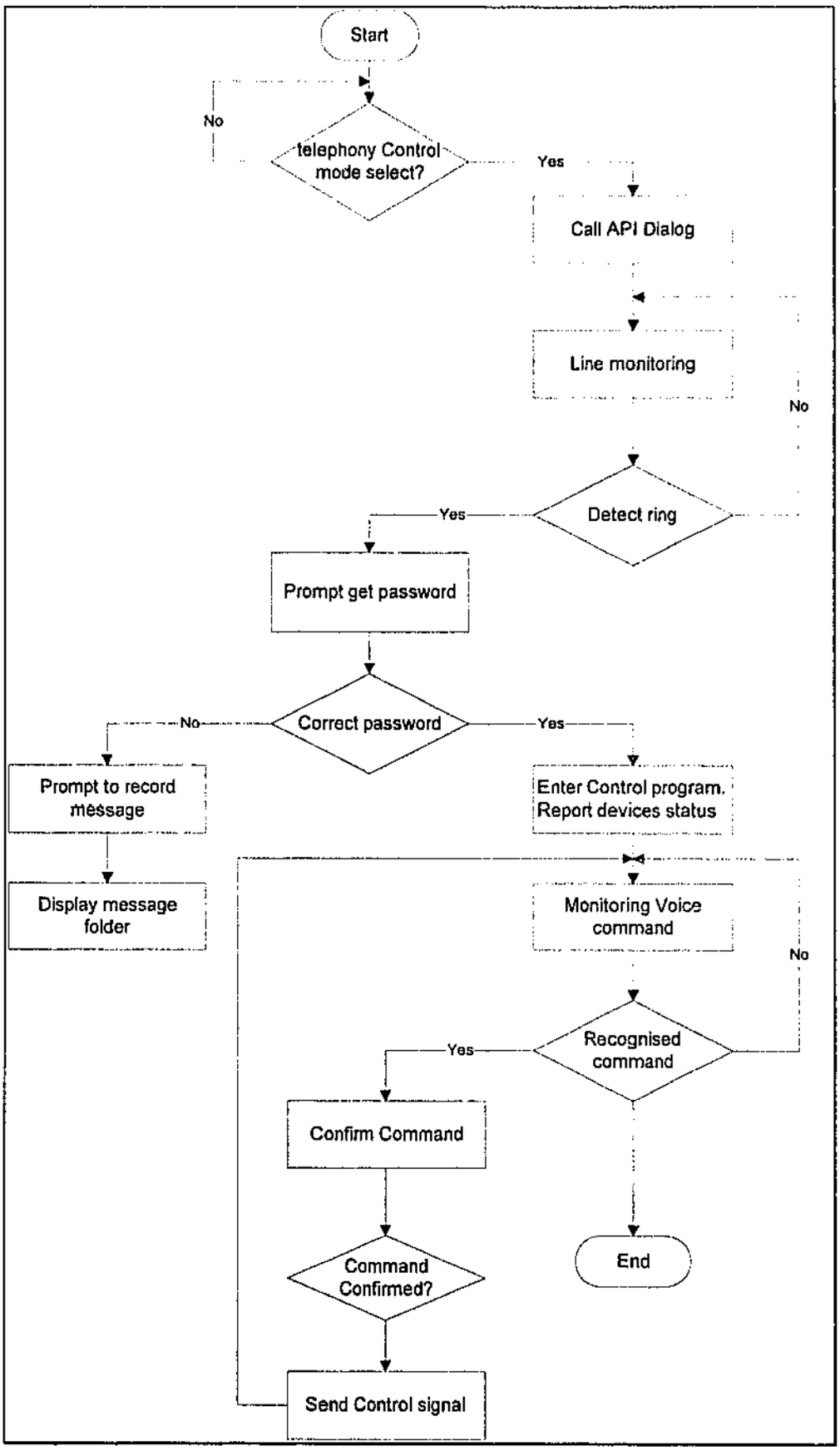


Figure 3-17 Telephony Control sub-routine (using modem)

g) Telephone line sub routine (using built-in hardware interface)

This sub routine is basically similar to the one described in 3.37, but in here the communication takes place via a built-in hardware interface module and the audio card. However, I have decided to limit it performance functions such as: request password, record caller message, accepting DTMF signal. This is because to providing such functions, large number of codes will be added to the program, and requires more computer resources and time to develop.

This routine is an experiment of using an ActiveX user control component of the Visual Basic program. A complete telephone line monitoring is built as a single control component. The user can monitors, disconnects the line when it is set to be active.

The program has preset values of number-of-ring before answer and idle time duration, but they can be set to a suitable value during program execution.

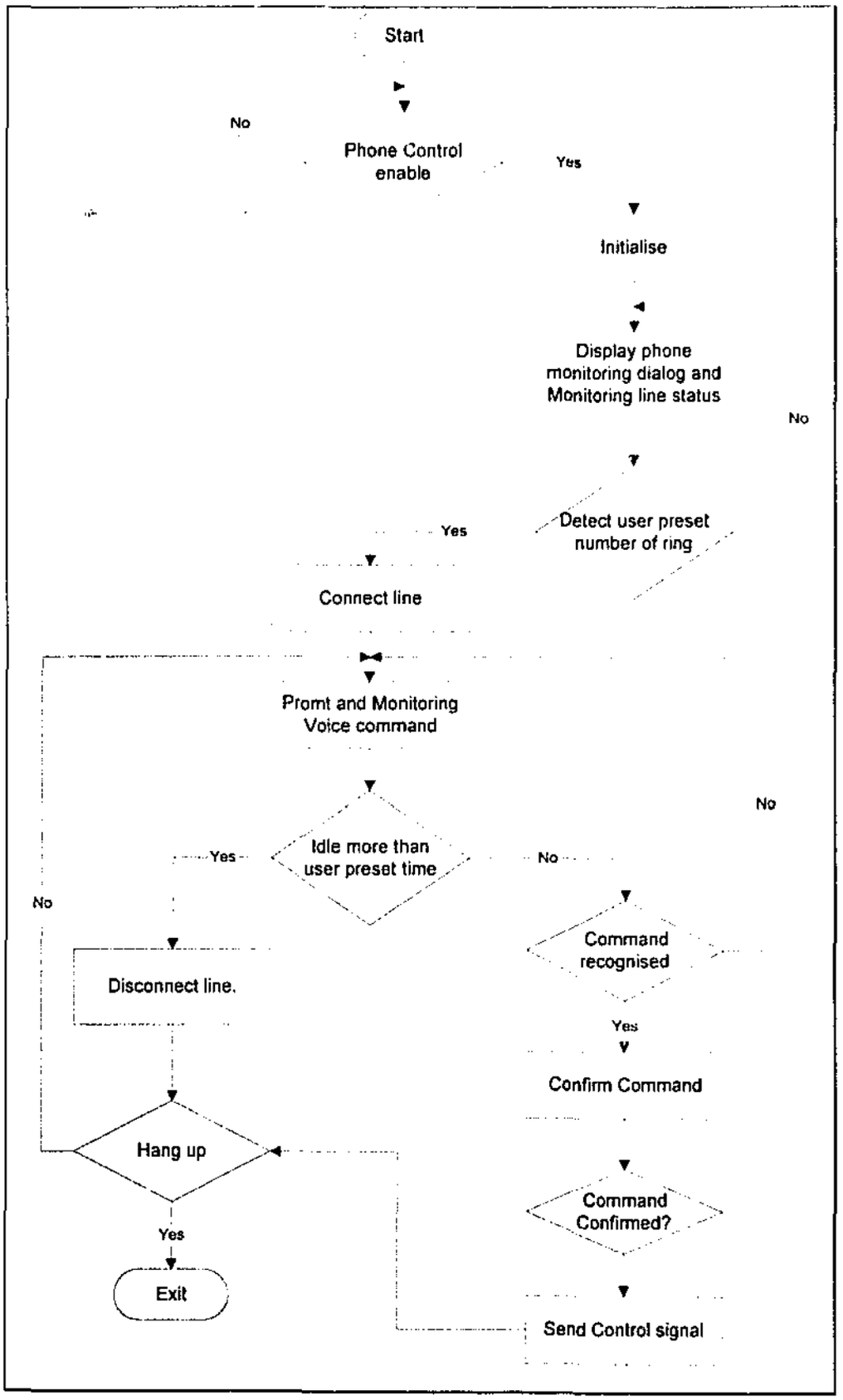


Figure 3-18 Telephone line sub routine (using built-in hardware interface)

h) Idle time detection sub routine

Idle-time detection is one of the functions of the Telephone line monitoring ActiveX component. When active, this sub routine will continuously check if there is a message sent from the main program about the start or end of the utterance, which is detected by the Voice Command API.

A time counter is used for measuring the idle state length in seconds. Whenever the utterance starts, the counter will be reset to zero, and starts to count up if the end of utterance is detected. At the set point, the program sends an off hook signal to disconnect the line. During the counting state, if the start of the utterance is detected, the counter then is reset immediately and the whole process is repeated.

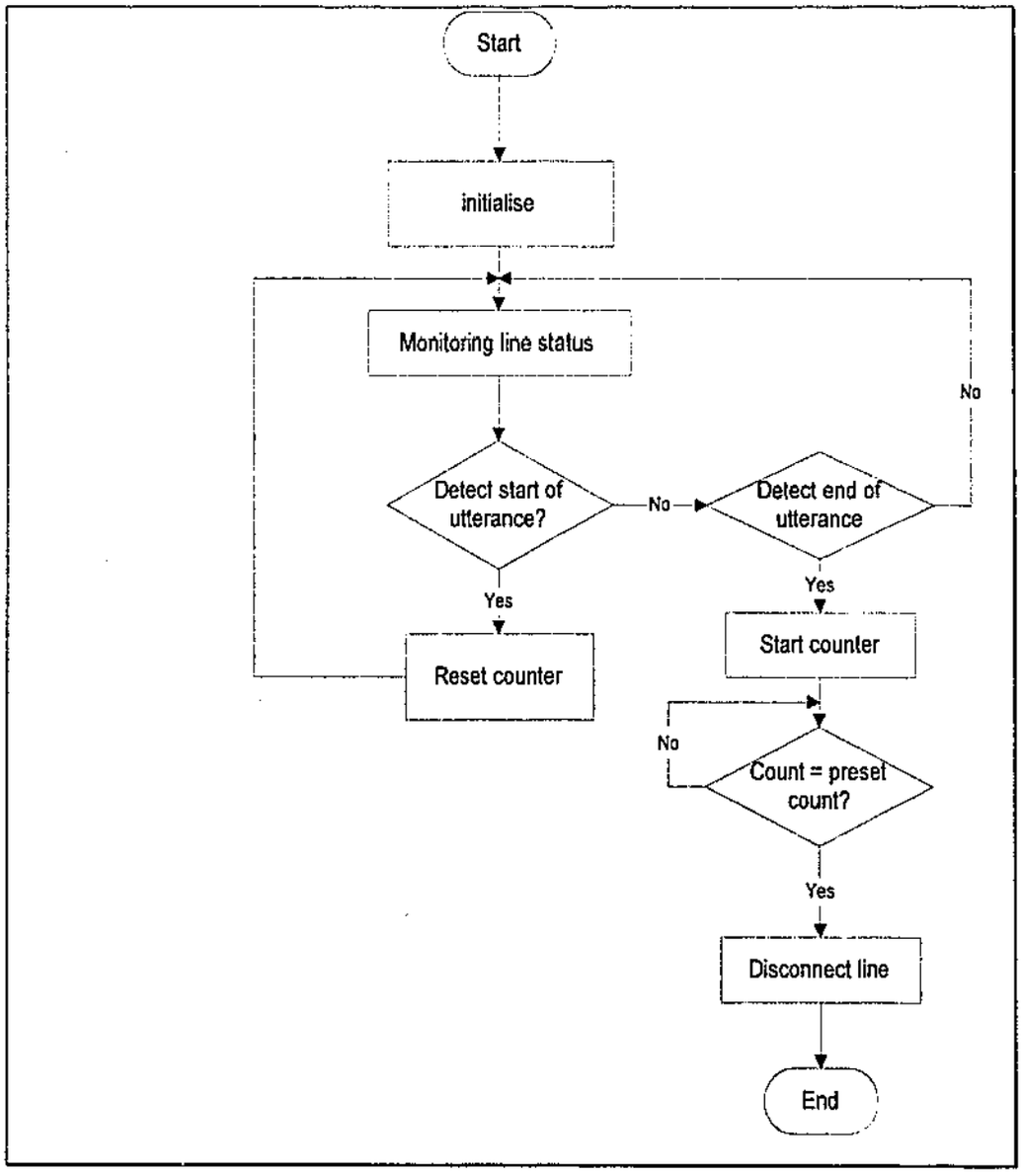


Figure 3-19 Idle-time detection sub routine

i) Synchronisation of voice recognition and Voice synthesis (apply for half duplex sound card)

The speech recognition and speech synthesis share together the same speech engine. Therefore, an application cannot speak and listen to the command at the same time. For system, which is installed a full-duplex sound card; there is no need to be concerned about synchronising between recognition and synthesis of the speech APIs. However, if the systems have installed half-duplex audio cards, synchronising the two above must be taken into account when developing voice application programs. This constrains is used to cause problems whilst designing this program, the system will crash easily just by simple mistake. This is quite difficult to avoid in case of development large, complex applications and lack of experience.

In fig. 3.14 below, the process of speech recognition and synthesis is a closed loop. When the synthesiser method is activated, the recogniser needs to be de-activated and vice versa. They can be set to activate or de-activate through their properties '**.Deactivate**(*Menu*)' or '**. Activate**(*Menu*)'. When either of them is set de-activate, they stop listening or speaking and release the sound card and system resource. If a method is called and the corresponding recogniser/ synthesiser has not yet been activated, an error is generated.

Synchronising the voice recognition and voice synthesis is better be done by using their *Events* such as **.SpeakingDone**( ), **.SpeakingStart**( ), **.CommandRecognise**( )

It is found that whenever the error is generated even through by some other causes, there certainly is a corruptions in system resource, reset the system is necessary in this case and better than try to end the running tasks via the Ctr-Alt-Del.

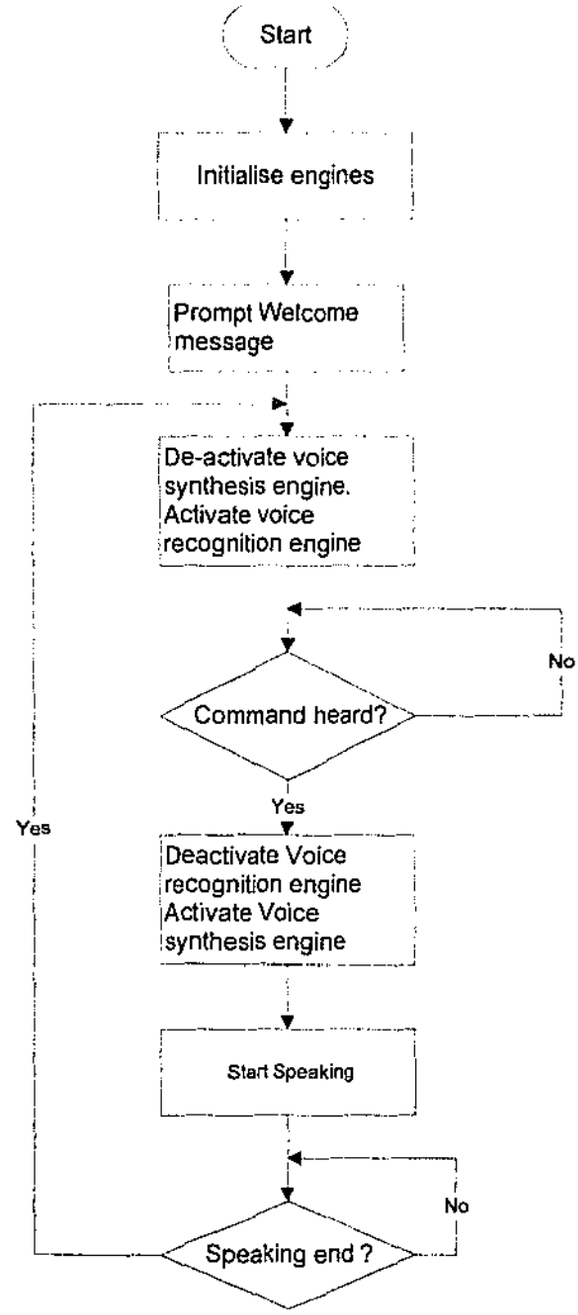


Figure 3-20 Synchronisations of voice recognition and Voice synthesis

Part 4 Implementation and testing

4.1 Software implementation

4.1.1 Object s in VB Visual Basic

Visual Basic provides the fastest and easiest way to create applications for Windows environmental it has a complete set of tools to simplify rapid application development.

The "Visual" part refers to the method used to create the graphical user interface (GUI). Rather than writing numerous lines of code to describe the appearance and location of interface elements, you simply add pre-built objects into place on screen.

The "Basic" part refers to the BASIC (Beginners All-Purpose Symbolic Instruction Code) language, a language used by more programmers than any other language in the history of computing (Microsoft's MSDN Library, 1999). Visual Basic has evolved from the original BASIC language and now contains several hundred statements, functions, and key words, many of which relate directly to the Windows GUI. The objects in visual basic are:

- Forms: Forms are objects that expose properties, which define their appearance; methods define their behaviour; events define their interaction with the user. By setting the properties of the form and writing Visual Basic code to respond to its events, the programmer customises the object to meet the requirements of an application.
- Controls: Controls are objects that are contained within form objects. Each type of control has its own set of properties, methods and events that make it suitable for a particular purpose. Some of the controls, the programmer can use in an applications are best suited for entering or displaying text. Other controls allow access other applications and process data as if the remote application is part of the program code.
- Object Arrays: is a group of the same type of control or form sharing the same name and events.

- System objects: Visual Basic provides several special objects that are neither forms nor controls. These include the Menu, Clipboard, Debug, Screen and Printer object. Each of these objects allows the programmer to access some underlying capability in Windows. These objects can also have properties and methods associated with them.

  - Properties, Methods, Events: Visual Basic forms and controls are objects, which expose their own properties, methods and events.

  Properties can be thought of as an object's attributes.

  Methods are very much like regular language statements, but they act directly on an object. Each method may have one or more arguments detailing specifically how the method will operate, eg. **ObjectName.MethodName** [*arguments*]

  Events are predefined VB procedures and can be as its responses when the user or program code performs some action on a control. Each control has its own unique set of events. Each event reacts to a different action.

4.1.2 Main Program functions

Having described about Speech SDK and Visual Basic program, in this section, we will focus and discuss on implementing Voice command in VB plus some of the main program's functions by code examples such as: Send control signal, Get number, Monitoring telephone line. Refer to Appendix 1 for complete program coding.

a) Initialise Speech Engines

Listing 4b.1 Initialise Speech engines and Load Main control GUI. (Abstracted)

```
'*************************************************************************
'Sub-procedure name: MDIForm_Load
'Purposes:           Loads Main control interfaces, and initialises engines.
'Function called:    LoadNewDoc, Menu_Create, SetVol(volCtrl.lMaximum,
'                    Speech_SayIt.
'Corrupted Global variables: none
'Input:              LoadResStrings
'Output:             Menu Files
'*************************************************************************

Private Sub MDIForm_Load()
Dim i%

    IdleTCounts = 0
```

```
        LoadResStrings Me

        engine = TextToSpeech.Find("Mfg=Microsoft;Gender=1; Style=4"
'Select the engine, SAPI style. This is synonymous with
'doing TextToSpeech.CurrentMode = engine
        TextToSpeech.Select engine

' Open the mixer with deviceID 0.
        rc = mixerOpen(hmixer, 0, 0, 0, 0)
                If ((MMSYSERR_NOERROR <> rc)) Then
                        MsgBox "Couldn't open the mixer."
                        Exit Sub
                End If

' Get the waveout volume control
        OK = GetVolumeControl(hmixer, _
        mixerline_componenttype_dst_speakers, _
        mixercontrol_controltype_volume, _
         volCtrl)
        Call SetVol(volCtrl.lMaximum / 2)
        Call LoadNewDoc

'Initialise Voice command engine
        Vcommand.initialized = 1
'Create Voice command menu
        Call Menu_Create(gMyMenu)
'Engine must be enable and command menu must be activated
        Vcommand.Enabled = 1
        Vcommand.Activate gMyMenu

End Sub
```

Discussion

The main tasks of this Sub-procedure are to create System file Menu, selects and initialises speech recognition and speech synthesis engines used when the program is executed.

The program's pull down menu is assigned at run time and done by loading the corresponding data string stored in 'Resource String' to each sub-menu.

The Text-to speech engine is select by create an object to handle the object passed from the method **TextToSpeech.Find**(RankList). This object is represented the engine found by the above method, it is then selected by the method **TextToSpeech.Select** engine ans used by the program.

The string "**Vcommand.initialized** = 1" is use to initialise the Speech Recognition engine, which loads the engine into memory.

However, in order to use the Voice command, a Voice menu object must be created and activated to represent a voice menu for this application. This is done through two function calls: create menu by Call **Menu_Create**(*gMyMenu*), then activate it by **Vcommand.Activate** (*gMyMenu*).

Detecting the existing of Audio device interface is also importance, as the program is designed mainly based on the use of Audio Interface. In the above example, this is detected by method **mixerOpen( ),** if an error returned by the method this mean the program can not detect the Audio device in the user system.

b) Create Voice Menu Command

Listing 4b.2 Create Voice Menu Command

```
'*********************************************************************
'Function name:          Menu_Create
'Purposes:               Create voice command menu from the database.
'Function called:        None
'Corrupted Global variables:  none
'Input:                  Device Database, field "device name".
'Output:                 gMenu (Voice command menu for the program control)
'*********************************************************************
Sub Menu_Create(gMyMenu As Long)
Dim X%

gMyMenu = fMainForm.Vcommand.MenuCreate(App.EXEName, "state1", 4)
  fMainForm.Data.Recordset.MoveFirst

  For X% = 0 To 17
      fMainForm.Vcommand.AddCommand gMyMenu, 1,_
      fMainForm.Data.Recordset.Fields("Device_Name"), "when you say"+_
      fMainForm.Data.Recordset.Fields("Device_Name"), "listen list", 0, ""
      MainForm.Data.Recordset.MoveNext
  Next X%

End Sub
```

Discussion

This function creates a voice menu for the Voice Command, and is done by the use of the Voice Command Control property '**Vcommand.MenuCreate**(*Application. flag)*'.

Devices' name, which have been stored in the Devices database are extracted and added to the list of command strings in the Voice menu data base one after another. Thus if the Voice recognition engine has heard the command, which matched one in the command set in the Voice menu, an event will occur and the program then can take an appropriate action.

The Voice menu must be activated before using Command-recognise to issue an event control, if not an error will be generated and the system may be "crashed".

c) Do Speaking

Listing 4b.3 Synthesis text

```
'*********************************************************************
'Function name:        Speech_SayIt
'Purposes:             Synthesise text string.
'Function called:      None
'Corrupted Global variables:  none
'Input:                String
'Output:               Spoken words
'*********************************************************************
Sub Speech_SayIt (ByVal szValue As String)

  fMainForm.TextToSpeech.Speak szValue

End Sub
```

Discussion

This is the simplest function used in the program and consisting only one line of code. The Voice Text property '**TextToSpeech.Speak** (*string*)' takes the string value from the calling program and just translates that to the audible sound.

In using half-duplex audio card, detecting Speaking Start and End of the Speech synthesis must be done in order to synchronise both the Speech recognition and Speech synthesis.

d) Recognise Command

Listing 4b.4 Command recognition

```
'*********************************************************************
'Procedure name:       Vcommand_CommandRecognize
```

```
'Purposes:              Generates Event.
'Function called:       Speech_SayIt, Control
'Corrupted Global variables:  S      'Device Index detected by this procedure
'Input:                 Command      'heard from user
'Output:                Audible sound
'*****************************************************************
Private Sub Vcommand_CommandRecognize(ByVal ID As Long, ByVal CmdName
As String, ByVal Flags As Long, ByVal Action As String, ByVal NumLists As Long,
ByVal ListValues As String, ByVal Command As String)


Dset = False
Heard = False

With frmDocument
        For i% = 0 To 17                ' compare with 18 command in voice menu

        Select Case Command

         Case .DevName(i).Caption:

                If i <= 15 Then         ' index of device from 0 to 15
                    S = i
                    Call Speech_SayIt("You selected device name " &
                    .DevName(i).Caption & " yes or no ?")
                    Heard = True
                ElseIf (i = 16) Then    'invisible command string in database: "Yes, _
                                        'No"
                    Dset = True
                    Heard = True
                    Call Speech_SayIt("Heard Command")
                    Call Control(S, Dset)
                End If

         End Select
        Next i%
 End With

                If Heard = False Then
                    Call Speech_SayIt("Please! Try again")
                End If
End Sub
```

Discussion

This procedure enables the program to respond to the user's command when the recognition engine heard and recognised a spoken phrase. This is done by using

CommandRecognise (..) property. This property creates an event when a command is recognised from the assigned command set.

When the device's name is recognised as defined from the data set, the program must first request verification from the user before processing the command by calling the **Control** ( ) function. In the above procedure, the heard command will be repeated by confirm with the user via Yes/ No command string.

Identify the recognised command can be done via other parameters such as use the data in *Action* (a string parameter that contains action data to accompany the recognised command, and are application specific), or the identifier *ID* (Identifier of the command that is recognised). But in this case, these parameters can not reflects the name of the device to be controlled.

4.2 Hardware implementation

Like many other practical projects, implementation of hardware required to go through number of steps from initial stage as construct and testing prototype circuit to testing and modification final assembly circuit. This section will explain and discuss the processes of implementing the hardware as part of the project.

4.2.1 Prototyping

This is the initial and also an important step of the process implementing the designed circuit.

Partial functions of the circuits are built and functional test on the Pre-drill prototype boards such as: decoupling circuit, serial to parallel converter, encoder, switching circuit. A few modifications are made and updated on the schematic diagram.

4.2.2 PCB design

After testing and update the circuits, PCB now can be designed and made. The PCB art works are done using software package named Protel Design System version 3.5. The following are steps of designing the PCBs:

- Step1: Create schematic diagram from the schematic editor package. At this stage, wiring and connection between components must be made. Assign components Footprint, which is PCB patterns for the component and will be used for Net List generation. As the PCB Design package has a reference library with

components' pattern setting based on industrial standards. The names of the patterns in the PCB library are referred as the Footprint for the selected component on the schematic diagram. For example, resistors used on the circuit have power rating of 0.25W; the pattern in the PCB library AXIAL 0.3 can be used as Footprint for them.

- Step2: Create Net List

The Net list is an ASCII text file in the Protel format. The typical Net list format includes descriptions of parts, such as the designator and package type combined with the pin-to-pin connections that define each net.

- Step3: Loading Net List on the PCB design and create PCB artwork.

Few jumpers and wire links are made on the PCB, as there are few crossing connections as single-sided boards are used. The physical layout of the components is arranged according to their group functions for easy testing purpose.

Conductor consideration: due to dielectric breakdown. A 1.8mm conductor's width applies for the high voltage from the main power line, 0.3mm conductors widths are for all other low voltage and digital signals activities.

4.2.3 Circuit construction

To produce a functional circuit, individual components are assembling on the PCB and soldered component leads to the etched copper conductors.

For easy managing in soldering the components to the board, the lower profile components such as resistors, diodes are assembled first and then the IC and so on. Transformer is placed last and firmly on the board and check polarity before soldered as if remove the transform, it will require much more heat to the soldered leads than other components and that may damage the pads.

The 25 pins "D" type chassis mounted connector is connected to eight pins female terminal connectors and two telephone-socket wires.

Final checking the construction circuit is carried on. There are two important steps:

Firstly, to ensure there is no dry joint, solder joint or wrong component oriented and position.

Secondly, continuity tests thoroughly and ensures no short circuit on the voltage supply.

4.2.4 Testing

As most of the project 's functions have been prototyped and tested at the initial stage of the design process (which are the Computer interface, Serial to Parallel converter, Encoder, Power supply, and the Switching circuit) before construction takes place. At this stage, the circuit functions must be tested as a whole system rather than being isolated and tested individually. The following is the testing procedure:

a) Visual inspection:

Before supply power to the circuits, final visual inspection is done to ensure components have been correctly assembled on boards, no short circuit on the PCB design, nor solder bridge.

b) Power Supply test:

After visual inspection, the Transmitter is power up. Voltage level at the output of the regulator and individual IC at its power-input pin is checked if a +5volts supply presented. The voltage at pin 15 of the LM 1893 must be 18 volts.

c) Testing Computer interface:

A communication cable is connected to the PC with the simple control program, which designed for testing purpose.

Digital signal is sent to the controller periodically. The voltage levels at pin 4 of IC4, IC8, IC1, IC7 are observed with the use of a digital Oscilloscope to check if the De-coupling circuit working in correct manner.

d) Serial to parallel converter and Encoder functions:

The output of the IC4 (represent the Clock signal) and the output of IC8 (is a Control Data) are fed to the shift register. The test-program now needs to send data only one bit at a time.

The output of the 74LS164 (Qo to Q7) is checked to see if the data correctly shifted. There is problem at this stage due to this device does not working as specified in the

data sheet provided. A small modification is made on the circuit to reduce the supply voltage to this device.

The Encoder output signal at pin 15 is also captured on the oscilloscope. If the device function correctly, the encoded data on the screen of the oscilloscope will be a string of short pulses and longer pulses depend on the digits at the input of the device. The short pulse represents for the encoded logic zero, and the longer pulse represents for the encoded of logic on1.

e) Modulation function

This test is done following the procedure provided by the manufacture.

First, trim Fo by putting the chip in the TX mode, setting a logical high data input, and measuring the TX high frequency, 1.022 * Fo. Adjust R O on pin 18 for F = 1.022Fo = 127,750 Hz.

Second, the line transformer is tuned. The chip is placed in the TX mode, a resistive line load is connected to disable the ALC by reducing tank voltage swing below its limit. FSK data is then passed through the tank so that the tank envelope may be adjusted for equal amplitude for high and low data frequency by adjusting the slug in the isolation transformer T2.

This test is also repeated on the receiver circuits, and the LM1893 of the receiver need to be set in transmit mode (Pin5 at logic high) in order to carrier the test.

f) Override function

By switch the SW2 to override position, the relay on the receiver circuit must be energised and the LED 4 should be turned on, which means the relay is functioning.

g) Final Testing

In the final test, both the Controller and Receiver are plugged in to the main sockets and the control program is executed.

The control data need to be sent periodically to the Controller during the final test. In order the system to work, the two waveforms, one at the output of the encoder of the controller and one at the decoder of the receiver should be exactly the same. This is done by tunning the slug in the coupling transform of both the transmitter and receiver. At certain point of adjustment, the two waveforms are identical and the relay will be energised if the address sent from the Controller matches with the local address of the Receiver.

4.3 Problems encountered and solutions

There are quite few problems occurred over the period of design and implementing the project:

4.3.1 Problems in designing and development of hardware:

Some of component values derived from the technical data are not available from suppliers, such as 50 K resistors. 5,100 pF. Therefore, we have to find what available and can be used from the supplier and modified the circuit. For example, to achieve 50 K resistor, two 100 K resistors must placed in parallel.

The PCB is not good in quality, so that few pads and tracks are damage when modify and change components on the PCB.

During testing phase, it is found that the shift register (74LS164) does not work with the 5volts supply specified from the technical data. Even through, this circuit has been prototyped, tested separately and working fine with supply voltage from the parallel port. The controller is then modified, a variable resistor is added to the circuit and adjusted the supply voltage to the 74LS 164 down to about 4.2 volts. This is also done on the IC4 (opto-coupling device), which provided clock signal to the 74LS164.

A 10K resistors pull-up is added to the circuit to between the Encoder and Modulator in the Controller and between the Decoder and Demodulator at the Receiver to provide adequate pull-up current drive for the encoded data.

4.3.2 With designing Control program

Designing the program is the most difficult and troublesome part of the project implementation. This is because of some reasons as follows:

1. Not satisfied the requirement of using the Speech SDK package in term of knowledge, such as C++, experience with COM (Component Object Model), Object Linking and Embedding (OLE), and understanding Win32 application programming interface (API).
2. Not known programming in Visual Basic before start the project.
3. There has been no textbook found or documents available discuss about the Speech SDK, the only one we could use here is the manual provided with the SDK. Microsoft has an On-line help MSDN library, however the document is also a manual and wrote for Visual C++ reference.

System often crashes during development phase, and the main reason is miss synchronisation between the speech recognition and speech synthesis engine. The speech engine functions will not work after the program is reloaded if previously exit the program does not completely unload the engine and there is no error indicated.

Part 5 Conclusions

This project has been carried out under the supervision of Dr Daryoush Habibi. The principal aims of this project are to design and implement a PC-based voice-command control system, which allows electrical devices to be controlled remotely. The system allows the use of voice command as well as standard input devices such as PC mouse and keyboard. The project employs the speech development kit (Speech SDK) developed by Microsoft and the Carrier Current Transceiver LM 1893 designed by National Semiconductor.

The project is motivated by the fact that many existing controlling circuits suffer from limits such as cable length and signal interference. LM 1893 Carrier Current Transceiver is an attractive device that can be used to overcome such limits by using the existing main power line as communication medium. In addition, systems that accept voice commands have become increasingly popular in areas such as telephone banking, passenger intercom on trains and voice mail. Microsoft Speech SDK 4.0 is a programming toolkit for speech synthesis and recognition that has been widely used in anticipation that user interface using speech will soon be intergrated into PC environments. Speech recognition is an essential component of any application that requires hands-free operation; it also can provide an alternative to the keyboard for users who are unable or prefer not to use one. Users with repetitive-stress injuries or those who cannot type may use speech recognition as the sole means of controlling the computer.

The Speech SDK contains six ActiveX controls for speech recognition, dictation, speech synthesis, and telephon, which can be used in Visual Basic applications. *Voice Commands* and *Voice Text* provide high-level interfaces to the speech engine. These interfaces are somewhat limited slower than the direct APIs. Howeve, they provide automatic resource (e.g. memory) allocation and out-of-process sharing between voice applications.

The *Telephony Application Programming Interface* (TAPI) is one of the most significant Speech APIs. The telephony API model is designed to provide an abstract layer for access to all telephony services across Windows platforms. The aim of TAPI is to allow programmers to write applications that work regardless of the physical telephone medium available to the PCs. Applications written using TAPI to gain direct access to telephone-line services work the same on analog or digital phone lines.

Applications that use TAPI can generate a full set of dialling tones and flash-hook functions (like that of the simple analog handset found in most homes), and can also communicate with sophisticated multi-line digital desktop terminals used in high-tech offices.

5.1 Recommendations of applications and further directions

5.1.1 Applications

The system allows the user to control remote electrical devices by entering voice commands. These commands are interpreted by the program into control signals, which are then sent to the remote devices through power line. As a result, the system is useful in domestic settings, particularly for those having difficulty with movement such as the elderly or disable. The system is also suitable for industrial applications such as building energy management or where wiring, decoration and high signal interference are issues of concern.

5.1.2 Further directions

At this stage, control can be carried out in one direction. Extra time-switching circuits can be added to both the Controller and the Receiver to reverse their function so that two-way communications are possible. This is simple if only two parties are involved, but in the case of multiple transmitters and one receiver, timing and identifying the message source can be difficult.

The LM 1893 is designed for use with low line impedance, and the carrier input and output are tied together on the same pin (pin 10). Furthermore, the coupling transform serves as a filter for incoming signals. Effected by these factors, matching impedance between system and the line is uncertain. The signal power loss due to mismatch will be varied depending on the type and condition of the line. Increasing the transmission range can be accomplished by the use of repeater.

The LM 1893 can be replaced by another device of the same familyLM 2893, which allows matching impedance on both transmitter and receiver to increase transmission range and receiver sensitivity as this device has separate receive/ transmit paths.

The software component is currently a working protope. However, Visual Basic code can be optimised further to enable more efficient use of system resources. In conjunction with hardware modifications, extra functions can be added to the program such as controlling camera, video capture and displaying device locations on a map. The system will then be suitable for security applications.

Telephony API provides much more functions than what has been used in the project. Running on powerful PCs, the program can be implemented as a *PBX server* or a digital answering machine, whilst still remain its main control functions. This approach is promising because of the high cost of PBX and digital answering machines.

Appendix

Appendix 1: Tools and equipment used

Equipment

1. Soldering station.
2. Pre-drilled prototype Board
3. Digital Oscilloscope
4. DMM
5. Hand-drill kit

Software tools

1. Visual Basic V6.0
2. HTML compiler
3. Speech SDK 4.0
4. Protel Schematic-Diagram Design
5. Protel PCB design
6. Microsoft FrontPage

Appendix 2: Part lists

Part Cross Reference Report For Receiver

Designator	Component Library Reference Sheet
C1	Ctc
C2	10n
C3	560p
C4	100n
C5	100n
C6	1n
C7	100n
C8	33N
C9	220n
C10	220n
C11	470u
C12	100n
C13	100n
D1	Bridge rectifier diode
D2	TRANZORB SA40
D3	1N4148
D4	LED
D5	1N4148
JP1	HEADER 3
JP3	HEADER 3
Q1	NPN
R1	100R
R2	100R
R3	1K
R5	50K
R6	100K
R7	4K7
R8	3K3
R9	10K
R10	5K6
R12	300R
R13	4R7
RESP1	RESPACK 4 .7K
RL1	RELAY -SPDT

RL2	RELAY-SPDT
SW1	SW DIP-5
T1	
T2	TRANS2
U1	LM1893A
U2	MC145027
U5	MC7805T
VR1	5K

Specifications For PCB Receiver

On 3-Jun-2000 at 14:00:18

Components on board 42

Layer	Route	Pads	Tracks	Fills	Arcs	Text
Bottom Layer		0	327	6	0	2
Top Overlay		0	216	0	5	83
Bottom Overlay		0	0	0	0	1
Keep Out Layer		0	2	0	0	0
Multi Layer		162	0	0	0	0
Total		162	545	6	5	86

Plated Hole Size	Pads	Vias
0.6mm (23.622mil)	22	0
0.635mm (25mil)	1	0
0.7112mm (28mil)	29	0
0.762mm (30mil)	5	0
0.8128mm (32mil)	104	0
1.016mm (40mil)	1	0
Total	162	0

Track Width	Count
0.2032mm (8mil)	6
0.254mm (10mil)	227
0.3048mm (12mil)	178
0.42mm (16.535mil)	84
0.854mm (33.622mil)	8
0.86mm (33.858mil)	36
2.4mm (94.488mil)	2
8.2mm (322.835mil)	4

Total 545

Part Cross Reference Report For : Transmitter

Designator	Component Library ReferenceSheet
C1	Ctc
C2	10n
C3	560p
C4	100n
C5	100n
C6	1n
C7	100n
C8	33N
C9	220n
C10	220n
C11	470u
C12	100n
C13	1uF
C14	100nF
D1	TRANZORB SA40
D2	LED
D3	BR
D4	BR
D5	DIODE
D6	ZENER2
D7	LED
D8	4733
D9	4733
J1	CON8
J2	CON2
JP1	HEADER 3
Plug1	Audio plug
Q1	BC337
R1	100R
R2	100R
R3	1K
R4	1K
R5	50K
R6	100K
R7	10K
R8	3K3
R9	10K
R10	1K
R11	4K7

R12	300R
R13	5K6
R14	1K
R15	4.7R
R16	100R
RL1	RELAY-SPST
T1	240V-18V
T2	COUPLING TRANSFORMER
T3	COUPLING TRANSFORMER
U1	4N25
U2	45026
U3	74164
U4	A
U5	MC7805T
U6	LM1893A
U7	4N25
U8	4N25
VR1	5K

Specifications For PCB Transmitter

Layer	Route	Pads	Tracks	Fills	Arcs	Text
Top Layer		0	4	0	0	0
Bottom Layer		0	499	21	0	2
Top Overlay		0	312	0	11	114
Multi Layer		209	0	0	0	0
Total		209	815	21	11	116

Total	209

Appendix 3: Project specifications and features

Power dissipation:	Transmit mode	1.66W
	Receive mode	1.33W
Supply voltage:		240 V
Operating temperature		-40 to 85 oC
Working Frequency		125 KHz

- **Features**

Use Voice to control remote devices.

Accept DTMF signals, and voice command from telephone line.

Can control upto 32 devices.

Has function as a telephone answering machine.

Database is updated in runtime.

Allows selecting different voice prompt and speech engine.

The time of last events will be recorded.

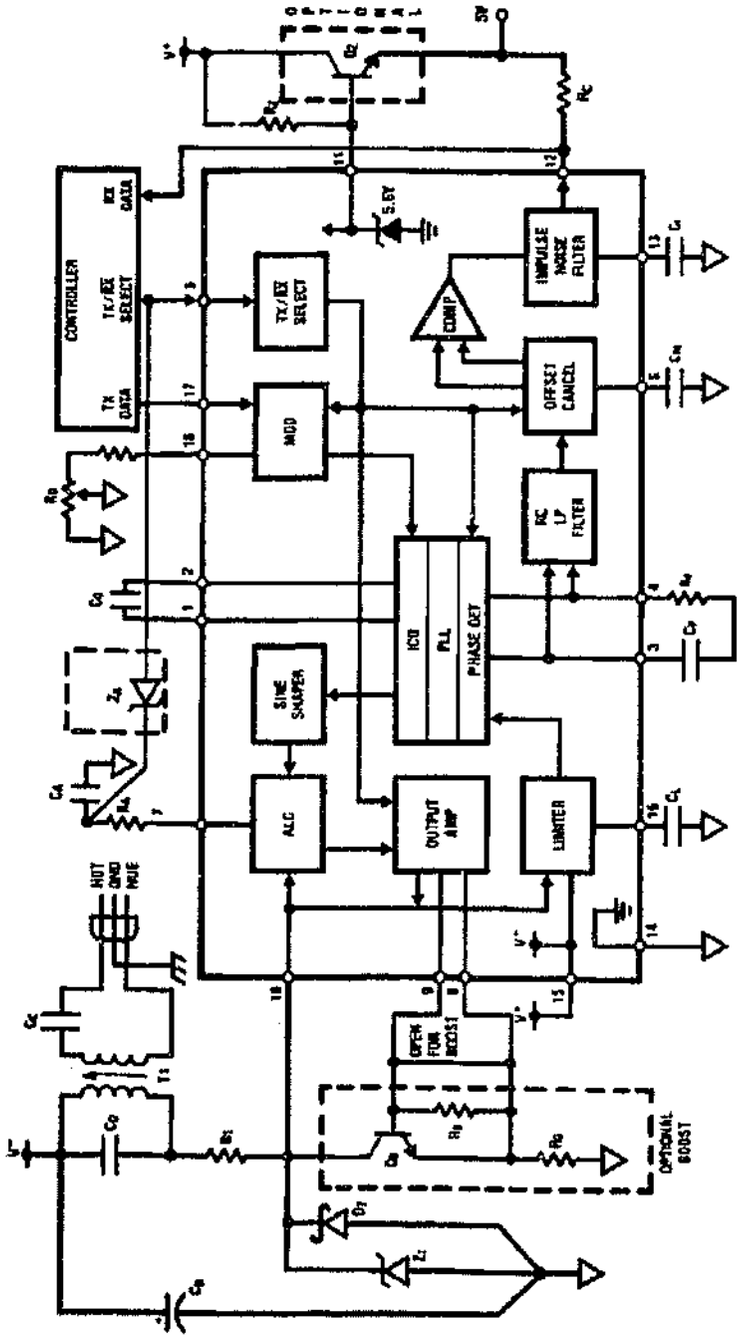
- Applications:

Energy management systems

Security and alarm systems.

Home convenience control, appliance control

Appendix 4: Selecting some particular components' values for LM 1893



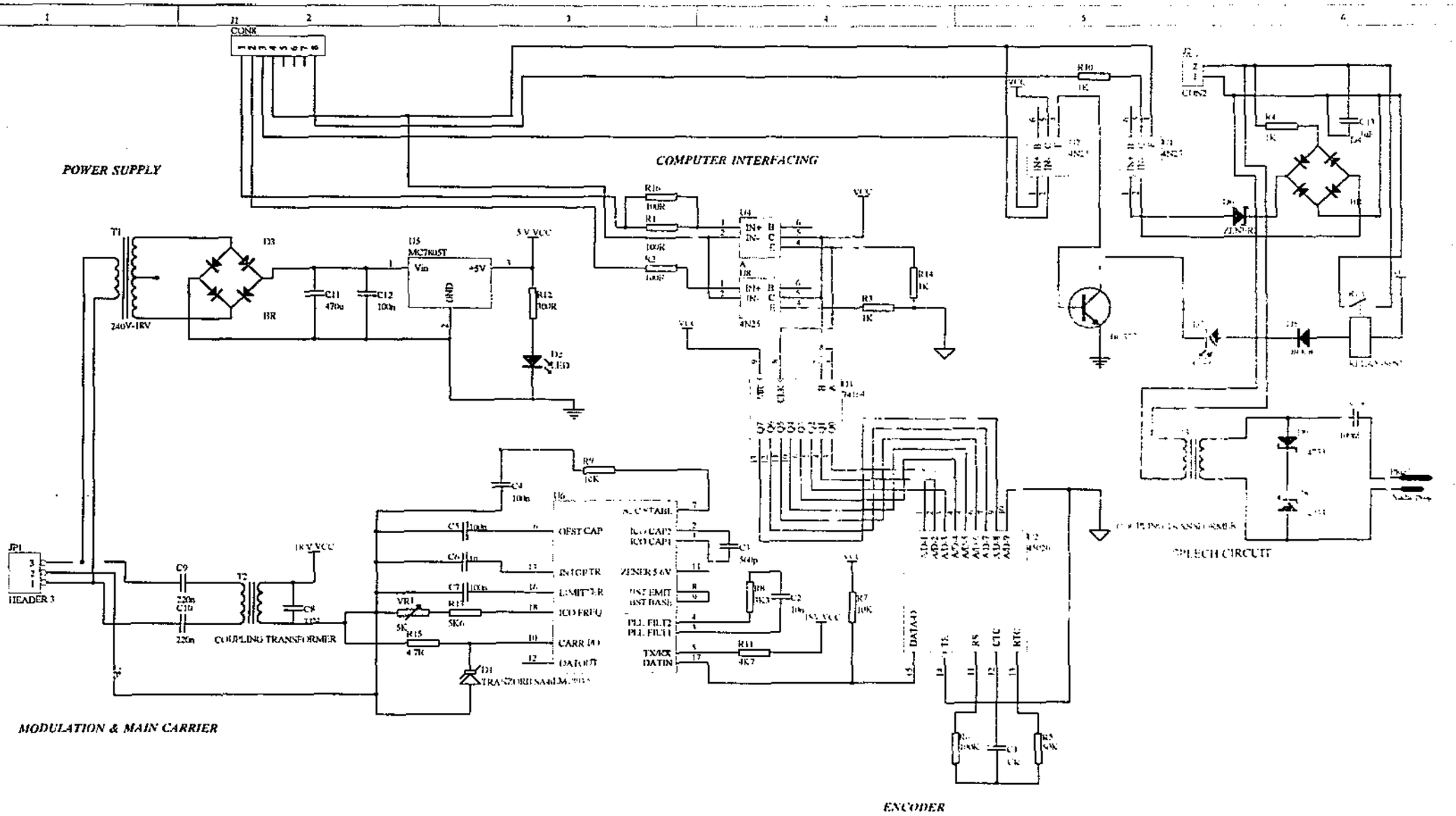
Block diagram of the LM 1893 and Power line interface

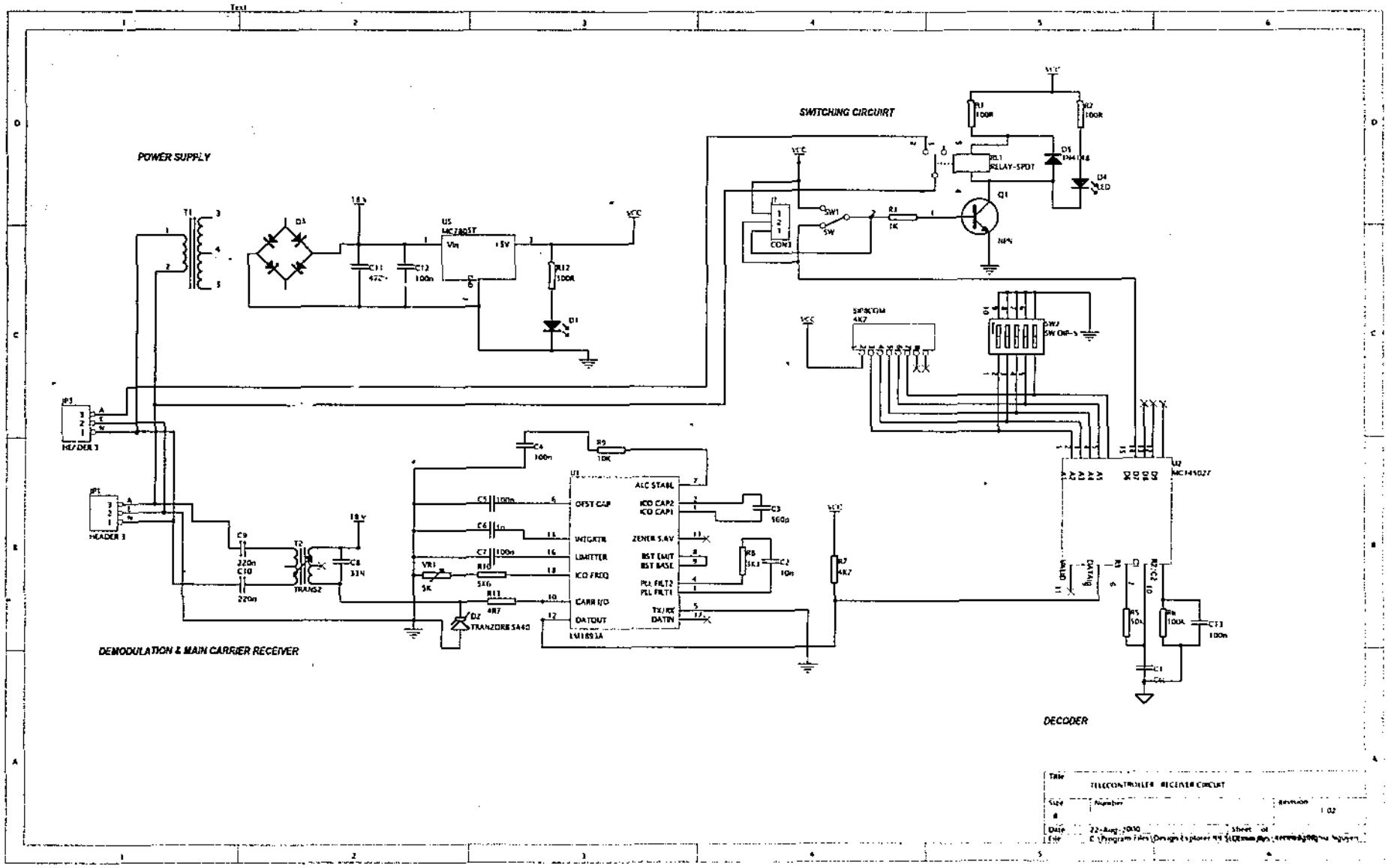
The following table is the summary of components used for the LM 1893 for the carrier frequency 125KHz recommended by the manufacture. Positions of components are shown on the block diagram of the LM 1893 above. Note that for different carrier frequency, value of components used can be draw directly from corresponding function diagram provided from the technical data.

Application Information

#	Recommended Value	Purpose	Effect of making the component value:		Notes
			Smaller	Larger	
C_O R_O	560 pF 6.2 kΩ	Together, C_O and R_O set ICO F_O.	Increases F_O Increases F_O <5.6 k not recommended.	Decreases F_O Decreases F_O >7.6 k not recommended.	±5% NPO ceramic. Use low TC 2 k pot and 5.6 k fixed R. Poor F_O TC with <5.6 k R_O.
C_F R_F	0.047 µF 3.3 kΩ	PLL loop filter pole PLL loop filter zero	Less noise immune, higher f_{DATA}, more PLL stability. PLL less stable, allows less C_F. Less ringing.	More noise immune, lower f_{DATA}, less PLL stability. PLL more stable, allows more C_F. More ringing.	Depending on R_F value and F_O, PLL unstable with large C_F. See Apps. Info. C_F and R_F values not critical.
C_C	0.22 µF	Couples F_O to line, C_C and T_1 low-pass attenuates 60 Hz.	Low TX line amplitude. Less 60 Hz T_1 current. Less stored charge.	Drives lower line Z. More 60 Hz T_1 current. More stored charge.	≥250 V non-polar. Use $2C_C$ on hot and neutral for max. line isolation, safety.
C_Q T_1	0.033 µF Use recommended XFMR	Tank matches line Z, bandpass filters, isolates from line, and attenuates transients.	Tank F_O up or increase L of T_1 for constant F_O. Smaller L: higher F_O or increase C_C; decreased F_O line pull.	Tank F_O down or decrease L of T_1 for constant F_O. Larger L: lower F_O or decrease C_C; increased F_O line pull.	100 V nonpolar, low TC, ±10% High large-signal Q needed. Optimize for low F_O line pull with control of F_O TC and Q.
C_A R_A	0.1 µF 10 kΩ	ALC pole ALC zero	Noise spikes turn ALC off. Less stable ALC.	Slower ALC response. More stable ALC.	R_A optional. ALC stable for C_A ≥100 pF.
C_L	0.047 µF	Limiter 50 kHz pole, 60 Hz rejection.	Higher pole F, more 60 Hz reject. F_O attenuation?	Lower pole F, less 60 Hz reject, more noise BW.	Any reasonably low TC cap. 300 pF guarantees stability.
C_M	0.47 µF	Holds RX path V_{OS}	Less noise immune, shorter V_{OS} hold, faster V_{OS} acquisition, shorter preamble.	More noise immune, longer V_{OS} hold, slower V_{OS} acquisition, longer preamble.	Low leakage ±20% cap. Scale with f_{DATA}.
C_I	0.047 µF	Rejects short pulses like impulse noise.	Less impulse reject, less delay, more pulse jitter.	More impulse reject, more delay, less pulse jitter.	C_I charge time ½ bit nom. Must be <1 bit worst-case.
R_C	10 kΩ	Open-col. pull-up	Less available sink I.	Less available source I.	R_C ≥1.5 kΩ on 5.6 V
R_Z	12 kΩ	5.6 V Zener bias	Larger shunt current, more chip dissipation.	Smaller shunt current, less V^+ current draw.	1<I_Z<30 mA recommended. (Chip power-up needs 5.6 V)
Z_T	≥44 V BV <60 V peak	Transient clamp	Z_T failure, higher series R-excess peak V, Zener and chip damage, less ruggedness.	Z_T costly, lower series R gives enhanced transient clamp, more ruggedness.	Recommend Zener rated for ≥500 W for 1 ms.
R_T D_T	4.7 Ω ≥44V BV	Transient I limit Over-drive Clamp	Damage Z_T, pull up V^+. Failure on Transient	Excessive TX attenuation. Costly	Carbon comp. recommended. IRF 11DQ05 or 1N5819
R_B Q_B R_G	180 Ω Power NPN 1.1 Ω	Base bleed Boost gain device Current setting R	Faster, lower THD I_O. Excessive T_J and V_{SAT}. More I_O, need higher h_{fe}.	Inadequate turn-off speed. More rugged, but costly. Less I_O, lower min. h_{fe}.	Boost optional. Q_B F(−3 dB) of >200 MHz. R_B > 24 Ohm. $I_O = 70[(10+R_G)/R_G]$ mApp.
C_B	≥47 µF	Supply bypass	Transients destroy chip.	Less supply spike.	V^+ never over abs. max.
Z_A	5.1V	Stop ALC charge in RX mode	Excess ALC current flow	ALC RX charging not inhibited over T_J	Z_A optional - 5.1V ±20% low leakage type

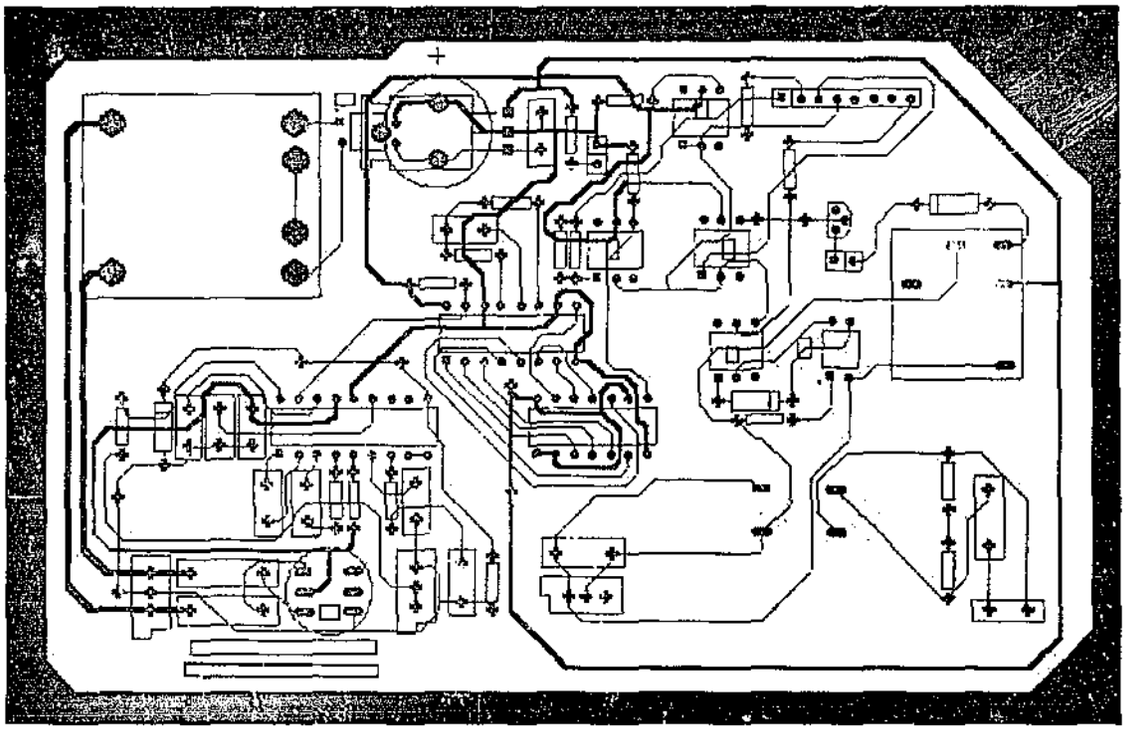
Appendix 5: Circuit diagram


POWER SUPPLY
COMPUTER INTERFACING
SPEECH CIRCUIT
MODULATION & MAIN CARRIER
ENCODER
CONN
T1
240V-18V
BR
MC7805T
5 V VCC
LED
4N25
ZENER
HEADER 3
COUPLING TRANSFORMER
OSC CAP
INTGPTR
LIMITTR
ICO FREQ
CARR IO
DATOUT
ZENER 5.6V
DATA43
TRANSCEIVER

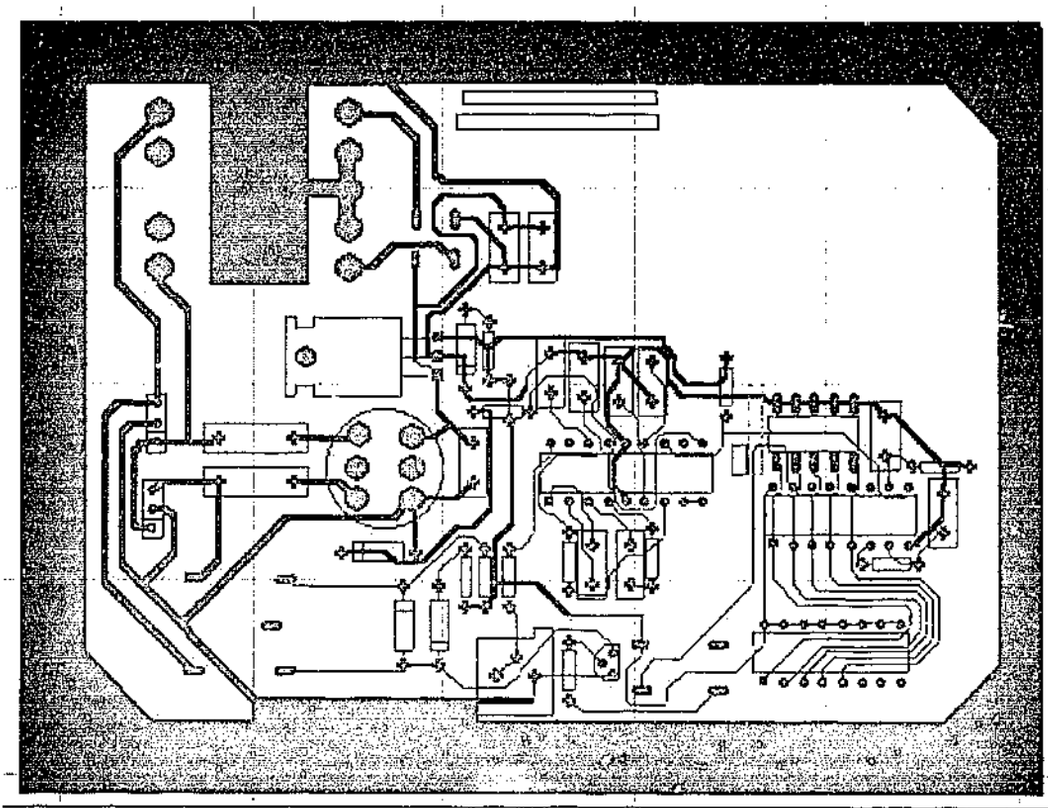

POWER SUPPLY
SWITCHING CIRCUIRT
DEMODULATION & MAIN CARRIER RECEIVER
DECODER
TELECONTROLLER RECEIVER CIRCUIT

Appendix 6: PCB Artwork

Transmitter PCB Art work (Top view)



Receiver PCB Art work (Top view)



Appendix 7: Speech SDK 4.0 (Abstracted)

This appendix contains summary properties of three ActiveX controls using in the project: Voice Command, Voice Text and Telephony API

- **Voice Command control**

Properties	Descriptions
AutoGainEnable	The current automatic gain value for a Voice Command site
AwakeState	The awake state for a Voice Command site. This property is TRUE .If the site is awake or FALSE if it is asleep.
CountCommands	Returns the number of commands on menu. Read only.
Device	The device identifier of the wave-in audio device currently used by the Voice Command site.
Enabled	TRUE if speech recognition is enabled for the site or FALSE if it is disabled. Must be set to TRUE to enable listening for the object.
EnableMenu	Enables the menu created with MenuCreate.
hWnd	Window handle. Read only.
Initialized	Equals 1 if the control has been initialised, 0 if not.
LastError	Result code from the last method or property invocation. Read only.
MenuCreate	Creates a Voice Menu object to represent a new or existing voice menu for an application. Read only.
Microphone	The name of the microphone used by a Voice Command site.
Speaker	The name of the current speaker for a

	Voice Command site
Properties	**Descriptions**
SRMode	The GUID of the speech recognition mode used for the site.
SuppressExceptions	When set to 1, exceptions will never occur.
Threshold	The threshold level of the speech recognition engine used by a Voice Command site.
Methods	**Descriptions**
Activate	Activates a voice menu so that its commands can be recognised.
AddCommand	New commands are added to the end of the menu
CmdMimic	Supplies a voice-aware application. Causes the command engine to act as if the recogniser had heard the command.
Deactivate	Tells recognizer to stop listening and release the microphone/sound card resource.
EnableItem	Permanently enables or disables a menu items.
GeneralDlg	Pprovides the user with full access to engine-specific controls.
GetCommand	Retrieves information about a command
LexiconDlg	Allows the user to display and edit his or her pronunciation lexicon.
ListGet	Retrieves the phrases stored in the current list (by **ListSet**) for the selected voice menu
ListSet	Sets the phrases in a list for a voice command
MenuDelete	Deletes a menu from the Voice Menu database.
ReleaseMenu	Releases a voice menu from memory.
Remove	Removes the specified commands from the voice menu.
SetCommand	Sets information for a command in either the global or application specific command set.

Methods	Descriptions
TrainGeneralDlg	Allows to trains speech recognition engine, recognition accuracy should be better for that particular user.
TrainMenuDlg	Train the engine for the selected menu.
TrainMicDlg	Trains the speech recognition engine with training for a microphone
Events	**Descriptions**
AttribChanged	Occurs when a site attribute has changed
ClickIn	Occurs when the user clicks in the object's icon.
CommandOther	Occurs when a spoken phrase is either recognised as being from another application's command set or is not recognised
CommandRecognize	Occurs when a spoken phrase is recognised as being from the application's command set.
CommandStart	Occurs when recognition processing has begun for a command.
Interference	Occurs when the engine cannot recognise speech properly for a known reason.
MenuActivate	Sent when a menu's activation state is changed.
UtteranceBegin	Occurs when the speech recognition engine has detected the beginning of an utterance or sound.
UtteranceEnd	Occurs when an utterance is finished.
VUMeter	Notifies the application of the current VU level, which indicates the loudness of the digital-audio stream.

Appendix 8: Bibliography

[1] National Semiconductor (1989). *Carrier Curent Transceivers LM1893.* Special Purpose Linear Devices. pp 5.136-5.158, pp. 2.393-2.399

[2] Motorola (1993). *Encoder / Decoder MC 145026- MC 145027.* Communications Device Data. pp. 2.461-2.477 .

[3] Motorola (1995). *Modulator/ Demodulator.* Analog Interface ICs. Vol. 2. pp. 8.38-8.48.

[4] Microsoft. (1998). Microsoft Speech SDK 4.0. Reference information for AciveX developer [on line]. Available WWW.http//Microsoft.com

[5] Stephen J. Bigelow. (1992). *Electronic speech circuits.* Understanding Telephone Electronics. pp 79-105. SAMS: Prentice Hall Computer Publishing.

[6] R.Villanuci & A. Avtgis & W.F.megow (1986). Electronic Techniques Shop practices and Construction. Prentice Hall, Inc.

[7] M. Sanderson. (1988). *The organization of Electronic System, Power Suppliers, Control Circuitry.* Electronic Devices A Top-Dowm Systems Approach. pp. 2-13, pp. 460-490, pp 510-561. . Prentice Hall, Inc

[8] D.E.Taylor. (1972). *Attenuators and filters, Transmission Lines.* Linear Circuit Theory. pp197-254. John Wiley & Sons.

[9] Ferrel G.Stremler. (1992). Introduction to Communication Systems. Addison-Wesley.

[10] Ronald J. Tocci. (1988). Digital Systems Principles and Applications.Englewood Cliffs. Prentice Hall, Inc

[11] Ronald A.Reis. (1991). Digital Electronics Through Project Analysis. pp 497-519. Macmillan, Inc.

[12] C.W Davison. (1989). Transmission Lines for Communications. Imdedance Matching. pp.122-151. Halsted Press-John Wiley & sons.

[13] Tomi Engdahl. (1998). *Telephone line audio interface circuits.*[online]. Available at http:\\electronics\Circuits\Phones\teleinterface.html

[14] Tomi Engdahl. (1999). Electronics circuit designed. [online]. Available at http://www.hut.fi/Misc/Electronics/faq/sfnet.harrastus.audio+video/

[15] PC hardware projects page . Available at www.epanorama.net

[16] Dino Esposito. (1999).The Microsoft Speech SDK. [online]. Available at http://premium.microsoft.com/isapi/devonly/prodinfo/msdnprod/msdnlib.idc?theURL=/msdn/library/sdkdoc/dnplatf/d21/s17a.htm.

[17] Keith Westley. (1999). Voice Command Enabling Your Software . [online]. Available at http://www.codeguru.com/cgibin/addpage

[18] Rob Thayer. (1999). Visusl Basic 6 Unleashed –Propfessional Reference Edition. Sams Publishing.

[19] Dan Appleman. (1999).Visual Basic Programmer's Guide to the Win32 API. Sams Publishing.

[20] B.Potter, Taylor Maxwell, Bryon Scott. (1993). Visual Basic Supperbible. Secon Edition.The Waite Group, Inc.

[21] Mitch G. Van Thurston, Jr. (1995). Windows 95 Multimedia Programming. pp. 117-214. M&T Books: MIS Press, Inc

[22] Paul Bonner. (1993). Visual Basic Utilities. Emeryville: Ziff-Davis Press.

[23] Taylor Maxwell, Jeff W., Ronald M. Michael Regelski. (1995). Using Visual Basic 4. *Data Access*. pp. 9-279. Que Corporation.

[24] Steven Holzner. (1999). Visual Basic 6, Core Language- Little Black Book. The Coriolis Group-Technology Press.

[25] Pei An. (1995, Volumn 26, No 3 & 4). *Computer Radio Control for Home Automation*. ETI electronics magazine. pp. 57-65

[26] (July 1996). Eleiktor Electronics Magazine. *50 Circuits, Ideas, and Tips. Signal- Control switch I & II*. pp. 89-99.

[27] *SAA Wiring Rules*. AS 3000-1991. Standards Australia

Appendix 9: Program coding

Main Form

```
---------------------------------------------------------------------------------
Public IdleTCounts%
Dim S%  ' command index detected by Voice recognition
Dim Dset As Boolean
Dim Heard As Boolean
Dim t As Integer

Private Declare Sub CopyMemory Lib "kernel32" Alias
"RtlMoveMemory" _
(hpvDest As Any, hpvSource As Any, ByVal cbCopy As Long)
Private Declare Function OSWinHelp% Lib "user32" Alias
"WinHelpA" (ByVal hwnd&, ByVal HelpFile$, ByVal wCommand%,
dwData As Any)
'*****************************************************************
'Sub-procedure name: MDIForm_Load
'Purposes:    Loads Main control interfaces, and initialises engines.
'Function called: LoadNewDoc, Menu_Create, SetVol(volCtrl.lMaximum,
'       Speech_SayIt.
'Corrupted Global variables:  none
'Input:      LoadResStrings
'Output:     Menu Files
'*****************************************************************
Private Sub MDIForm_Load()
    Dim i%

    IdleTCounts = 0
    LoadResStrings Me
Me.Left = GetSetting(App.Title, "Settings", "MainLeft", 1000)
Me.Top = GetSetting(App.Title, "Settings", "MainTop", 1000)
Me.Width = GetSetting(App.Title, "Settings", "MainWidth",
6500)
Me.Height = GetSetting(App.Title, "Settings", "MainHeight",
6500)
```

```
 engine = TextToSpeech.Find("Mfg=Microsoft;Gender=1; Style=4")
 Rem Now Select the engine, SAPI style. This is synonymous
with doing TextToSpeech1.CurrentMode = engine


    TextToSpeech.Select engine
    ' Open the mixer with deviceID 0.
    rc = mixerOpen(hmixer, 0, 0, 0, 0)
    If ((MMSYSERR_NOERROR <> rc)) Then
        MsgBox "Couldn't open the mixer."
        Exit Sub
    End If
    ' Get the waveout volume control
    OK = GetVolumeControl(hmixer, _

MIXERLINE_COMPONENTTYPE_DST_SPEAKERS,MIXERCONTROL_CONTROLTYPE_
VOLUME, volCtrl)
    Call SetVol(volCtrl.lMaximum / 2)
    Call LoadNewDoc
    Vcommand.initialized = 1
    Call Menu_Create(gMyMenu)
    Vcommand.Enabled = 1
    Vcommand.Activate gMyMenu
    'The following codes will reset the control
    For i% = 0 To 15
    Reset (i%)
    Next i%
    Call Speech_SayIt("System Initialization completed.")

End Sub
'*********************************************************************
```

Private Sub **LoadNewDoc()**

```
  Dim dbname As String
  dbname = App.Path & "\Control.mdb"
  With Data

    .DatabaseName = dbname
    .RecordSource = "DEVICES"
```

```
        .Refresh
    End With
    frmDocument.Show
End Sub
'*********************************************************************
```

Private Sub **MDIForm_Unload(Cancel As Integer)**

```
    If Me.WindowState <> vbMinimized Then
        SaveSetting App.Title, "Settings", "MainLeft", Me.Left
        SaveSetting App.Title, "Settings", "MainTop", Me.Top
        SaveSetting App.Title, "Settings", "MainWidth",
Me.Width
        SaveSetting App.Title, "Settings", "MainHeight",
Me.Height
    End If
    ' close the database
    With Data
        .Recordset.Close
        .Database.Close
    End With
    Vcommand.ReleaseMenu gMyMenu
    Unload DlgSetup
    Unload frmDocument
    Set fMainForm = Nothing
    Unload Me
End Sub
'*********************************************************************

Private Sub mnuToolDatabase_Click()
     DevicesDTBase.Show
End Sub
```

```
Private Sub mnuToolSetup_Click()
    DlgSetup.Show vbModal, Me
End Sub
'*********************************************************************
```

'Procedure name: tbToolBar_ButtonClick

'Purposes: Access menu functions.

'Function called: None

```
'Corrupted Global variables:    none
'Input:
'Output:       Depent on which menu is selected
'*********************************************************************

Private Sub tbToolBar_ButtonClick(ByVal Button As
MSComCtlLib.Button)

    On Error Resume Next
    Select Case Button.Key
        Case "New"
            LoadNewDoc
        Case "Save"
            mnuFileSave_Click
        Case "Print"
            mnuFilePrint_Click
        Case "Find"
            'ToDo: Add 'Find' button code.
            MsgBox "Add 'Find' button code."
        Case "Help"
            'ToDo: Add 'Help' button code.
            MsgBox "Add 'Help' button code."
    End Select
End Sub
```

```
Private Sub mnuHelpAbout_Click()
    frmAbout.Show vbModal, Me
End Sub
```

```
Private Sub mnuHelpSearchForHelpOn_Click()
Dim nRet As Integer

    'if there is no helpfile for this project display a
message to the user
    'you can set the HelpFile for your application in the
    'Project Properties dialog

    If Len(App.HelpFile) = 0 Then
```

```
        MsgBox "Unable to display Help Contents. There is no
       Help associated with this _project.", vbInformation,
       Me.Caption
        Call Speech_SayIt("Unable to display Help Contents. There
       is no Help associated with this project.")
    Else
        On Error Resume Next
        nRet = OSWinHelp(Me.hwnd, App.HelpFile, 261, 0)
        If Err Then
            MsgBox Err.Description
            Call Speech_SayIt(Err.Description)
        End If
    End If
End Sub
```

```
Private Sub mnuHelpContents_Click()
    Dim nRet As Integer

'if there is no helpfile for this project display a message to
'the user
    If Len(App.HelpFile) = 0 Then
        MsgBox "Unable to display Help Contents. There is no
Help associated with this project.", vbInformation, Me.Caption
    Else
        On Error Resume Next
        nRet = OSWinHelp(Me.hwnd, App.HelpFile, 3, 0)
        If Err Then
            MsgBox Err.Description
        End If
    End If
End Sub
```

```
Private Sub mnuViewStatusBar_Click()
    mnuViewStatusBar.Checked = Not mnuViewStatusBar.Checked
    sbStatusBar.Visible = mnuViewStatusBar.Checked
End Sub
```

```
Private Sub mnuViewToolbar_Click()
    mnuViewToolbar.Checked = Not mnuViewToolbar.Checked
    tbToolBar.Visible = mnuViewToolbar.Checked
End Sub
```

```
Public Sub mnuFileExit_Click()
    Call Exits
End Sub
```

```
Private Sub mnuFileSend_Click()
    'ToDo: Add 'mnuFileSend_Click' code.
    MsgBox "Add 'mnuFileSend_Click' code."
End Sub
```

```
Private Sub mnuFilePrint_Click()

    On Error Resume Next
    If ActiveForm Is Nothing Then Exit Sub
    With dlgCommonDialog
        .DialogTitle = "Print"
        .CancelError = True
        .Flags = cdlPDReturnDC + cdlPDNoPageNums
     If ActiveForm.rtfText.SelLength = 0 Then
         .Flags = .Flags + cdlPDAllPages
     Else
            Flags = .Flags + cdlPDSelection
      End If
      ShowPrinter
       If Err <> MSComDlg.cdlCancel Then
            ActiveForm.rtfText.SelPrint .hDC
       End If
    End With
End Sub
```

```
Private Sub mnuFilePrintPreview_Click()
    'ToDo: Add 'mnuFilePrintPreview_Click' code.
    MsgBox "Add 'mnuFilePrintPreview_Click' code."
End Sub

Private Sub mnuFileProperties_Click()
    'ToDo: Add 'mnuFileProperties_Click' code.
    MsgBox "Add 'mnuFileProperties_Click' code."
End Sub

Private Sub mnuFileSave_Click()
    Dim sFile As String

    If Left$(ActiveForm.Caption, 8) = "Document" Then
        With dlgCommonDialog
            .DialogTitle = "Save"
            .CancelError = False
'ToDo: set the flags and attributes of the common dialog
'control
            .Filter = "All Files (*.*)|*.*"
            .ShowSave

            If Len(.filename) = 0 Then
                Exit Sub
            End If

            sFile = .filename
        End With
        ActiveForm.rtfText.SaveFile sFile
    Else
        sFile = ActiveForm.Caption
        ActiveForm.rtfText.SaveFile sFile
    End If
End Sub

Private Sub mnuFileNew_Click()
    'LoadNewDoc
End Sub
```

```
Private Sub Timer1_Timer()
      IdleTCounts = IdleTCounts + 1
End Sub
```

'Procedure name: Vcommand_CommandOther

'Purposes: Generates Event.

'Function called: Speech_SayIt, Control

'Corrupted Global variables: S 'Device Index detected by this procedure

'Input: Command 'heard from user

'Output: Audible sound

'Description: Used to confirm command heard

```
'*********************************************************************
Private Sub Vcommand_CommandOther(ByVal CmdName As String,
ByVal Command As String)
 Select Case Command
 Case ""
    Call Speech_SayIt("Command was not recognised. Please say
again")
    Case "Yes"
        Call Control(S, Dset)
        Call Speech_SayIt("Heard Command")
    End Select
End Sub
```

'Procedure name: Vcommand_CommandRecognize

'Purposes: Generates Event.

'Function called: Speech_SayIt, Control

'Corrupted Global variables: S 'Device Index detected by this procedure

'Input: Command 'heard from user

'Output: Audible sound

```
'*********************************************************************
```

Private Sub **Vcommand_CommandRecognize**(ByVal ID As Long, ByVal CmdName As String, ByVal Flags As Long, ByVal Action As String, ByVal NumLists As Long, ByVal ListValues As String, ByVal Command As String)

```
Dset = False
Heard = False
```

```
    With frmDocument
    For i% = 0 To 17

    Select Case Command

        Case .DevName(i).Caption:

            If i <= 15 Then
            S = i
            Call Speech_SayIt("You selected device name " &
           .DevName(i).Caption & "  yes or  no ?")
            Heard = True
            ElseIf (i = 16) Then
            Dset = True
            Heard = True
            Call Speech_SayIt("Heard Command")
            Call Control(S, Dset)
            End If
    End Select
    Next i%
    End With

    If Heard = False Then
                Call Speech_SayIt("Please! Try again")
    End If
            Debug.Print Heard
End Sub
```

'Procedure name: phone1_DoPhoneCall

'Purposes: Generates Event.

'Function called: GetNumber,Speech_SayIt, Control

'Corrupted Global variables: S 'Device Index detected by this procedure

'Input: Command, DTMF from telephone line 'heard from user

'Output: Audible sound, executes Command

'Description: Is the main function of telephone control

```
Private Sub phone1_DoPhoneCall(ByVal lineID As Long)
'This function gets called each time the phone is answered,
for each phone line.
'Because Telephony supports multiple phone lines, this
function may be called simultaneously
'several times, each version (identified by lineID) running on
a different thread.
Dim result As Long
Dim size As Long
Dim wave As Long
Dim wavefile() As Byte
Dim filename As String

Dim sPassKey As String
Dim CFirstFour As String
Dim CNextSix As String
Dim CLastFive As String
Dim CExpDate As String

Dim S As String
Dim ss As String
Dim sDTMF As String
Dim Command As String
Dim P As String
On Error Resume Next
    size = 0

DlgSetup.toggle = 0

    'this line causes a recorded wave file to be used in place
of the text wherever it is spoken.
    '(if you comment out this line, the text-to-speech engine
will be used instead)
    'This feature exists so you can prototype your program
with text-to-speech, then use
    'the wave list editor to do recordings of the string.  See
the docs for more details.
    phone1.WaveAddFromListString lineID, "[FromFile]" +
vbNewLine + App.Path + _
        "\hianswer.wav=Hi. We can't come to the phone right
now. Please leave a message at the beep."
```

```
    'this is the heart of the program.  The prompts are spoken
(or played if the waveadd worked),
    'and the callers message is stored in wave.  See the docs
for definitions of the settings used.

phone1.Speak lineID, "Hi. Press one to leave message or two to
enter control mode after the beep."

PassKey:
'call our helper routine which gets numbers. See below
   S = GetNumber(lineID, 4, "Please say or enter the four
digit passkey.", _
                            "What is the passkey?", _
                            "This is a sample. Just make up a
four digit number.", _
                            "This is a sample. Just make up a
four digit number.")

  If (S = "") Then
    GoTo done
  End If

 If (S = "1 2 3 4 ") Or (S = "1   2   3   4 ") Then

phone1.Speak lineID, "the number you entered are "
phone1.Speak lineID, S
phone1.Speak lineID, "Please enter the device number"
phone1.Speak lineID, "The following devices are off"
For i% = 0 To 1
If frmDocument.CmdOff(i%).FontBold = True Then
phone1.Speak lineID, "device number"
phone1.Speak lineID, Str(i%)
'phone1.Speak lineID, (Label1(i%).Caption)
MainControl.phone1.Speak lineID, "is off"
End If
Next i%

phone1.Speak lineID, "The following devices are on"
For i% = 0 To 1
```

```
If frmDocument.CmdOn(i%).FontBold = True Then
phone1.Speak lineID, "device number"
phone1.Speak lineID, Str(i%)
'phone1.Speak lineID, (Label1(i%).Caption)
phone1.Speak lineID, "is on"
End If
Next i%

SelectControl:
S = "100"
While S <> "*"
S = GetNumber(lineID, 2, "Please say or enter the two digit
for divice number.", _
                          "What is the number?", _
                          "", _
                          "What is the number?.")
If (S = "") Or (S = "*") Then
    GoTo done
  End If
phone1.Speak lineID, "the number you select are "
phone1.Speak lineID, S
phone1.YesNoFromString lineID, "[Prompts]" + vbNewLine + _
                               "Main=is this correct?" +
vbNewLine + _
                                "where=getting first four"
+ vbNewLine + _
                                "[Settings]" + vbNewLine +
_
                                "CanGoBack=1" + vbNewLine,
result, P

   If (result < 0) Then
      GoTo done
   ElseIf (result <> 1) Then
    GoTo SelectControl
   End If

Command = S
```

```
 Select Case Command

Case "0 0 ", "0   0   "
   If frmDocument.CmdOn(0).FontBold = False Then
       frmDocument.DON (0)
       phone1.Speak lineID, (frmDocument.DevName(0).Caption)
       phone1.Speak lineID, "is on"
   ElseIf frmDocument.CmdOn(0).FontBold = True Then
       frmDocument.DOFF (0)
       phone1.Speak lineID, (frmDocument.DevName(0).Caption)
       phone1.Speak lineID, "is off"
   End If

 Case "0 1 ", "0   1   "
   If frmDocument.CmdOn(1).FontBold = False Then
       frmDocument.DON (1)
       phone1.Speak lineID, (frmDocument.DevName(1).Caption)
       phone1.Speak lineID, "is on"
   ElseIf frmDocument.CmdOn(1).FontBold = True Then
       frmDocument.DOFF (1)
       phone1.Speak lineID, (frmDocument.DevName(1).Caption)
       phone1.Speak lineID, "is off"
   End If

   Case "0 2 ", "0   2   "
  If frmDocument.CmdOn(2).FontBold = False Then
       frmDocument.DON (2)
       phone1.Speak lineID, (frmDocument.DevName(2).Caption)
       phone1.Speak lineID, "is on"
   ElseIf frmDocument.CmdOn(2).FontBold = True Then
       frmDocument.DOFF (2)
       phone1.Speak lineID, (frmDocument.DevName(2).Caption)
       phone1.Speak lineID, "is off"
   End If

   Case "0 3 ", "0   3   "
   If frmDocument.CmdOn(3).FontBold = False Then
       frmDocument.DON (3)
```

```
        phone1.Speak lineID, (frmDocument.DevName(3).Caption)
        phone1.Speak lineID, "is on"
    ElseIf frmDocument.CmdOn(3).FontBold = True Then
        frmDocument.DOFF (3)
        phone1.Speak lineID, (frmDocument.DevName(3).Caption)
        phone1.Speak lineID, "is off"
    End If

   Case "0 4 ", "0   4   "
   If frmDocument.CmdOn(4).FontBold = False Then
        frmDocument.DON (4)
        phone1.Speak lineID, (frmDocument.DevName(4).Caption)
        phone1.Speak lineID, "is on"
    ElseIf frmDocument.CmdOn(4).FontBold = True Then
        frmDocument.DOFF (4)
        phone1.Speak lineID, (frmDocument.DevName(4).Caption)
        phone1.Speak lineID, "is off"
    End If

    Case "0 5 ", "0   5   "
    If frmDocument.CmdOn(5).FontBold = False Then
        frmDocument.DON (5)
        phone1.Speak lineID, (frmDocument.DevName(5).Caption)
        phone1.Speak lineID, "is on"
    ElseIf frmDocument.CmdOn(5).FontBold = True Then
        frmDocument.DOFF (5)
        phone1.Speak lineID, (frmDocument.DevName(5).Caption)
        phone1.Speak lineID, "is off"
    End If

    Case "0 6 ", "0   6   "
    If frmDocument.CmdOn(6).FontBold = False Then
       frmDocument.DON (6)
       phone1.Speak lineID, (frmDocument.DevName(6).Caption)
       phone1.Speak lineID, "is on"
    ElseIf frmDocument.CmdOn(6).FontBold = True Then
        frmDocument.DOFF (6)
        phone1.Speak lineID, (frmDocument.DevName(6).Caption)
```

```
    phone1.Speak lineID, "is off"
End If

Case "0 7 ", "0   7   "
If frmDocument.CmdOn(7).FontBold = False Then
    frmDocument.DON (7)
    phone1.Speak lineID, (frmDocument.DevName(7).Caption)
    phone1.Speak lineID, "is on"
ElseIf frmDocument.CmdOn(7).FontBold = True Then
    frmDocument.DOFF (7)
    phone1.Speak lineID, (frmDocument.DevName(7).Caption)
    phone1.Speak lineID, "is off"
End If

Case "0 8 ", "0   8   "
If frmDocument.CmdOn(8).FontBold = False Then
    frmDocument.DON (8)
    phone1.Speak lineID, (frmDocument.DevName(8).Caption)
    phone1.Speak lineID, "is on"
ElseIf frmDocument.CmdOn(8).FontBold = True Then
    frmDocument.DOFF (8)
    phone1.Speak lineID, (frmDocument.DevName(8).Caption)
    phone1.Speak lineID, "is off"
End If

    Case "0 9 ", "0   9   "
If frmDocument.CmdOn(9).FontBold = False Then
    frmDocument.DON (9)
    phone1.Speak lineID, (frmDocument.DevName(9).Caption)
    phone1.Speak lineID, "is on"
ElseIf frmDocument.CmdOn(9).FontBold = True Then
    frmDocument.DOFF (9)
    phone1.Speak lineID, (frmDocument.DevName(9).Caption)
    phone1.Speak lineID, "is off"
End If

Case "1 0 ", "1   0   "
If frmDocument.CmdOn(10).FontBold = False Then
```

```
    frmDocument.DON (10)
    phone1.Speak lineID, (frmDocument.DevName(10).Caption)
    phone1.Speak lineID, "is on"
ElseIf frmDocument.CmdOn(10).FontBold = True Then
    frmDocument.DOFF (10)
    phone1.Speak lineID, (frmDocument.DevName(10).Caption)
    phone1.Speak lineID, "is off"
End If

Case "1 1 ", "1   1   "
If frmDocument.CmdOn(11).FontBold = False Then
    frmDocument.DON (11)
    phone1.Speak lineID, (frmDocument.DevName(11).Caption)
    phone1.Speak lineID, "is on"
ElseIf frmDocument.CmdOn(11).FontBold = True Then
    frmDocument.DOFF (11)
    phone1.Speak lineID, (frmDocument.DevName(11).Caption)
    phone1.Speak lineID, "is off"
End If

    Case "1 2 ", "1   2   "
If frmDocument.CmdOn(12).FontBold = False Then
    frmDocument.DON (12)
    phone1.Speak lineID, (frmDocument.DevName(12).Caption)
    phone1.Speak lineID, "is on"
ElseIf frmDocument.CmdOn(12).FontBold = True Then
    frmDocument.DOFF (10)
    phone1.Speak lineID, (frmDocument.DevName(12).Caption)
    phone1.Speak lineID, "is off"
End If

    Case "1 3 ", "1   3   "
If frmDocument.CmdOn(13).FontBold = False Then
    frmDocument.DON (13)
    phone1.Speak lineID, (frmDocument.DevName(13).Caption)
    phone1.Speak lineID, "is on"
ElseIf frmDocument.CmdOn(13).FontBold = True Then
    frmDocument.DOFF (13)
```

```
        phone1.Speak lineID, (frmDocument.DevName(13).Caption)
        phone1.Speak lineID, "is off"
    End If

        Case "1 4 ", "1   4   "
    If frmDocument.CmdOn(14).FontBold = False Then
        frmDocument.DON (14)
        phone1.Speak lineID, (frmDocument.DevName(14).Caption)
        phone1.Speak lineID, "is on"
    ElseIf frmDocument.CmdOn(14).FontBold = True Then
        frmDocument.DOFF (14)
        phone1.Speak lineID, (frmDocument.DevName(14).Caption)
        phone1.Speak lineID, "is off"
    End If

        Case "1 5 ", "1   5   "
    If frmDocument.CmdOn(15).FontBold = False Then
        frmDocument.DON (15)
        phone1.Speak lineID, (frmDocument.DevName(15).Caption)
        phone1.Speak lineID, "is on"
    ElseIf frmDocument.CmdOn(15).FontBold = True Then
        frmDocument.DOFF (15)
        phone1.Speak lineID, (frmDocument.DevName(15).Caption)
        phone1.Speak lineID, "is off"
    End If
 End Select
Wend
Else
  GoTo answermachine

answermachine:
    phone1.RecordFromString lineID, "[Prompts]" + vbNewLine +
_
                                "Main=Hi. We can't come to the
phone right now. Please leave a message at the beep." +
vbNewLine + _
"[Settings]" + vbNewLine + "BetweenUtt=10000" + vbNewLine + _
"InitialBuf=30000" + vbNewLine + "ReallocBuf=30000" +
vbNewLine + "MaxBuf=300000" + vbNewLine + _
```

```
"NoAnswerTime=10" + vbNewLine , result, wave, size


    If (size <> 0) Then
'The following "casts" the wave to an array (by making a copy
of it) so we can save it out to disk.
        ReDim wavefile(size)
        CopyMemory wavefile(0), ByVal wave, size


'free the wave as soon as possible so we dont pig up memory
        phone1.FreeWave wave


'use the date and time as the filename, and put into the
Messages directory
      filename = App.Path + "\Messages" + "\Message left at " +
      Format(Now, "hh mm ss AMPM") + " " + Format(Now, " mmm d
      yyyy") + ".wav"


'write the wave data out to disk. you can double click on the
file to play it with sound recorder.
        Open filename For Binary Access Write As #1
        Put #1, , wavefile
        Close #1
    End If
  End If
    'hang up and wait for next call
done:
   phone1.Speak lineID, "goodbye"
Done2:
DlgSetup.toggle = 1


End Sub
```

'Function name: GetNumber

'Purposes: Generates Event.

'Function called: GetNumber,Speech_SayIt, Control

'Corrupted Global variables: S 'Device Index detected by this procedure

'Input: lineID,numdigits,grammar string 'heard from user

'Output: Received Command in integer format

'Description: Is used to get command from user

```
Private Function GetNumber(lineID As Long, numdigits As
Integer, Main As String, main2 As String, where As String,
help As String) As String
Dim result As Long
Dim S As String
Dim tstr As String

Dim NumGram As String

NumGram = ""
For i = 1 To numdigits
    NumGram = NumGram + "<0..9> "" """
Next i
phone1.GrammarFromString lineID, "[Prompts]" + vbNewLine + _
                                  "Main=" + Main + vbNewLine + _
                                  "Main.2=" + main2 + vbNewLine
                                  + _
                                  "Where=" + where + vbNewLine +
                                  _
                                  "Help=" + help + vbNewLine +
                                  _
                                  "[DTMFString]" + vbNewLine +
                                  _
                                  "Count=" + Str(numdigits) +
                                  vbNewLine + _
                                  "0=0" + vbNewLine + _
                                  "1=1" + vbNewLine + _
                                  "2=2" + vbNewLine + _
                                  "3=3" + vbNewLine + _
                                  "4=4" + vbNewLine + _
                                  "5=5" + vbNewLine + _
                                  "6=6" + vbNewLine + _
                                  "7=7" + vbNewLine + _
                                  "8=8" + vbNewLine + _
                                  "9=9" + vbNewLine + _
                                  "*=*" + vbNewLine + _
                                  "[<MyGrammar>]" + vbNewLine +
_"<MyGrammar>= " + NumGram + vbNewLine, result, S

GetNumber = ""
```

```
tstr = S
For i = 1 To Len(tstr)
    GetNumber = GetNumber + Mid(tstr, i, 1) + " "
Next i
End Function
```

'Reset or Initialize the Controls all to OFF state

```
Private Sub Reset(index As Integer)
        frmDocument.CmdOff(index).FontBold = True
        frmDocument.CmdOn(index).FontBold = False
        Call SendControl(index + 1, 0)
End Sub
```

'procedure name: Vcommand_UtteranceBegin
'Purposes: Together with Vcommand_UtteranceEnd detect idle time on line .
'Function called:
'Corrupted Global variables:
'Input: Audio signal from line
'Output: Enent

```
Private Sub Vcommand_UtteranceBegin()
      IdleTCounts = -5
      'Timer1.Enabled = False
End Sub
```

```
Private Sub Vcommand_UtteranceEnd()
      IdleTCounts = 0
      Timer1.Enabled = True
End Sub
```

'procedure name: Control
'Purposes: Executes Command.
'Function called:
'Corrupted Global variables: S%,Dset
'Input: S%,Dset
'Output: S%,Dset

```
Private Sub Control(ByVal S%, ByVal Dset As Boolean)
    With frmDocument
    If Dset = True Then
        If .CmdOn(S).FontBold = False Then
                Call .CmdOn_Click(S)
                Dset = False
        ElseIf .CmdOn(S).FontBold = True Then
                Call .CmdOff_Click(S)
                Dset = False
                End If
     End If
End With
End Sub
```

- **Main Control Interface Form**

```
Dim Y%
Dim t%
Public PCtr_flage%
```

```
'Sub-procedure name: CmdOff_Click
'Purposes:    Send OFF signal to LPT1.
'Function called: UpdateCommandRec,SendControl
'       Speech_SayIt.
'Corrupted Global variables:  none
'Input:       Device's index
'Output:      Control signal to LPT1
```

```
Public Sub CmdOff_Click(index As Integer)
    Dim RetBVal As Boolean
    RetBVal = FindRecord(fMainForm.Data, index + 1)

    LblTime(index).Caption = Time        ' Set time
    Call UpdateCommandRec(False)
    Call SendControl(index + 1, False)
    CmdOff(index).FontBold = True
    CmdOn(index).FontBold = False
```

```
    If (Not fMainForm.Data.Recordset.fields("Status")) Then
LblTime(index).ForeColor = vbBlack
    If DlgSetup.Check2.Value = 1 Then
    Call Speech_SayIt(fMainForm.Data.Recordset.fields(1) & "
is " & fMainForm.Data.Recordset.fields("status_text"))
   End If
End Sub
```

'Sub-procedure name: CmdOn_Click

'Purposes: Send On signal to LPT1.

'Function called: UpdateCommandRec,SendControl

' Speech_SayIt.

'Corrupted Global variables: none

'Input: Device's index

'Output: Control signal to LPT1

```
Public Sub CmdOn_Click(index As Integer)
    Dim RetBVal As Boolean
    RetBVal = FindRecord(fMainForm.Data, index + 1)
    LblTime(index).Caption = Time        ' Set time
    Call UpdateCommandRec(True)
    Call SendControl(index + 1, True)
    CmdOff(index).FontBold = False
    CmdOn(index).FontBold = True
    If (fMainForm.Data.Recordset.fields("Status")) Then
LblTime(index).ForeColor = vbRed
    If DlgSetup.Check2.Value = 1 Then
    Call Speech_SayIt(fMainForm.Data.Recordset.fields(1) & "
is " & fMainForm.Data.Recordset.fields("status_text"))
   End If
End Sub
```

```
Private Sub DevName_DblClick(index As Integer)
    DlgSetup.Show vbModal
End Sub
```

```
Private Sub runtop()
    ' Advance animation one frame.

    Y = Y + 1: If Y = 15 Then Y = 0
```

```
    Picture1.Picture = PictureClip1.GraphicCell(Y)
    'x = x + 1: If x = 5 Then x = 0
    'Picture4.Picture = PictureClip2.GraphicCell(x)
   'Form1.Icon = Image1(x).Picture

End Sub

Private Sub runicon()
    t = t + 1: If t = 5 Then t = 0
    fMainForm.Icon = Image1(t).Picture
End Sub
```

```
Private Sub Form_Load()
    'Call FormLoad
    Picture1.Picture = PictureClip1.GraphicCell(0)
    fMainForm.Icon = Image1(0).Picture
   ' UserControl11.Flage = 2
End Sub
```

```
Private Sub Form_Resize()
    On Error Resume Next
    Call FormLoad
End Sub
```

'Sub-procedure name: DOFF

'Purposes: Send On signal to LPT1.

'Note: same as CmdOff_Click but for using with Telephony control

```
Public Sub DOFF(index As Integer)
    Dim RetBVal As Boolean
    RetBVal = FindRecord(fMainForm.Data, index + 1)

    LblTime(index).Caption = Time        ' Set time
    Call UpdateCommandRec(False)
    Call SendControl(index + 1, False)
    CmdOff(index).FontBold = True
    CmdOn(index).FontBold = False
```

```
    If (Not fMainForm.Data.Recordset.fields("Status")) Then
LblTime(index).ForeColor = vbBlack
    Call fMainForm.TextToSpeech_SpeakingStarted
End Sub
```

```
Public Sub DON(index As Integer)
    Dim RetBVal As Boolean
    RetBVal = FindRecord(fMainForm.Data, index + 1)

    LblTime(index).Caption = Time        ' Set time
    Call UpdateCommandRec(True)
    Call SendControl(index + 1, True)
    CmdOff(index).FontBold = False
    CmdOn(index).FontBold = True
    If (fMainForm.Data.Recordset.fields("Status")) Then
LblTime(index).ForeColor = vbRed
    'Call SaveRefeData(fMainForm.Data, index, 7)
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
Set frmDocument = Nothing
End Sub
```

```
Private Sub PhoneCtrMode_Click()
    Timer1.Interval = 1000
If PhoneCtrMode.Value = 1 Then
     Call Speech_SayIt("Telephone  monitoring is enable ")
     P_monotor.PCtr_enable = True
     P_monotor.Visible = True
     PhoneCtrMode.Enabled = False       'disable the control
     check box
     P_monotor.IdleTime = 30
     P_monotor.RingSet = 5
     P_monotor.P_InputSet = 190
     Else
     Call Speech_SayIt("Telephone monitoring  is  disable")
End If
End Sub
```

```
Private Sub Timer1_Timer()
Dim t As Boolean

P_monotor.IdleTimeCount = fMainForm.IdleTCounts
Debug.Print fMainForm.IdleTCounts
     If  P_monotor.PCtr_enable = False Then
       frmDocument.PhoneCtrMode.Value = 0
       PhoneCtrMode.Enabled = True
     End If
End Sub
```

```
Private Sub Timer2_Timer()
    If DlgSetup.AnimatCheck = 1 Then
        runicon
        If DlgSetup.toggle = 1 Then runtop
    End If
End Sub
```

- **Set up Form**

```
'Public gMyMenu As Long
Dim click3%, click4%
Dim click1%, click2%
'Public P_flage As Integer
Public toggle%
Option Explicit
```

```
Private Sub CancelButton_Click()
    Unload Me
End Sub
```

```
Private Sub CmbVoiceType_Click()
    Rem Each time somebody selects a new voice/engine/mode
from the combo box,
    Rem select that voice as the active speaker.
    fMainForm.TextToSpeech.CurrentMode =
CmbVoiceType.ListIndex + 1
End Sub
```

```
Private Sub cmdApply_Click()
    fMainForm.Vcommand.ReleaseMenu gMyMenu
    fMainForm.Data.Refresh
    Call ChangeMode
    Call FormLoad
    Call Menu_Create(gMyMenu)
    fMainForm.Vcommand.Enabled = 1
    fMainForm.Vcommand.Activate gMyMenu
    Call OKButton_Click
End Sub
```

```
Private Sub Command1_Click()
    fMainForm.TextToSpeech.Speak TxtTest.Text
End Sub
```

```
Private Sub Command2_Click()

'The 'Selection Engines...' button
    DlgSetup.Visible = False
    fMainForm.phone1.ChooseEngineDialog  'this lets the user
set the TTS voice used, as well as the voice engine used.
    DlgSetup.Visible = True
End Sub
```

```
Private Sub Command3_Click()
    Call Vcommand1.TrainGeneralDlg(Me.hwnd, "TrainEngine")
End Sub
```

```
Private Sub Form_Load()
    Dim X%
    Dim dbname As String

    dbname = App.Path & "\Control.mdb"

    With Data
        .DatabaseName = dbname
        .RecordSource = "DEVICES"
        .Refresh

    End With
    Call Set_Grid(0)
    Set tbsOptions.SelectedItem = tbsOptions.Tabs(1)
        SldVolume.Max = 100
        SldVolume.Min = 0
        SldVolume.SmallChange = 1
        SldVolume.Value = 50
        click1 = 0
        click2 = 1
        click3 = 0
        click4 = 0
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
    Set DlgSetup = Nothing
    Unload Me
End Sub
```

```
Private Sub GridText_GotFocus()

    With GridText
        .SelStart = 0
        .SelLength = Len(.Text)

    End With
End Sub
```

```
Private Sub GridText_KeyDown(KeyCode As Integer, shift As
Integer)

    Select Case KeyCode
        Case vbKeyUp
            If (MSFlexGrid.Row = 1) Then Exit Sub
            GridText.Visible = False
            MSFlexGrid.Row = MSFlexGrid.Row - 1
        Case vbKeyDown
            If (MSFlexGrid.Row = 16) Then Exit Sub
            GridText.Visible = False
            MSFlexGrid.Row = MSFlexGrid.Row + 1
        Case vbKeyLeft
            If (MSFlexGrid.Col = 1) Then Exit Sub
            GridText.Visible = False
        Case vbKeyRight
            If (MSFlexGrid.Col = MSFlexGrid.Cols - 1) Then
Exit Sub
            GridText.Visible = False
        Case vbKeyEscape
            GridText.Visible = False
    End Select
End Sub
```

```
Private Sub GridText_KeyPress(KeyAscii As Integer)
    Dim Tmpstr$

KeyAscii = TextBoxKeysFilter("~!`@#$%^&*()-_=+|\/:;,.",
KeyAscii)
    Select Case KeyAscii
        Case vbKeyReturn
            Tmpstr$ = GridText.Text
            Call SaveTextBox(Tmpstr$)
            KeyAscii = 0
            GridText.Visible = False
    End Select
End Sub
Private Sub GridText_LostFocus()
```

```
    Dim Tmpstr$
    Dim NewStr$

    If (MSFlexGrid.Col = 1) Then Exit Sub
        Tmpstr$ = MSFlexGrid.Text
        NewStr$ = GridText.Text
    If (Tmpstr$ <> NewStr$) Then
        Call SaveTextBox(NewStr$)
    End If

End Sub
```

```
Private Sub MSFlexGrid_EnterCell()
    Dim CellTop%, CellLeft%
    Dim CellWidth%, CellHeight%

    ' if select col 1 then do nothing
    If (MSFlexGrid.Col = 1) Then
        GridText.Visible = False
        Exit Sub
    End If

    Select Case MSFlexGrid.Col
        Case 1
            GridText.Visible = False
            Exit Sub
        Case 2:     GridText.MaxLength = 12
        Case 11:    GridText.MaxLength = 50
    End Select

    CellTop% = MSFlexGrid.CellTop
    CellLeft% = MSFlexGrid.CellLeft
    CellWidth% = MSFlexGrid.CellWidth
    CellHeight% = MSFlexGrid.CellHeight
    ' display the text box
    Call ShowTextBox(CellLeft%, CellTop%, CellWidth%,
CellHeight%)
    ' grap the content of the cell
```

```
    GridText.Text = "" & MSFlexGrid.Text
End Sub

'Set Setup interface invisible

Private Sub OKButton_Click()
    DlgSetup.Visible = False
End Sub

'Set volumn
Private Sub SldVolume_Scroll()
    Dim ValSet As Long
    ValSet = (SldVolume.Value * volCtrl.lMaximum) /
SldVolume.Max
    Call SetVol(ValSet)
End Sub

'Position frame on tab strip

Private Sub tbsOptions_Click()

    Select Case tbsOptions.SelectedItem.index

    Case 1
        SetupType = SetupDeviceType
        frameData.Left = 240
        frameSpeech.Left = -20000
        frameMisc.Left = -20000
        FrameAni.Left = -20000
    Case 2
        SetupType = SetupVoiceType
        frameSpeech.Left = 240
        frameData.Left = -20000
        frameMisc.Left = -20000
        FrameAni.Left = -20000
        Call SetVoice
    Case 3
        frameMisc.Left = 240
```

```
        frameData.Left = -20000
        frameSpeech.Left = -20000
        FrameAni.Left = -20000
    Case 4
        FrameAni.Left = 240
        frameData.Left = -20000
        frameSpeech.Left = -20000
        frameMisc.Left = -20000
    End Select
End Sub
```

```
Private Sub Check1_Click()
    If Check1.Value = 1 Then
        Check2.Value = 0
        frmDocument.Picture1.Enabled = True
        toggle = 1
        fMainForm.phone1.initialized = 1
    End If
    click1 = Check1.Value
    click2 = Check2.Value
    click3 = Check3.Value
    click4 = Check4.Value
End Sub
```

```
Private Sub Check2_Click()

    If Check2.Value = 1 Then
        Check1.Value = 0
        Check3.Value = 0
        Check4.Value = 0
        frmDocument.Picture1.Enabled = False
    End If
    click1 = Check1.Value
    click2 = Check2.Value
    click3 = Check3.Value
    click4 = Check4.Value
End Sub
```

```
Private Sub Check3_Click()

    If Check3.Value = 1 Then
        Check2.Value = 0
        Check4.Value = 0
        Check1.Value = 1
    End If
    click1 = Check1.Value
    click2 = Check2.Value
    click3 = Check3.Value
    click4 = Check4.Value
End Sub
```

```
Private Sub Check4_Click()

     If Check4.Value = 1 Then
        Check2.Value = 0
        Check1.Value = 1
        Check3.Value = 0
    End If
    click1 = Check1.Value
    click2 = Check2.Value
    click3 = Check3.Value
    click4 = Check4.Value
End Sub
```

```
'Sub-procedure name:  ChangeMode
'Purposes:       Change the Control mode.
'Function called:   CallDialog, ViewDirectory
'Corrupted Global variables:    none
'Input:
'Output:         Phone Control dialog if phone Ctr mode is
```

```
Private Sub ChangeMode()
Dim ring As Long

    click1 = Check1.Value
    click2 = Check2.Value
    click3 = Check3.Value
    click4 = Check4.Value

    If click2 = 1 And click1 = 0 Then
        frmDocument.Enabled = True
        frmDocument.Picture1.Visible = False
    ElseIf click1 = 1 And click2 = 0 Then
        fMainForm.phone1.AnswerAfterRings = ring
        frmDocument.Enabled = False
        Call ViewDirectory
        frmDocument.Picture1.Visible = True

        If click3 = 1 Then  'And click1 = 1 Then
            fMainForm.phone1.initialized = 1 'run on emulator
            fMainForm.phone1.CallDialog
            Check1.Value = 0
            frmDocument.Picture1.Visible = False
            frmDocument.Enabled = True
        End If
        If click4 = 1 Then  'And click1 = 1 Then
            fMainForm.phone1.initialized = 2 'run on real
phone
            fMainForm.phone1.CallDialog
            Check1.Value = 0
            frmDocument.Picture1.Visible = False
            frmDocument.Enabled = True
        End If
    End If
End Sub
```

- **Monitoring Call**

```
Dim ring As Integer
Public counts As Integer

Private Sub Command1_Click()
      frmDocument.CmdOn_Click (0)
End Sub
```

```
Private Sub Command2_Click()
      Out &H378, 0
      Delay 1000
      Out &H378, 16
End Sub
```

```
Private Sub Form_Load()

      Let ring = 0
      Let counts = 0
      Text1.Text = 0
      Text2.Text = ""
      Timer2.Interval = 1000
      Timer1.Interval = 1000
      Out &H378, 16
      Timer2.Enabled = True
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
      Unload Me
End Sub
```

```
Private Sub Hang_Up_Click()
      Out &H378, 0
      Timer2.Enabled = True
      Text2.Text = "Disconnect"
      Timer1.Enabled = False
```

```
      frmDocument.PhoneCtrMode.Value = 0
      Unload Me
      frmDocument.Enabled = True
      DlgSetup.P_flage = 0
End Sub
```

```
Private Sub Timer1_Timer()

 ' set  idle time interval is 30 seconds
  Text1 = Text1 + 1
  counts = counts + 1
  If (Text1 + Str(counts)) = (Text1 + Str(30)) Then
      Text2.Text = "Connection time out"
      Out &H378, 0
       Timer2.Enabled = True
       Timer1.Enabled = False
  End If
      'DlgSetup.P_flage = 0
End Sub
```

```
Private Sub Timer2_Timer()
  Dim P_status As Integer

  counts = 0
    If Text1.Text <> 0 Then
        Text1 = Text1 + 1
    End If

    P_status = Inp(&H379)
    Debug.Print P_status

    If P_status >= 24 Then
        ring = ring + 1
        Text2.Text = "ring" & Str(ring)
    End If
'answer call
    If ring = 5 Then
```

```
        ring = 0
        Out &H378, 20   'phone connect
        'DlgSetup.P_flage = 1
        Timer2.Enabled = False
        Text2.Text = ""
        Timer1.Enabled = True
    End If

End Sub
```

- **Data Base Form**

```
Private Const MARGIN_SIZE = 60      ' in Twips
' variables for enabling column sort
Private m_iSortCol As Integer
Private m_iSortType As Integer

' variables for column dragging
Private m_bDragOK As Boolean
Private m_iDragCol As Integer
Private xdn As Integer, ydn As Integer
```

```
Private Sub Form_Load()
    Dim i As Integer
    Dim j As Integer
    Dim m_iMaxCol As Integer
    Dim wide As Integer

    datPrimaryRS.Visible = False

    With MSHFlexGrid1

        .Redraw = False
        ' place the columns in the right order
        .ColData(0) = 0
        .ColData(1) = 1
        .ColData(2) = 2
```

```
        .ColData(7) = 3
        .ColData(3) = 4
        .ColData(4) = 5
        .ColData(5) = 6
        .ColData(6) = 7
        .RowHeight(17) = 0
        .RowHeight(18) = 0
        ' loop to re-order the columns
        For i = 0 To .Cols - 1
            m_iMaxCol = i                    ' find the highest
value starting from this column
            For j = i To .Cols - 1
                If .ColData(j) > .ColData(m_iMaxCol) Then
m_iMaxCol = j
            Next j
            .ColPosition(m_iMaxCol) = 0      ' move the column
with the max value to the left
        Next i

        ' set grid's column widths
        .ColWidth(0) = 400
        .ColWidth(1) = 1800
        .ColWidth(2) = 1200
        .ColWidth(3) = 3200
        .ColWidth(4) = 1500
        .ColWidth(5) = 1500
        .ColWidth(6) = 1500
        .ColWidth(7) = 1500

        ' set grid's style
        .AllowBigSelection = True
        .FillStyle = flexFillRepeat

        ' make header bold
        .Row = 0
        .Col = 0
        .RowSel = .FixedRows - 1
        .ColSel = .Cols - 1
        .CellFontBold = True
```

```
        .AllowBigSelection = False
        .FillStyle = flexFillSingle
        .Redraw = True

    End With

    For i% = 0 To 7
    wide = wide + MSHFlexGrid1.ColWidth(i)

    Next i
  DevicesDTBase.Width = wide
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
Unload Me
End Sub
```

```
Private Sub MSHFlexGrid1_MouseDown(Button As Integer, shift As
Integer, X As Single, Y As Single)
'--------------------------------------------------------------

' code in grid's DragDrop, MouseDown, MouseMove, and MouseUp
events enables column dragging
'--------------------------------------------------------------
    If MSHFlexGrid1.MouseRow <> 0 Then Exit Sub

    xdn = X
    ydn = Y
    m_iDragCol = -1      ' clear drag flag
    m_bDragOK = True
End Sub
```

```
Private Sub MSHFlexGrid1_MouseMove(Button As Integer, shift As
Integer, X As Single, Y As Single)
'--------------------------------------------------------------

' code in grid's DragDrop, MouseDown, MouseMove, and MouseUp
events enables column dragging
'--------------------------------------------------------------
```

```
    ' test to see if we should start drag
    If Not m_bDragOK Then Exit Sub
    If Button <> 1 Then Exit Sub                          '
wrong button
    If m_iDragCol <> -1 Then Exit Sub                     '
already dragging
    If Abs(xdn - X) + Abs(ydn - Y) < 50 Then Exit Sub     '
didn't move enough yet
    If MSHFlexGrid1.MouseRow <> 0 Then Exit Sub           ' must
drag header

    ' if got to here then start the drag
    m_iDragCol = MSHFlexGrid1.MouseCol
    MSHFlexGrid1.Drag vbBeginDrag
End Sub
```

```
Private Sub MSHFlexGrid1_MouseUp(Button As Integer, shift As
Integer, X As Single, Y As Single)
'--------------------------------------------------------------
' code in grid's DragDrop, MouseDown, MouseMove, and MouseUp
events enables column dragging
'--------------------------------------------------------------
    m_bDragOK = False
End Sub
```

```
Private Sub MSHFlexGrid1_DblClick()
'--------------------------------------------------------------
' code in grid's DblClick event enables column sorting
'--------------------------------------------------------------
    Dim i As Integer

    ' sort only when a fixed row is clicked
    If MSHFlexGrid1.MouseRow >= MSHFlexGrid1.FixedRows Then
Exit Sub

    i = m_iSortCol                    ' save old column
    m_iSortCol = MSHFlexGrid1.Col     ' set new column

    ' increment sort type
    If i <> m_iSortCol Then
```

```
        ' if clicking on a new column, start with ascending
sort
        m_iSortType = 1
    Else
        ' if clicking on the same column, toggle between
ascending and descending sort
        m_iSortType = m_iSortType + 1
    If m_iSortType = 3 Then m_iSortType = 1
    End If
    DoColumnSort
End Sub
```

```
Sub DoColumnSort()
' does Exchange-type sort on column m_iSortCol
'-------------------------------------------------------------
    With MSHFlexGrid1
        .Redraw = False
        .Row = 1
        .RowSel = .Rows - 1
        .Col = m_iSortCol
        .Sort = m_iSortType
        .Redraw = True
    End With
End Sub
```

```
Private Sub Form_Resize()

    Dim sngButtonTop As Single
    Dim sngScaleWidth As Single
    Dim sngScaleHeight As Single

    On Error GoTo Form_Resize_Error

    With Me
        sngScaleWidth = .ScaleWidth
        sngScaleHeight = .ScaleHeight

        ' move Close button to the lower right corner
```

```
        With .cmdClose
        sngButtonTop = sngScaleHeight - (.Height + MARGIN_SIZE)
        .Move sngScaleWidth - (.Width+MARGIN_SIZE),sngButtonTop
        End With

        .MSHFlexGrid1.Move MARGIN_SIZE, _
            MARGIN_SIZE, _
            sngScaleWidth - (2 * MARGIN_SIZE), _
            sngButtonTop - (2 * MARGIN_SIZE)

    End With
    Exit Sub
Form_Resize_Error:
    ' avoid error on negative values
    Resume Next
End Sub
```

```
Private Sub cmdClose_Click()
    Unload Me
End Sub
```

- **Module Send Data to LPT**

```
'Declare Inp and Out for port I/O
Public Declare Function Inp Lib "inpout32.dll" _
Alias "Inp32" (ByVal PortAddress As Integer) As Byte
Public Declare Sub Out Lib "inpout32.dll" _
Alias "Out32" (ByVal PortAddress As Integer, ByVal Value As
Byte)
```

- **Module Update data base**

```
Option Explicit

Function FindRecord(DataSource As Data, Ref As Integer) As
Boolean
    With DataSource.Recordset
```

```
        .MoveFirst
        .FindFirst ("REF=" & Ref)
    End With
End Function
'*****************************************
' Update the current record by field name
'*****************************************
Function UpdateDB(DataSource As Data, FieldName As String,
DataVal As Variant) As Boolean
    With DataSource.Recordset
        .Edit
        .fields(FieldName).Value = DataVal
        .Update
    End With
End Function

Sub UpdateCommandRec(ValSet As Boolean)

    ' update fo the current record
    Dim sxValue$
    Dim RetBVal As Boolean

    If (ValSet) Then sxValue$ = "On"
    If (Not ValSet) Then sxValue$ = "Off"
    RetBVal = UpdateDB(fMainForm.Data, "Time", Time)
    RetBVal = UpdateDB(fMainForm.Data, "Status", ValSet)
    RetBVal = UpdateDB(fMainForm.Data, "status_text",
sxValue$)
End Sub
```

- **Module Load form**

```
Sub FormLoad()
    Dim ScreenWidth As Long
    Dim ScreenHeight    As Long
    Dim X%

    With frmDocument
```

```
        ScreenWidth = .ScaleWidth - 300
        ScreenHeight = .ScaleHeight - 200
        .Frame1.Move 10, 8700, frmDocument.Width - 40, 1000
        .PhoneCtrMode.Move 220, 9000
        .P_monotor.Move 2500, 8900
        .DevName(0).Move 220, 1500, 2500, 350
        .DevName(8).Move ((ScreenWidth / 2) + 220), 1500,
2500, 350
        .CmdOn(0).Move .DevName(0).Width + 350, 1500
        .CmdOn(8).Move ((ScreenWidth / 2) + 220) +
.DevName(8).Width + 350, 1500
        .CmdOff(0).Move .CmdOn(0).Width + .CmdOn(0).Left +
250, 1500
        .CmdOff(8).Move .CmdOn(8).Width + .CmdOn(8).Left +
250, 1500
        .LblTime(0).Move .CmdOff(0).Width + .CmdOff(0).Left +
250, 1500
        .LblTime(8).Move .CmdOff(8).Width + .CmdOff(8).Left +
250, 1500

        For X% = 1 To 7

            .DevName(X%).Move 220, .DevName(X% - 1).Top +
(ScreenHeight / 10), 2500, 350
            .CmdOn(X%).Move .DevName(X%).Width + 350,
.DevName(X% - 1).Top + (ScreenHeight / 10)
            .CmdOff(X%).Move .CmdOn(X%).Width +
.CmdOn(X%).Left + 250, .DevName(X% - 1).Top + (ScreenHeight /
10)
            .LblTime(X%).Move .CmdOff(X%).Width +
.CmdOff(X%).Left + 250, .DevName(X% - 1).Top + (ScreenHeight /
10)
        Next X%

        For X% = 9 To 15

            .DevName(X%).Move ((ScreenWidth / 2) + 220),
.DevName(X% - 1).Top + (ScreenHeight / 10), 2500, 350
            .CmdOn(X%).Move ((ScreenWidth / 2) + 220) +
.DevName(X).Width + 350, .DevName(X% - 1).Top + (ScreenHeight
/ 10)
```

```
            .CmdOff(X%).Move .CmdOn(X%).Width +
.CmdOn(X%).Left + 250, .DevName(X% - 1).Top + (ScreenHeight /
10)
            .LblTime(X%).Move .CmdOff(X%).Width +
.CmdOff(X%).Left + 250, .DevName(X% - 1).Top + (ScreenHeight /
10)


        Next X%


        ' Load device name from db


        fMainForm.Data.Recordset.MoveFirst
        For X% = 0 To 17


            .DevName(X%).Caption = "" &
fMainForm.Data.Recordset.fields("Device_Name")
            fMainForm.Data.Recordset.Edit
            If (fMainForm.Data.Recordset.fields("Status"))
Then

fMainForm.Data.Recordset.fields("status_text").Value = "On"
            Else

fMainForm.Data.Recordset.fields("status_text").Value = "Off"
            End If
            If
(IsNull(fMainForm.Data.Recordset.fields("Time"))) Then
                .LblTime(X%).Caption = Time
            Else
                .LblTime(X%).Caption =
fMainForm.Data.Recordset.fields("Time").Value
            End If
            fMainForm.Data.Recordset.Update
            fMainForm.Data.Recordset.MoveNext


        Next X%


        'For Y% = 16 To 17


        '    .DevName(Y%).Caption = "" &
fMainForm.Data.Recordset.Fields("Device_Name")
```

```
        'Next Y%
    End With

End Sub
```

- **Module Load Main MDI form and initialise**

```
Private Declare Function waveOutGetNumDevs Lib "winmm.dll" ()
As Long

Public Const SetupDeviceType = 0
Public Const SetupVoiceType = 1
Public Const MsgBoxMode = 0
Public Const VoiceMsgMode = 1

Public engine As Long

Public gMyMenu As Long           ' voice command handle
Public hmixer As Long           ' mixer handle
Public volCtrl As MIXERCONTROL ' waveout volume control
Public micCtrl As MIXERCONTROL ' microphone volume control
Public rc As Long               ' return code
Public OK As Boolean            ' boolean return code
Public vol As Long

Public fMainForm As frmMain
Public SetupType As Integer
Public ModeName As String
Public MsgMode As Integer

Sub Main()
    Dim fLogin As New frmLogin
    Dim i%

    ' detect sound card
    i = waveOutGetNumDevs()
    If (i = 0) Then
        MsgBox "Sound Card not found.", vbInformation
```

```
        End
    End If

    fLogin.Show vbModal
    If Not fLogin.OK Then
        'Login Failed so exit app
        End
    End If
    Unload fLogin

    frmSplash.Show
    frmSplash.Refresh
    Set fMainForm = New frmMain
    Load fMainForm
    Unload frmSplash

    fMainForm.Show
End Sub
```

```
Sub LoadResStrings(frm As Form)
    On Error Resume Next
    Dim obj As Object
    Dim fnt As Object
    Dim sCtlType As String
    Dim nVal As Integer

    'set the form's caption
    frm.Caption = LoadResString(CInt(frm.Tag))

    'set the font
    Set fnt = frm.Font
    fnt.Name = LoadResString(20)
    fnt.size = CInt(LoadResString(21))

    'set the controls' captions using the caption
    'property for menu items and the Tag property
    'for all other controls
    For Each ctl In frm.Controls
```

```
        Set ctl.Font = fnt
        sCtlType = TypeName(ctl)
        If sCtlType = "Label" Then
            ctl.Caption = LoadResString(CInt(ctl.Tag))
        ElseIf sCtlType = "Menu" Then
            ctl.Caption = LoadResString(CInt(ctl.Caption))
        ElseIf sCtlType = "TabStrip" Then
            For Each obj In ctl.Tabs
                obj.Caption = LoadResString(CInt(obj.Tag))
                obj.ToolTipText =
LoadResString(CInt(obj.ToolTipText))
            Next
        ElseIf sCtlType = "Toolbar" Then
            For Each obj In ctl.Buttons
                obj.ToolTipText =
LoadResString(CInt(obj.ToolTipText))
            Next
        ElseIf sCtlType = "ListView" Then
            For Each obj In ctl.ColumnHeaders
                obj.Text = LoadResString(CInt(obj.Tag))
            Next
        Else
            nVal = 0
            nVal = Val(ctl.Tag)
            If nVal > 0 Then ctl.Caption = LoadResString(nVal)
            nVal = 0
            nVal = Val(ctl.ToolTipText)
            If nVal > 0 Then ctl.ToolTipText =
LoadResString(nVal)
        End If
    Next
End Sub
```

- **Module Send Control Data**

```
---------------------------------------------------------------

Option Explicit

'Declare Inp and Out for port I/O
Private Declare Function Inp Lib "inpout32.dll" Alias "Inp32"
(ByVal PortAddress As Integer) As Byte
Private Declare Sub Out Lib "inpout32.dll" Alias "Out32"
(ByVal PortAddress As Integer, ByVal Value As Byte)
'*************************************************************
Sub Delay(Value As Long)

    Do While Value > 0
        Value = Value - 1
    Loop
End Sub
```

```
Sub SendControl(index As Integer, CtrlVal As Boolean)
    Dim mask, tim As Integer
    ReDim Data(0 To 12) As Integer
    Dim clock1, clock2 As Integer
    Dim i%
    Dim P_control%

    Data(6) = 0
    'frmDocument.Picture1.Cls
    'PMainControl.Picture1.Cls
    tim = 1

    For i% = 1 To 12
        Data(i%) = 0
    Next i%
    clock1 = 2
    clock2 = 0
    If (CtrlVal) Then Data(6) = 1

    For i% = 1 To 5
```

```
        Data(i%) = index Mod 2
        mask = index \ 2
        index = mask
    Next i%
    Data(0) = 1
    Rem send data

    If frmDocument.PCtr_flage = 1 Then
    P_control% = 20
    Else: P_control% = 0
    End If

    Out &H378, (0 + P_control%)
    Delay (tim)

    For i% = 8 To 1 Step -1

        Out &H378, (Data(i%) + P_control%)
        Delay (30000)

        Out &H378, (Data(i%) + (2 + P_control%))
        Delay (15000)
        Out &H378, (0 + P_control%)
        Delay (15000)

        'frmDocument.Picture1.Print Data(i%)
        'PMainControl.Picture1.Print Data(i%)
        Out &H378, (0 + P_control%)
    Next i%
End Sub
```

- **Module Provide Set up functions**

```
Option Explicit

Sub Set_Grid(GridType As Integer)
    Dim X%
    With DlgSetup.MSFlexGrid
```

```
        For X% = 1 To .Cols - 1
            .ColAlignment(X%) = flexAlignCenterCenter
        Next X%

        Select Case GridType
        Case 0
            .ColWidth(0) = 300
            .ColWidth(1) = 500          ' Ref
            .ColWidth(2) = 2200         ' Device_Name
            .RowHeight(17) = 0
            .RowHeight(18) = 0
            For X% = 3 To .Cols - 2
                .ColWidth(X%) = 0
            Next X%
            .ColWidth(11) = .Width - .ColWidth(0) -
.ColWidth(1) - .ColWidth(2) - 100
        Case 1
        End Select
    End With
End Sub
```

```
Sub ShowTextBox(Left As Integer, Top As Integer, _
                Width As Integer, Height As Integer)
    With DlgSetup
        .GridText.Move Left, Top + 125, Width, Height
        .GridText.Visible = True
        .GridText.SetFocus
    End With
End Sub
```

```
Sub DisableTextBox()
        DlgSetup.GridText.Visible = False
End Sub
```

```
Function TextBoxKeysFilter(FieldKey As String, KeyAscii As
Integer) As Integer
    Dim CharIn$

    CharIn$ = Chr$(KeyAscii)
    If (InStr(FieldKey, CharIn$) > 0) Then
        TextBoxKeysFilter = 0
    Else
        TextBoxKeysFilter = KeyAscii
    End If
End Function
```

```
Sub SaveTextBox(DataStr As String)
    Dim RetBVal As Boolean
    Dim RowRef%

    With DlgSetup
        RowRef% = .MSFlexGrid.Row
        RetBVal = FindRecord(.Data, RowRef%)
        .Data.Recordset.Edit
        .Data.Recordset.fields(1).Value = DataStr
        .Data.Recordset.Update
        .MSFlexGrid.Text = DataStr

    End With
End Sub
```

```
Sub SetVoice()
    Dim X%

    For X% = 1 To fMainForm.TextToSpeech.CountEngines
        ModeName = fMainForm.TextToSpeech.ModeName(X%)
        DlgSetup.CmbVoiceType.AddItem ModeName
    Next X%

    fMainForm.TextToSpeech.Speed = 150
    DlgSetup.CmbVoiceType.ListIndex =
fMainForm.TextToSpeech.CurrentMode - 1
```

```
    DlgSetup.TxtTest.Text = "The quick brown fox jumps over
the lazy dog."
End Sub
```

- **Module Speech**

```
'Function name: Menu_Create
'Purposes:      Create voice command menu from the database.
'Function called:   None
'Corrupted Global variables:    none
'Input:         Device Database, field "device name".
'Output:        gMenu (Voice command menu for the program
control)
```

```
Sub Menu_Create(gMyMenu As Long)
    Dim X%
    gMyMenu = fMainForm.Vcommand.MenuCreate(App.EXEName,
"state1", 4)
    fMainForm.Data.Recordset.MoveFirst
    For X% = 0 To 17
        'fMainForm.Vcommand.AddCommand gMyMenu, 1,
frmDocument.DevName(X%).Caption, "when you say" +
frmDocument.DevName(X%).Caption, "listen list", 0, ""
        fMainForm.Vcommand.AddCommand gMyMenu, 1,
fMainForm.Data.Recordset.fields("Device_Name"), "when you say"
+ fMainForm.Data.Recordset.fields("Device_Name"), "listen
list", 0, ""
    fMainForm.Data.Recordset.MoveNext

    Next X%
End Sub
```

```
'Function for speech engine to synthesis the
'resource string.
'************************************
Sub Speech_SayIt(ByVal szValue As String)
   fMainForm.TextToSpeech.Speak szValue
End Sub
```

- **Module Set volume functions**

```
---------------------------------------------------------------
      Public Const MMSYSERR_NOERROR = 0
      Public Const MAXPNAMELEN = 32
      Public Const MIXER_LONG_NAME_CHARS = 64
      Public Const MIXER_SHORT_NAME_CHARS = 16
      Public Const MIXER_GETLINEINFOF_COMPONENTTYPE = &H3&
      Public Const MIXER_GETCONTROLDETAILSF_VALUE = &H0&
      Public Const MIXER_GETLINECONTROLSF_ONEBYTYPE = &H2&
      Public Const MIXERLINE_COMPONENTTYPE_DST_FIRST = &H0&
      Public Const MIXERLINE_COMPONENTTYPE_SRC_FIRST = &H1000&

      Public Const MIXERLINE_COMPONENTTYPE_DST_SPEAKERS = _
                     (MIXERLINE_COMPONENTTYPE_DST_FIRST + 4)

      Public Const MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE = _
                     (MIXERLINE_COMPONENTTYPE_SRC_FIRST + 3)

      Public Const MIXERLINE_COMPONENTTYPE_SRC_LINE = _
                     (MIXERLINE_COMPONENTTYPE_SRC_FIRST + 2)

      Public Const MIXERCONTROL_CT_CLASS_FADER = &H50000000
      Public Const MIXERCONTROL_CT_UNITS_UNSIGNED = &H30000

      Public Const MIXERCONTROL_CONTROLTYPE_FADER = _
                     (MIXERCONTROL_CT_CLASS_FADER Or _
                     MIXERCONTROL_CT_UNITS_UNSIGNED)

      Public Const MIXERCONTROL_CONTROLTYPE_VOLUME = _
                     (MIXERCONTROL_CONTROLTYPE_FADER + 1)

      Declare Function mixerClose Lib "winmm.dll" _
                     (ByVal hmx As Long) As Long

      Declare Function mixerGetControlDetails Lib "winmm.dll"
_
                     Alias "mixerGetControlDetailsA" _
                     (ByVal hmxobj As Long, _
```

```
                pmxcd As MIXERCONTROLDETAILS, _
                ByVal fdwDetails As Long) As Long

    Declare Function mixerGetDevCaps Lib "winmm.dll" _
                Alias "mixerGetDevCapsA" _
                (ByVal uMxId As Long, _
                ByVal pmxcaps As MIXERCAPS, _
                ByVal cbmxcaps As Long) As Long

    Declare Function mixerGetID Lib "winmm.dll" _
                (ByVal hmxobj As Long, _
                pumxID As Long, _
                ByVal fdwId As Long) As Long

    Declare Function mixerGetLineControls Lib "winmm.dll" _
                Alias "mixerGetLineControlsA" _
                (ByVal hmxobj As Long, _
                pmxlc As MIXERLINECONTROLS, _
                ByVal fdwControls As Long) As Long

    Declare Function mixerGetLineInfo Lib "winmm.dll" _
                Alias "mixerGetLineInfoA" _
                (ByVal hmxobj As Long, _
                pmxl As MIXERLINE, _
                ByVal fdwInfo As Long) As Long

    Declare Function mixerGetNumDevs Lib "winmm.dll" () As
Long

    Declare Function mixerMessage Lib "winmm.dll" _
                (ByVal hmx As Long, _
                ByVal uMsg As Long, _
                ByVal dwParam1 As Long, _
                ByVal dwParam2 As Long) As Long

    Declare Function mixerOpen Lib "winmm.dll" _
                (phmx As Long, _
                ByVal uMxId As Long, _
```

```
                    ByVal dwCallback As Long, _
                    ByVal dwInstance As Long, _
                    ByVal fdwOpen As Long) As Long

    Declare Function mixerSetControlDetails Lib "winmm.dll"
_(ByVal hmxobj As Long, _pmxcd As MIXERCONTROLDETAILS, _
ByVal fdwDetails As Long) As Long

    Declare Sub CopyStructFromPtr Lib "kernel32" _
                    Alias "RtlMoveMemory" _
                    (struct As Any, _
                    ByVal ptr As Long, ByVal cb As Long)

    Declare Sub CopyPtrFromStruct Lib "kernel32" _
                    Alias "RtlMoveMemory" _
                    (ByVal ptr As Long, _
                    struct As Any, _
                    ByVal cb As Long)

    Declare Function GlobalAlloc Lib "kernel32" _
                    (ByVal wFlags As Long, _
                    ByVal dwBytes As Long) As Long

    Declare Function GlobalLock Lib "kernel32" _
                    (ByVal hmem As Long) As Long

    Declare Function GlobalFree Lib "kernel32" _
                    (ByVal hmem As Long) As Long

    Type MIXERCAPS

       wMid As Integer                    '  manufacturer id
       wPid As Integer                    '  product id
       vDriverVersion As Long             '  version of the
driver
       szPname As String * MAXPNAMELEN    '  product name
       fdwSupport As Long                 '  misc. support
bits
```

```
        cDestinations As Long              '  count of
destinations

      End Type

      Type MIXERCONTROL

        cbStruct As Long            '  size in Byte of
MIXERCONTROL
        dwControlID As Long         '  unique control id for
mixer device
        dwControlType As Long       '
MIXERCONTROL_CONTROLTYPE_xxx
        fdwControl As Long          '
MIXERCONTROL_CONTROLF_xxx
        cMultipleItems As Long      '  if
MIXERCONTROL_CONTROLF_MULTIPLE set
        szShortName As String * MIXER_SHORT_NAME_CHARS   '
short name of control
        szName As String * MIXER_LONG_NAME_CHARS         '
long name of control
        lMinimum As Long            '  Minimum value

        ' Obtain a line corresponding to the component type
        rc = mixerGetLineInfo(hmixer, mxl,
MIXER_GETLINEINFOF_COMPONENTTYPE)

        If (MMSYSERR_NOERROR = rc) Then
            mxlc.cbStruct = Len(mxlc)
            mxlc.dwLineID = mxl.dwLineID
            mxlc.dwControl = ctrlType
            mxlc.cControls = 1
            mxlc.cbmxctrl = Len(mxc)

            ' Allocate a buffer for the control
            hmem = GlobalAlloc(&H40, Len(mxc))
            mxlc.pamxctrl = GlobalLock(hmem)
            mxc.cbStruct = Len(mxc)

            ' Get the control
            rc = mixerGetLineControls(hmixer, _
```

```
                                        mxlc, _
MIXER_GETLINECONTROLSF_ONEBYTYPE)
            If (MMSYSERR_NOERROR = rc) Then
                GetVolumeControl = True
                ' Copy the control into the destination
structure
                CopyStructFromPtr mxc, mxlc.pamxctrl,
Len(mxc)
            Else
                GetVolumeControl = False
            End If
            GlobalFree (hmem)
            Exit Function
        End If
        GetVolumeControl = False
    End Function
```

```
    Function SetVolumeControl(ByVal hmixer As Long, _
                              mxc As MIXERCONTROL, _
                              ByVal volume As Long) As Boolean
    'This function sets the value for a volume control.
Returns True if successful

        Dim mxcd As MIXERCONTROLDETAILS
        Dim vol As MIXERCONTROLDETAILS_UNSIGNED

        mxcd.item = 0
        mxcd.dwControlID = mxc.dwControlID
        mxcd.cbStruct = Len(mxcd)
        mxcd.cbDetails = Len(vol)

        ' Allocate a buffer for the control value buffer
        hmem = GlobalAlloc(&H40, Len(vol))
        mxcd.paDetails = GlobalLock(hmem)
        mxcd.cChannels = 1
        vol.dwValue = volume
        ' Copy the data into the control value buffer
        CopyPtrFromStruct mxcd.paDetails, vol, Len(vol)
```

```
        ' Set the control value
        rc = mixerSetControlDetails(hmixer, _
                                    mxcd, _

MIXER_SETCONTROLDETAILSF_VALUE)
        GlobalFree (hmem)
        If (MMSYSERR_NOERROR = rc) Then
            SetVolumeControl = True
        Else
            SetVolumeControl = False
        End If
    End Function
```

```
Sub SetVol(zlvalue As Long)
    SetVolumeControl hmixer, volCtrl, zlvalue
End Sub
```

```
Sub SetMicVol(zlvalue As Long)
    SetVolumeControl hmixer, micCtrl, zlvalue
End Sub
```

- **Module Show Record Message**

```
'------------------------------------------------------------
Global Const Modal = 1
Public Sub ViewDirectory()
'This function simply causes the Explorer to display the
Messages directory (and creates it if needed),
'so each time a new message is recorded, an icon appears that
can be double clicked on.
On Error Resume Next
Dim slash As String
If Right(App.Path, 1) = "\" Then
    slash = ""
Else
    slash = "\"
End If
    mdir = App.Path + slash + "Messages"
```

```
    MkDir mdir
    ChDrive mdir
    ChDir mdir
    Shell "command.com /c start .", vbMinimizedNoFocus
    Shell "cmd /c start .", vbMinimizedNoFocus
End Sub
```

- **Show About dialog**

```
' Reg Key Security Options...
Const KEY_ALL_ACCESS = &H2003F

' Reg Key ROOT Types...
Const HKEY_LOCAL_MACHINE = &H80000002
Const ERROR_SUCCESS = 0
Const REG_SZ = 1                          ' Unicode nul
terminated string
Const REG_DWORD = 4                       ' 32-bit number

Const gREGKEYSYSINFOLOC = "SOFTWARE\Microsoft\Shared Tools
Location"
Const gREGVALSYSINFOLOC = "MSINFO"
Const gREGKEYSYSINFO = "SOFTWARE\Microsoft\Shared
Tools\MSINFO"
Const gREGVALSYSINFO = "PATH"

Private Declare Function RegOpenKeyEx Lib "advapi32" Alias
"RegOpenKeyExA" (ByVal hKey As Long, ByVal lpSubKey As String,
ByVal ulOptions As Long, ByVal samDesired As Long, ByRef
phkResult As Long) As Long
Private Declare Function RegQueryValueEx Lib "advapi32" Alias
"RegQueryValueExA" (ByVal hKey As Long, ByVal lpValueName As
String, ByVal lpReserved As Long, ByRef lpType As Long, ByVal
lpData As String, ByRef lpcbData As Long) As Long
Private Declare Function RegCloseKey Lib "advapi32" (ByVal
hKey As Long) As Long
```

```
Private Sub Form_Load()
    LoadResStrings Me
    lblVersion.Caption = "Version " & App.Major & "." &
App.Minor & "." & App.Revision
    lblTitle.Caption = App.Title
End Sub
```

```
Private Sub cmdSysInfo_Click()
        Call StartSysInfo
End Sub
```

```
Private Sub cmdOK_Click()
     Set frmAbout = Nothing
           Unload Me
End Sub
```

```
Public Sub StartSysInfo()
    On Error GoTo SysInfoErr
        Dim rc As Long
        Dim SysInfoPath As String

        ' Try To Get System Info Program Path\Name From
Registry...
        If GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFO,
gREGVALSYSINFO, SysInfoPath) Then
        ' Try To Get System Info Program Path Only From
Registry...
        ElseIf GetKeyValue(HKEY_LOCAL_MACHINE,
gREGKEYSYSINFOLOC, gREGVALSYSINFOLOC, SysInfoPath) Then
                ' Validate Existance Of Known 32 Bit File
Version
                If (Dir(SysInfoPath & "\MSINFO32.EXE") <> "")
Then
                        SysInfoPath = SysInfoPath &
"\MSINFO32.EXE"

                ' Error - File Can Not Be Found...
                Else
                        GoTo SysInfoErr
                End If
```

```
        ' Error - Registry Entry Can Not Be Found...
        Else
                GoTo SysInfoErr
        End If
        Call Shell(SysInfoPath, vbNormalFocus)

        Exit Sub
SysInfoErr:
        MsgBox "System Information Is Unavailable At This
Time", vbOKOnly
End Sub
```

```
Public Function GetKeyValue(KeyRoot As Long, KeyName As
String, SubKeyRef As String, ByRef KeyVal As String) As
Boolean
        Dim i As Long
' Loop Counter
        Dim rc As Long
' Return Code
        Dim hKey As Long
' Handle To An Open Registry Key
        Dim hDepth As Long
'
        Dim KeyValType As Long
' Data Type Of A Registry Key
        Dim tmpVal As String
' Tempory Storage For A Registry Key Value
        Dim KeyValSize As Long
' Size Of Registry Key Variable
'------------------------------------------------
' Open RegKey Under KeyRoot {HKEY_LOCAL_MACHINE...}
'------------------------------------------------
        rc = RegOpenKeyEx(KeyRoot, KeyName, 0, KEY_ALL_ACCESS,
hKey) ' Open Registry Key

        If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError
' Handle Error...

        tmpVal = String$(1024, 0)
' Allocate Variable Space
        KeyValSize = 1024
' Mark Variable Size
```

```
'-------------------------------------------------
' Retrieve Registry Key Value...
'-------------------------------------------------
        rc = RegQueryValueEx(hKey, SubKeyRef, 0, KeyValType,
tmpVal, KeyValSize)     ' Get/Create Key Value

        If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError
' Handle Errors

        tmpVal = VBA.Left(tmpVal, InStr(tmpVal, VBA.Chr(0)) -
1)
'-------------------------------------------------
' Determine Key Value Type For Conversion...
'-------------------------------------------------
        Select Case KeyValType
' Search Data Types...
        Case REG_SZ
' String Registry Key Data Type
                KeyVal = tmpVal
' Copy String Value
        Case REG_DWORD
' Double Word Registry Key Data Type
                For i = Len(tmpVal) To 1 Step -1
' Convert Each Bit
                        KeyVal = KeyVal + Hex(Asc(Mid(tmpVal,
i, 1)))    ' Build Value Char. By Char.
                Next
                KeyVal = Format$("&h" + KeyVal)
' Convert Double Word To String
        End Select

        GetKeyValue = True
' Return Success
        rc = RegCloseKey(hKey)
' Close Registry Key
        Exit Function
' Exit

GetKeyError:    ' Cleanup After An Error Has Occured...
        KeyVal = ""
' Set Return Val To Empty String
```

```
        GetKeyValue = False
' Return Failure
        rc = RegCloseKey(hKey)
' Close Registry Key
End Function
```

- **Login function**

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias
"GetUserNameA" (ByVal lpbuffer As String, nSize As Long) As
Long

Public OK As Boolean
Private Sub Form_Load()
    Dim sBuffer As String
    Dim lSize As Long
    LoadResStrings Me
    sBuffer = Space$(255)
    lSize = Len(sBuffer)
    Call GetUserName(sBuffer, lSize)
    If lSize > 0 Then
        txtUserName.Text = Left$(sBuffer, lSize)
    Else
        txtUserName.Text = vbNullString
    End If
End Sub
```

```
Private Sub cmdCancel_Click()
    OK = False
    Me.Hide
End Sub
```

```
Private Sub cmdOK_Click()
    'ToDo: create test for correct password
    'check for correct password

    If txtPassword.Text = "" Then
        OK = True
```

```
        Me.Hide
    Else
        MsgBox "Invalid Password, try again!", , "Login"
        txtPassword.SetFocus
        txtPassword.SelStart = 0
        txtPassword.SelLength = Len(txtPassword.Text)
    End If
End Sub
```