A THESIS ON

SYNTHESIS OF SPEECH FROM BANGLA TEXT

BY

MD. RABIUL HASAN

Roll No. 930531P

Reg. No. 93742

Session 1992-'93-'94

SUPERVISOR

MD. ABDUS SATTAR

Assistant Professor

Department of C.S.E., BUET.

A thesis submitted to the Department of Computer Science and Engineering, BUET in partial fulfilment of the requirements for the degree of Master of Science in Computer Science and Engineering .

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DHAKA, BANGLADESH.
OCTOBER, 1999.

'SYNTHESIS OF SPEECH FROM BANGLA TEXT'

A thesis submitted by

MD. RABIUL HASAN, Roll no. 930531P, Session 1992-'93-'94

for partial fulfilment of the requirements for the degree of

Master of Science in

Computer Science and Engineering held on 17-10-1999.

Approved as to the style and contents by :

(1) Md. Abdus Sattar — Chairman
Assistant Professor, CSE Department, BUET. — (Supervisor)

(2) Dr. Mohammad Kaykobad — Member
Head and Professor, CSE Department, BUET. — (Ex-Officio)

(3) Dr. Md. Abul Kashem Mia — Member
Assistant Professor, CSE Department, BUET.

Saidur

(4) Dr. Md. Saidur Rahman — Member
Assistant Professor, CSE Department, BUET.

(5) Dr. M. Rezwan Khan — Member
Professor, EEE Department, BUET. — (External)

Declaration

This is declared that the thesis work on 'SYNTHESIS OF SPEECH FROM BANGLA TEXT' has been done by me under the guidance of MD. ABDUS SATTAR and has not been submitted elsewhere for the award of any degree or diploma except for publication.

Countersigned
Signature of the Supervisor

(MD. ABDUS SATTAR)

Signature of the Candidate

MHasan

(MD. RABIUL HASAN)

Acknowledgement

I wish to express my deepest graditude to my guide MD. ABDUS SATTAR, Assistant Professor, Department of Computer Science and Engineering, BUET, DHAKA for his meditative, affectionate and careful guidance which rights for a great contribution incarrying out my job. His keen interest and vast experience in this field have influenced me to complete this work. Though the job was so difficult, I was able to successfully complete the job due to his assistance, encouragement and valuable advice at every stage.

Sincere graditude are also propounded in favour of my respected teacher DR. MOHAMMAD KAYKOBAD, Professor and Head, Department of Computer Science and Engineering, BUET for his inspiration, administrative support and animated co-operation.

Finally, I extend my heartiest thanks to all of my teachers, friends and other fellows specially to DAISY PERVIN, Asst. Engineer, PDB who helped me in various stages in carrying out this thesis work.

DHAKA
OCTOBER, 1999.

Author

Abstract

Bangla text to speech synthesizer can generate speech sounds from arbitrary text input. Bangla text to speech synthesizer is developed using the method that stores speech sounds of parts of words. The parts of words i.e. basic sounds units are recorded using sound card, microphone and sound recorder utility software. The pronunciation rules and the determination of basic sound units of Bangla words are studied extensively as part of the current research which is the field of Bangla phoneme research. At first Bangla text to speech synthesizer accepts the character from keyboard and identifies the sound unit and plays that. This task is repeated until the word is completed. Then the text is normalized by defining the wave files from the adjacent keys as per pronunciation rule and searched. When identification of the sound units for a particular Bangla word is completed, all the sound units are merged together into a speech file to form the complete speech signal for that Bangla word and finally the merged wave file is played. Note that Bangla text to speech synthesizer is interfaced with the application/word processing software so that at first the message (generated by pressing key) is retrieved from application's message queue to Bangla text to speech synthesizer.

Contents

CHAPTER 1. INTRODUCTION PAGE NO.

1.1 Introduction to speech technology.................................... 01

1.2 Literature review and present state of the problem.............. 06

1.3 Objectives with specific aims and possible outcomes........... 08

1.4 Applications of Bangla text to speech synthesizer............... 09

1.5 Organization of the thesis paper....................................... 11

CHAPTER 2. METHODOLOGY OF TEXT TO SPEECH SYNTHESIZER

2.1 Introduction... 12

2.2 Phoneme method.. 12

2.3 Segment storage strategies.. 13

2.4 Method used for Bangla text to speech synthesizer.............. 14

CHAPTER 3. SPEECH DATA PREPARATION

3.1 Digital audio file.. 16

3.2 Musical Instrument Digital Interface (MIDI)...................... 18

3.2.1 MIDI Network.. 18

3.2.2 MIDI music data.. 18

3.2.3 MIDI messages.. 19

3.3 File used for development of Bangla text to speech 20
synthesizer.

3.4 Sound file format... 21

3.5 Recording the wave file... 23

3.6 File packing and compression..................................... 24

3.7 Sound Blaster... 26

3.7.1 The chips of sound blaster.................................. 26

CHAPTER 4. TEXT TO SPEECH SYNTHESIZER PAGE NO.

4.1 Text to speech synthesizer.. 28

4.2 Developed Bangla text to speech synthesizer.................... 32

4.3 Wave file searching technique....................................... 33

4.4 Playing the wave file 36

4.5 Developed text normalizer.. 37

4.6 Wave file merging technique... 42

CHAPTER 5. INTERFACING WITH APPLICATION SOFTWARE

5.1 Introduction.. 46

5.2 Message loop... 47

5.3 Message queue... 51

5.4 Interfacing with application software............................ 52

5.5 Keyboard translator... 57

CHAPTER 6. CONCLUSION

6.1 Results of developed Bangla text to speech.. 59
synthesizer

6.2 Suggestions for further research in this field.................... 60

6.3 Conclusion... 62

References... 64

Appendices

Appendix A : Bangla alphabets....................................... 66

Appendix B : Bangla sound units and their filenames.............. 67

Appendix C : Bangla symbols and their codes....................... 71

Appendix D : Bangla Bijoy keyboard layout......................... 76

Appendix E : Program of Bangla text to speech...................... 78
synthesizer.

List of figures

Figure	Description	Page no.
3.1	Analog to digital conversion.	17
3.2	Digital to analog conversion.	17
3.3	MIDI message structure.	19
3.4	Silence block packing.	25
4.1	Generalised block diagram of text to speech synthesizer.	29
4.2	Generalised block diagram of developed Bangla text to speech synthesizer.	32
4.3	Generalised block diagram of play procedure of the wave file.	32
4.4	Memory mapping of the wave file.	33
4.5	Program flowchart of play procedure of the wave file.	35
4.6	Program flowchart of developed text normaliser. (Wave file defining and searching technique)	39
4.7	Program flowchart of developed text normaliser. (Last wave file defining and searching technique)	41
4.8	Block diagram of the wave file merging technique.	42
4.9	Program flowchart of wave file merging technique.	43
5.1	wParam and lParam of WM_CHAR message.	48
5.2	WM_CHAR lParam coding.	49
5.3	Program flowchart of main module of Bangla text to speech synthesizer.	50
5.4	Message queues.	51
5.5	Interface of Bangla text to speech synthesizer with application/word processing software.	52
5.6	Program flowchart of hook procedure.	56
5.7	Block diagram of the keyboard translator.	57
5.8	Program flowchart of keyboard translator.	58

Chapter 1

Introduction

This chapter describes the basic information of speech technology. Section 1.1 discusses the introductory information of speech technology. The developments in speech technology are also included in this section. Section 1.2 discusses the literature review and present state of the problem. Section 1.3 introduces the objectives with specific aims and possible outcomes. Section 1.4 describes the applications of Bangla text to speech synthesizer. Section 1.5 includes the organization of the thesis paper.

1.1 Introduction to speech technology

Speech is the most natural form of human communication. Speech consists of a continuously changing complex sound wave linking the speaker's mouth and the hearer's ear. Sound is due to the vibratory motion of an object. The vibrations may be rapid or slow. These vibrations are recognized as audible sound if they lie within certain limits 20 cycles/sec to 20,000 cycles/sec known as the limits of audibility. The physical properties [1] of speech are as follows.

- Intensity,
- Pitch i.e. intonation,
- Frequency spectrum i.e. quality, and
- Duration.

Intensity is a measure of loudness of sound. Intensity of sound wave at any point is defined as the amount of energy flowing per second through a unit area about that point at right angles to the direction of flow. It depends on amplitude, distance of the sounding body, density of the medium, area of the vibrating source. Actually amplitude determines the loudness of sound. Pitch means fundamental frequency of variation of the vocal cord. Pitch is a measure of acuteness or shrillness of a sound. For example, a woman's voice is more shrill than that of a man. So her voice is said to be higher pitch than that of a man. Pitch is independent of loudness and quality. Wave length determines pitch. Quality distinguishes between two notes of same pitch and intensity but coming from two different sources. Duration is the length in time taken by sound wave.

Speech is also the natural way for human beings to communicate with a computer. For two-way communication, the computer must speak like a person. Part of the requirement to reach this goal is to give the computer the ability to generate natural and fluent speech in response to any input text.

Speech technology as well as speech processing is one of the most exciting areas of signal processing. The developments in speech technology [2] include

- Speech analysis and synthesis,
- Speech coding,
- Speech enhancement,
- Speech recognition,
- Spoken language understanding,
- Speaker identification and verification, and
- Multimodal communication.

Speech analysis and synthesis

The areas of speech synthesis [3] include

- Generation of speech from text,
- Voice conversion, and
- Modification of speech attributes such as time scaling and articulatory minic.

Text to speech synthesis takes text as input and generates human like speech as output.

Voice conversion refers to the technique of changing one person's voice to another, from person A to person B or from male to female and vice versa.

In many applications, it is useful to be able to change the time scale of a signal (to speed up or slow down the speech signal without changing the pitch) or to change the "mood" of the speech (making it sound happy or sad). This signal processing technique has appeared in animation and computer graphics applications.

Speech coding

Homer Dudley's pioneering work [2] was motivated by the need to increase the communication capacity (number of channels) in a telephone network (which was analog then). The term "bandwidth compression" was generally used to refer to such a task. Today, most if not all of the telephone network is digital and hence, speech bandwidth compression translates into speech coding, which aims at representing the speech signal in binary digits (bits) with highest efficiency (i.e. highest quality of the reconstructed signal with least number of bits).

Digital encoding of speech begins with an analog to digital conversion device that samples the analog speech wave form at an appropriate rate (usually 8,000 samples per second for telephone bandwidth speech) and then represents the amplitude of each sample digitally. In communication systems, this is called pulse code modulation (PCM). Typically, each wave form sample is represented by 12-16 bits, resulting in a rate of 96-128 thousand bits per second (kb/s). Research in speech coding attempts to find methods to increase the efficiency in transmission and storage while maintaining the speech quality. Aside from efficient transmission, speech coding is also essential for achieving secure communications.

In general, speech coder attributes [2] can be described in terms of the following four classes.

- Bit rate,
- Complexity,
- Delay, and
- Quality.

The bit rate is the communication channel bandwidth (kb/s) at which the coder operates. Complexity refers to the computational complexity of the speech coder. Delay refers to the communications delay caused by the coder. Quality refers to a large number of attributes. As bit rates are lowered, speech coders become more speech specific and give less faithful renditions of other sounds.

Today, speech coding finds a diverse range of applications such as cellular telephony, voice mail, multimedia messaging, digital answering machines, packet telephony, audio-visual teleconferencing and of course many other applications in the internet arena.

Speech enhancement

The idea that vocoder principles could be used to improve the quality of a speech signal corrupted by additive noise [2]. The basic idea was to generate a signal with a fine structure as close as possible to that of the original speech signal, but with an envelope that attenuates the signal between formant peaks. Although the idea was shown to be feasible, the quality attained was not very good.

Since those early days, variants of this idea have been proposed and implemented by several authors. The common features of all these implementations are to split the noisy speech signal into frequency regions by passing it through a filter bank and attenuating the output of each channel by a factor depending on the estimated signal to noise ratio in that channel. The main differences between these various proposals are the methods used to estimate the level of noise and of speech in various frequency bands.

Enhancement of speech signals in noise has been quite useful in telephony applications.

Speech recognition

Speech recognition technology has made it possible for computers to follow human voice commands and even understand human languages and converts a speech wave form into words.

Spoken language understanding

Spoken language understanding as undertaken at present involves integrating speech recognition (what are the words?) and natural language understanding (what do those words mean?).

Speaker verification and identification [2]

Speaker recognition is the process of automatically recognizing a speaker by using speaker-specific information included in his or her speech. This technique can be used to verify the identity claimed by people accessing systems; that is, it enables control of access to various services by voice. Applicable services include voice dialling, banking over a telephone network, telephone shopping, database access services, information and reservation services, voice mail, security control for confidential information and remote access to computers.

Speaker recognition can be classified into speaker identification and speaker verification. Closed -set speaker identification is the process of determining which of the registered speakers a given utterance comes from. Speaker verification is the process of accepting or rejecting the identify claim of a speaker. Most of the applications in which voice is used to confirm the identity claim of a speaker require speaker verification.

From spoken language to multimodal communication [2]

Human machine communication (HMC) is evolving from text interface (i.e., keyboard and screen display) to spoken language (automatic speech recognition and understanding) to multimodal communication involving different senses (audio, visual, tactile or even gestural) with synergy. Human communication includes the perception or production of a message or of an action as an explicit or implicit cognitive process. For perception, there are the "five senses": hearing, vision, touch, taste and smell, with reading as a specific visual operation, and speech perception as a specific hearing operation. For production, it includes sound (speech or general sound production) and vision (generation of drawing, graphics or more typically, written messages). Cognition includes the means to understand or to generate a message or an action from a knowledge source.

Research interest in speech processing today has gone well beyond the simple notion of mimicking the human vocal apparatus (which still intrigues many researchers). The scope (both breadth and depth) of speech research today has become much larger due to advances in mathematical tools (algorithms), computers, and the almost limitless potential applications of speech processing in modern communication systems and networking. Conversely, speech research has been viewed as an important driving force behind many of the advances in computing and software engineering, including digital signal processors (DSPs). Such a synergetic relationship will continue for years to come.

1.2 Literature review and present state of the problem

Research on speech technology began in 1865. One of the first speech researchers was Alexander Graham Bell, who experimented with various analogues of speech waves as early as 1865. His work led to the invention of the telephone, for which he was awarded the Volta Prize in 1880 and the Bell Telephone Company has hosted one of the most important speech research laboratories up to the present day. In 1940, at the Bell Laboratories, an apparatus was designed which allowed the spectrum of a speech wave to be analysed against time - the spectrograph. To understand the acoustic form of speech it is essential to study the complex changes that take place across the utterance, so this breakthrough meant that at last a piece of speech could be analytically described [1].

A few years later, people began to develop apparatus which would play back speech from a simple spectrographic picture, as was done at the Haskins Laboratory in 1947. This could be regarded as the real beginning of speech synthesis technology, since the human voice was being analysed into a set of data and reconstructed at a later time. By 1960 many academic institutions around the world shared techniques of synthesizing human speech using minicomputer. In 1971 a new digital technique 'Linear prediction' was developed for analysing and synthesizing speech using computer [1].

A text to speech synthesizer was first developed for the English language in 1960. Following English, text to speech synthesizers of Telegu and Hindi languages have also been developed in our sub-continent [4].

The special series for the 50th anniversary for the Signal Processing Society [2] covers the history and current status of the field of speech processing research and describes future contributions of speech processing.

A neural network based approach to synthesize F0 information for Mandarin text to speech is discussed in [5]. The basic idea is to use neural networks to model the relationship between linguistic features, extracted from input text and parameters representing the pitch contour of syllables.

The application of Bangla in computer is not new now. But there is no satisfactory development in speech technology of it. Bangla text to speech converter has been developed by Islam [6]. But this software is applicable only for a particular Bangla word processor 'Amar Bangla' which was developed in the Department of Electrical and Electronic Engineering, BUET and is not WINDOWS based which is popular now.

A group of researchers of the Applied Physics Department of the University of Rajshahi are trying to find the formant frequencies of Bangla phonemes [7].

The confirmation of the correctness of typed Bangla character during word processing can be done by developing Bangla text to speech synthesizer. But Bangla text to speech synthesizer must be interfaced with any other application/word processing software. In this case, user can correct the character or word immediately. But previously, by checking the proof copy the character or word was to be corrected.

It is cumbersome to use the computer for blind person. If the keyboard layout, computer operation and any application/word processing software are in control and Bangla text to speech synthesizer is included to the applications/word processing software with Bangla font then it is easy to process Bangla word for blind person.

So Bangla text-to-speech synthesizer should be developed and interfaced with any WINDOWS based application/word processing software without knowing the internal software structure of the well known word processors like Microsoft Word or other word processors.

1.3 Objectives with specific aims and possible outcomes

The main objective of this research is to develop a system so that user can understand easily about the correctness of the typed Bangla character during word processing. For two-way communication, the computer must speak like a person. Part of the requirement to reach this goal is to give the computer the ability to generate natural and fluent speech in response to any input text. By developing the text to speech synthesizer, the computer informs Bangla character what is typed by generating the sound through sound card and speaker and then plays after synthesis of speech from Bangla text which will be helpful for blind person during word processing. Also the pronunciation rules and the determination of basic sound units of Bangla word are studied extensively as a part of the current research which is the field of Bangla phoneme research.

In this research, conversion of arbitrary text input to voice output actually involves three separate tasks.

(i) At first, accepting the character from keyboard and is to detect the character and identify the sound unit and play this file. This task is repeated until the word is completed.

(ii) The second task is to normalize the text by defining the wave file from the adjacent keys as per pronunciation rule and search that.

(iii) Finally, merge the sound units together to form the complete speech signal for that Bangla word and play this file to a listening device (a speaker) to produce the audible speech sound.

1.4 Applications of Bangla text to speech synthesizer

The Bangla text to speech synthesizer can be used in a variety of applications. They are discussed below.

(a) As an assisting tool for a blind person in typing

A blind person does not know which key means which digit or letter in a keyboard. So blind person might make mistake in typing. By hearing the sound, a blind person can understand which key has been pressed.

(b) As a language tool for the dumb

The Bangla text to speech synthesizer can be used as a language tool for the dumb who can write, but can not speak. If the desired texts are given as a input from the keyboard of a PC, the text to speech synthesizer will convert these texts into audiable speech sounds. Thus, a dumb can communicate with others using Bangla text to speech synthesizer.

(c) Generation of speech from arbitrary text input

For development of Bangla text to speech synthesizer, the sound units are recorded. Later on, Bangla text to speech synthesizer will generate the sounds from arbitrary text input. This may have manifold applications, some of which are mentioned below.

(i) Voice generation of a person

Once the basic sound units of a particular person are recorded, the Bangla text to speech synthesizer simply requires to input the corresponding text from the keyboard of a PC, to produce any speech sound in that person's voice.

(ii) As a talking computer in the classroom

In educational institutions, a teacher becomes so busy that he can not be present in the classroom. He may input the texts of his lecture during off time and Bangla text to speech synthesizer will take the class on his absence. Thus, it will be a talking computer in the class room .

(iii) To produce actor's/actress's voice in an unfinished Bangla film

If the sound units of a actor/actress are once recorded, his/her voice may be generated by using the Bangla text to speech synthesizer even his/her death. Also, the voice of a person of young age can be generated at a later stage. This facility may be used along with the video processing systems to finish the dialogues of an unfinished Bangla film due to sudden death or illness of an actor/actress.

(iv) For advertisement and announcement purposes in public places

Bangla text to speech synthesizer can be used for advertisement and announcement purposes in public places like at a bus stop, airport, railway station, hospital, fair etc. It can also act as a receptionist at a party or function.

(v) In telecommunication and electronic mail systems

Bangla text to speech synthesizer can be used in telecommunication and electronic mail systems to produce audio messages from the corresponding text inputs.

(vi) For generating speech sounds to develop various video games and education programs on a PC

Bangla text to speech synthesizer can be used to develop useful education programs like teaching Bangla alphabets and numerical digits and also to generate Bangla sounds for the video games on a PC.

1.5 Organization of the thesis paper

The thesis paper has been organized in the following way.

This chapter (Chapter 1) describes the introductory information about speech technology, literature review and present state of the problem, objectives with specific aims and outcomes, applications of Bangla text to speech synthesizer.

Chapter 2 describes the methodology of text to speech synthesizer.

Chapter 3 describes about the preparation of speech data. It is discussed about the digital audio file and their format, MIDI files, recording and compression techniques. It is also discussed about the sound blaster at the end of this chapter.

Chapter 4 introduces a generalized block diagram of text to speech synthesizer and discusses about the wave file searching, playing, text normalizing and merging technique.

Chapter 5 discusses about the concept of message driven programming, the interface of Bangla text to speech synthesizer with application/word processing software and keyboard translator.

Chapter 6 describes the results of Bangla text to speech synthesizer and comments on the topics of further research in this field.

Chapter 2

Methodology of text to speech synthesizer

This chapter describes the methodology of text to speech synthesizer. Section 2.1 introduces the types of the methodology of text to speech synthesizer. Section 2.2 discusses about the phoneme method. Section 2.3 discusses about the segment storage strategy method. Section 2.4 describes the method used for developing Bangla text to speech synthesizer.

2.1 Introduction

Language conveys meaning by stringing together discrete symbolic units at several concurrent levels. Sequences of sound form words and when words are assembled together they form sentences. The combination of these units is governed at each level by a set of principles. The science of linguistic provides rigorous methods for identifying these principles and represents them in a formal manner. The methodology of text to speech synthesizer [1] is as follows.

- Phoneme method, and
- Segment storage strategies.

2.2 Phoneme method

Phoneme analysis of a particular language is important for synthetic voice generation. Phoneme may be viewed as a sequence of segmental units at a linguistic level [1,6]. These phonemes are abstract linguistic units and may not be directly observed in the speech signal. Phonetics is concerned with sound of human speech. Phoneme analysis means to derive the phonetic structure of an utterance directly from the speech signal.

Conversion of arbitrary text input to voice output actually involves three separate tasks. The first consists of accepting a sequence of characters, identifying the phonetic components of the required message and extracting information about its syntactic structure. The second part of the process is to match the phonetics symbols stored in the phonetics inventory. The computer program is generally based on the rules for converting phonetic information to acoustic information. For this reason, the synthesis process is often referred to as 'speech synthesis by rule'. So thirdly, the phonetic symbols are linked together and send the resulting coded wave form to a voice output device. Also the prosody (variations in intonation, pitch, rhythm or timing and intensity or loudness) [6] of the speech must be determined directly from the textual representation. Pitch means fundamental frequency of variation of the vocal cord. Intonation means the rise and fall of the pitch of the voice in speaking, which is an element of meaning in language.

So text to speech synthesizer using phoneme method requires two major tasks. Translation of the text into phonetic symbols and determination of the prosody from the text are very complex task. So phoneme analysis of a particular language is very difficult and difficult software development is required. But by using the phonetic symbols, natural sound can be produced.

2.3 Segment storage strategies

The multi-level linguistic message supports a number of possible strategies [1] for their storage and retrieval regardless of which voice output technology is employed in a particular system. They are as follows.

- Sentence storage,
- Word and phrase storage, and
- Storing parts of words i.e. syllables.

Sentence storage

In the sentence storage system, the voice of a live speaker is recorded with the desired inflection and personality and then processed to derive a matrix of parameters from which the original utterance may be reconstructed. Sentence storage offers the highest level of naturalness currently possible for synthetic speech. But this system is not flexible because it requires more memory space to store repeated words or phrases.

Word and phrase storage

In word or phrase storage system, it stores the similar words or phrases just for one time which reduces the number of the words to be stored to a greater extent. Besides these advantages the word and phrase storage system has contextual variation problem, variation in intonation and duration which provide important clues about syntactic structure. Also slapping together words and phrases from different sentences recorded at different times can also result in choppy 'sing song' effect since the pitch and duration of words will not necessarily match.

Storing parts of words

This method also requires to store a small number of sound units compared to sentence storage or word and phrases storage system. Also contextual variation problem can be solved by storing parts of words, but in this system if correct variant of a prefix or suffix has been selected there remains the problem of matching its intonation to that of the words to which it is attached.

2.4 Method used for developing Bangla text to speech synthesizer

Bangla text to speech synthesizer can be developed by using either phoneme method or segment storage strategies mehod. Bangla text to speech synthesizer using phoneme method requires two major tasks. Translation of the text into phonetic symbols and determination of the prosody from the text are very complex task. So we have discarded this method. Bangla text to speech synthesizer can also be developed by using any of the segment storage strategies method like sentence storage, word and phrase storage and storing parts of words (i.e. syllables). Sentence storage is not flexible because it requires more memory space to store repeated words or phrases. The word and phrase storage system has contextual variation problem, variation in intonation and duration. This variation results 'sing song' effect. So we have not used the sentence storage and word or phrase storage methods. Third segment storage strategy 'storing parts of words' is more appropriate [1] for development Bangla text to speech synthesizer. Because

(a) This method requires to store a small number of sound units compared to sentence storage or word and phrase storage system.

(b) Contextual variation problem can be solved by using storing parts of words method.

(c) There are fixed pronunciation rules for a specific arrangement of Bangla letters in a word. On the contrary, there are no such definite rules for the English language. The English letters has a different pronunciation rule according to the position of the letter in the word, and therefore, is pronounced differently according to the use of words. A particular arrangement of Bangla letters obeys the same pronunciation rule despite its position in the word. This helps the method that stores parts of words.

However in storing parts of words method, there remains the problem of matching its intonation to that of the words to which it is attached [1].

Finally, we again say that we have used the storing parts of word method for development of Bangla text to speech synthesizer. The parts of words i.e. basic sound units have to be recorded. In this method, the sound units will be detected from the text input and will be merged together as per Bangla pronunciation rule to form a complete word.

Chapter 3

Speech data preparation

In this chapter, a description is given on the preparation of speech data. Section 3.1 discusses about the digital audio file. Section 3.2 discusses about the musical instrument digital interface (MIDI). Section 3.3 describes the file used for development of Bangla text to speech synthesizer. Section 3.4 includes the sound file format. Section 3.5 introduces the recording of the wave file. Section 3.6 discusses about the file packing and compression technique. Section 3.7 describes the chips of sound blaster.

3.1 Digital audio file [8]

Digital audio files are files that contain sound converted to digital form by analog to digital converter (ADC) with pulse code modulation (PCM) technique so it can be stored in computer's memory or disk. Once sound has been converted to digital form it can be easily modified with an editor. Figure 3.1 below depicts how analog to digital conversion is performed.

The quality of digital audio sound depends on two key factors.

- Sampling rate (number of samples taken per second), and
- Sampling size.

The higher sampling rate performs the better the sound quality. The sampling rate must be at least twice the highest frequency component. But there are some problems to use the highest sampling rates. First, highest sample rate requires a lot of storage capacity. Each sample consumes 1 byte of memory or disk space. At a sample rate of 6,000 Hz, one minute of recording will fill a 360k disk. At the peak sound blaster pro sampling rate of 44,100 Hz (for monaural sound) or 22,050 Hz (for stereo), an empty 10 MB hard disk will be filled in just four minutes!. Second, it is not used too high a sampling rate if the files are planed to pack (compress).

The other factor in quality of digital audio is the sampling size. The sound card may be 8 bit (a maximum of 256 steps) or 16 bit (256X256 or 65,535 steps) which varies the signal strength. The human ear perceives the difference between these two sampling size. The ears are most sensitive at detecting pitch (frequency) but also quite sensitive to sound intensity. Human ears are capable of detecting sounds that vary in intensity by orders of magnitude and 8 bits sound is perceived as lethargic and noisy in comparison to 16 bits sound.

When the digitized audio file is played back, the digital data is converted from a stream of byte values to an analog electrical wave by hardware called a DAC, a digital to analog converter represented at figure 3.2. The analog signal, sound consists of a pressure wave is then fed to the speakers or headphone which moves through a medium, such as air. These analog to digital and digital to analog conversion are done by digital sound processor (DSP) chip of sound blaster which is described at the end of this chapter.

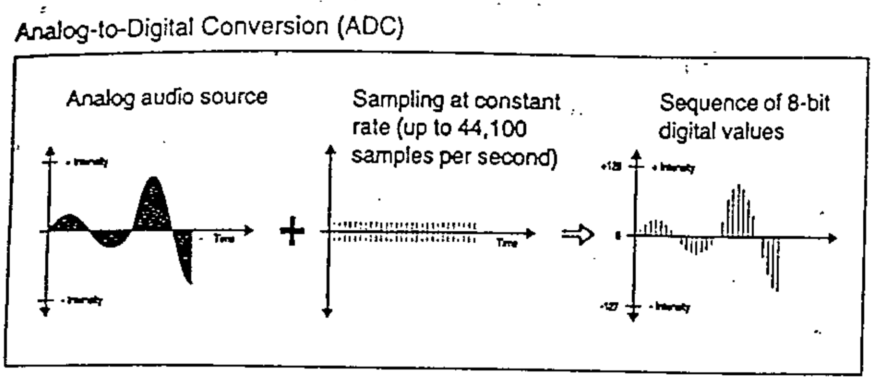

Analog-to-Digital Conversion (ADC)
Analog audio source
+ Intensity
- Intensity
Time
Sampling at constant rate (up to 44,100 samples per second)
Time
Sequence of 8-bit digital values
+128
+ Intensity
0
-127
- Intensity

Figure 3.1 Analog to digital conversion.

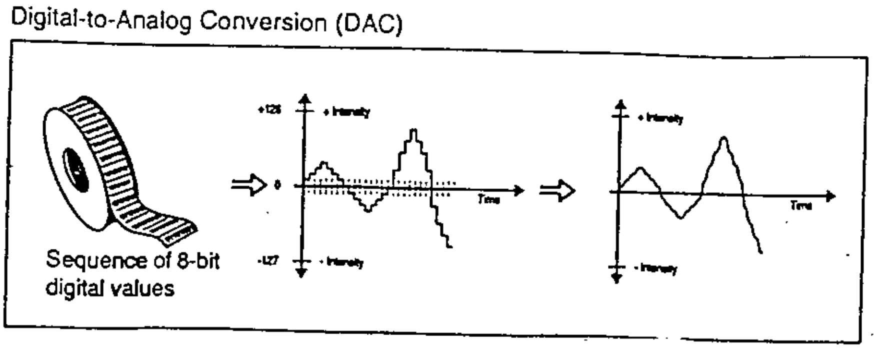

Digital-to-Analog Conversion (DAC)
Sequence of 8-bit digital values
+128
+ Intensity
0
Time
-127
- Intensity
+ Intensity
Time
- Intensity

Figure 3.2 Digital to analog conversion.

3.2 Musical Instrument Digital Interface (MIDI)[9]

The Musical Instrument Digital Interface (MIDI) has completely reshaped the electronic music world, by delivering sophisticated music recording and performing capability to amateur musicians. MIDI actually makes no sound at all. MIDI is just a protocol that enables computers, synthesizers, keyboards and other musical devices to communicate with each other. At the hardware level, MIDI provides a simple asynchronous serial interface that transmits data in 10 bit chunks at a rate of 31.25 kilo baud (31,250 bits per second). The ten bits include a start bit, eight data bits and a stop bit.

3.2.1 MIDI Network [8]

A MIDI network is a music network that connects computer and musical instruments that have a MIDI interface by inserting the sound blaster card and an external MIDI device such as a MIDI keyboard. Almost every sound card of PC includes some kind of built in synthesizer.

MIDI consists of a digital protocol for representing musical notes and actions and a network protocol for transmitting music data between MIDI compatible devices. The digital protocol allows MIDI to describe music in a digital fashion that is, as a sequence of byte value with musical meaning. These byte values can be recorded on disk by a digital computer. The network protocol refers to the network software protocol used to pass music data between MIDI devices including personal computer.

3.2.2 MIDI music data [8]

Rather than musical sounds, MIDI consists of instructions on how to play music when a sequencer program stores music as a MIDI file on disk, it records MIDI instructions that specify what instrument to play, what key to press, with how much strength and when to press it.

3.2.3 MIDI messages [9]

MIDI devices communicate by sending each other messages. The most commonly messages used are Note Number (what note to play), Note On (when to play a note, Velocity (how hard to hit the note), Note off (when to release a note) and Channel Number (what instrument should play the note). Message is divided into the following two general categories.

- Channel message, and
- System message.

Channel message

The first category, channel messages includes voice messages and mode messages. They are transmitted on individual channels rather than globally to all devices in the MIDI network.

As shown in figure 3.3 a MIDI message includes a status byte and up to two data bytes. It is easy to identify a status byte because all status bytes have their most significant bit set to 1. Conversely the most significant bit of any data byte is set to 0. This convention holds for all standard MIDI messages and not just channel messages.

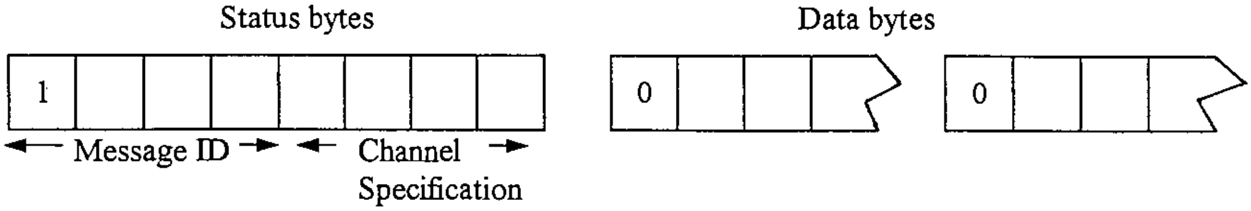


Figure 3.3 MIDI message structure.

MIDI devices transmit all messages on the same cable, regardless of their channel assignments. The four low order bits of each status byte identify which channel it belongs to. Four bits produce 16 possible combinations, so MIDI supports 16 channels over a single cable string. The three remaining bits identify the message. Three bits encode eight possible combinations, so channel message could come in eight flavours, but they don't. A status byte with all four high order bits set to 1 indicates a system common message - the second general category. So there are only seven channel messages.

System messages

The second general category of MIDI messages is the system messages, which include system common messages, system real time messages and system exclusive messages. These messages carry information that is not channel specific, such as timing signals for synchronization, positioning information in pre-recorded MIDI sequences and detailed set-up information for the destination device.

3.3 File used for development of Bangla text to speech synthesizer

Advantages of MIDI file over digital audio file [8]

(a) MIDI is the most economical type of multimedia sound. Digital audio file contain actual sound, recorded in digital form by taking thousands of samples each sound. MIDI music, on the other hand, consists only instructions on how to play an instrument. Digital audio files may require millions of bytes of data to play just a few minutes of music but it is played hours of music with a MIDI file of just a few thousand bytes of data.

(b) Another advantage is that it allows to change the MIDI music data. The MIDI file is the computer equivalent of sheet music that is read by a musician. A MIDI sequencer program displays the music composition on the screen, showing the notes for each instrument on its own track.

Disadvantages of MIDI file over digital audio file [9,10]

(a) The major disadvantage of MIDI is that the quality of sound it produces depends entirely on the synthesizer on which it is played, whether that is a sound card or an external synthesizer .

(b) Requires MIDI hardware to record the wave file.

But we have no MIDI hardware. For these reasons, we have used the digital audio file for development of text to speech synthesizer.

There are many digital audio files [8] such as

- Voice file by a file name extension of .VOC,
- Microsoft wave file by a file name extension of .WAV,
- Creative Music file by a file name extension of .CMF,
- Sound Blaster Instrument file by a file name extension of .SBI, and
- Sound Blaster Instrument Bank file by a file name extension of .IBK.

Windows supports the wave file and it is possible to play the wave file easily with a few lines of code by VISUAL C++ multimedia software. So we have used the wave file from many digital audio files for development of Bangla text to speech synthesizer.

3.4 Sound file format

Sound file format is

Header	Data

The header is introduced at the very beginning of the file that identifies the file's type. For digital audio files, this header characterizes the file as wave or voice; designates whether the raw data should be played as stereo or monaural and so forth. Data is raw data of music.

Microsoft wave file format has been discussed here. The wave form audio file is organized in RIFF (Resource Interchange File Format) structure [8].

RIFF file format

The basic building block of a RIFF file is called a chunk, which is formatted as follows.

<rID> <rLen> <rData>

◊ <rID> 'RIFF' identifies the representation of the chunk data (4 bytes).

◊ <rLen> is the length of data in the chunk. (4 bytes)

◊ <rData> is the RIFF data chunk.

WAVE form definition

The wave form of a RIFF data chunk is further divided into chunks. It must always contain a format chunk followed by a data chunk.

<rData>=<wID><Format Chunk><Data Chunk>

where <wID> 'WAVE' identifies the data as wave form audio (4 bytes).

WAVE format chunk

The format chunk contains data which specifies the format of the data contained in the data chunk. The syntax of the format chunk is as follows.

<Format Chunk> =<fID> <flen><wFormatTag><nChannels><nSamplesPerSec>
<nAvgBytesperSec><nBlockAlign><nBitsPerSample>

◊ <fID> 'fmt' identifies the block as a format chunk (4 bytes)

◊ <fLen> length of Data in the format chunk (4 bytes)

◊ <wFormatTag> indicates the wave format category of the file (2 bytes)
For example, 1 = Pulse Code Modulation (PCM) format.

◊ <nChannels> indicates the number of channels for output (2 bytes)
For example, 1 = mono, 2 = stereo

◊ <nSamplePerSec> indicates sampling rate (in samples per second) at which each channel should be played back (4 bytes)

◊ <nAvgBytesPerSec>indicates the average number of bytes per second that the data should be transferred (4 bytes).

<nAvgBytesPerSec>=nChannels * nSamplesPerSec*(nBitsPerSample/8)

◊ <nBlockAlign> indicates the block alignment (in bytes) of the data in the Data Chunk. Play back software needs to process a multiple of <nBlockAlign> bytes of data at a time, so that the value of <nBlockAlign> can be used for buffer alignment (2 bytes).

<nBlockAlign>=nChannels*(nBitsPerSample/8)

◊ nBitsPerSample (2 bytes)

WAVE Data Chunk

The data chunk contains the actual .WAV audio data. The format of the data depends on the <wFormatTag> value stored in the Format Chunk.

<Data Chunk> = <dId> <dLen> <dData>

where

◊ <dId> 'DATA' identifies the block as data chunk (4 bytes)

◊ <dLen> indicates the length of data in the data chunk (4 bytes)

◊ <dData> is the actual wave form data.

3.5 Recording the wave file

Most popular sound files are recorded as 8 bit or 16 bit pulse code modulated (PCM) data.

The total number of byte in the file depends on the recording frequency and the total length of the recording. The recording frequency, also called the sample rate, is the number of times sound is sampled within one second. In 8 bits PCM format, each sample yields a single byte. Higher frequencies or sample rates, produces higher quality recordings. The higher quality recording of digital audio files requires more disk space.
The question arising for recording has to do with the segmentation of the wave file. Let us consider the Bangla word 'আমাদের'. The basic sound units of 'আমাদের' are অ, া, ম, ে, দ and র. So the basic sound units অ, া, ম, ে, দ and র are recorded for playing Bangla word 'আমাদের' individually. Now for playing the merged wave file of Bangla word 'আমাদের', the basic sound units are আ, মা, দে and র্ । So total basic sound units অ, আ or া, ম, ে, দ, র, মা, দে and র্ are recorded for Bangla word 'আমাদের'. Remember that র্ is not recorded

directly. At first অর্ is recorded then to get র্ , অ is removed from অর্ by using utility software.

Note that these sound units are also used for another any Bangla word when necessary. A list showing the names of the basic sound units and the corresponding file names is included in Appendix B. A particular combination of these sound units will form the pronunciation of a Bangla word.

To avoid background electrical noise during recording, it is best to shut off any computers, air conditioning, fluorescent light, fan or any other machinery in the vicinity of the sound booth.

We have used the sound recorder utility software to record the wave file for development the Bangla text to speech synthesizer. For this, sound card and microphone must be needed.

3.6 File packing and compression

The higher quality recording of digital audio files requires more disk space. The higher quality recording means highest sample rates. Each sample consumes 1 byte of memory or disk space. At a sample rate of 6,000 Hz, one minute of recording will fill a 360k disk. At the peak sound blaster pro sample rate of 44,100 Hz (for stereo), an empty 10 MB hard disk will be filled in just four minutes!. So two techniques [8] for packing and compression are available for requiring the reduced disk space.

- Silence block packing, and
- Data block packing.

Silence block packing

Silence Block packing is a technique for replacing stretches of silence or near silence with a special marker that represents a period of silence. Digital audio files containing speech is much more compact by eliminating the silent data. Figure 3.4 shows the audio data that can be replaced by silence block.

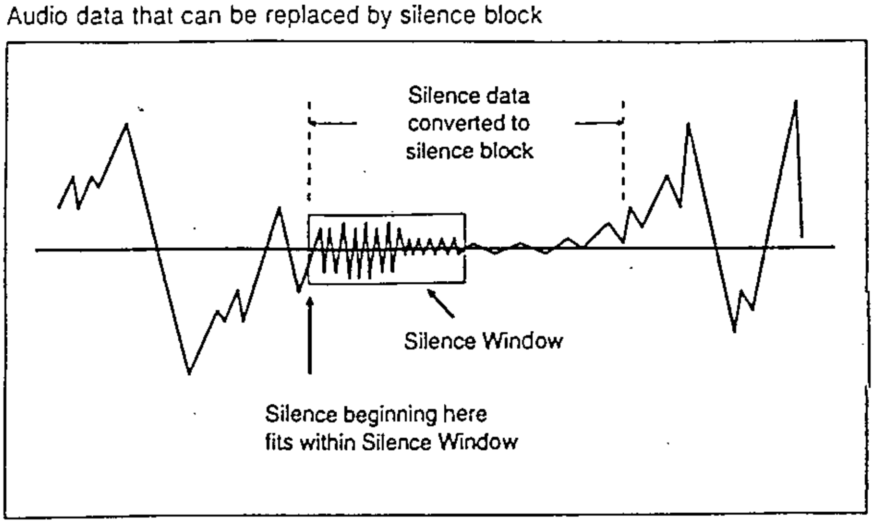

Audio data that can be replaced by silence block
Silence data converted to silence block
Silence Window
Silence beginning here fits within Silence Window

Figure 3.4 Silence block packing.

During recording of wave file, a silent block is added to the recorded file for time required of file saving in hard disk. So when two or more wave files are merged for speech synthesis then silence block is to be inserted into merged wave file. This silence block must be removed and we have removed this silence block by CREATIVE WAVE STUDIO utility. The recorded wave file is also edited by this CREATIVE WAVE STUDIO software such as amplify, copy, cut and paste etc.

Data block packing

Data block packing is a two step process. First the digital audio file is compressed without the loss of information. Second the file is decompressed when played and original file is restored. Many different compression algorithms are used, after the sound files have been recorded. The sound blaster cards are capable of playing the digital audio files in which the data blocks are compressed. That means decompression is done by the sound blaster cards' hardware during playback, by means of the digital signal processor (DSP) chip. The beginning of the file (the header) must not be compressed, since it contains information that identifies the file. Only the data blocks within the file are compressed.

The size of the wave file can be reduced by the new audio compression technique. Human ear cannot detect all audio frequency. The principle of audio compression is to cut off the frequency which is out of audibility limits of human ear.

3.7 Sound blaster

To develop a Bangla text to speech synthesizer, a sound card is needed to record the basic sound units and also to play back these.

3.7.1 The chips of sound blaster

The main chips of sound blaster include the digital sound processor (DSP), FM synthesizer and Mixer etc [8].

Digital Sound Processor (DSP) chip

The most versatile chip on the sound blaster is the DSP (Digital Sound Processor) chip; it processes all the commands that come from an application. The DSP chip must also instruct all the other sound chips on sound blaster in order to produce sounds.

When a presentation program wants to play notes through the FM synthesizer, the DSP must accept the data from the computer and instruct the FM chip how to play the music and converts it from digital to analog form to near the sound.

The DSP is responsible for sending and receiving the MIDI data used by electronic keyboards and synthesizers. The DSP also performs the analog to digital and digital to analog functions that allow to do digital recording and play back of music, sound effects and speech.

Some digital sound files are stored in a compressed format to save disk space and must be decompressed. The DSP can play these sound files by decompressing the data as it arrives from computer.

FM synthesizer chip

The FM synthesizer is a modern invention for producing a wide range of sounds, both music and special effects. The FM chip is responsible for synthesizing the sounds of musical instruments. This chip can play up to 11 instruments simultaneously. A wealth musical sounds could be created by the mixing of low frequency (audible to the human ear) speech and music with a pure, very high frequency electrical wave using the FM (frequency modulation) techniques.

The mixer

Another new chip introduced on the pro cards is the Mixer. This chip allows to adjust and mix the sounds from the microphone, line in, CD input & the digital sound output and to control volumes. This way it can be heard a blend of sounds such as music playing in the background while speech and sound effects blast away in the foreground on the sound blaster 1.x/2.0, it can be only heard one thing at a time.

Chapter 4

Text to speech synthesizer

This chapter discusses about the text to speech synthesizer. Section 4.1 introduces a generalized block diagram of text to speech synthesizer. Section 4.2 introduces the block diagram of developed Bangla text to speech synthesizer. Section 4.3 describes the wave file searching technique. Section 4.4 discusses how the wave file is played. Section 4.5 introduces the developed text normalizer. Section 4.6 discusses the wave file merging technique.

4.1 Text to speech synthesizer

Text to speech synthesizer is a device which takes text as a input and generate the synthetic speech by controlling the parameters and a model. If the model and parameters are sufficiently accurate then the production of intelligible synthetic speech would be possible. The basic goal in text to speech synthesis is to convert unrestricted text input into natural sounding speech.

Key problems in this area include

- To study the pronunciation rule and determine the basic sound units of Bangla words.
- To produce the natural sound of Bangla words.
- To develop the mathematical model of sound units of Bangla words.
- Introducing the text normalizer including syntax parser and abbreviation interpreter.
- Handling an imperative, interrogative, exclamatory sentences and joint letter.
- Conversion of text of a sentences or a paragraph into speech continuously.

A generalised block diagram of text to speech synthesizer [1] is shown in figure 4.1.

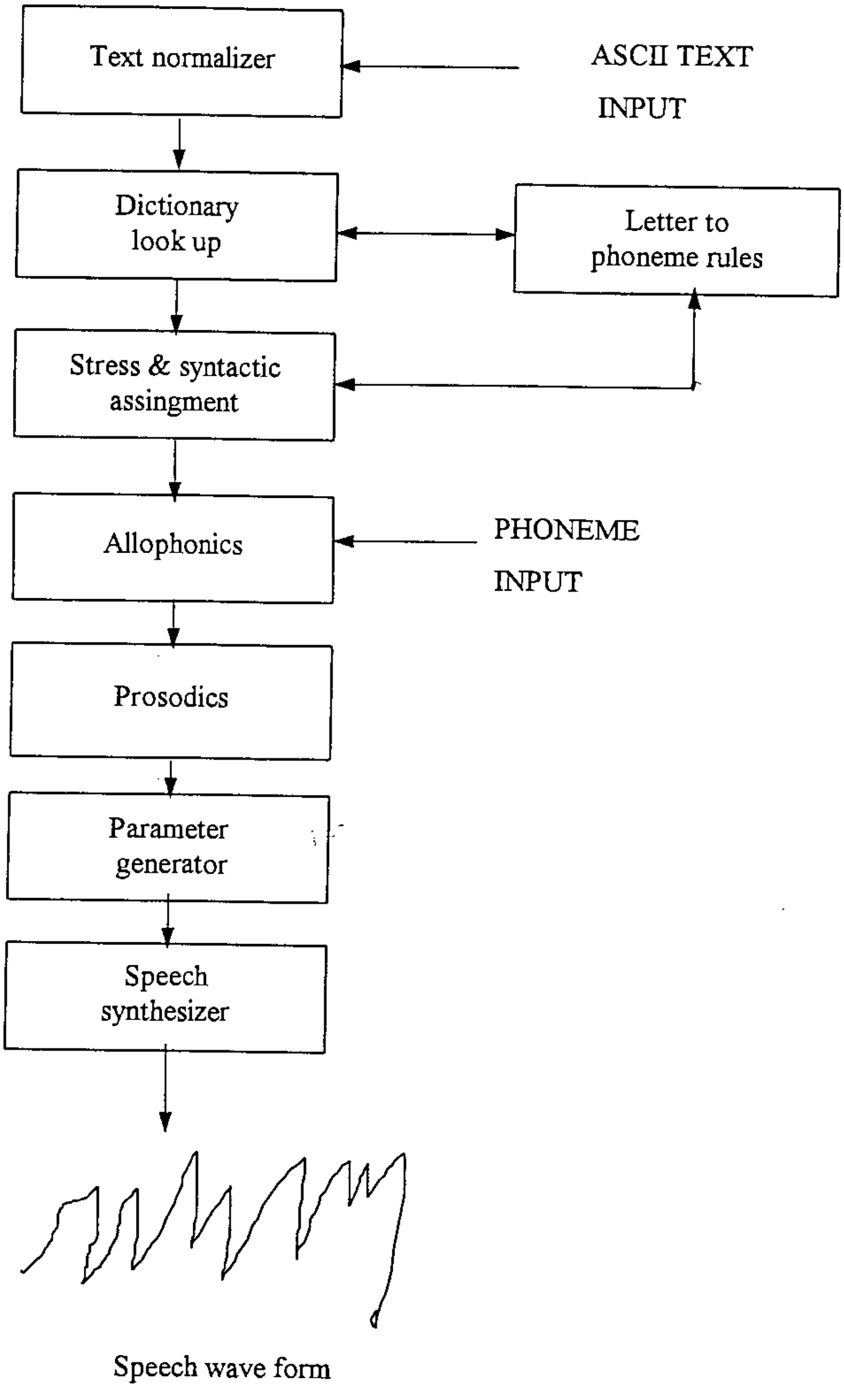


Figure 4.1 Generalised block diagram of text to speech synthesizer.

Text normalizer

By text normalizer, the Bangla word 'মা' should be pronounced as 'maa' and not as 'ma', 'aa' and also a string such as ১,২৩৪ should be spoken as এক হাজার দুই শত চৌত্রিশ and not as এক কমা দুই তিন চার if it is to be clearly understood by the general listener. Text normalizers which must make decisions must also be controllable by the user so that incorrect decisions can be rectified. Text normalisation can also handle syntax parsing, abbreviations and forms like ২২তম and ১২ঃ০০.

(i) Syntax parsing

Further obstacles to the development of a rule set for interpreting spelling are provided by pairs of words which are spelled the same but pronounced differently. As for example, Bangla text to speech synthesizer produces the correct speech sound of Bangla word 'বার' when it represents a day, However, when the same word 'বার' is used to represent a number (which represents twelve in Bangla), the synthesizer will still pronounce the word as to mean 'day' of week. The only real solution to this problem would be to devise an algorithm capable of determining the syntactic structure of a given sentence and assigning each word to its proper grammatical category.

(ii) Abbreviation interpreting

Another function 'abbreviation interpreting' should be included in Bangla text to speech synthesizer. We can use the abbreviation such as মিঃ, ডঃ etc in Bangla sentences. But the exception rules must be accomplished in Bangla text to speech synthesizer which convert the abbreviation into spoken form such as মিঃ to মিষ্টার.

Letter-to-sound conversion

Accurate conversion of normalised text to equivalent sounds requires both a set of rules and a list of exceptions to those rules. This rule set must be augmented by 'dictionary' which associates exceptional text strings with appropriate phoneme strings. A sample strategy for identifying word boundaries would involve replacing spaces, commas, full stops (periods) etc with phonetic symbols which are then part of the input to subsequent rule of sets. Since the look-up process is usually much faster than application of the rules, no speed penalty is paid for extensive exceptions lists. The size of the memory available can be important, however.

Stress and syntactic marking

Very natural speech can only be produced if the converter knows what it is saying user enters phonemes strings with stress marks to permit specification of correct pronunciation and stress when the automatic results are not acceptable.

Allophonics and prosodics

The actual sound (or phone) produced by operation of a rule is called an allophone of the phoneme. Allophonics variation is determined by syllabic stress as well as by the identity of neighbouring phonemes. That means, each sound produced by a human vocal tract is influenced by the other sounds surrounding it, so a text-to-speech product must operate similarly to speak naturally. This naturalness is enhanced by incorporation of proper pitch and amplitude variations at the clause and sentence level. These cues can help the listener identify questions, for instance.

Final speech production

Once the sound string to be produced is fully specified, it must be converted into an electrical signal. If algorithmic smoothing is used, this process includes generation of the data that drives a filter representing the vocal tract.

4.2 Developed Bangla text to speech synthesizer

For development of Bangla text to speech synthesizer, at first the wave file is to be recorded by any utility software. Then key is pressed and keyboard translator translates the key according to Bangla code. After this, play procedure of the wave file is executed. In play procedure of the wave file, at first, according to key pressing, the wave file is searched and played individually. Then the wave file is defined from the adjacent keys by text normalizer and searched. After this, searched wave files are merged and merged wave file is played. These procedures are shown in figure 4.2 and 4.3.

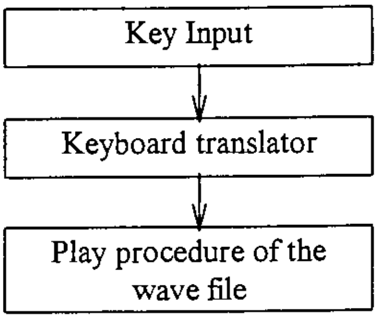


Figure 4.2 Generalized block diagram of developed Bangla text to speech synthesizer.

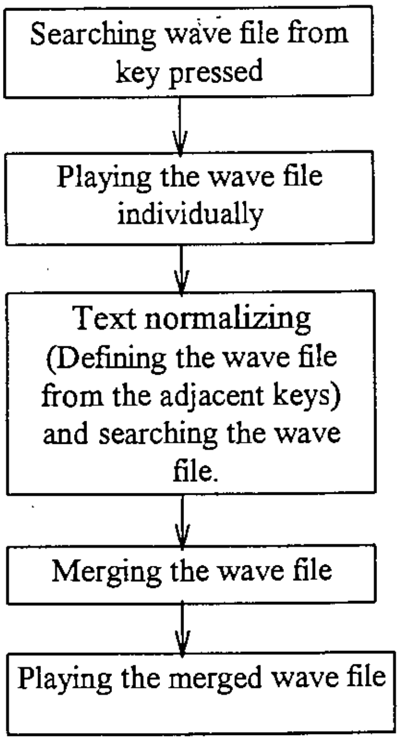

Searching wave file from key pressed
Playing the wave file individually
Text normalizing (Defining the wave file from the adjacent keys) and searching the wave file.
Merging the wave file
Playing the merged wave file

Figure 4.3 Generalized block diagram of play procedure of the wave file.

4.3 Wave file searching technique

Like in English alphabet, there are three types of letter in Bangla alphabet.

- Vowel,
- Vowel auxiliary, and
- Consonant.

There are 11 vowels, 9 vowel auxiliaries and 39 consonants in Bangla alphabet. A list of Bangla vowels, vowel auxiliaries, consonants and other symbols used to form a Bangla word are given in Appendix A.

The memory mapping of the wave file is shown in figure 4.4.

SET 0	কো	খো	....	য়ো
	0			38
SET 1	কা	খা	....	য়া
	39			77
SET 2	কু	খু	....	য়ু
	78			116
SET 3	কী	খী	....	য়ী
	117			155
SET 4	কে	খে	....	য়ে
	156			194
SET 5	কি	খি	....	য়ি
	195			233
SET 6	ক	খ	....	য়
	234			272
SET 7	অ	ই	..... ঔ	া
	273		282	283
SET 8	ক্	খ্	.....	য়্
	284			322

Figure 4.4 Memory mapping of the wave file.

Set 0 to set 8 describe all possible sound units to form a complete word and is mapped into memory in this sequence.

If there is no necessary of a particular sound unit to form any Bangla word, then the memory space of that particular sound unit's location is reserved for keeping the other sound units at right position.

Wave files are searched by direct searching technique. So the wave files (from set 0 to set 6) are searched by the following equation.

Position = Bangla code of the key pressed – 75
+ (Difference between 1st and last consonant code + 1) X set ... (4.1)

Note that Bangla code of 'ক' is 75 for sulekha, proshun font which is shown in Appendix C.

For set 7 except ী, the equation will be
Position = Bangla code of the key pressed + 208 ... (4.2)

When the last wave is from ক to য় then set 8 is applicable. w is the number of sound units then the equation will be
Position = 50 + Position of sound unit at (w-1)th ... (4.3)

These equations may be changed if we change the memory mapping of the wave file.

The wave file searching technique has been shown in the program flowchart of play procedure of the wave file (figure 4.5).

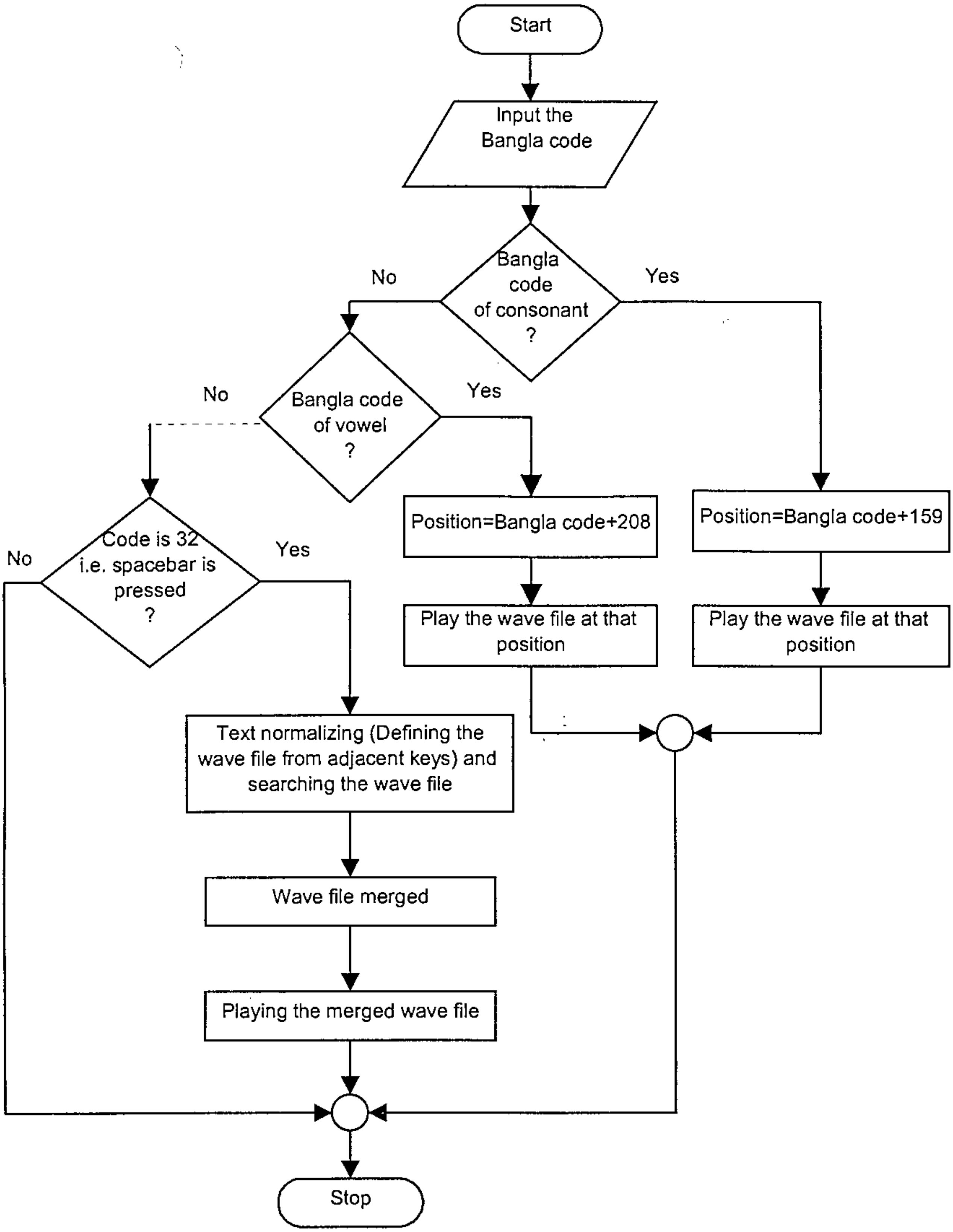

Start
Input the Bangla code
Bangla code of consonant ?
No
Yes
Bangla code of vowel ?
No
Yes
Code is 32 i.e. spacebar is pressed ?
No
Yes
Position=Bangla code+208
Position=Bangla code+159
Play the wave file at that position
Play the wave file at that position
Text normalizing (Defining the wave file from adjacent keys) and searching the wave file
Wave file merged
Playing the merged wave file
Stop

Figure 4.5 Program flowchart of play procedure of the wave file

4.4 Playing the wave file

There are three easy ways to play the wave files.

(a) Message Beep: Plays only sounds configured in the registry for warnings and errors.

(b) sndPlaySound: Plays sounds directly from .WAV files or from memory buffers.

The syntax of sndPlaySound function [9,10] is as follows.

```
BOOL sndPlaySonnd (LPCTSTR lpszSoundName,  // file name
                        UINT uFlages ) ;         // SND - option Flags
```

The first parameter contains a full path name pointing to a .WAV file. sndPlaySound() first searches the [Sounds] section of WIN.INI for a matching string. If it does not find one, it looks on the disk for a matching file, If it still does not find a match, it plays the "system default" sound.

sndPlaySound() requires enough memory to load the full sound into memory. It works best with sound files no larger than about 100 KB.

The second parameter expects a flag controlling how the sound is played. Some possible values are as follows.

SND_SYNC: The sound is played synchronously and the function does not return until the sound ends.

SND_ASYNC: The sound is played asynchronously and the function returns immediately after beginning the sound. To terminate an asynchronously played sound, call sndPlaySound() with the SoundName parameter set to NULL.

SND_NODEFAULT: If the sound can not be found, the function returns silently without playing the default sound.

SND_MEMORY: The parameter specified by SoundName points to an in memory image of a wave form sound.

SND_LOOP: The sound continues to play repeatedly until sndPlaySound() is called again with the lpszSoundName parameter set to NULL. SND_ASYNC flag must be also specified to loop sounds.

SND_ NOSTOP: If a sound is currently playing, the function immediately returns FALSE without playing the requested sound.

(c) PlaySound: New in Win 32. It does not play sounds from memory.

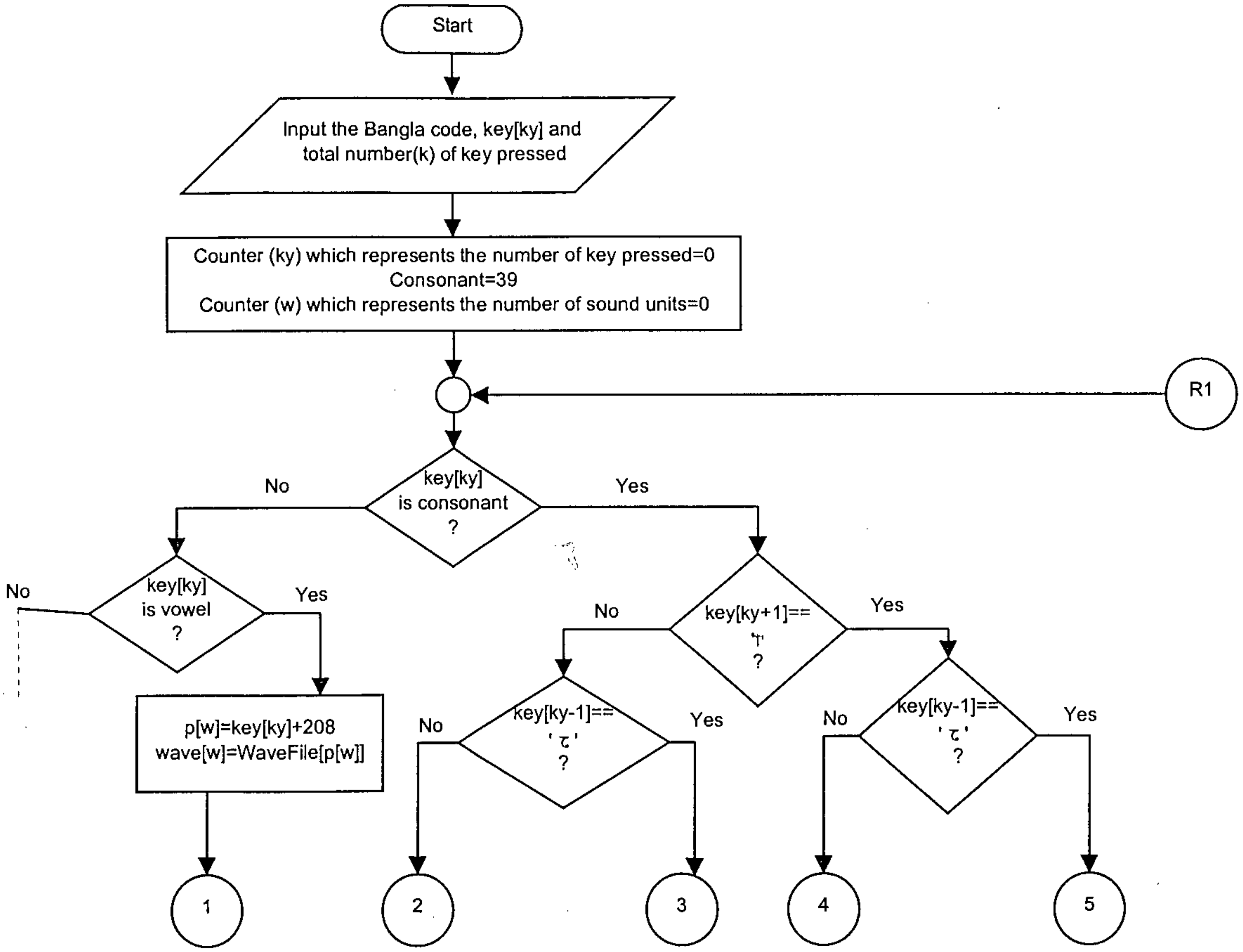
4.5 Developed text normalizer

Bangla has a very rich vocabulary. There is a fixed pronunciation rule for a specific arrangement of Bangla letters in a word. The pronunciation rule is introduced by using text normalizer which defines the wave file from the adjacent keys in our software. The pronunciation of each letter of Bangla alphabet (vowels and consonants) as one reads them produces unique sound.

In Bangla language, generally vowels may occur as a first letter and sometimes used singly as a Bangla word such as এ (pronounced as 'A') and ঐ (pronounced as 'Oi'). There are also vowel auxiliaries in Bangla language and have different symbols [া (Aa-kar), ে (A-kar), ু (U-kar), ি(E-kar) etc.].

A consonant may take its position at the beginning, at the middle or at the end of a Bangla word. Two or more consonants may form a Bangla word (examples বক, কলম etc.). However, in most of the cases, vowels, vowel auxiliaries and consonants combine together to form a Bangla word. A consonant has a vowel auxiliary before and/or after it. The pronunciation of the combination of a consonant and vowel auxiliary depends on the particular consonant and the particular vowel auxiliary being used. The Bangla word 'মা' (pronounced as 'Maa') is recorded and this wave file is defined and searched when ম (Ma) is followed by an া (A-kar). So by text normaliser the Bangla word 'মা' should be pronounced as 'maa' and not as 'ma' ,'aa'. Similarly 'মে ' (pronounced as English word 'Mae') is defined and searched when vowel auxiliary ে (A-kar) is placed before ম (Ma). This wave file defining from the adjacent keys by text normaliser and searching technique using ে, া and ো have been shown in the program flow chart (figure 4.6). In this discussion, we can say that 11 vowels and 39 consonants are not sufficient to pronounce all the Bangla words.

There is another instance of varied pronunciation, when a vowel or a consonant appears at the middle or end of a word without any vowel auxiliary. Its pronunciation depends on the ending sound of the utterance of the letter immediately preceding it. Let us take the Bangla word কাক ('Kaak') as an example. Here the first sound unit is কা ('Kaa') and the second sound unit, if pronounced in isolation is ক ('Kau'). If these two sound units are now combined together, the word will be pronounced as 'Kaakau'. However, as we know, the actual pronunciation of this word is 'Kaak', that is, the utterance কা ('Kaa') + ক্ ('aak') actually produce the sound. This pronunciation rule of Bangla language is introduced by text normalizer shown in the program flowchart (figure 4.7). The consonant (ক to য়) are mapped from 234 to 272 position in the memory. If the consonant (ক to য়) appears at the end of the word then TextNormaliser2 function is executed. The position of the last wave file is also determined by the equation 4.3 shown in wave file searching technique.


Start
Input the Bangla code, key[ky] and total number(k) of key pressed
Counter (ky) which represents the number of key pressed=0
Consonant=39
Counter (w) which represents the number of sound units=0
R1
key[ky] is consonant ?
No
Yes
key[ky] is vowel ?
No
Yes
p[w]=key[ky]+208
wave[w]=WaveFile[p[w]]
key[ky+1]== '্' ?
No
Yes
key[ky-1]== ' τ ' ?
No
Yes
key[ky-1]== ' τ ' ?
No
Yes
1
2
3
4
5

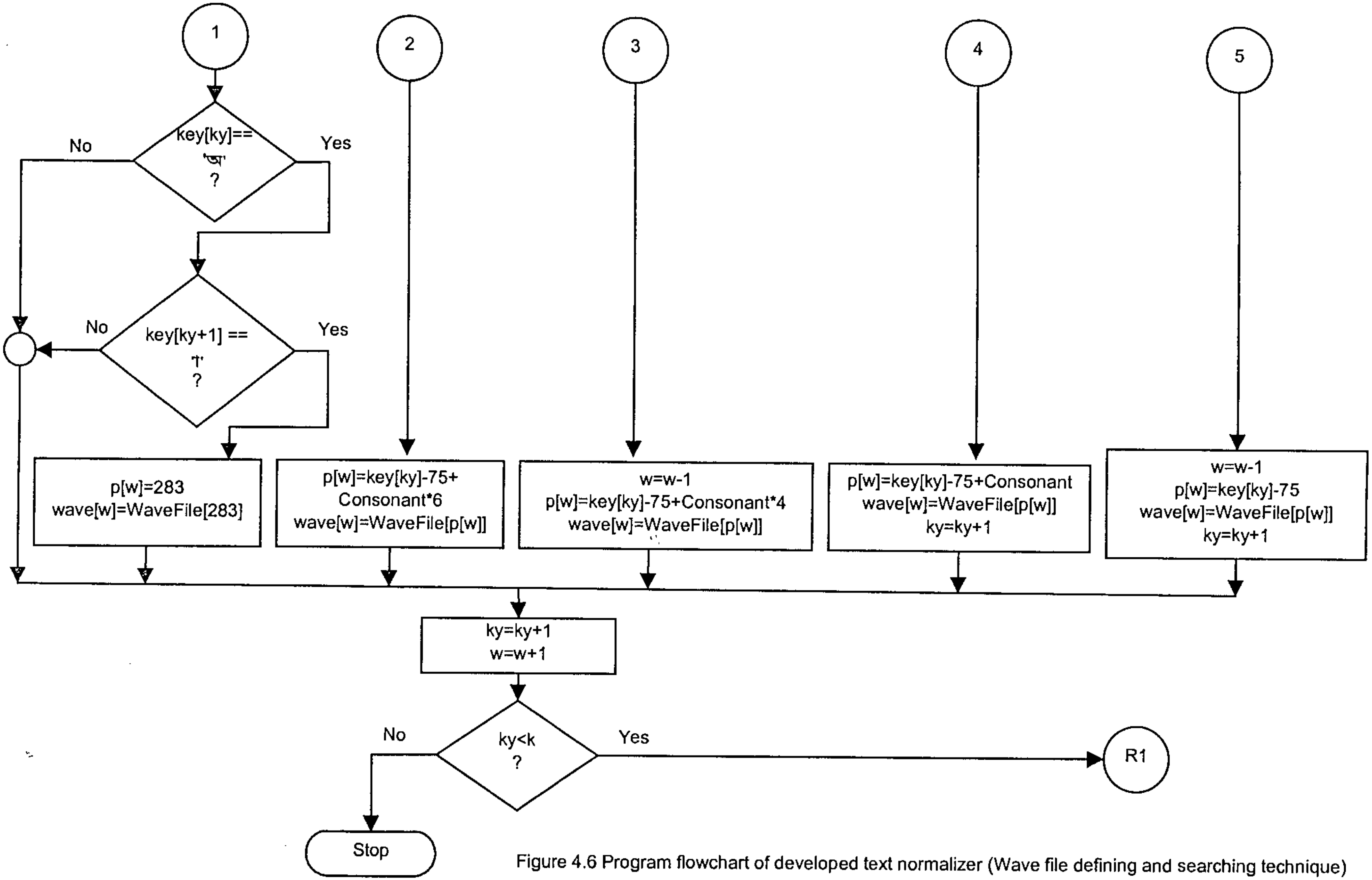


Figure 4.6 Program flowchart of developed text normalizer (Wave file defining and searching technique)

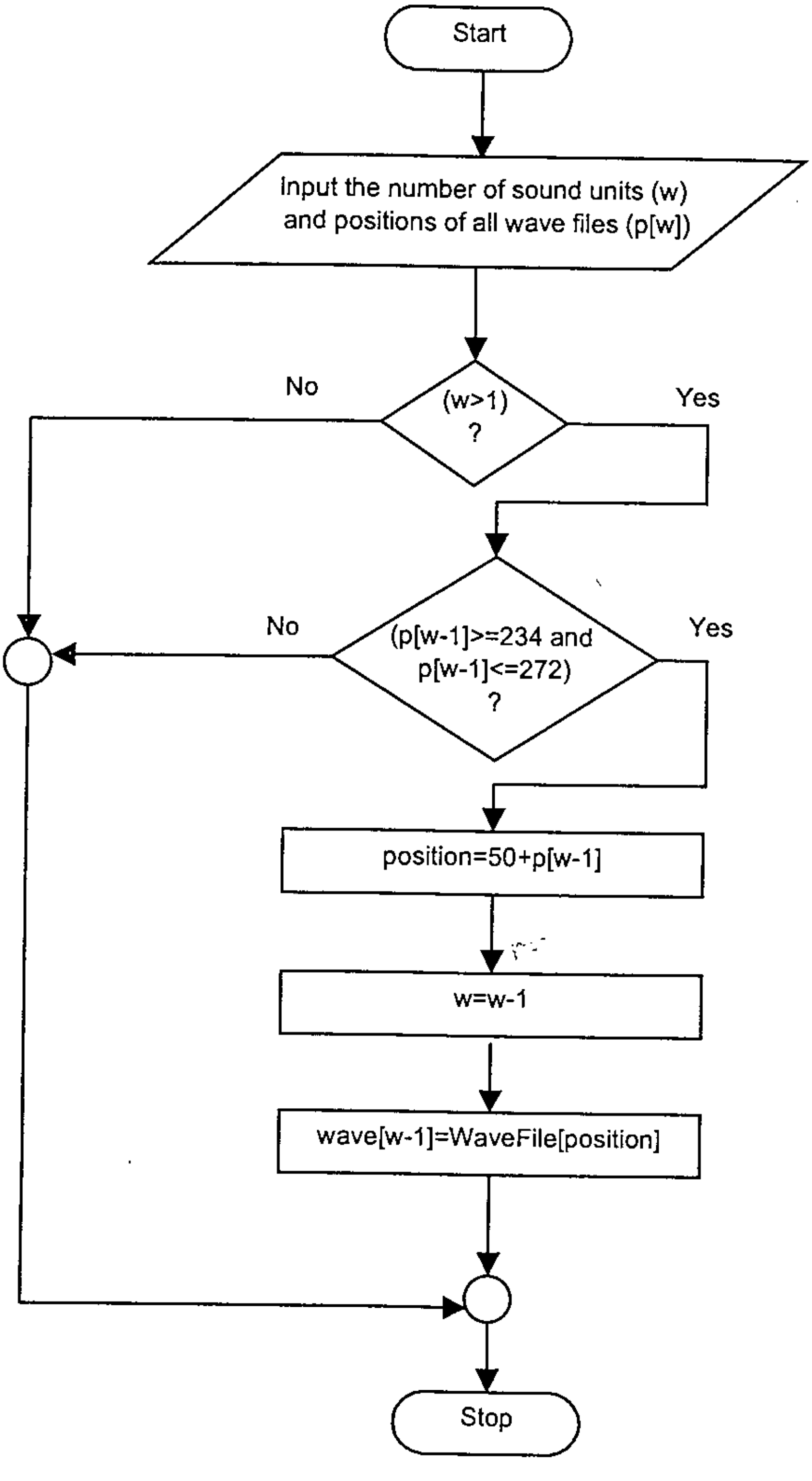

Start
Input the number of sound units (w)
and positions of all wave files (p[w])
No
(w>1)
?
Yes
No
(p[w-1]>=234 and
p[w-1]<=272)
?
Yes
position=50+p[w-1]
w=w-1
wave[w-1]=WaveFile[position]
Stop

Figure 4.7 Program flowchart of developed text normalizer
(Last wave file defining and searching technique)

4.6 Wave file merging technique

A few Bangla words need single sound unit. However, most of the Bangla words require multiple sound units to form the corresponding speech sounds.

After assigning the sound units for each speech segment of a word, it is necessary to merge the sound units together to reconstruct the speech sound corresponding to the word. Depending on the sound units present in a word, the file merging technique differs. An individual file corresponding to a sound unit has its own header, which contains useful information about the type and length of the sound file. This information is vital for producing correct speech sound when the sound file is played back. When two such sound files are merged together to form a new sound file, the headers of both the files are omitted and new header is inserted with the new chunk size and data size followed by the speech information of the two sound units. When three sound units are merged together, another new header is generated omitting the headers of each sound unit and merging all speech information sequentially. The block diagram and program flow chart of wave file merging technique are shown in the figure 4.8 and 4.9 respectively.

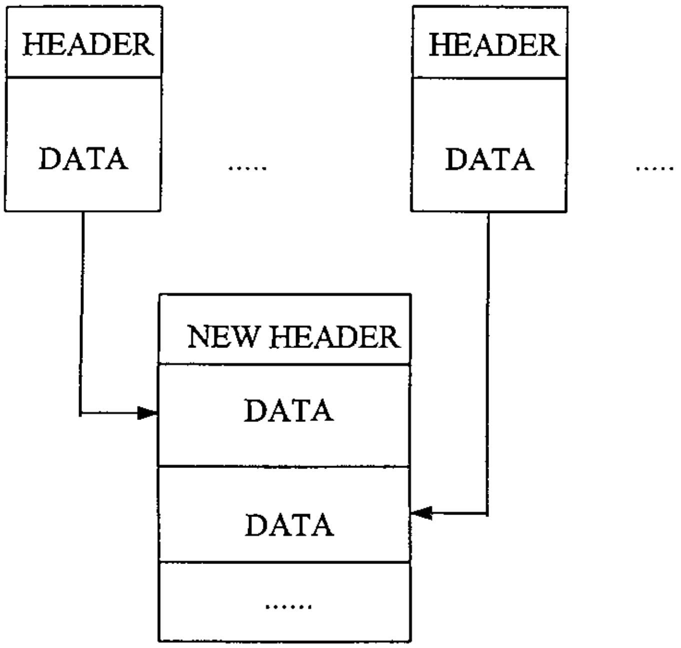
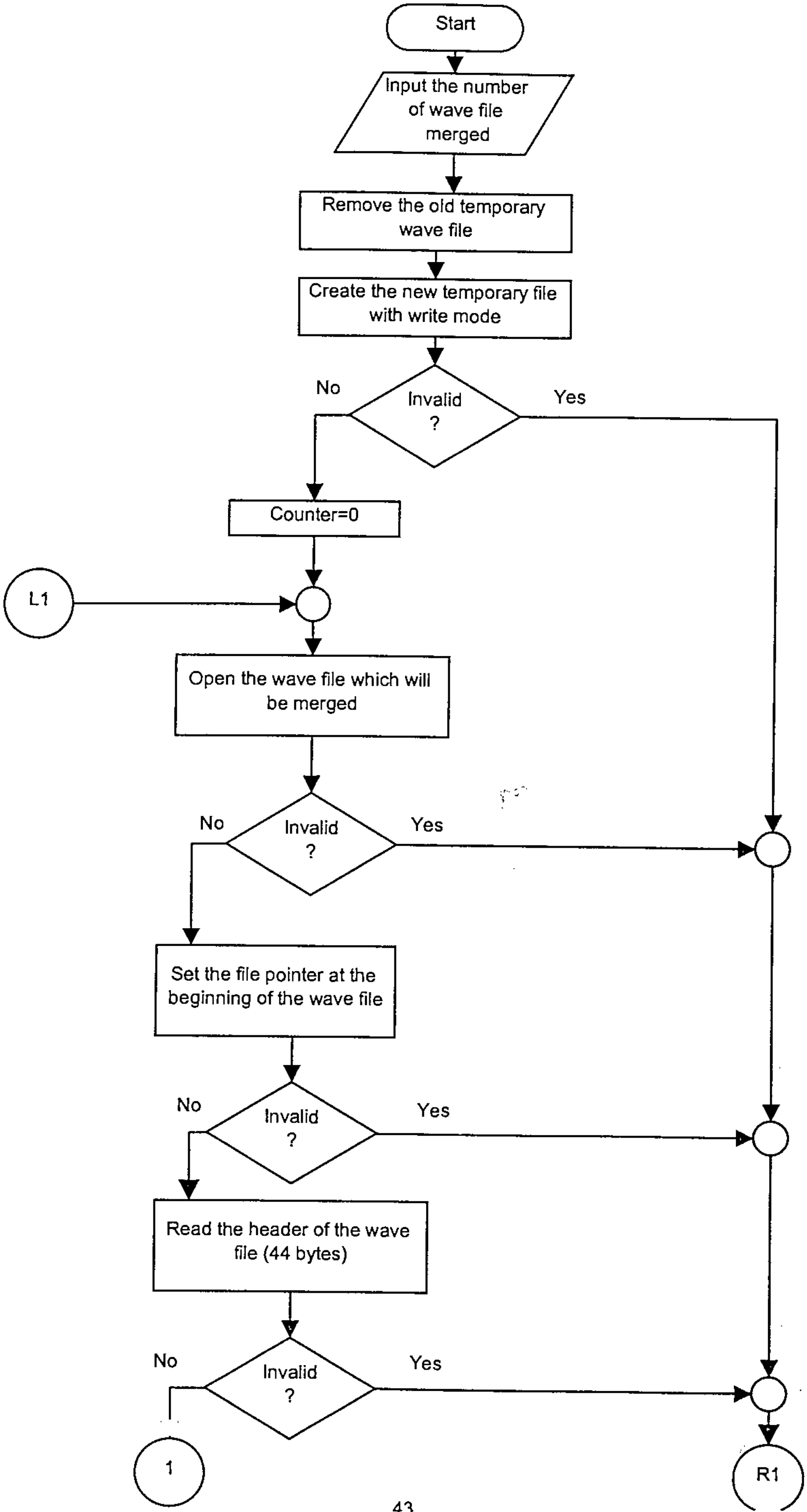
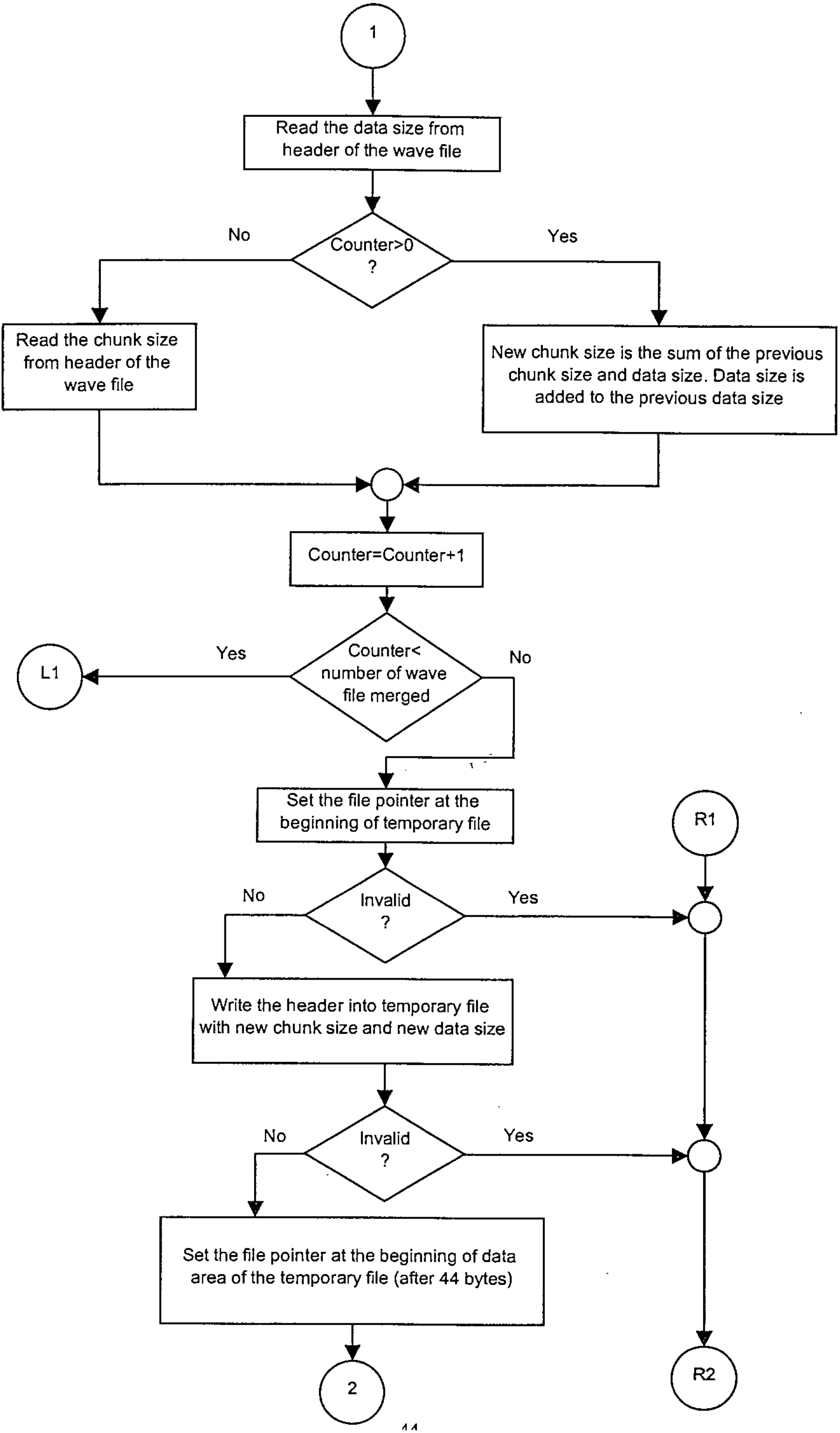


Figure 4.8 Block diagram of the wave file merging technique.


Start
Input the number of wave file merged
Remove the old temporary wave file
Create the new temporary file with write mode
Invalid ?
No
Yes
Counter=0
L1
Open the wave file which will be merged
Invalid ?
No
Yes
Set the file pointer at the beginning of the wave file
Invalid ?
No
Yes
Read the header of the wave file (44 bytes)
Invalid ?
No
Yes
1
R1


1
Read the data size from header of the wave file
No
Counter>0 ?
Yes
Read the chunk size from header of the wave file
New chunk size is the sum of the previous chunk size and data size. Data size is added to the previous data size
Counter=Counter+1
Yes
L1
Counter< number of wave file merged
No
Set the file pointer at the beginning of temporary file
R1
No
Invalid ?
Yes
Write the header into temporary file with new chunk size and new data size
No
Invalid ?
Yes
Set the file pointer at the beginning of data area of the temporary file (after 44 bytes)
2
R2

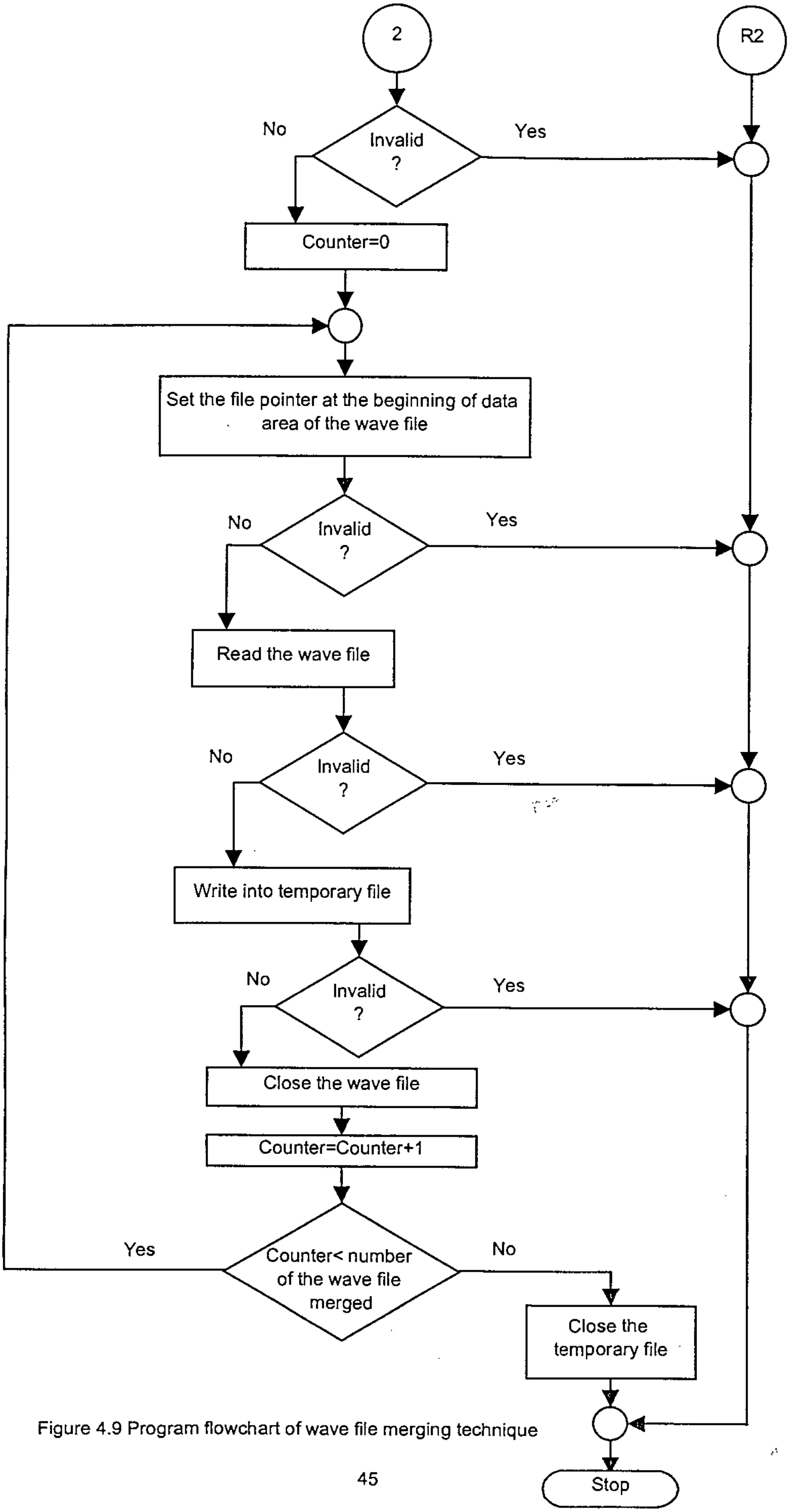


Figure 4.9 Program flowchart of wave file merging technique

Chapter 5

Interfacing with application software

This chapter discusses the concept of message driven programming. Section 5.1 introduces the basic idea of message driven programming, Section 5.2 discusses the message loop of windows programming. Section 5.3 discusses the message queuing technique of windows programming. Section 5.4 discusses the interface between Bangla text to speech synthesizer and application software. Section 5.5 describes the keyboard translator.

5.1 Introduction

Message driven programming (also known as event driven programming) is a process by which various subprocessess and/or applications communicate. In windows, messages are the process used by windows itself to manage a multitasking system and to share keyboards, mouse and other resources by distributing information to applications, application instances and processes within an application.

Instead of telling the computer what to do one step at a time, windows programs are structured to wait there until the program receives a message from windows. Messages can be sent to the application's program by keystrokes (pressed or released) or other external system events, such as selecting a menu or moving or clicking the mouse. The application program must interpret this message and then act on it. This is the basic concept of windows programming. The program of Bangla text to speech synthesizer has been developed on windows programming using 'VISUAL C++' language [10, 11, 13].

5.2 Message loop

The message loop of windows programming is the heart of a windows application. The message loop is as follows.

```
While  (GetMessage (&msg, NULL, O, O))
{
        TranslateMessage (&msg);
        DispatchMessage (&msg);
}
```

Windows passes all of the messages to the program via the functions in this loop. Windows keeps a message queue for each running application. When a key or a mouse button is pressed (input events), windows translates the input event into a message, then places the message in that application's message queue. The application retrieves these messages with the message loop.

GetMessage() pulls the next waiting message from windows into MSG structure pointed to by msg. The second parameter of GetMessage() 'NULL' instructs the function to retrieve any of the messages for any window that belongs to the application. The last two parameters, O and O, tell GetMessage() not to apply any message filters. Message filters can restrict retrieved messages to specify categories such as key strokes or mouse moves. GetMessage() examines msg and returns zero only if the message member of msg is WM_QUIT. When WM_QUIT is encountered, the program terminates, returning the value of msg.wParam. The MSG structure contains the following fields (defined in WINDOWS.H) [10,12].

```
typedef  struct tag MSG
{
HWND hwnd;          /* window handle */
UINT message;       /* 16 bit message ID */
WPARAM wParam;      /* 32 bit (double word) message parameter */
LPARAM lParam;      /* 32 bit (long) message parameter */
DWORD time;          /* time when message was placed in the queue */
POINT pt;            /* x,y mouse co- ordinates at the time the message was placed in the
                        message queue */
} MSG;
```

TranslateMessage() posts the corresponding WM_CHAR code on the applications message queue when a virtual key (WM_KEYUP or WM_KEYDOWN) code is received.

In windows, the wParam and lParam arguments accompanying each keyboard event message carry the event character code, the scan code, all of the shift state information just mentioned and, in addition, an eight bit key repeat count. Figure 5.1 illustrates the components of the wParam and lParam arguments [9] for WM_CHAR.

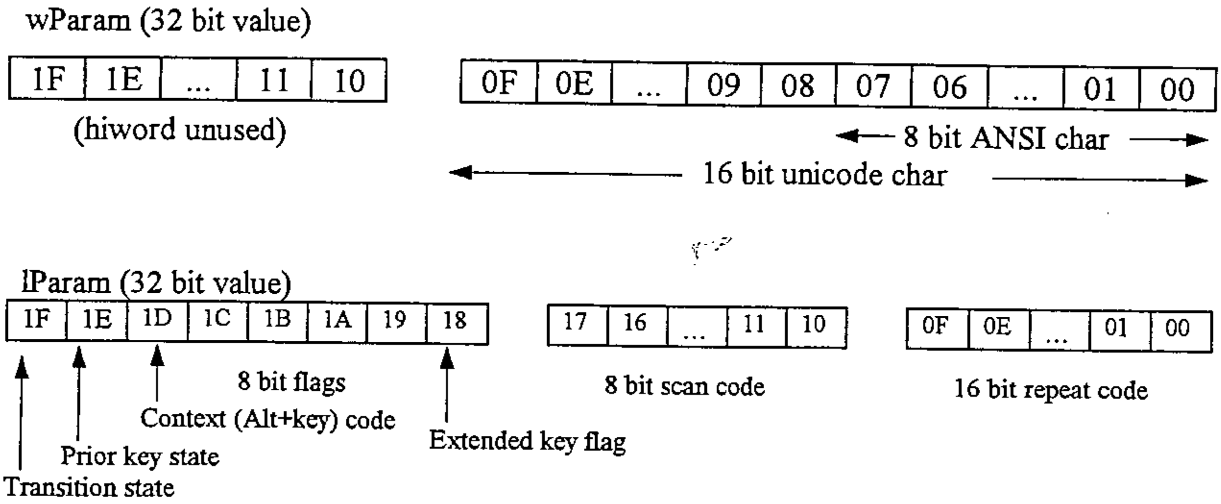


Figure 5.1 wParam and lParam of WM_CHAR message

lParam meaning [11] is shown in figure 5.2.

Repeat code		
0 - 15 (low order word)	:	The repeat count. This is the number of times the character is repeated because the user holds down a key.
Scan code		
16 – 23	:	The keyboard scan code.
Flags		
24	:	1 if an extended key, such as a function key or a key on the numeric keypad.
25 – 28	:	Not available.
29	:	1 if the 'Alt' key is held down when the key is pressed, 0 if not.
30	:	1 if the key is down before the message is sent, 0 if not.
31	:	1 if the key is being released, 0 if the key is being pressed.

Figure 5.2 WM_CHAR lParam coding.

DispatchMessage() sends the message to windows. When window's receives the message from DispatchMessage, it dispatches the message to the window's window procedure.

The program flowchart of main module of Bangla text to speech synthesizer which is related to message loop is shown in figure 5.3.

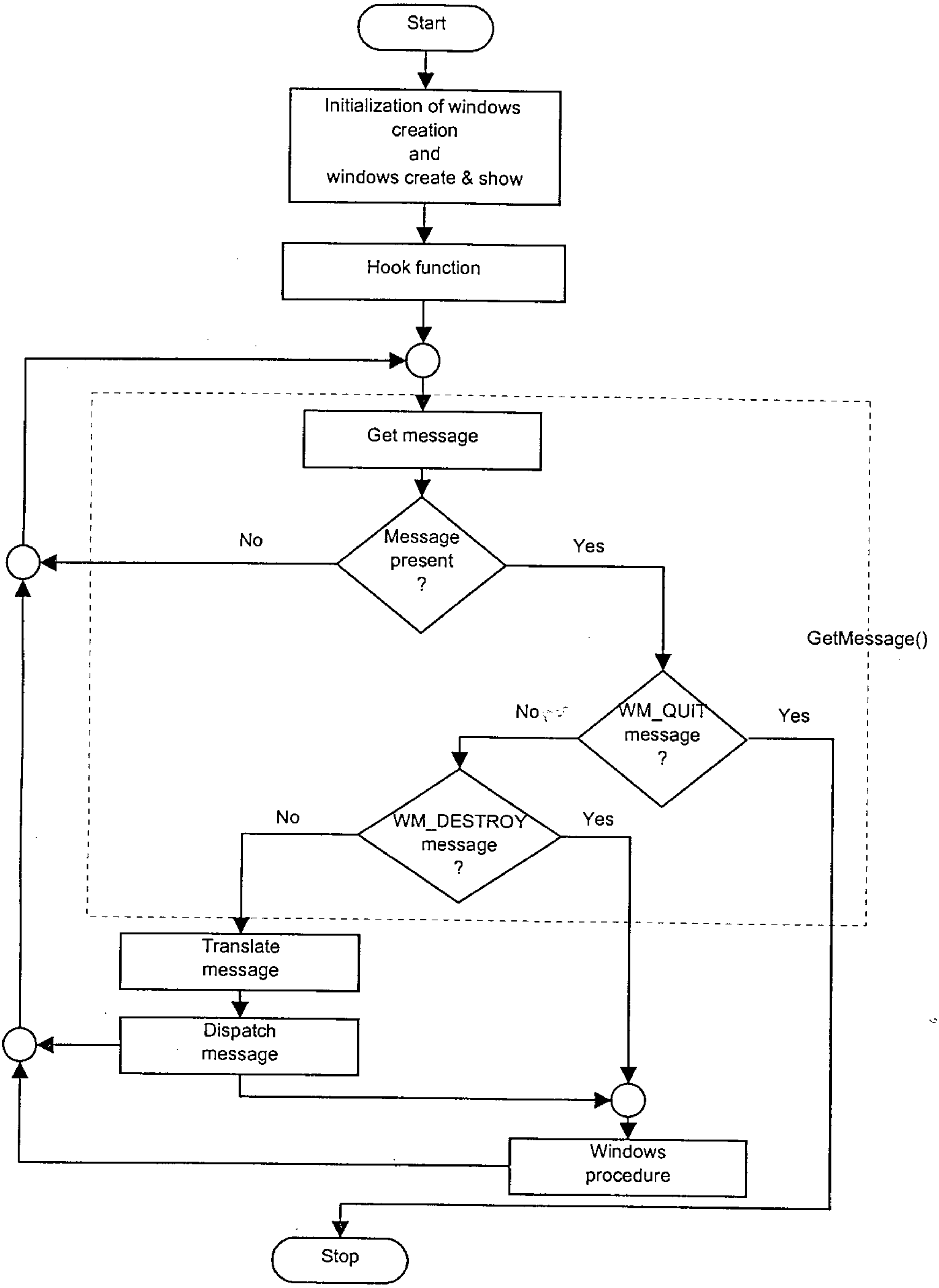


Figure 5.3 Program flowchart of main module of Bangla text to speech synthesizer

5.3 Message Queue

Since Windows has to control the execution of all active programs, all messages pass through windows before being dispatched to the appropriate application program. Within windows, the messages are processed by two queues (see figure 5.4). The first queue, the system queue, is where messages are originally placed until they can be read by Windows. After Windows examines a message, it is placed in a queue for one of the active programs. The queues for the application programs are called the application queues. Notice that the individual windows do not have queues; it is the applications that have queues. The main procedure of the program reads the application queue and then dispatches the message to the appropriate window routine within that application. The queues are generally quite short, normally three or fewer messages. Normally, the queue`s function is FIFO; that is, the first message in is the first out. Messages are processed in the order in which they are received [13].

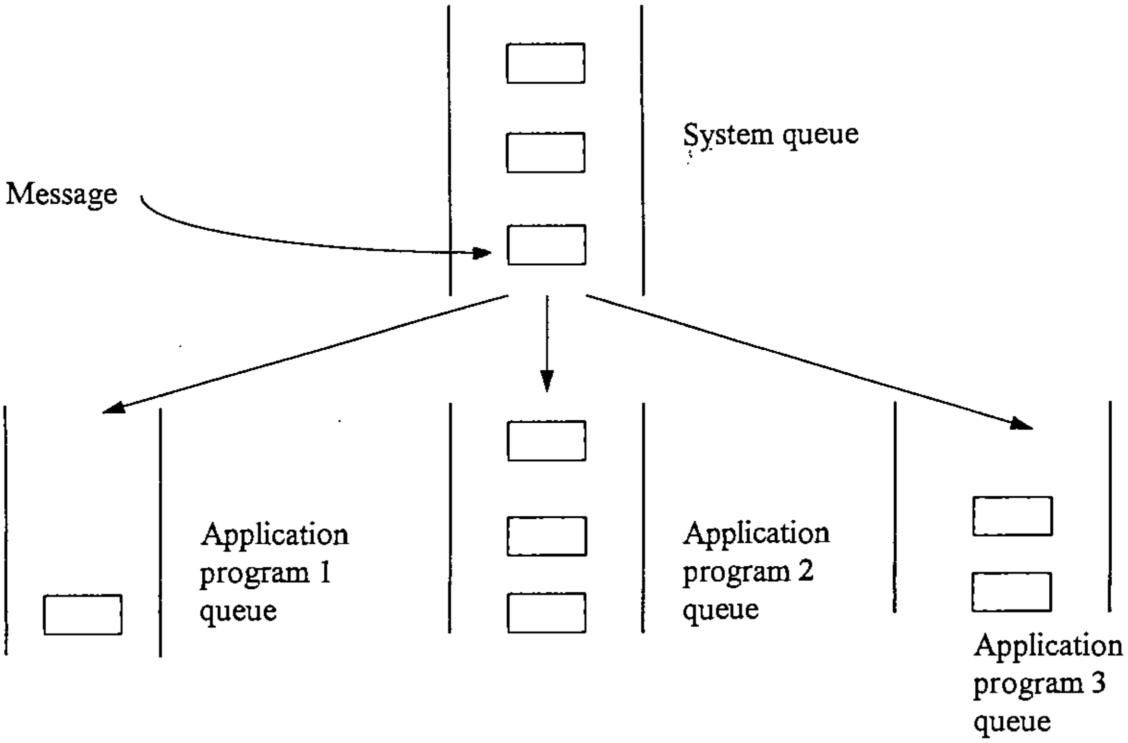

System queue
Message
Application program 1 queue
Application program 2 queue
Application program 3 queue

Figure 5.4 Message queues.

5.4 Interfacing with application software

When a key or a mouse button is pressed (input events) in any application/word processing software, windows translates the input event into a message, then places the message in the application's message queue. Then Bangla text to speech synthesizer will be interfaced with the application/word processing software so that at first the message is retrieved from application's message queue to Bangla text to speech synthesizer. Now Bangla text-to-speech synthesizer works according to the message and software. Then this message is sent back to the application's message queue. After this, application/word processing software takes this messages and works according to message and software. This procedure is shown in figure 5.5.

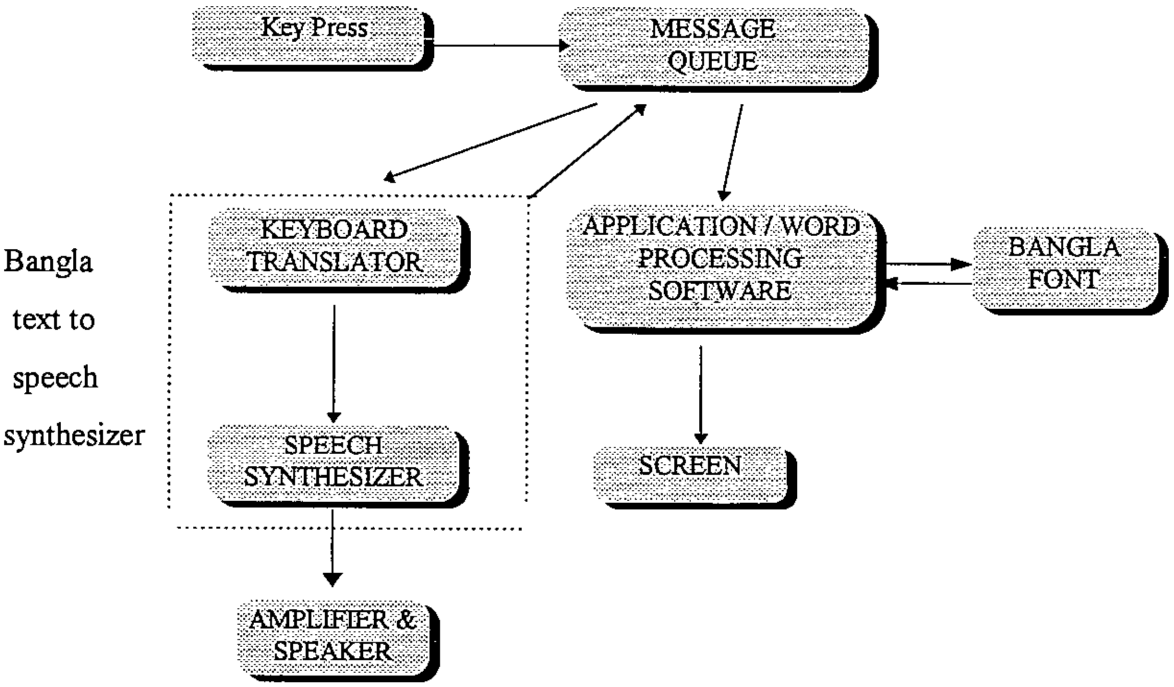


Figure 5.5 Interface of Bangla text to speech synthesizer with application/word processing software.

At first for retrieving the message from application's message queue to Bangla text to speech synthesizer, Bangla text to speech synthesizer must be hooked by the hook function. Hooks are one of the most powerful features of Windows. They give a way to trap various events that are about to occur with respect to the application or the entire windows system.

Windows now offer 12 hooks. If many hooks are installed, windows has to notify all of them. An application installs a hook by calling SetWindowsHookEx. The SetWindowsHookEx function installs an application defined hook procedure into a hook chain. A hook procedure can monitor events associated either with a specific thread or with all threads in the system. The prototype of the hook function appears below.

```
HHook  SetWindowsHookEx ( int   idHook,              // type of hook to install
                          HookProc  lpfn,            // address of hook procedure
                          HINSTANCE  hMod            // handle of application instance
                          DWORD   dwThread ID // identify of thread to install
                          );                           hook for
```

When this function is called, Windows allocates a block of memory containing an internal data structure describing the newly installed hook. All of these data structures are linked together to form a linked list with each new node (data structure) being placed at the front. After the node has been inserted into the list, SetWindowsHookEx returns a handle to the node. This value must be saved by the application needs to refer to the hook. idHook parameters specifies the type of hook procedure to be installed. This parameter can be one of the following values.

```
WH_CALLWINDPROC
WH_CALLWINDRPOCRET
WH_CBT
WH_DEBUG
WH_GETMESSAGE
WH_JOURNALPLAYBACK
WH_JOURNALRECORD
WH_KEYBOARD
WH_MOUSE
WH_MSGFILTER
WH_SHELL
WH_SYSMSG FILTER
```

In Bangla text to speech synthesizer, we have used WH_GETMESSAGE which installs a hook procedure that monitors messages posted to a message queue.

The hook function used in Bangla text to speech synthesizer appears bellow.

```
HHOOK  hHook = NULL;
hHook = SetWindowsHookEx (WH_GETMESSAGE,
                          (HOOKPROC) GetMsgProc, NULL, O);
```

The GetMsgProc hook procedure is an application defined or library defined call back function that the system calls whenever the GetMessage function has retrieved a message from an application message queue. Before passing the retrieved message to the destination window procedure, the system passes the message to the hook procedure. The GetMsgProc hook procedure can examine or modify the message. After the hook procedure returns control to the system, the GetMessage function returns the message, along with any modifications, to the application that originally called it. The prototype of GetMsgProc hook procedure appears bellow.

```
LRESULT CALLBACK GetMsgProc (     int code,              // hook code
                                  WPARAM    WParam, // removal flag
                                  LPARAM    lParam   // address of -
                                  );        // structure with message
```

- Parameter code specifies whether the hook procedure must process the message. if code is HC_ACTION, the hook procedure must process the message. If code is less than zero, the hook procedure must pass the message to the CallNextHookEx function without further processing and should return the value returned by CallNextHookEx.

- wParam specifies whether the message has been removed from the queue. This parameter can be one of the following values.

PM_NOREMOVE	specifies that the message has not been removed from the queue.
PM_REMOVE	specifies that the message has been removed from the queue.

- lParam points to an MSG structure that contains details about the message.

The prototype of CallNextHookEx is as follows.

```
LRESULT Call Next HooksEx ( HHOOK   hhook,
                            int  ncode,
                            WPARAM  wParam,
                            LPARAM  lParam );
```

When an application no longer needs it hooks, they are removed by calling UnhookWindowsHookEx.

The prototype of UnhookWindowsHookEx is as follows.

```
BOOL UnhookWindowsHookEx (HHOOK   hHook);
```

The hHook parameter specifies the handle of the hook. The handle was originally returned from the call to SetWindowsHooksEx when the hook was first installed.

There are keyboard translator and play procedure of the wave file in hook procedure. The program flowchart of hook procedure is shown in figure 5.6.

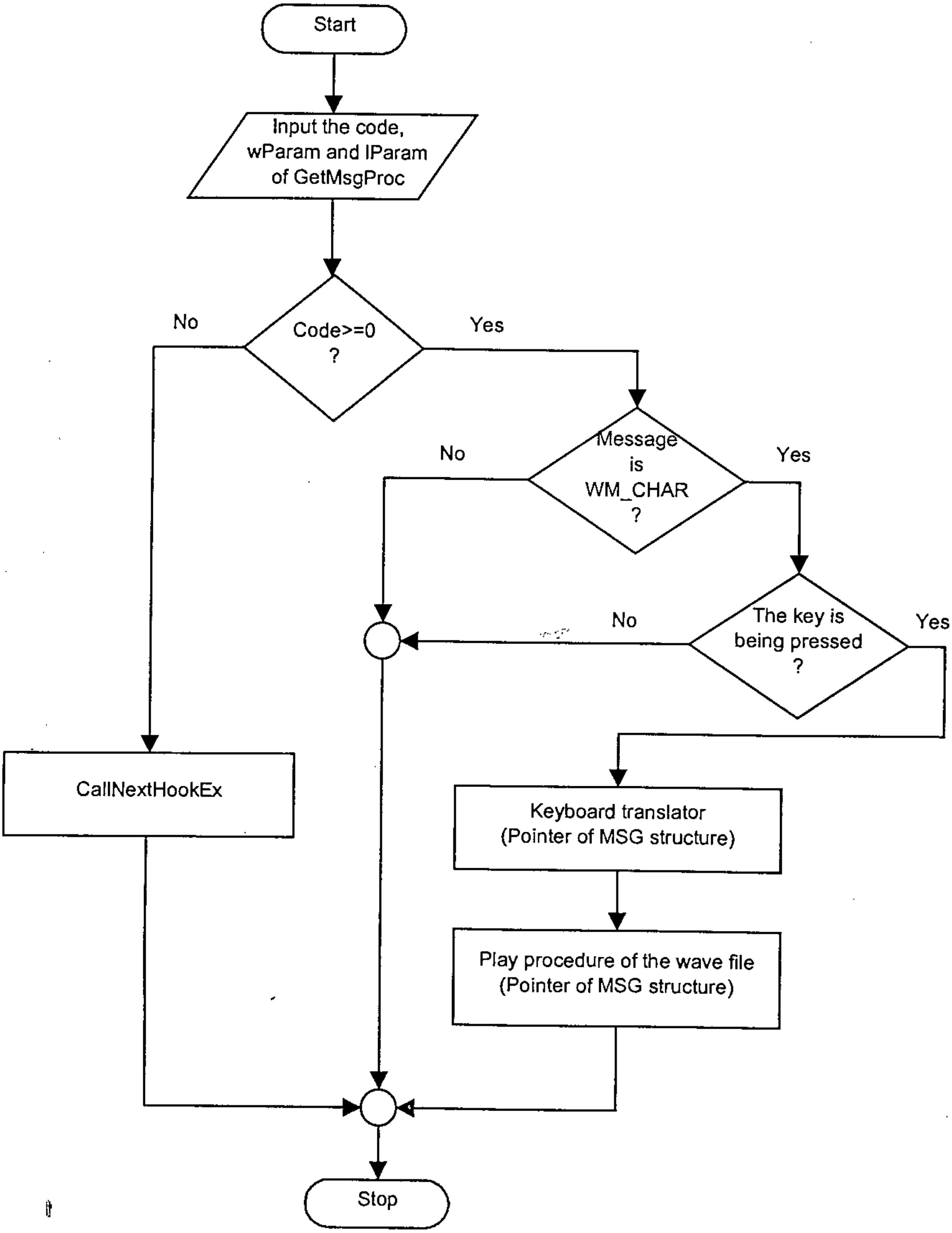

Start
Input the code, wParam and lParam of GetMsgProc
Code>=0 ?
No
Yes
Message is WM_CHAR ?
No
Yes
The key is being pressed ?
No
Yes
CallNextHookEx
Keyboard translator (Pointer of MSG structure)
Play procedure of the wave file (Pointer of MSG structure)
Stop

Figure 5.6 Program flowchart of hook procedure

5.5 Keyboard translator

Keyboard translator translates ASCII code to BANGLA code which is shown in figure 5.7.

ASCII code
KEYBOARD TRANSLATOR
BANGLA code

Figure 5.7 Block diagram of the keyboard translator.

When a key is pressed, Windows translates the input event into a message, then places the message in that program's message queue. The message of a character key pressed on the keyboard is WM_CHAR. This message is generated by TranslateMessage() function in the program's message loop. wParam parameter of WM_CHAR contains the ASCII value of the key pressed.

For Bangla character, there must be a code which is not fixed up now. But we have considered the codes of the popular Bangla font sulekha, proshun etc. which are shown in Appendix C. In these codes, the code of Bangla character 'ক' is 75. The popular Bangla keyboard layouts are Lekhani, Munir, Jatiya, Bijoy etc. But now there is no standard Bangla keyboard layout. We have considered the Bijoy keyboard layout (shown in Appendix D) for development of Bangla text to speech synthesizer. For typing the Bangla Character 'ক' in Bijoy keyboard layout, j (106) key is pressed. Then the parameter of wParam of WH_CHAR message is 106. But for typing Bangla character 'ক', wParam value must be 75. So when pressing the j(106) key, the wParam value must be changed by 75 for typing 'ক'. This is keyboard translator. Note that inspite of wrong typing, backspace is pressed for deletion of wrong character and during merging the wave files, these wrong characters are omitted. For Bangla text to speech synthesizer, we need application/word processing software with Bangla font and Bangla text to speech synthesizer has built in keyboard translator. The program flowchart of the keyboard translator is shown in figure 5.8.

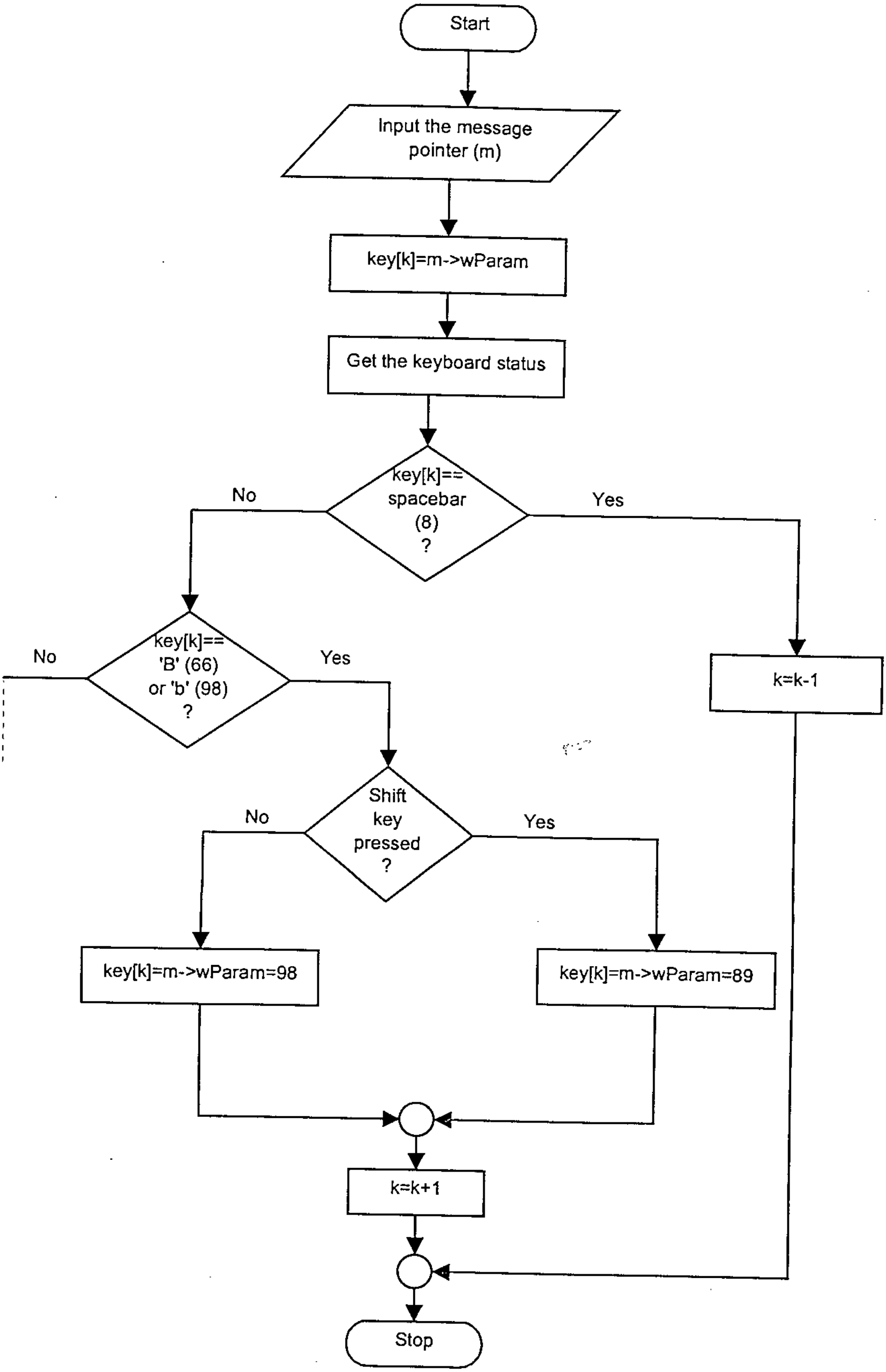


Figure 5.8 Program flowchart of keyboard translator

Chapter 6

Conclusion

Chapter 6 is the conclusion chapter. Section 6.1 describes the results of developed Bangla text to speech synthesizer. Section 6.2 discusses the suggestions for further research in this field. Section 6.3 discusses the conclusion of the thesis.

6.1 Results of developed Bangla text to speech synthesizer

Bangla text to speech synthesizer can generate sounds from arbitrary text inputs. We have developed the generalized software for synthesis of speech from Bangla text and tested this software which works properly. At first Bangla text to speech synthesizer accepts the character from keyboard and identifies the sound unit and plays that. This task is repeated until the word is completed. Then the text is normalized by defining the wave files from the adjacent keys as per pronunciation rule and searched. When identification of the sound units for a particular Bangla word is completed, all the sound units are merged together into a speech file to form the complete speech signal for that Bangla word and finally the merged wave file is played. Note that Bangla text to speech synthesizer is interfaced with the application/word processing software so that at first the message (generated by pressing key) is retrieved from application's message queue to Bangla text to speech synthesizer.

If we store more basic sound units, then Bangla text to speech synthesizer can work properly from any input text.

As Bangla text to speech synthesizer merges different sound units together to form a word, there remains the problem of intonation (pitch). Slapping the sound units together from different words, recorded at different times, may also result in a choppy 'sing song' effect since the pitch and duration of words will not necessarily match.

6.2 Suggestions for further research in this field

There may be modifications of Bangla text to speech synthesizer to suit specific needs. These and the other topics of research are described below.

(a) To produce natural sound

As Bangla text to speech synthesizer merges different sound units together to form a word, there remains the problem of intonation (pitch). Slapping the sound units together from different words, recorded at different times, may also result in a choppy 'sing song' effect since the pitch and duration of words will not necessarily match. So further research may be carried out to produce natural sounds by reducing the 'sing song' effect and the problem of intonation. In this regard, the envelop of the wave file will be determined and carrier frequency will be added to the wave file for obtaining the same pitch of the wave file.

(b) To develop the mathematical model of Bangla Sound units

Research may be carried out to develop proper mathematical model of Bangla sound units, which would enable the current Bangla text to speech synthesizer to produce synthetic Bangla speech.

(c) To determine the basic Bangla sound units of all Bangla words

For development of Bangla text to speech synthesizer, the basic Bangla sound units of all Bangla words must be determined which is the field of Bangla phoneme research.

(d) Handling an imperative, interrogative, exclamatory sentences and joint letter

Research may be carried out to handle an imperative, interrogative, exclamatory sentences and joint letter which express the feeling of the speaker differently at different situations.

(e) Introducing the text normalizer including syntax parser and abbreviation interpreter

Research may also be carried out to introduce the text normalizer. By text normaliser, a string such as ১,২৩৪ should be spoken as এক হাজার দুই শত চৌত্রিশ and not as এক কমা দুই তিন চার if it is to be clearly understood by the general listener. Text normalisation can also handle syntax parsing, abbreviations and forms like ২২তম and ১২ঃ00.

(i) Syntax parsing

Further obstacles to the development of a rule set for interpreting spelling are provided by pairs of words which are spelled the same but pronounced differently. As for example, Bangla text to speech synthesizer produces the correct speech sound of Bangla word 'বার' when it represents a day, However, when the same word 'বার' is used to represent a number (which represents twelve in Bangla), the synthesizer will still pronounce the word as to mean 'day' of week. The only real solution to this problem would be to devise an algorithm capable of determining the syntactic structure of a given sentence and assigning each word to its proper grammatical category.

(ii) Abbreviation interpreting

Another function 'abbreviation interpreting' should be included in Bangla text to speech synthesizer. We can use the abbreviation such as মিঃ, ডঃ etc in Bangla sentences. But the exception rules must be accomplished in Bangla text to speech synthesizer which convert the abbreviation into spoken form such as মিঃ to মিষ্টার.

(f) To play sentences or paragraph continuously

A provision can be kept in the Bangla text to speech synthesizer to generate play files corresponding to text inputs of a sentences or paragraph, and then the speech files of the paragraph or page should be played back sequentially at a time. This will produce speech sounds of the texts of a paragraph or of a page continuously. However, this method would increase the hard disk surface requirement to store many speech files and would take more time to convert text to speech file before the actual playing of the files.

(g) Reducing the sound file size by compression

The higher quality recording of digital audio files requires more disk space. So research may be carried out to reduce the sound file size by compression during recording and file is decompressed when playing. The size of the wave file can be reduced by the new audio compression technique. Human ear cannot detect all audio frequency. The principle of audio compression is to cut off the frequency which is out of audibility limits of human ear.

(h) To develop the application software

Research may be carried out to develop application softwares and education tools for PC using the speech sounds produced by the Bangla text to speech synthesizer.

(i) Handling exceptions

Research may be carried out to handle exceptions when many problems might have remained unnoticed.

6.3 Conclusion

Bangla text to speech synthesizer can generate speech sounds from arbitrary text input. Bangla text to speech synthesizer has been developed using the method that stores speech sounds of parts of words. The parts of words i.e. basic sounds units have been recorded using sound card, microphone and sound recorder utility software. Though the pronunciation rules and the determination of basic sound units of Bangla words have been studied shortly as part of the current research yet Bangla phoneme specialist can do this extensively. At first Bangla text to speech synthesizer accepts the character from keyboard and identifies the sound unit and plays that. This task is repeated until the word is completed. Then the text is normalized by defining from the adjacent keys as per pronunciation rule and searched. When identification of the sound units for a particular Bangla word is completed, all the sound units are merged together into a speech file to form the complete speech signal for that Bangla word and finally the merged wave file are played. Note that Bangla text to speech synthesizer will be interfaced with the application/word processing software so that at first the message (generated by pressing key) is retrieved from application's message queue to Bangla text to speech synthesizer.

A generalized software has been developed for synthesis of speech from Bangla text which can generate any speech sounds from any text inputs. But if we store more basic sound units, then Bangla text to speech synthesizer can work properly from any input text.

As Bangla text to speech synthesizer merges different sound units together to form a word, there remains the problem of intonation (pitch). Slapping the sound units together from different words, recorded at different times, may also result in a choppy 'sing song' effect since the pitch and duration of words will not necessarily match.
We do not handle an imperative, interrogative, exclamatory sentences and joint letter which express the feeling of the speaker differently at different situations.

We do not introduce the text normalizer including syntax parser and abbreviation interpreter.

The code of Bangla character is not fixed now. Also there is no standardisation of Bangla keyboard layout. So problem was arised during development of Bangla text to speech synthesizer.

Many other problems might have remained unnoticed. These problems will, hopely, be eliminated in the subsequent versions of Bangla text to speech synthesizer.

References

[1] Bristow, Geoff, "Electronic Speech Synthesis", McGraw-Hill Book Company, 1st Edition, 1984, pp: 19-47, 94-113.

[2] Chen, Tsuhan, "The past, present and futer of speech processing", IEEE Signal Processing Magazine, May 1998, Vol. 15, No. 3, pp 24-48.

[3] "Speech synthesis", IEEE Signal Processing Magazine, July 1997, Vol. 14, No. 4.

[4] Sinha, D.K. (ed), "Special issue on computer applications in processing of indian languages and scripts", JIETE, Vol. 30, No. 6, November 1984.

[5] Hwang, S.H. and Chen, S.H., "Neural network based F0 text to speech syntherizer for Mandarin", IEE Procedings, VISION, IMAGE AND SIGNAL PROCESSING, Vol. 141, Number 6, December 1994.

[6] Islam, Md. Nazrul, "Development of a Bangla text to speech converter", M.Sc. Engg. thesis, Department of Electrical & Electronic Engg., BUET, April, 1995.

[7] Uddin, M. Jamal and Sobhan, M.A., "Studies on the effects of number of periods on the formants frequencies of Bangla voice using Hamming window", 38th Annual Convention of IEB, 1994.

[8] Heimlich, Rich and Golden, David M. and Luk, Ivan and Ridge, Peter M., "Sound Blaster : The Official Book", Osborne McGraw-Hill, 1993, pp:4-57, 428-444.

[9] Aitken, Peter and Jarol, Scott, "Visual C++ Multimedia", Comdex Computer Publishing, 3rd Edition, January 1997, pp: 86-92, 108-111.

[10] Ezzell, Ben and Blaney, Jim, "NT 4/Windows 95 Developer's Hand Book", BPB book centre, First Edition, 1997, pp : 16-183, 1230-1295.

[11] Conger, James L., "Windows API Bible", Galgotia Publications Pvt Ltd.,1996, pp: 1- 349.

[12] Pappas, Chris H. and Murry, William H., "Visual C++ 5: The Complete Reference", Tata McGraw Hill Publishing Company Limited, New Delhi, 1998, pp : 682-732.

[13] Townsend, Cart, "Advanced MS-DOS Expert Technique for Programmers", Howard W. Sams and Company, First Edition, pp 359-405.

Appendix A

Bangla Alphabets

Vowels of Bangla word

অ আ ই ঈ উ ঊ ঋ এ ঐ ও ঔ

Vowel auxiliaries of Bangla word

া ি ী ু ূ ৃ ে ৈ ৗ

Consonants of Bangla word

ক খ গ ঘ ঙ চ ছ জ ঝ ঞ

ট ঠ ড ঢ ণ ত থ দ ধ ন

প ফ ব ভ ম য র ল শ ষ

স হ ড় ঢ় য় ৎ ং ঃ ঁ

Other Symbols

্ র্ ্য ্র

Appendix B

Bangla Sound units and their file names

Serial No.	Bangla Sound Unit	File Name
1	কো	w0001.wav
2	খো	w0002.wav
3	গো	w0003.wav
4	ঘো	w0004.wav
5	*	
6	চো	w0006.wav
7	ছো	w0007.wav
8	জো	w0008.wav
9	ঝো	w0009.wav
10	*	
11	টো	w0011.wav
12	ঠো	w0012.wav
13	ডো	w0013.wav
14	ঢো	w0014.wav
15	ণো	w0015.wav
16	তো	w0016.wav
17	*	
18	*	
19	*	
20	*	
21	থো	w0021.wav
22	দো	w0022.wav
23	ধো	w0023.wav
24	নো	w0015.wav
25	পো	w0025.wav
26	ফো	w0026.wav
27	বো	w0027.wav
28	ভো	w0028.wav
29	মো	w0029.wav
30	যো	w0008.wav
31	রো	w0031.wav

Serial No.	Bangla Sound Unit	File Name
32	লো	w0032.wav
33	শো	w0033.wav
34	ষো	w0033.wav
35	সো	w0033.wav
36	হো	w0036.wav
37	*	
38	*	
39	য়ো	w0039.wav
40	কা	w0041.wav
41	খা	w0042.wav
42	গা	w0043.wav
43	ঘা	w0044.wav
44	*	
45	চা	w0046.wav
46	ছা	w0047.wav
47	জা	w0048.wav
48	ঝা	w0049.wav
49	*	
50	টা	w0050.wav
51	ঠা	w0051.wav
52	ডা	w0052.wav
53	ঢা	w0053.wav
54	ণা	w0054.wav
55	তা	w0055.wav
56	*	
57	*	
58	*	
59	*	
60	থা	w0060.wav
61	দা	w0061.wav
62	ধা	w0062.wav

Serial No.	Bangla Sound Unit	File Name
63	না	w0054.wav
64	পা	w0064.wav
65	ফা	w0065.wav
66	বা	w0066.wav
67	ভা	w0067.wav
68	মা	w0068.wav
69	যা	w0047.wav
70	রা	w0070.wav
71	লা	w0071.wav
72	শা	w0072.wav
73	ষা	w0072.wav
74	সা	w0072.wav
75	হা	w0075.wav
76	*	
77	*	
78	য়া	w0078.wav
79	কু	w0079.wav
80	খু	w0080.wav
81	গু	w0081.wav
82	ঘু	w0082.wav
83	*	
84	চু	w0084.wav
85	ছু	w0085.wav
86	জু	w0086.wav
87	ঝু	w0087.wav
88	*	
89	টু	w0089.wav
90	ঠু	w0090.wav
91	ডু	w0091.wav
92	ঢু	w0092.wav
93	ণু	w0093.wav
94	তু	w0094.wav
95	*	
96	*	
97	*	
98	*	
99	থু	w0099.wav
100	দু	w0100.wav
101	ধু	w0101.wav
102	নু	w0093.wav
103	পু	w0103.wav
104	ফু	w0104.wav
105	বু	w0105.wav
106	ভু	w0106.wav
107	মু	w0107.wav
108	যু	w0086.wav
109	রু	w0109.wav
110	লু	w0110.wav
111	শু	w0111.wav
112	ষু	w0111.wav
113	সু	w0111.wav
114	হু	w0114.wav
115	*	
116	*	
117	য়ু	w0117.wav
118	কী	w0118.wav
119	খী	w0119.wav
120	গী	w0120.wav
121	ঘী	w0121.wav
122	*	
123	চী	w0123.wav
124	ছী	w0124.wav
125	জী	w0125.wav
126	ঝী	w0126.wav
127	*	
128	টী	w0128.wav
129	ঠী	w0129.wav
130	ডী	w0130.wav
131	ঢী	w0131.wav
132	ণী	w0132.wav
133	তী	w0133.wav
134	*	
135	*	
136	*	
137	*	
138	থী	w0138.wav

Serial No.	Bangla Sound Unit	File Name
139	দী	w0139.wav
140	ধী	w0140.wav
141	নী	w0132.wav
142	পী	w0142.wav
143	ফী	w0143.wav
144	বী	w0144.wav
145	ভী	w0145.wav
146	মী	w0146.wav
147	যী	w0125.wav
148	রী	w0148.wav
149	লী	w0149.wav
150	শী	w0150.wav
151	ষী	w0150.wav
152	সী	w0150.wav
153	হী	w0153.wav
154	*	
155	*	
156	য়ী	w0156.wav
157	কে	w0157.wav
158	খে	w0158.wav
159	গে	w0159.wav
160	ঘে	w0160.wav
161	*	
162	চে	w0162.wav
163	ছে	w0163.wav
164	জে	w0164.wav
165	ঝে	w0165.wav
166	*	
167	টে	w0167.wav
168	ঠে	w0168.wav
169	ডে	w0169.wav
170	ঢে	w0170.wav
171	ণে	w0171.wav
172	তে	w0172.wav
173	*	
174	*	
175	*	
176	*	

Serial No.	Bangla Sound Unit	File Name
177	থে	w0177.wav
178	দে	w0178.wav
179	ধে	w0179.wav
180	নে	w0171.wav
181	পে	w0181.wav
182	ফে	w0182.wav
183	বে	w0183.wav
184	ভে	w0184.wav
185	মে	w0185.wav
186	যে	w0164.wav
187	রে	w0187.wav
188	লে	w0188.wav
189	শে	w0189.wav
190	ষে	w0189.wav
191	সে	w0189.wav
192	হে	w0192.wav
193	*	
194	*	
195	য়ে	w0195.wav
196	কি	w0118.wav
197	খি	w0119.wav
198	গি	w0120.wav
199	ঘি	w0121.wav
200	*	
201	চি	w0123.wav
202	ছি	w0124.wav
203	জি	w0125.wav
204	ঝি	w0126.wav
205	*	
206	টি	w0128.wav
207	ঠি	w0129.wav
208	ডি	w0130.wav
209	ঢি	w0131.wav
210	ণি	w0132.wav
211	তি	w0133.wav
212	*	
213	*	
214	*	

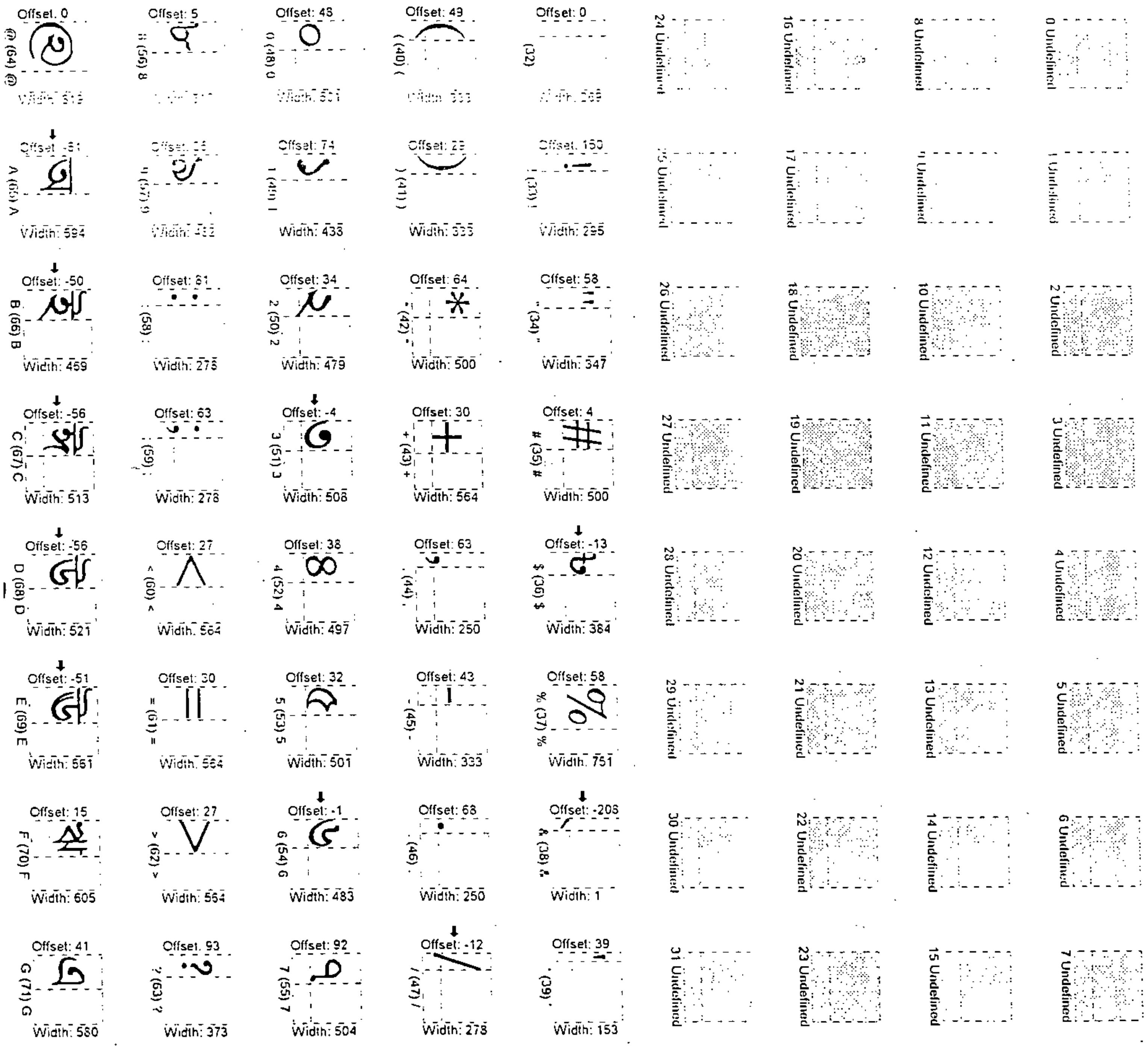
Serial No.	Bangla Sound Unit	File Name
215	*	
216	থি	w0138.wav
217	দি	w0139.wav
218	ধি	w0140.wav
219	নি	w0132.wav
220	পি	w0142.wav
221	ফি	w0143.wav
222	বি	w0144.wav
223	ভি	w0145.wav
224	মি	w0146.wav
225	যি	w0125.wav
226	রি	w0148.wav
227	লি	w0149.wav
228	শি	w0150.wav
229	ষি	w0150.wav
230	সি	w0150.wav
231	হি	w0153.wav
232	*	
233	*	
234	য়ি	w0156.wav
235	ক	w0235.wav
236	খ	w0236.wav
237	গ	w0237.wav
238	ঘ	w0238.wav
239	*	
240	চ	w0240.wav
241	ছ	w0241.wav
242	জ	w0242.wav
243	ঝ	w0243.wav
244	*	
245	ট	w0245.wav
246	ঠ	w0246.wav
247	ড	w0247.wav
248	ঢ	w0248.wav
249	ণ	w0249.wav
250	ত	w0250.wav
251	*	
252	*	
253	*	
254	*	
255	থ	w0255.wav
256	দ	w0256.wav
257	ধ	w0257.wav
258	ন	w0249.wav
259	প	w0259.wav
260	ফ	w0260.wav
261	ব	w0261.wav
262	ভ	w0262.wav
263	ম	w0263.wav
264	য	w0242.wav
265	র	w0265.wav
266	ল	w0266.wav
267	শ	w0267.wav
268	ষ	w0267.wav
269	স	w0267.wav
270	হ	w0270.wav
271	*	
272	*	
273	য়	w0273.wav
274	অ	w0274.wav
275	ই	w0275.wav
276	ঈ	w0276.wav
277	উ	w0277.wav
278	ঊ	w0278.wav
279	ঋ	w0279.wav
280	এ	w0280.wav
281	ঐ	w0281.wav
282	ও	w0282.wav
283	ঔ	w0283.wav
284	া	w0284.wav
285	দের	w0285.wav

* Reserved for future use.

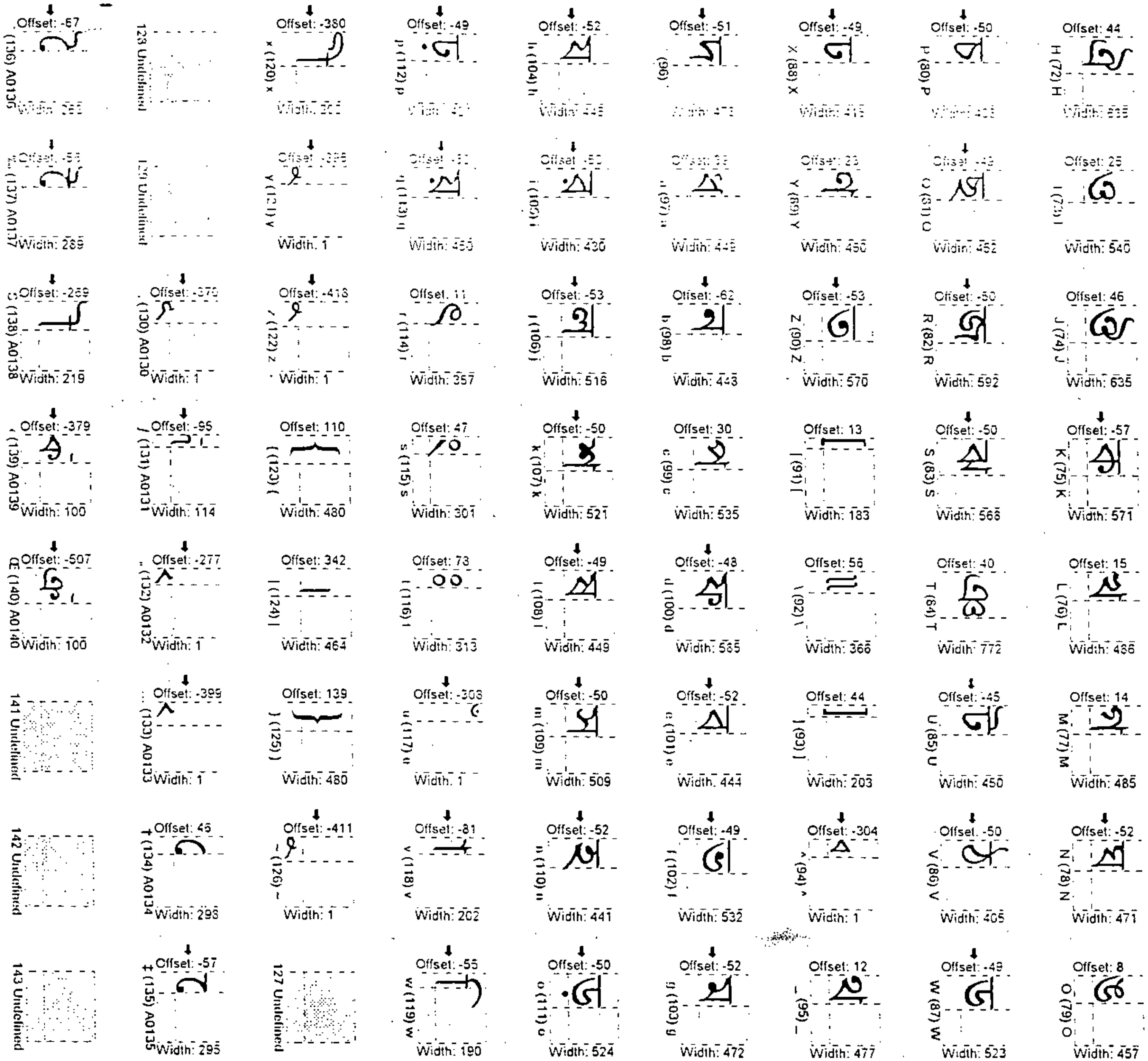
Appendix C

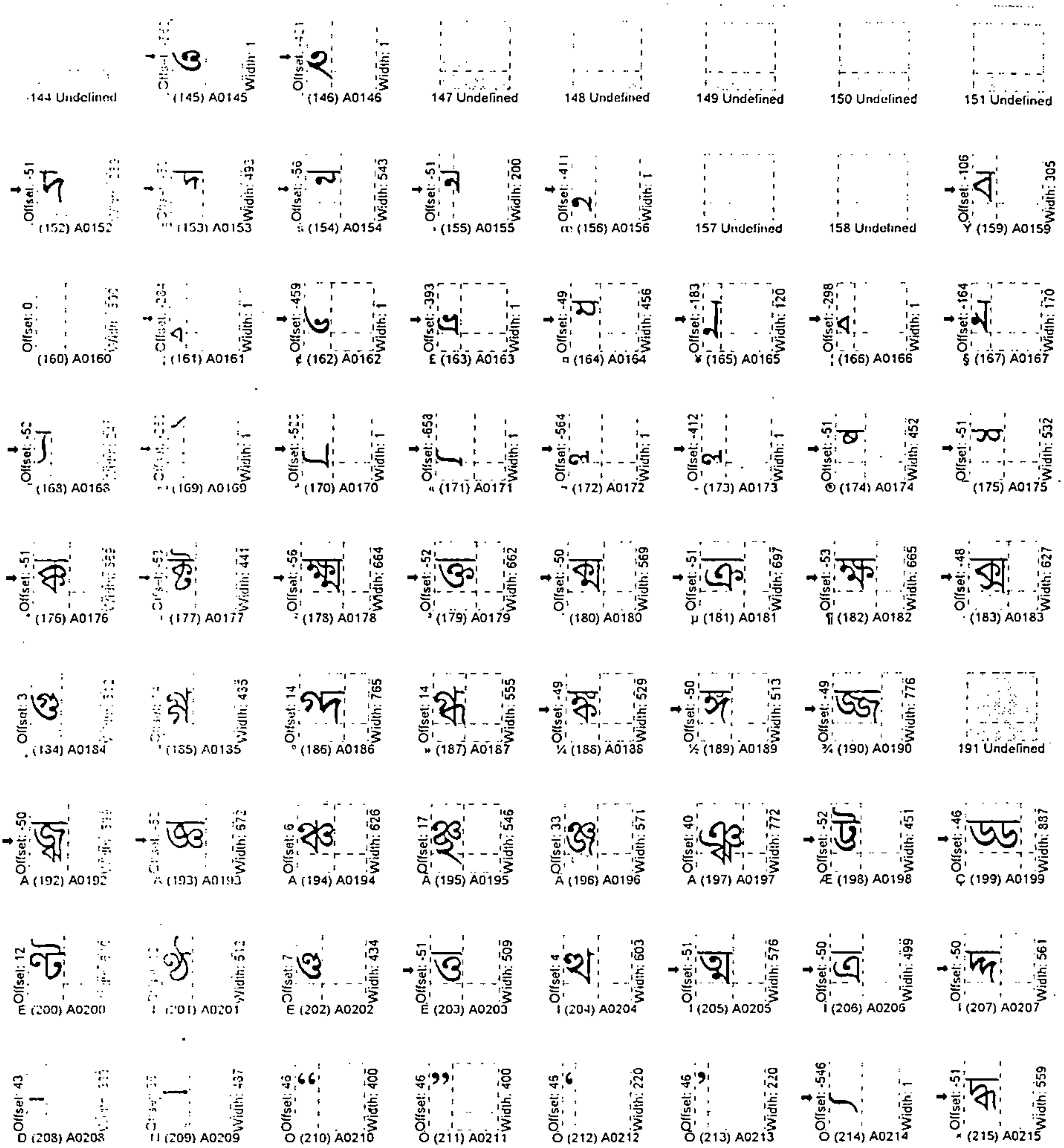
Bangla symbols and their codes

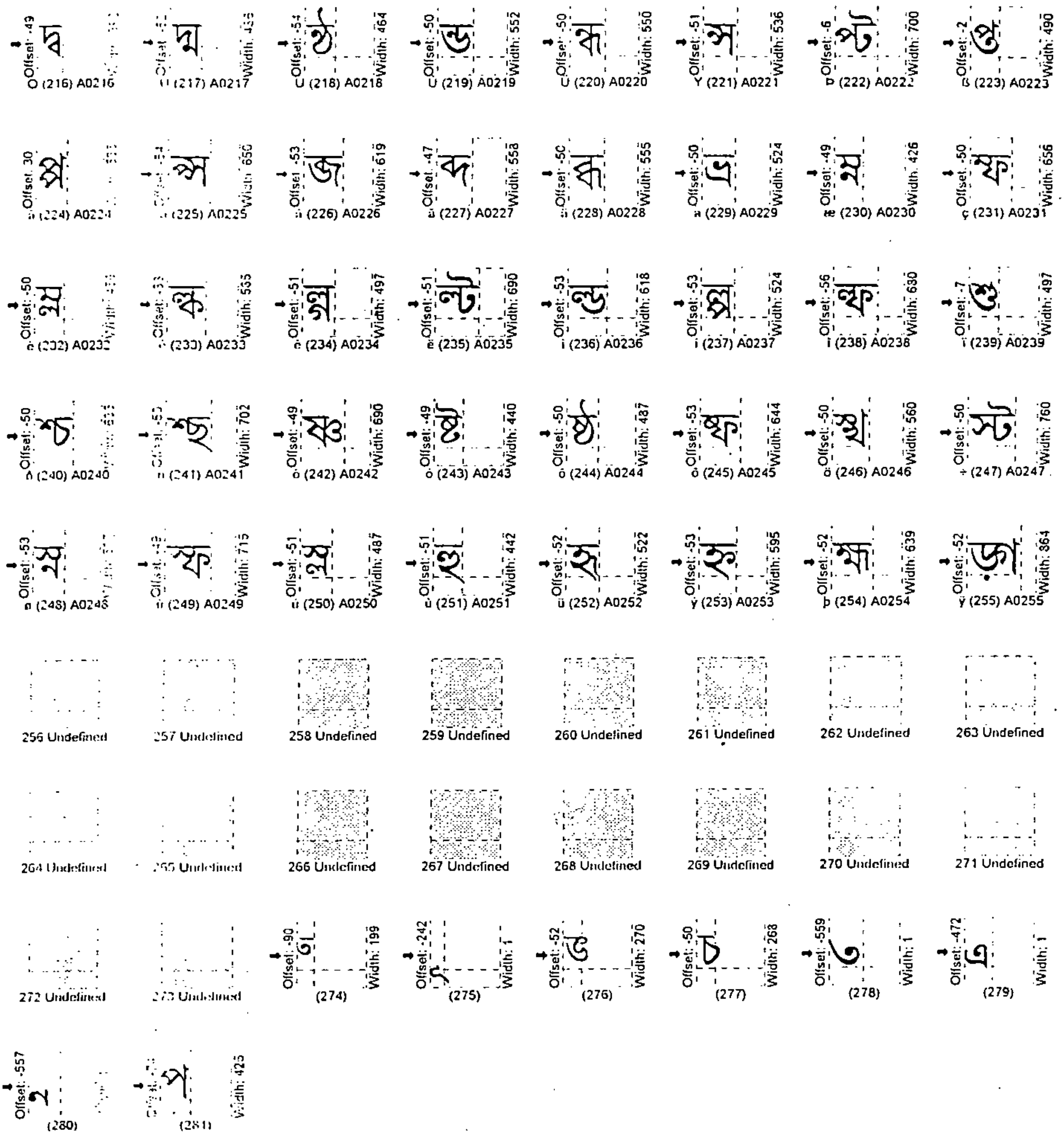
(SulekhaT font)



Offset. 0 @ (64) @ Width: 516	Offset: 5 8 (56) 8 [illegible]	Offset: 48 0 (48) 0 Width: 501	Offset: 49 ((40) (Width: 333	Offset: 0 (32) Width: 269	24 Undefined	16 Undefined	8 Undefined	0 Undefined
Offset: -61 A (65) A Width: 594	Offset: 36 9 (57) 9 Width: 432	Offset: 74 1 (49) 1 Width: 433	Offset: 23) (41)) Width: 333	Offset: 150 ! (33) ! Width: 295	25 Undefined	17 Undefined	9 Undefined	1 Undefined
Offset: -50 B (66) B Width: 459	Offset: 61 : (58) : Width: 278	Offset: 34 2 (50) 2 Width: 479	Offset: 64 * (42) * Width: 500	Offset: 58 " (34) " Width: 347	26 Undefined	18 Undefined	10 Undefined	2 Undefined
Offset: -56 C (67) C Width: 513	Offset: 63 ; (59) ; Width: 278	Offset: -4 3 (51) 3 Width: 508	Offset: 30 + (43) + Width: 564	Offset: 4 # (35) # Width: 500	27 Undefined	19 Undefined	11 Undefined	3 Undefined
Offset: -56 D (68) D Width: 521	Offset: 27 < (60) < Width: 564	Offset: 38 4 (52) 4 Width: 497	Offset: 63 , (44) , Width: 250	Offset: -13 $ (36) $ Width: 384	28 Undefined	20 Undefined	12 Undefined	4 Undefined
Offset: -51 E (69) E Width: 561	Offset: 30 = (61) = Width: 564	Offset: 32 5 (53) 5 Width: 501	Offset: 43 - (45) - Width: 333	Offset: 58 % (37) % Width: 751	29 Undefined	21 Undefined	13 Undefined	5 Undefined
Offset: 15 F (70) F Width: 605	Offset: 27 > (62) > Width: 564	Offset: -1 6 (54) 6 Width: 483	Offset: 68 . (46) . Width: 250	Offset: -206 & (38) & Width: 1	30 Undefined	22 Undefined	14 Undefined	6 Undefined
Offset: 41 G (71) G Width: 580	Offset: 93 ? (63) ? Width: 373	Offset: 92 7 (55) 7 Width: 504	Offset: -12 / (47) / Width: 278	Offset: 39 ' (39) ' Width: 153	31 Undefined	23 Undefined	15 Undefined	7 Undefined


Offset: -67
(136) A0136
Width: 285
128 Undefined
Offset: -380
x (120) x
Width: 305
Offset: -49
p (112) p
Width: 401
Offset: -52
h (104) h
Width: 445
Offset: -51
` (96) `
Width: 473
Offset: -49
X (88) X
Width: 415
Offset: -50
P (80) P
Width: 403
Offset: 44
H (72) H
Width: 635
Offset: -56
(137) A0137
Width: 289
129 Undefined
Offset: -395
y (121) y
Width: 1
Offset: -51
q (113) q
Width: 455
Offset: -50
i (105) i
Width: 430
Offset: 33
a (97) a
Width: 449
Offset: 23
Y (89) Y
Width: 450
Offset: -49
Q (81) Q
Width: 452
Offset: 25
I (73) I
Width: 540
Offset: -269
(138) A0138
Width: 219
Offset: -370
(130) A0130
Width: 1
Offset: -416
z (122) z
Width: 1
Offset: 11
r (114) r
Width: 357
Offset: -53
j (106) j
Width: 516
Offset: -62
b (98) b
Width: 443
Offset: -53
Z (90) Z
Width: 570
Offset: -50
R (82) R
Width: 592
Offset: 46
J (74) J
Width: 635
Offset: -379
(139) A0139
Width: 100
Offset: -95
/ (131) A0131
Width: 114
Offset: 110
{ (123) {
Width: 480
Offset: 47
s (115) s
Width: 301
Offset: -50
k (107) k
Width: 521
Offset: 30
c (99) c
Width: 535
Offset: 13
[(91) [
Width: 183
Offset: -50
S (83) S
Width: 566
Offset: -57
K (75) K
Width: 571
Offset: -507
Œ (140) A0140
Width: 100
Offset: -277
(132) A0132
Width: 1
Offset: 342
| (124) |
Width: 464
Offset: 73
t (116) t
Width: 313
Offset: -49
l (108) l
Width: 449
Offset: -48
d (100) d
Width: 565
Offset: 56
\ (92) \
Width: 366
Offset: 40
T (84) T
Width: 772
Offset: 15
L (76) L
Width: 436
141 Undefined
Offset: -399
(133) A0133
Width: 1
Offset: 139
} (125) }
Width: 480
Offset: -308
u (117) u
Width: 1
Offset: -50
m (109) m
Width: 509
Offset: -52
e (101) e
Width: 444
Offset: 44
] (93)]
Width: 203
Offset: -45
U (85) U
Width: 450
Offset: 14
M (77) M
Width: 485
142 Undefined
Offset: 46
† (134) A0134
Width: 298
Offset: -411
~ (126) ~
Width: 1
Offset: -81
v (118) v
Width: 202
Offset: -52
n (110) n
Width: 441
Offset: -49
f (102) f
Width: 532
Offset: -304
^ (94) ^
Width: 1
Offset: -50
V (86) V
Width: 405
Offset: -52
N (78) N
Width: 471
143 Undefined
Offset: -57
‡ (135) A0135
Width: 295
127 Undefined
Offset: -55
w (119) w
Width: 190
Offset: -50
o (111) o
Width: 524
Offset: -52
g (103) g
Width: 472
Offset: 12
_ (95) _
Width: 477
Offset: -49
W (87) W
Width: 523
Offset: 8
O (79) O
Width: 457


144 Undefined
(145) A0145 Width: 1
(146) A0146 Width: 1
147 Undefined
148 Undefined
149 Undefined
150 Undefined
151 Undefined
Offset: -51 (152) A0152
(153) A0153 Width: 493
Offset: -56 (154) A0154 Width: 543
Offset: -51 (155) A0155 Width: 200
Offset: -411 (156) A0156 Width: 1
157 Undefined
158 Undefined
Offset: -106 Ÿ (159) A0159 Width: 305
Offset: 0 (160) A0160
(161) A0161 Width: 1
Offset: -459 ¢ (162) A0162 Width: 1
Offset: -393 £ (163) A0163 Width: 1
Offset: -49 ¤ (164) A0164 Width: 456
Offset: -183 ¥ (165) A0165 Width: 120
Offset: -298 ¦ (166) A0166 Width: 1
Offset: -164 § (167) A0167 Width: 170
Offset: -52 (168) A0168
(169) A0169 Width: 1
(170) A0170 Width: 1
Offset: -658 « (171) A0171 Width: 1
Offset: -564 (172) A0172 Width: 1
Offset: -412 - (173) A0173 Width: 1
Offset: -51 ® (174) A0174 Width: 452
Offset: -51 (175) A0175 Width: 532
Offset: -51 (176) A0176
(177) A0177 Width: 441
Offset: -56 (178) A0178 Width: 664
Offset: -52 (179) A0179 Width: 662
Offset: -50 (180) A0180 Width: 569
Offset: -51 µ (181) A0181 Width: 697
Offset: -53 ¶ (182) A0182 Width: 665
Offset: -48 · (183) A0183 Width: 627
Offset: 3 (184) A0184
(185) A0185 Width: 435
Offset: 14 ° (186) A0186 Width: 765
Offset: 14 » (187) A0187 Width: 555
Offset: -49 ¼ (188) A0188 Width: 529
Offset: -50 ½ (189) A0189 Width: 513
Offset: -49 ¾ (190) A0190 Width: 776
191 Undefined
Offset: -50 À (192) A0192
(193) A0193 Width: 672
Offset: 6 Â (194) A0194 Width: 626
Offset: 17 Ã (195) A0195 Width: 546
Offset: 33 Ä (196) A0196 Width: 571
Offset: 40 Å (197) A0197 Width: 772
Offset: -52 Æ (198) A0198 Width: 451
Offset: -46 Ç (199) A0199 Width: 887
Offset: 12 È (200) A0200
(201) A0201 Width: 512
Offset: 7 Ê (202) A0202 Width: 434
Offset: -51 Ë (203) A0203 Width: 509
Offset: 4 Ì (204) A0204 Width: 603
Offset: -51 Í (205) A0205 Width: 576
Offset: -50 Î (206) A0206 Width: 499
Offset: -50 Ï (207) A0207 Width: 561
Offset: 43 Ð (208) A0208
(209) A0209 Width: 437
Offset: 46 Ò (210) A0210 Width: 400
Offset: 46 Ó (211) A0211 Width: 400
Offset: 46 Ô (212) A0212 Width: 220
Offset: 46 Õ (213) A0213 Width: 220
Offset: -546 Ö (214) A0214 Width: 1
Offset: -51 × (215) A0215 Width: 559

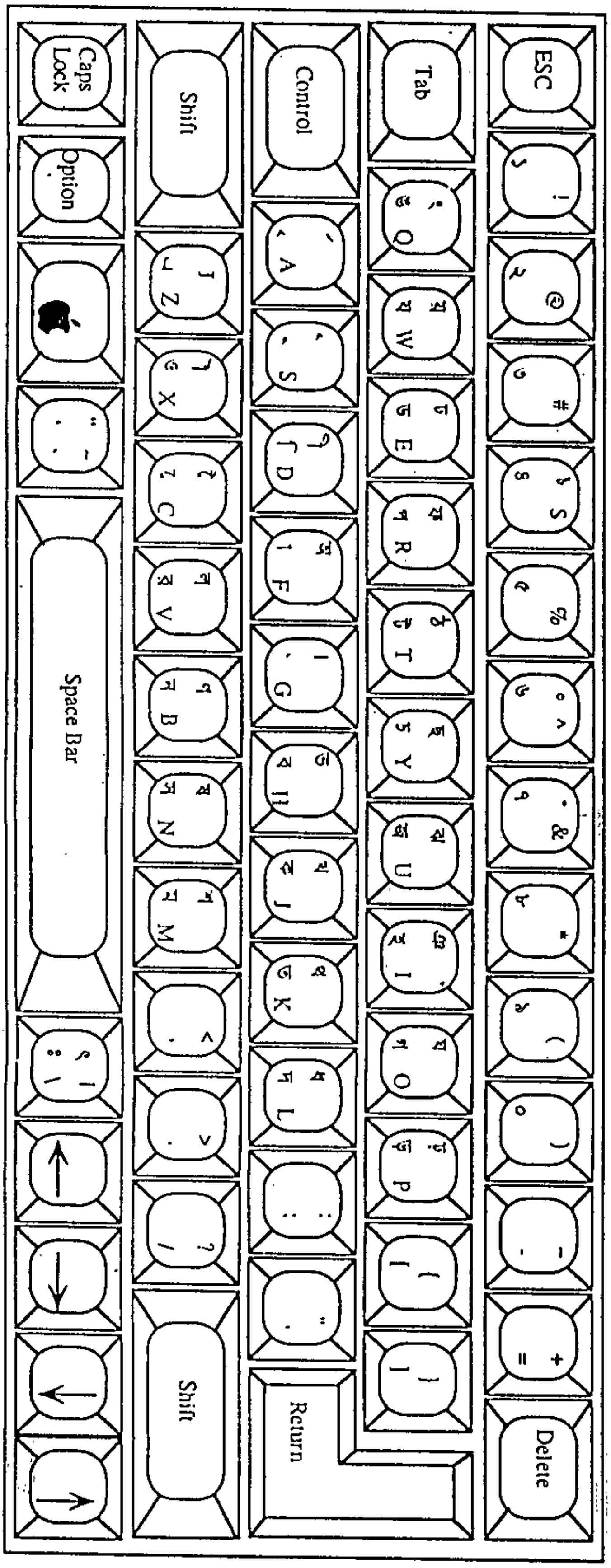


Offset: -49
Ø (216) A0216
(217) A0217
Width: 433
Offset: -54
Width: 464
Ú (218) A0218
Offset: -50
Width: 552
Û (219) A0219
Offset: -50
Width: 550
Ü (220) A0220
Offset: -51
Width: 536
Ý (221) A0221
Offset: -6
Width: 700
Þ (222) A0222
Offset: -2
Width: 490
ß (223) A0223
Offset: 30
à (224) A0224
Width: 650
á (225) A0225
Offset: -53
Width: 619
â (226) A0226
Offset: -47
Width: 558
ã (227) A0227
Offset: -50
Width: 555
ä (228) A0228
Offset: -50
Width: 524
å (229) A0229
Offset: -49
Width: 426
æ (230) A0230
Offset: -50
Width: 656
ç (231) A0231
Offset: -50
è (232) A0232
Offset: -33
Width: 535
÷ (233) A0233
Offset: -51
Width: 497
ê (234) A0234
Offset: -51
Width: 690
ë (235) A0235
Offset: -53
Width: 618
ì (236) A0236
Offset: -53
Width: 524
í (237) A0237
Offset: -56
Width: 630
î (238) A0238
Offset: -7
Width: 497
ï (239) A0239
Offset: -50
ð (240) A0240
Width: 702
ñ (241) A0241
Offset: -49
Width: 690
ò (242) A0242
Offset: -49
Width: 440
ó (243) A0243
Offset: -50
Width: 487
ô (244) A0244
Offset: -53
Width: 644
õ (245) A0245
Offset: -50
Width: 560
ö (246) A0246
Offset: -50
Width: 760
÷ (247) A0247
Offset: -53
ø (248) A0248
Width: 715
ù (249) A0249
Offset: -51
Width: 487
ú (250) A0250
Offset: -51
Width: 442
û (251) A0251
Offset: -52
Width: 522
ü (252) A0252
Offset: -53
Width: 595
ý (253) A0253
Offset: -52
Width: 639
þ (254) A0254
Offset: -52
Width: 364
ÿ (255) A0255
256 Undefined
257 Undefined
258 Undefined
259 Undefined
260 Undefined
261 Undefined
262 Undefined
263 Undefined
264 Undefined
265 Undefined
266 Undefined
267 Undefined
268 Undefined
269 Undefined
270 Undefined
271 Undefined
272 Undefined
273 Undefined
Offset: -90
Width: 199
(274)
Offset: -242
Width: 1
(275)
Offset: -52
Width: 270
(276)
Offset: -50
Width: 268
(277)
Offset: -559
Width: 1
(278)
Offset: -472
Width: 1
(279)
Offset: -557
(280)
Width: 425
(281)

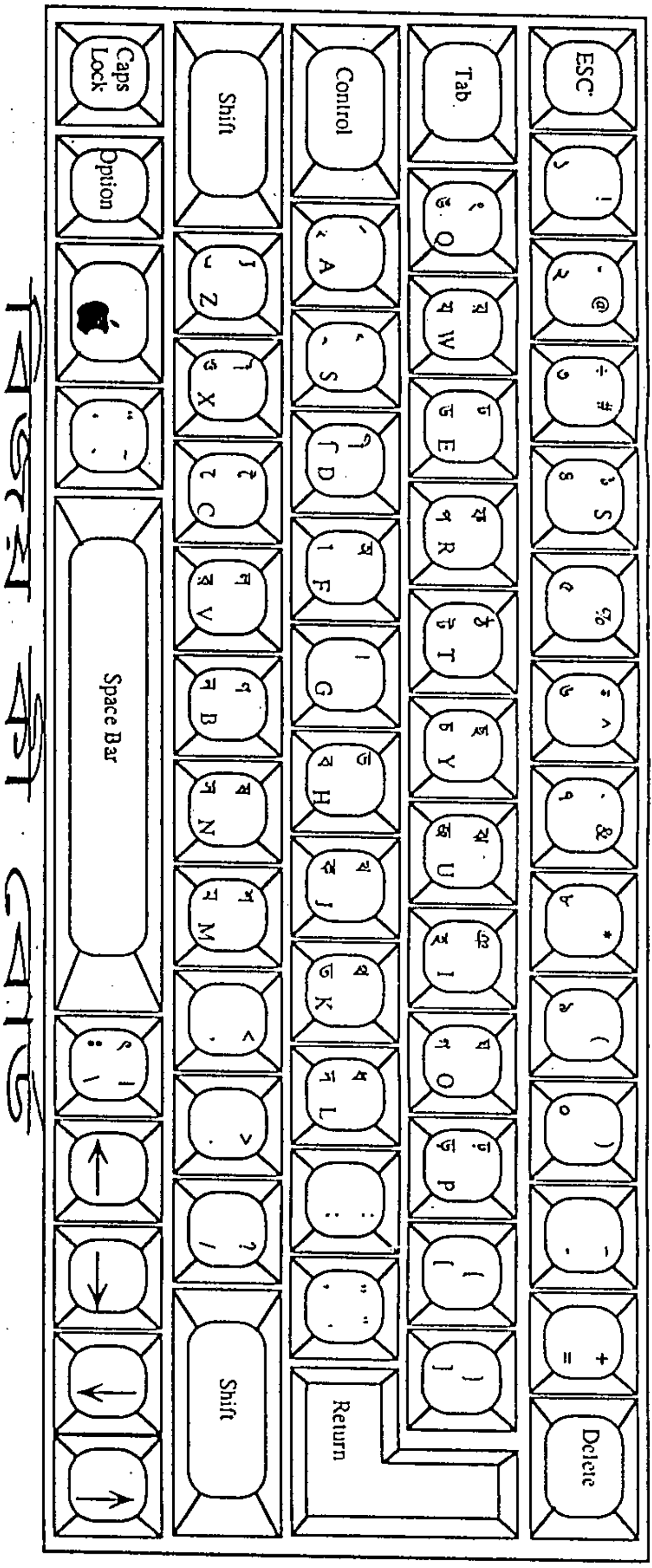
Appendix D

Bangla Bijoy Keyboard Layout

দ্বিতীয় সংস্করণ প্রথম সংস্করণ



বিজয় কী বোর্ড



Appendix E

Program of Bangla text to speech synthesizer

```
//Header file
# include <windows.h>
# include <windowsx.h>
# include <ddeml.h>
# include <string.h>
# include <mmsystem.h>
# include <stdlib.h>
# include <stdio.h>
# include <memory.h>
# include <math.h>
# include <io.h>

//Function prototype

int WINAPI WinMain (HINSTANCE,HINSTANCE,LPSTR,INT);
LRESULT CALLBACK SounderWndProc(HWND,UINT,UINT,LONG);
LRESULT CALLBACK GetMsgProc(int,WPARAM,LPARAM);
VOID KbdXlator (MSG *);
VOID PlayProc (MSG *);
VOID TextNormaliser1();
VOID TextNormaliser2();
VOID WaveFileMerge();

char szAppName[]="Sound";                    //Application name.
HHOOK hhook=NULL;                            //Handle of the hook function.


//Wave file assign.
char  * WaveFile[285] ={
                                    "c:\\wave\\w0001.wav",
                                    "c:\\wave\\w0002.wav",
                                    "c:\\wave\\w0003.wav",
                                    "c:\\wave\\w0004.wav",
                                    "c:\\wave\\w0005.wav",
                                    "c:\\wave\\w0006.wav",
                                    "c:\\wave\\w0007.wav",
                                    "c:\\wave\\w0008.wav",
                                    "c:\\wave\\w0009.wav",
                                    "c:\\wave\\w0010.wav",
                                    "c:\\wave\\w0011.wav",
                                    "c:\\wave\\w0012.wav",
                                    "c:\\wave\\w0013.wav",
                                    "c:\\wave\\w0014.wav",
                                    "c:\\wave\\w0015.wav",
                                    "c:\\wave\\w0016.wav",
                                    "c:\\wave\\w0017.wav",
                                    "c:\\wave\\w0018.wav",
                                    "c:\\wave\\w0019.wav",
                                    "c:\\wave\\w0020.wav",
"c:\\wave\\w0021.wav",
"c:\\wave\\w0022.wav",
"c:\\wave\\w0023.wav",
"c:\\wave\\w0015.wav",
"c:\\wave\\w0025.wav",
"c:\\wave\\w0026.wav",
"c:\\wave\\w0027.wav",
"c:\\wave\\w0028.wav",
"c:\\wave\\w0029.wav",
"c:\\wave\\w0008.wav",
"c:\\wave\\w0031.wav",
"c:\\wave\\w0032.wav",
"c:\\wave\\w0033.wav",
"c:\\wave\\w0033.wav",
"c:\\wave\\w0033.wav",
"c:\\wave\\w0036.wav",
"c:\\wave\\w0037.wav",
"c:\\wave\\w0038.wav",
"c:\\wave\\w0039.wav",
"c:\\wave\\w0040.wav",
"c:\\wave\\w0041.wav",
"c:\\wave\\w0042.wav",
"c:\\wave\\w0043.wav",
"c:\\wave\\w0044.wav",
"c:\\wave\\w0045.wav",
"c:\\wave\\w0046.wav",
"c:\\wave\\w0047.wav",
"c:\\wave\\w0048.wav",
"c:\\wave\\w0049.wav",
"c:\\wave\\w0050.wav",
"c:\\wave\\w0051.wav",
"c:\\wave\\w0052.wav",
"c:\\wave\\w0053.wav",
"c:\\wave\\w0054.wav",
"c:\\wave\\w0055.wav",
"c:\\wave\\w0056.wav",
"c:\\wave\\w0057.wav",
"c:\\wave\\w0058.wav",
"c:\\wave\\w0059.wav",
"c:\\wave\\w0060.wav",
"c:\\wave\\w0061.wav",
"c:\\wave\\w0062.wav",
"c:\\wave\\w0054.wav",
"c:\\wave\\w0064.wav",
"c:\\wave\\w0065.wav",
"c:\\wave\\w0066.wav",
"c:\\wave\\w0067.wav",
"c:\\wave\\w0068.wav",
"c:\\wave\\w0047.wav",
"c:\\wave\\w0070.wav",
"c:\\wave\\w0071.wav",
"c:\\wave\\w0072.wav",
"c:\\wave\\w0072.wav",
"c:\\wave\\w0072.wav",
"c:\\wave\\w0075.wav",
"c:\\wave\\w0076.wav",
"c:\\wave\\w0077.wav",
"c:\\wave\\w0078.wav",
"c:\\wave\\w0079.wav",
"c:\\wave\\w0080.wav",
"c:\\wave\\w0081.wav",
"c:\\wave\\w0082.wav",
"c:\\wave\\w0083.wav",
"c:\\wave\\w0084.wav",
"c:\\wave\\w0085.wav",
"c:\\wave\\w0086.wav",
"c:\\wave\\w0087.wav",
"c:\\wave\\w0088.wav",
"c:\\wave\\w0089.wav",
"c:\\wave\\w0090.wav",
"c:\\wave\\w0091.wav",
"c:\\wave\\w0092.wav",
"c:\\wave\\w0093.wav",
"c:\\wave\\w0094.wav",
"c:\\wave\\w0095.wav",
"c:\\wave\\w0096.wav",
"c:\\wave\\w0097.wav",
"c:\\wave\\w0098.wav",
"c:\\wave\\w0099.wav",
"c:\\wave\\w0100.wav",
"c:\\wave\\w0101.wav",
"c:\\wave\\w0093.wav",
"c:\\wave\\w0103.wav",
"c:\\wave\\w0104.wav",
"c:\\wave\\w0105.wav",
"c:\\wave\\w0106.wav",
"c:\\wave\\w0107.wav",
"c:\\wave\\w0086.wav",
"c:\\wave\\w0109.wav",
"c:\\wave\\w0110.wav",
"c:\\wave\\w0111.wav",
"c:\\wave\\w0111.wav",
"c:\\wave\\w0111.wav",
"c:\\wave\\w0114.wav",
"c:\\wave\\w0115.wav",
"c:\\wave\\w0116.wav",
"c:\\wave\\w0117.wav",
"c:\\wave\\w0118.wav",
"c:\\wave\\w0119.wav",
"c:\\wave\\w0120.wav",
"c:\\wave\\w0121.wav",
"c:\\wave\\w0122.wav",
"c:\\wave\\w0123.wav",
"c:\\wave\\w0124.wav",
"c:\\wave\\w0125.wav",
"c:\\wave\\w0126.wav",
"c:\\wave\\w0127.wav",
"c:\\wave\\w0128.wav",
"c:\\wave\\w0129.wav",
"c:\\wave\\w0130.wav",
"c:\\wave\\w0131.wav",
"c:\\wave\\w0132.wav",
"c:\\wave\\w0133.wav",
"c:\\wave\\w0134.wav",
"c:\\wave\\w0135.wav",
"c:\\wave\\w0136.wav",
"c:\\wave\\w0137.wav",
"c:\\wave\\w0138.wav",
"c:\\wave\\w0139.wav",
"c:\\wave\\w0140.wav",
"c:\\wave\\w0132.wav",
"c:\\wave\\w0142.wav",
"c:\\wave\\w0143.wav",
"c:\\wave\\w0144.wav",
"c:\\wave\\w0145.wav",
"c:\\wave\\w0146.wav",
"c:\\wave\\w0125.wav",
"c:\\wave\\w0148.wav",
"c:\\wave\\w0149.wav",
"c:\\wave\\w0150.wav",
"c:\\wave\\w0150.wav",
"c:\\wave\\w0150.wav",
"c:\\wave\\w0153.wav",
"c:\\wave\\w0154.wav",
"c:\\wave\\w0155.wav",
"c:\\wave\\w0156.wav",
"c:\\wave\\w0157.wav",
"c:\\wave\\w0158.wav",
"c:\\wave\\w0159.wav",
"c:\\wave\\w0160.wav",
"c:\\wave\\w0161.wav",
"c:\\wave\\w0162.wav",
"c:\\wave\\w0163.wav",
"c:\\wave\\w0164.wav",
"c:\\wave\\w0165.wav",
"c:\\wave\\w0166.wav",
"c:\\wave\\w0167.wav",
"c:\\wave\\w0168.wav",
"c:\\wave\\w0169.wav",
"c:\\wave\\w0170.wav",
"c:\\wave\\w0171.wav",
"c:\\wave\\w0172.wav",
"c:\\wave\\w0173.wav",
"c:\\wave\\w0174.wav",
"c:\\wave\\w0175.wav",
"c:\\wave\\w0176.wav",
"c:\\wave\\w0177.wav",
"c:\\wave\\w0178.wav",
"c:\\wave\\w0179.wav",
"c:\\wave\\w0171.wav",
"c:\\wave\\w0181.wav",
"c:\\wave\\w0182.wav",
"c:\\wave\\w0183.wav",
"c:\\wave\\w0184.wav",
"c:\\wave\\w0185.wav",
"c:\\wave\\w0164.wav",
"c:\\wave\\w0187.wav",
"c:\\wave\\w0188.wav",
"c:\\wave\\w0189.wav",
"c:\\wave\\w0189.wav",
"c:\\wave\\w0189.wav",
"c:\\wave\\w0192.wav",
"c:\\wave\\w0193.wav",
"c:\\wave\\w0194.wav",
"c:\\wave\\w0195.wav",
"c:\\wave\\w0118.wav",
"c:\\wave\\w0119.wav",
"c:\\wave\\w0120.wav",
"c:\\wave\\w0121.wav",
"c:\\wave\\w0122.wav",
"c:\\wave\\w0123.wav",
"c:\\wave\\w0124.wav",
"c:\\wave\\w0125.wav",
"c:\\wave\\w0126.wav",
"c:\\wave\\w0127.wav",
"c:\\wave\\w0128.wav",
"c:\\wave\\w0129.wav",
"c:\\wave\\w0130.wav",
"c:\\wave\\w0131.wav",
"c:\\wave\\w0132.wav",
"c:\\wave\\w0133.wav",
"c:\\wave\\w0134.wav",
"c:\\wave\\w0135.wav",
"c:\\wave\\w0136.wav",
"c:\\wave\\w0137.wav",
"c:\\wave\\w0138.wav",
"c:\\wave\\w0139.wav",
"c:\\wave\\w0140.wav",
"c:\\wave\\w0132.wav",
"c:\\wave\\w0142.wav",
"c:\\wave\\w0143.wav",
"c:\\wave\\w0144.wav",
"c:\\wave\\w0145.wav",
"c:\\wave\\w0146.wav",
"c:\\wave\\w0125.wav",
"c:\\wave\\w0148.wav",
"c:\\wave\\w0149.wav",
"c:\\wave\\w0150.wav",
"c:\\wave\\w0150.wav",
"c:\\wave\\w0150.wav",
"c:\\wave\\w0153.wav",
"c:\\wave\\w0154.wav",
"c:\\wave\\w0155.wav",
"c:\\wave\\w0156.wav",
"c:\\wave\\w0235.wav",
"c:\\wave\\w0236.wav",
"c:\\wave\\w0237.wav",
"c:\\wave\\w0238.wav",
"c:\\wave\\w0239.wav",
"c:\\wave\\w0240.wav",
"c:\\wave\\w0241.wav",
"c:\\wave\\w0242.wav",
"c:\\wave\\w0243.wav",
"c:\\wave\\w0244.wav",
"c:\\wave\\w0245.wav",
"c:\\wave\\w0246.wav",
"c:\\wave\\w0247.wav",
"c:\\wave\\w0248.wav",
"c:\\wave\\w0249.wav",
"c:\\wave\\w0250.wav",
"c:\\wave\\w0251.wav",
"c:\\wave\\w0252.wav",
                                        "c:\\wave\\w0253.wav",
                                        "c:\\wave\\w0254.wav",
                                        "c:\\wave\\w0255.wav",
                                        "c:\\wave\\w0256.wav",
                                        "c:\\wave\\w0257.wav",
                                        "c:\\wave\\w0249.wav",
                                        "c:\\wave\\w0259.wav",
                                        "c:\\wave\\w0260.wav",
                                        "c:\\wave\\w0261.wav",
                                        "c:\\wave\\w0262.wav",
                                        "c:\\wave\\w0263.wav",
                                        "c:\\wave\\w0242.wav",
                                        "c:\\wave\\w0265.wav",
                                        "c:\\wave\\w0266.wav",
                                        "c:\\wave\\w0267.wav",
                                        "c:\\wave\\w0267.wav",
                                        "c:\\wave\\w0267.wav",
                                        "c:\\wave\\w0270.wav",
                                        "c:\\wave\\w0271.wav",
                                        "c:\\wave\\w0272.wav",
                                        "c:\\wave\\w0273.wav",
                                        "c:\\wave\\w0274.wav",
                                        "c:\\wave\\w0275.wav",
                                        "c:\\wave\\w0276.wav",
                                        "c:\\wave\\w0277.wav",
                                        "c:\\wave\\w0278.wav",
                                        "c:\\wave\\w0279.wav",
                                        "c:\\wave\\w0280.wav",
                                        "c:\\wave\\w0281.wav",
                                        "c:\\wave\\w0282.wav",
                                        "c:\\wave\\w0283.wav",
                                        "c:\\wave\\w0284.wav",
                                        "c:\\wave\\w0285.wav",
};
char TempFile[]="c:\\wave\\temp.wav";       // Merged wave file.

char * wave[10];                            //Wave file assign for merging.
int p[10];                                  //Position of the wavefile.
char key[20];                               //Key pressed.
int k,w=0;                                  //Counter of the key pressed and wave file.

int WINAPI WinMain (HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpcmdLine, int cmdShow)
{
        MSG msg;                //Message structure
        WNDCLASS wc;            //Window class
        HWND hWnd;              //Handle of the window.

//Window create initialize
                wc.style =CS_HREDRAW | CS_VREDRAW;              //Class styles.
                wc.lpfnWndProc =(WNDPROC) SounderWndProc;       //Function called by message loop.
                wc.cbClsExtra = 0;                              //No extra class data.
                wc.cbWndExtra = 0;                              //No extra window data.
                wc.hInstance=hInstance;                         //Instance value of the application.
                wc.hIcon=LoadIcon(NULL,IDI_APPLICATION);        //Handle for an icon.
                wc.hCursor=LoadCursor(NULL,IDC_ARROW);          //Handle to a cursor type.
                wc.hbrBackground=GetStockObject(WHITE_BRUSH);//Handle of the background painted.
                wc.lpszMenuName=NULL;
                wc.lpszClassName=szAppName;
```

```
//Register the window class.
            if (!RegisterClass (&wc))
                            return FALSE;

//Window create
                   hWnd = CreateWindow(szAppName,              //Class name
                                      "Speech Synthesizer",    //Window name.
                                      WS_OVERLAPPEDWINDOW,     //Window style.
                                      CW_USEDEFAULT,           //X position on the screen.
                                      CW_USEDEFAULT,           //Y position on the screen.0
                                      CW_USEDEFAULT,           //Width of the window.
                                      CW_USEDEFAULT,           //Height of the window. 0
                                      NULL,                    //No parent window handle.
                                      NULL,                    //Uses window class menu.
                                      hInstance,               //Instance handle.
                                      NULL );                  //Not required.
//Make Window visible.
                   ShowWindow(hWnd, SW_SHOWMINIMIZED);
                   UpdateWindow(hWnd);                         //Send first WM_PAINT message.

//Message hook
                   hhook=SetWindowsHookEx (WH_GETMESSAGE,(HOOKPROC) GetMsgProc,NULL,0);

// Message loop.
          while (GetMessage(&msg, NULL, 0, 0))
          {
            TranslateMessage(&msg);                     //Translates the virtual key to ASCII.
            DispatchMessage (&msg);                     //Send message to SounderProc.
          }

          return (msg.wParam);
}

LRESULT CALLBACK GetMsgProc (int code,WPARAM wParam,LPARAM lParam)
{

MSG * mp;                              //Pointer of message structure.
         LRESULT lresult=0;
         mp=(MSG *)lParam;             //Pointer of message structure.

   if (code>=0)
         {
                   if (mp->message==WM_CHAR)
                   {
                            if (!(mp->lParam & 0x80000000L))
                            {
                                      KbdXlator (mp);          //Keyboard translator.
                                      PlayProc (mp);           //Play the wave file.
                            }
                   }
         }
         else
                   lresult = CallNextHookEx (hhook,code,wParam,lParam);


return ((LRESULT) lresult);
}
```

```
//Keyboard translator.
VOID KbdXlator (MSG * m)

{
static char cKeyBuf[256];

        key[k]=(char) LOWORD(m->wParam);        //Store the key.
        //Assign if the Shift key is depressed.
        GetKeyboardState(cKeyBuf);

switch (key[k])
{
case  8:  //Backspace pressed for Cancel.
        if (k>0) k--;
        break;

case  66:           //B
case  98:           //b

        if (cKeyBuf[VK_SHIFT] & 0x80)
                key[k]=m->wParam=89;   //Shift + B or b.
        else
                key[k]=m->wParam=98;   // B or b.
        k++;
        break;

case  67:           //C
case  99:           //c

        if (cKeyBuf[VK_SHIFT] & 0x80)
        {
                if (cKeyBuf[0x47] & 0x80)
                        key[k]=m->wParam=72;            //G or g, Shift + C or c.

        }
        else
        {
                if (cKeyBuf[0x47] & 0x80)
                        key[k]=m->wParam=71;            //G or g, C or c.
                else
                        key[k]=m->wParam=127;           // C or c.
        }
        k++;
        break;

case  68:                   //D
case 100:                   //d
        if (cKeyBuf[VK_SHIFT] & 0x80)
        {
                if (cKeyBuf[0x47] & 0x80)
                        key[k]=m->wParam=67;            //G or g, Shift + D or d.
                else
                        key[k]=m->wParam=120;           //Shift + D or d.
        }
        else
        {
                if (cKeyBuf[0x47] & 0x80)
                        key[k]=m->wParam=66;            //G or g, D or d.
                else
                        key[k]=m->wParam=119; // D or d.
            }
            k++;
            break;

    case 69:                    //E
    case 101:                   //e

            if (cKeyBuf[VK_SHIFT] & 0x80)
                    key[k]=m->wParam=88;            //Shift + E or e.
            else
                    key[k]=m->wParam=87;            // E or e.

            k++;
            break;

    case 70:                    //F
    case 102:                   //f

            if (cKeyBuf[VK_SHIFT] & 0x80)
                    key[k]=m->wParam=65;            //Shift + F or f.
            else
                    key[k]=m->wParam=118;           // F or f.

            k++;
            break;

    case 71:                    //G
    case 103:                   //g
            m->wParam=31;
            break;

    case 72:                    //H
    case 104:                   //h

            if (cKeyBuf[VK_SHIFT] & 0x80)
                    key[k]=m->wParam=102;           //Shift + H or h.
            else
                    key[k]=m->wParam=101;           // H or h.

            k++;
            break;

    case 73:                    //I
    case 105:                   //i

            if (cKeyBuf[VK_SHIFT] & 0x80)
                    key[k]=m->wParam=84;            //Shift + I or i.
            else
                    key[k]=m->wParam=110;           // I or i.

            k++;
            break;

    case 74:                    //J
    case 106:                   //j
```

```
          if (cKeyBuf[VK_SHIFT] & 0x80)
                    key[k]=m->wParam=76;  //Shift + J or j.
          else
                    key[k]=m->wParam=75;  // J or j.

          k++;
          break;

case  75:                     //K
case 107:                     //k

          if (cKeyBuf[VK_SHIFT] & 0x80)
                    key[k]=m->wParam=95;  //Shift + K or k.
          else
                    key[k]=m->wParam=90;  // K or k.

          k++;
          break;

case  76:                     //L
case 108:                     //l

          if (cKeyBuf[VK_SHIFT] & 0x80)
                    key[k]=m->wParam=97;  //Shift + L or l.
          else
                    key[k]=m->wParam=96;  // L or l.

          k++;
          break;

case  77:                     //M
case 109:                     //m

          if (cKeyBuf[VK_SHIFT] & 0x80)
                    key[k]=m->wParam=107; //Shift + M or m.
          else
                    key[k]=m->wParam=103; // M or m.

          k++;
          break;

case  78:                     //N
case 110:                     //n

          if (cKeyBuf[VK_SHIFT] & 0x80)
                    key[k]=m->wParam=108; //Shift + N or n.
          else
                    key[k]=m->wParam=109; // N or n.

          k++;
          break;

case  79:                     //O
case 111:                     //o

          if (cKeyBuf[VK_SHIFT] & 0x80)
                    key[k]=m->wParam=78;  //Shift + O or o.
          else
```

```
                key[k]=m->wParam=77;  // O or o.

            k++;
            break;

    case  80:                   //P
    case 112:                   //p

            if (cKeyBuf[VK_SHIFT] & 0x80)
                key[k]=m->wParam=112; //Shift + P or p.
            else
                key[k]=m->wParam=111; // P or p.

            k++;
            break;


    case  81:                   //Q
    case 113:                   //q

            key[k]=m->wParam=79;  //Q or q.
            k++;
            break;

    case  82:                   //R
    case 114:                   //r

            if (cKeyBuf[VK_SHIFT] & 0x80)
                key[k]=m->wParam=100;           //Shift + R or r.
            else
                key[k]=m->wParam=99;            // R or r.

            k++;
            break;

    case  83:                   //S
    case 115:                   //s

            if (cKeyBuf[VK_SHIFT] & 0x80)
            {
                if (cKeyBuf[0x47] & 0x80)
                    key[k]=m->wParam=69;            //G or g, Shift + S or s.

            }
            else
            {
                if (cKeyBuf[0x47] & 0x80)
                    key[k]=m->wParam=68;            //G or g, S or s.
                else
                    key[k]=m->wParam=121;           // S or s.
            }
            k++;
            break;

    case  84:                   //T
    case 116:                   //t

            if (cKeyBuf[VK_SHIFT] & 0x80)
```

```
                key[k]=m->wParam=86;            //Shift + T or t.
        else
                key[k]=m->wParam=85;            // T or t.

        k++;
        break;

case  85:               //U
case 117:               //u

        if (cKeyBuf[VK_SHIFT] & 0x80)
                key[k]=m->wParam=83;            //Shift + U or u.
        else
                key[k]=m->wParam=82;            // U or u.

        k++;
        break;


case  86:               //V
case 118:               //v

        if (cKeyBuf[VK_SHIFT] & 0x80)
                key[k]=m->wParam=106;           //Shift + V or v.
        else
                key[k]=m->wParam=105;           // V or v.

        k++;
        break;

case  87:               //W
case 119:               //w

        if (cKeyBuf[VK_SHIFT] & 0x80)
                key[k]=m->wParam=113;           //Shift + W or w.
        else
                key[k]=m->wParam=104;           // W or w.

        k++;
        break;

case  88:               //X
case 120:               //x

        if (cKeyBuf[VK_SHIFT] & 0x80)
        {
                if (cKeyBuf[0x47] & 0x80)
                        key[k]=m->wParam=74;  //G or g, Shift + X or x.

        }
        else
                key[k]=m->wParam=73;            // X or x.

        k++;
        break;

case  89:               //Y
case 121:               //y
```

```
        if (cKeyBuf[VK_SHIFT] & 0x80)
                key[k]=m->wParam=81;            //Shift + Y or y.
        else
                key[k]=m->wParam=80;            // Y or y.

        k++;
        break;

        }
}


VOID PlayProc (MSG * m)
{
int position;
        if ((m->wParam>=75) && (m->wParam<=113))        //Consonant
        {
                position=m->wParam+159;
                sndPlaySound(WaveFile[position],SND_ASYNC|SND_NODEFAULT);
        }
        else
        {
                if (m->wParam==118)                 //Vowel auxiliary: AA-kar
                        sndPlaySound(WaveFile[283],SND_ASYNC|SND_NODEFAULT);
                else
                {
                        if ((m->wParam>=119) && (m->wParam<=121)) //Vowel auxiliary
                        {
                                position=m->wParam+155;
                                sndPlaySound(WaveFile[position],SND_ASYNC|SND_NODEFAULT);
                        }
                        else
                        {
                                if (m->wParam==127)      //Vowel auxiliary:A-kar
                                {
                                        sndPlaySound(WaveFile[279],SND_ASYNC|SND_NODEFAULT);
                                        m->wParam=135;
                                }
                                else
                                {
                                        if ((m->wParam>=65) && (m->wParam<=74))        //Vowel
                                        {
                                                position=m->wParam+208;
                                sndPlaySound(WaveFile[position],SND_ASYNC|SND_NODEFAULT);
                                        }
                                        else
                                        {
                                                if (m->wParam==32)        //If spacebar pressed.
                                                {
                                                    if (k>0) TextNormaliser1();//Define the wave file.

                                                                if (w>0)
                                                                {
                                                        TextNormaliser2();//Last wave file define.
                                                        WaveFileMerge();    //Wave file merged.
                                                                }
                                        //Play the merged wave file.
                                        sndPlaySound(TempFile,SND_ASYNC|SND_NODEFAULT);
                                                w=0;
                                                k=0;
                                                }
                                        }
                                }
                        }
                }
        }
}

VOID TextNormaliser1()
{
int step=2;                     //Step of key selection.
int ky;                         //Counter of key pressed.
int consonant=39;               //Total no of consonant.

 for (ky=0;ky<k;ky+=step)
 {
        if ((key[ky]>=75) && (key[ky]<=113))                         //Consonant
        {
            if (((ky+step)<k) && ((key[ky+step])==118))              //AA-kar
            {
              if  ((ky>(step-1)) && ((key[ky-step])==127))           //A-kar
             {
                w--;
                p[w]=key[ky]-75;
                wave[w]=WaveFile[p[w]];  //Consonant with O-kar
                ky+=step;
               }
               else
              {
                p[w]=key[ky]-75+consonant;
                wave[w]=WaveFile[p[w]];  //Consonant with AA-kar
                ky+=step;
               }
             }
             else
            {
              if (((ky+step)<k) && ((key[ky+step])==121))//U-kar
              {
                  p[w]=key[ky]-75+consonant*2;
                  wave[w]=WaveFile[p[w]];  //Consonant with U-kar
                  ky+=step;
               }
               else
                {
                        if (((ky+step)<k) && ((key[ky+step])==120))//EE-kar
                        {
                          p[w]=key[ky]-75+consonant*3;
                          wave[w]=WaveFile[p[w]];  //Consonant with EE-kar
                          ky+=step;
                        }
                        else
                        {
                                if ((ky>(step-1)) && ((key[ky-step])==127))//A-kar
                                {
```

```
                  w--;
                  p[w]=key[ky]-75+consonant*4;
                    wave[w]=WaveFile[p[w]];  //Consonant with A-kar
                }
            else
            {
                if ((ky>(step-1)) && ((key[ky-step])==119)) //E-kar
                {
                    w--;
                    p[w]=key[ky]-75+consonant*5;
                    wave[w]=WaveFile[p[w]];    //Consonant with E-kar
                }
                else
                {
                      p[w]=key[ky]-75+consonant*6;
                       wave[w]=WaveFile[p[w]];    //Consonant
                }
            }
        }
      }
   }
}
else
{
    if ((key[ky]>=65) && (key[ky]<=74))                    //Vowel
    {
        p[w]=key[ky]+208;
        wave[w]=WaveFile[p[w]];

        if (key[ky]==65)
        {
            if (((ky+step)<k) && ((key[ky+step])==118)) //AA-kar
            {
                p[w]=283;
                wave[w]=WaveFile[283]; //AA
                ky+=step;
            }

        }

    }
    else
    {
        if (key[ky]==118)                          //AA-kar
        {
            p[w]=283;
            wave[w]=WaveFile[283]; //AA
        }
        else
        {
            if ((key[ky]>=119) && (key[ky]<=121)) //Vowel auxiliary
            {
                p[w]=key[ky]+155;
               wave[w]=WaveFile[p[w]];
            }
            else
            {
                if (key[ky]==127)            //A-kar
```

```
                                        {
                                                p[w]=279;
                                                wave[w]=WaveFile[279];
                                        }
                                  }
                             }
                        }
                   }
              w++;
             }
         }

         /*VOID TextNormaliser2()
         {
                 int position;
                 int set;

                 if (w>1)
                 {
                         if ((p[w-1]>=234) && (p[w-1]<=272))
                         {

                                         position=50+p[w-1]+set
                                         w--;
                                         wave[w-1]=WaveFile[position];          //der
                         }
                 }

         }*/


         VOID TextNormaliser2()
         {
                 if (w>1)
                 {

                         if ((wave[w-2]==WaveFile[177]) && (wave[w-1]==WaveFile[264])) //de + ra
                         {
                                 w–;
                                 wave[w-1]=WaveFile[284];               //der
                         }
                 }

         }



         VOID WaveFileMerge()
         {
         HANDLE hOutFile, hInFile[10];
         BOOL bRESULT;
         DWORD dwPointer;

         //Initialize the number of bytes read.
         DWORD test1[10]={0,0,0,0,0,0,0,0,0,0};
         DWORD test2=0;
```

```
DWORD test3[10]={0,0,0,0,0,0,0,0,0,0};
DWORD test4[10]={0,0,0,0,0,0,0,0,0,0};

unsigned char h[44];              //Header Buffer.
int x;                            //Wave File Counter.
long dsize[10];                   //Data Size.
long tsize[10];                   //Total Data Size.
long size;                                // Do
long cksize;                      //Chunk Size.
char buff[100000];                //Temporary Buffer.

remove(TempFile);                         //Delete the temporary file.

//Temp. File Create.
hOutFile=CreateFile(TempFile,GENERIC_WRITE,1,NULL,CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL)
                                                                                           ;
if (hOutFile==INVALID_HANDLE_VALUE) return;

        for(x=0;x<w;x++)
        {
                //Wave File open.
                hInFile[x]=CreateFile(wave[x],GENERIC_READ,1,NULL,OPEN_EXISTING,
                                                        FILE_ATTRIBUTE_NORMAL,NULL);
                        if (hInFile[x]==INVALID_HANDLE_VALUE) return;

                //Pointer Set in the wave file.
                dwPointer=SetFilePointer(hInFile[x],0,NULL,FILE_BEGIN);
                        if (dwPointer==0xFFFFFFFF) return;

                //Header Read from the wave file..
                bRESULT=ReadFile(hInFile[x],h,44,&test1[x],NULL);
                        if (!bRESULT) return;

//File size calculation
                dsize[x]=*((long *)(h+40));
                tsize[x]=dsize[x];
                if (x>0)
                        {
                                cksize=cksize+tsize[x];
                                tsize[x]=tsize[x-1]+tsize[x];
                        }
                else
                                cksize=*((long *)(h+4));

                size=tsize[x];
        }

//Set the file size in header.
*((long *)(h+4))=cksize;
*((long *)(h+40))=size;

//Pointer Set in the temp.file.
dwPointer=SetFilePointer(hOutFile,0,NULL,FILE_BEGIN);
        if (dwPointer==0xFFFFFFFF) return;

//Header Written in the temp.file.
bRESULT=WriteFile(hOutFile,h,44,&test2,NULL);
        if (!bRESULT) return;
```

```
    //Set the pointer in the temp.file for data written.
    dwPointer=SetFilePointer(hOutFile,44,NULL,FILE_BEGIN);
            if (dwPointer==0xFFFFFFFF) return;

    //Data add
            for(x=0;x<w;x++)
            {
                    //Set the pointer in the wave file for data read.
                    dwPointer=SetFilePointer(hInFile[x],44,NULL,FILE_BEGIN);
                            if (dwPointer==0xFFFFFFFF) return;

                    //Read the data from wave file.
                    bRESULT=ReadFile(hInFile[x],buff,dsize[x],&test3[x],NULL);
                            if (!bRESULT) break;

                    //Write the data in temp. file.
                    bRESULT=WriteFile(hOutFile,buff,dsize[x],&test4[x],NULL);
                            if (!bRESULT) return;

                    //File Close.
                    if (hInFile[x] !=INVALID_HANDLE_VALUE)
                    {
                            CloseHandle (hInFile[x]);
                            hInFile[x]=INVALID_HANDLE_VALUE;
                    }
            }

            if (hOutFile !=INVALID_HANDLE_VALUE)
            {
                    CloseHandle (hOutFile);
                    hOutFile=INVALID_HANDLE_VALUE;
            }
}

//Function of the Message Loop.
LRESULT CALLBACK SounderWndProc(HWND hWnd, UINT msg,UINT wParam,
                                                                LONG lParam)
{
            switch (msg)

                    {

                    case WM_DESTROY:     //Stop Application.
                            UnhookWindowsHookEx(hhook);
                            hhook=NULL;
                            PostQuitMessage(0);
                            break;

            case WM_CLOSE:      //Windows close & Stop Application.
                            DestroyWindow(hWnd);
                            break;

            default :                   //Default windows message processing.
                            return DefWindowProc(hWnd, msg, wParam, lParam);

                    }
        return (0L);
}
```