10-29-2003

FPGA-based Implementation of Concatenative Speech Synthesis Algorithm

Praveen Kumar Bamini
University of South Florida

FPGA-based Implementation of Concatenative Speech Synthesis Algorithm

by

Praveen Kumar Bamini

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Engineering
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Srinivas Katkoori, Ph.D.
Murali Varanasi, Ph.D.
Sanjukta Bhanja, Ph.D.

Date of Approval:
October 29, 2003

Keywords: Processing, VHDL, Verification, Sequential Circuits, Waveforms

DEDICATION

To all my family members and friends who always believed me.

ACKNOWLEDGEMENTS

My sincere thanks to Dr. Srinivas Katkoori, who has provided me an opportunity to work with him as well as for the encouragement and necessary support he has rendered thorughout my masters program. I would also like to thank Dr. Varanasi and Dr. Sanjuktha Bhanja for being on my committee.

I also acknowledge my friends Ramnag, Sarath, Sudheer, Lolla, Sunil, Venu, Jyothi, Babu, Hariharan for their support. I really appreciate the support of my friends in the VCAPP group especially Saraju Prasad Mohanty, who constantly helped me during the course of this research work. I also thank Daniel Prieto, for the technical help he provided during my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FPGA-BASED IMPLEMENTATION OF CONCATENATIVE SPEECH SYNTHESIS ALGORITHM

Praveen Kumar Bamini

ABSTRACT

The main aim of a text-to-speech synthesis system is to convert ordinary text into an acoustic signal that is indistinguishable from human speech. This thesis presents an architecture to implement a concatenative speech synthesis algorithm targeted to FPGAs. Many current text-to-speech systems are based on the concatenation of acoustic units of recorded speech. Current concatenative speech synthesizers are capable of producing highly intelligible speech. However, the quality of speech often suffers from discontinuities between the acoustic units, due to contextual differences. This is the easiest method to produce synthetic speech. It concatenates prerecorded acoustic elements and forms a continuous speech element. The software implementation of the algorithm is performed in C whereas the hardware implementation is done in structural VHDL. A database of acoustic elements is formed first with recording sounds for different phones. The architecture is designed to concatenate acoustic elements corresponding to the phones that form the target word. Target word corresponds to the word that has to be synthesized. This architecture doesn't address the form discontinuities between the acoustic elements as its ultimate goal is the synthesis of speech. The Hardware implementation is verified on a Virtex (v800hq240-4) FPGA device.

CHAPTER 1

INTRODUCTION

1.1 Introduction to speech synthesis

Speech is the primary means of communication between people. Speech synthesis and recognition often called as voice input and output has a very special place in the man-machine communication world. The keyboard, touch-sensitive screen, the mouse, the joystick, image processing devices, etc., are well established media of communication with the computer. As language is most natural means of communication, speech synthesis and recognition are best means for communicating with the computer [1].

A sound wave is caused by the vibration of a molecule. Speech is a continuously varying sound wave which links speaker to listener. Sound travels in a medium such as air, water, glass, wood, etc., the most usual medium is air. When a sound is made, the molecules of air nearest to our mouths are disturbed. These molecules being disturbed in a manner which sets them oscillating about their point of rest. A chain reaction will begin as each molecule will propagate the above mentioned effect when it collides with other molecules in its surroundings. This chain reaction will eventually dissipate some distance from the speaker. The distance traveled solely depends on the energy initially imparted by vocal chords. The maximum distance a vibrating molecule moves from its point of rest, is known as the *amplitude* of vibration. In one complete cycle of motion a molecule starts from its point of rest, goes to maximum displacement at one side, then the other side, and finally returns to point of rest. The time taken for one complete cycle is called as *period*. The number of times this complete cycle occurs in one second is termed as *frequency*.

Sounds which possess same periods in successive cycles are called as *periodic sounds* and those that do not are *aperiodic sounds*. A common example for a periodic sound is the note produced by striking a tuning fork, where sound stays fairly constant throughout its span. The shape of a sound

wave can be influenced by the presence of harmonics. Fig. 1.1 represents a sinusoidal periodic sound waveform from point A to point B.

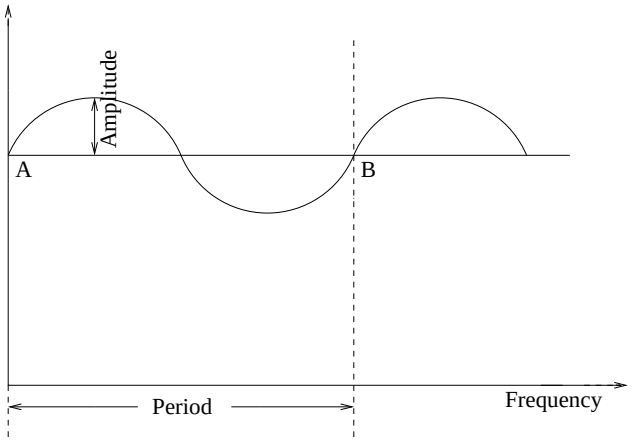

Figure 1.1. Representation of a sinusoidal periodic sound wave

Sound is a common form of multimedia used for many day to day applications such as games, presentations, and even operating system feedback provide audio source to a computer user. Analog to Digital (A/D) converters and Digital to Analog (D/A) converters are often used to interface such audio source to a computer speech synthesis. Automatic generation of speech waveforms, has been under development for several decades. Intelligibility, naturalness and variability are the three criteria used to distinguish human speech and synthetic speech. Intelligibility is the measure of understandability of speech. Naturalness is the measure of human factor in the speech, that is how close is the synthetic speech to human speech [2]. Recent progress in speech synthesis has produced synthesizers with very high intelligibility but the sound quality and naturalness still remain a major problem to be addressed. However, for several applications such as multimedia and telecommunications the quality of present synthesizers reached an adequate level.

Text-to-speech is a process through which text is rendered as digital audio and then “spoken”. Continuous speech is a set of complicated audio signals. This complexity of audio signals makes it difficult to produce them artificially. Speech signals are usually considered as voiced or unvoiced, but in some cases they are something between these two.

1.2 Applications of text-to-speech synthesis

- Text-to-speech transforms linguistic information stored as text into speech[3]. It is widely used in audio reading devices for the disabled such as blind people.
- Text-to-speech is most useful for situations when pre recording is not practical.
- Text-to-speech can be used to read dynamic text.
- Text-to-speech is useful for proofreading. Audible proofreading of text and numbers helps the user catch typing errors missed by visual proofreading.
- Text-to-speech is useful for phrases that would occupy too much storage space if they were prerecorded in digital-audio format.
- Text-to-speech can be used for informational messages. For example, to inform the user that a print job is complete, an application could say “Printing complete” rather than displaying a message and requiring the user to acknowledge it.
- Text-to-speech can provide audible feedback when visual feedback is inadequate or impossible. For example, the user’s eyes might be busy with another task, such as transcribing data from a paper document.
- Users that have low vision may rely on text-to-speech as their sole means of feedback from the computer.
- It’s always possible to use concatenated word/phrase text-to-speech to replace recorded sentences. The application designer can easily pass the desired sentence strings to the text-to-speech engine.

1.3 Attributes of text-to-speech synthesis

Output quality is one basic and most important attribute of any speech synthesis mechanism. It is often a common occurrence that a single system can sound good on one sentence and pretty bad on the next sentence. This makes it essential to consider the quality of the best sentences and the

percentage of sentences for which maximum quality is achieved. Here we consider four different families of speech generation mechanisms [4].

- *Limited-domain waveform concatenation.* This mechanism can generate very high quality speech with only a small number of recorded segments. This method is used in most interactive voice response systems.
- *Concatenative synthesis with no waveform modification.* This approach can generate good quality on a large set of sentences, but the quality can be bad, when poor concatenation takes place.
- *Concatenative synthesis with waveform modification.* This approach has more flexibility and less mediocrity as waveforms can be modified to allow better prosody match. The downside is that waveform modification degrades the overall quality of output speech.
- *Rule-based synthesis.* This mechanism provides uniform sound across different sentences, but the quality when compared to other mechanisms, is poor.

The characteristics of above methods are illustrated in the Figure 1.2.

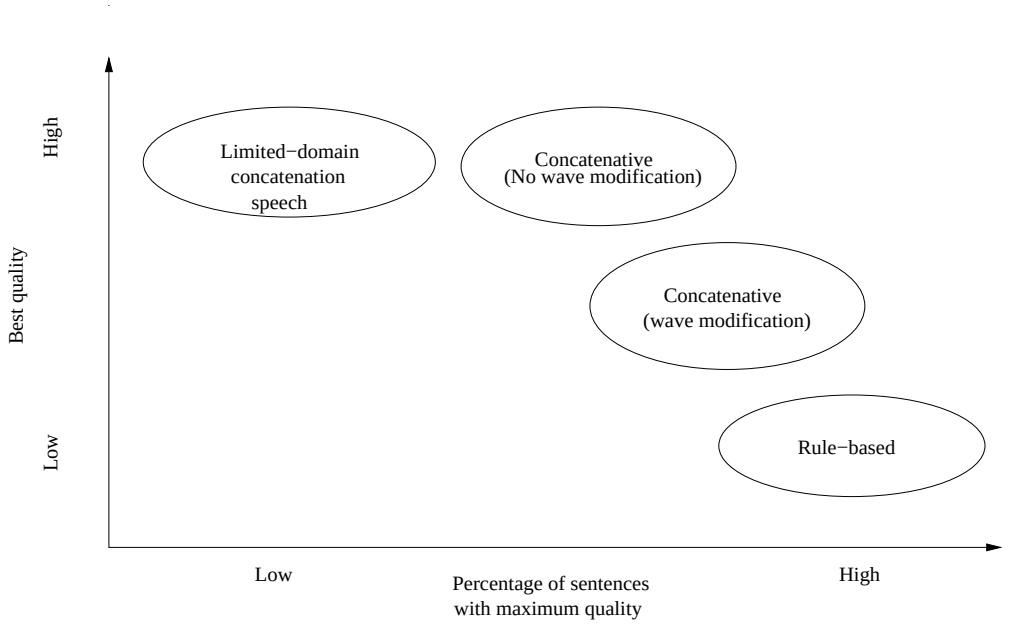

Figure 1.2. Quality and task independence in speech synthesis approaches

According to the speech generation model used, speech synthesis can be classified into three categories [4].

- Articulatory synthesis
- Formant synthesis
- Concatenative synthesis

Based on the degree of manual intervention in the design, speech synthesis can be classified into two categories [4].

- Synthesis by rule
- Data-driven synthesis

Articulatory synthesis and formant synthesis fall in the synthesis by rule category where as concatenative synthesis comes under data-driven synthesis category. *Formant synthesis* uses a source-filter model, where the filter is characterized by slowly varying formant frequencies of the vocal tract. *Articulatory synthesis* system makes use of a physical speech production model that includes all the articulators and their mechanical motions[1, 4]. It produces high quality speech as it tries to model human vocal cords. Truly speaking, *concatenative synthesis* is not exactly a speech synthesis mechanism, but it is one of the most commonly used text-to-speech systems around. This thesis work mainly deals with concatenative speech synthesis.

Basically there are two techniques that most speech synthesizers use for speech synthesis.

- Time domain technique
- Frequency domain technique.

In case of time domain technique the stored data represent a compressed waveform as a function of time. The main aim of this technique is to produce a waveform which may not resemble the original signal spectrographically, but it will be perceived to be same by the listener. This technique implementation compared to the frequency domain needs relatively simple equipment such as D/A converter and post sampling filter when Pulse Code Modulation (P.C.M.) is used. The quality of speech can be controlled by the sampling rate. Frequency domain synthesis is based on the modeling of human speech and it requires more circuitry [4].

The text-to-speech (TTS) synthesis by rule procedure involves two main phases.

- Text analysis phase
- Speech generation phase

The first one is text analysis, where the input text is transcribed into a phonetic or some other linguistic representation, and the second one is the generation of speech waveforms, where the acoustic output is produced from this phonetic and prosodic information. These two phases are usually called as high-level synthesis and low-level synthesis. The input text might be for example data from a word processor, standard ASCII from e-mail, a mobile text-message, or scanned text from a newspaper. The character string is then preprocessed and analyzed into phonetic representation which is usually a string of phonemes with some additional information for correct intonation, duration, and stress. Speech sound is finally generated with the low-level synthesizer by the information from high-level one.

1.4 Concatenative speech synthesis

The simplest way of producing synthetic speech is to play long prerecorded samples of natural speech, such as single words or sentences. In a concatenated word engine, the application designer provides recordings for phrases and individual words. The engine pastes the recordings together to speak out a sentence or phrase. If you use voice-mail then you've heard one of these engines speaking, "[You have] [three] [new messages]". The engine has recordings for "You have", all of the digits, and "new messages". A text-to-speech engine that uses synthesis generates sounds similar to those created by the human vocal cords and applies various filters to simulate throat length, mouth cavity, lip shape, and tongue position. This concatenation method provides high quality and naturalness. However, the downside is that it usually requires more memory capacity than other methods. The method is very suitable for some announcing and information systems. However, it is quite clear that we cannot create a database of all words and common names in the world. It is maybe even inappropriate to call this speech synthesis because it contains only recordings. Thus, for unrestricted speech synthesis (text-to-speech) we have to use shorter pieces of speech signal, such as syllables, phonemes, diphones, or even shorter segments.

A text-to-speech engine that uses subword concatenation links short digital-audio segments together and performs intersegment smoothing to produce a continuous sound. Subword segments

are acquired by recording many hours of a human voice and painstakingly identifying the beginning and ending of phonemes. Although this technique can produce a more realistic voice, it is laborious to create a new voice. This approach requires neither rules nor manual tuning. An utterance is synthesized by concatenating together several speech segments [5, 6].

When two speech segments, not adjacent to each other are concatenated there can be spectral discontinuities. Spectral and prosodic discontinuities result due to unmatching of formants and pitch at concatenation point, respectively [6, 4]. This may be due to lack of identical context for the units, intra-speaker variation, or errors in segmentation. This problem can be reduced by techniques such as spectral smoothing, manual adjustment of unit boundaries, use of longer units to reduce the e number of concatenations, manual reselection of units, etc. The downside of using spectral smoothing is that it decreases the naturalness of the resulting synthesized speech [6]. This problem is clearly illustrated in Figures 1.3 and 1.4.

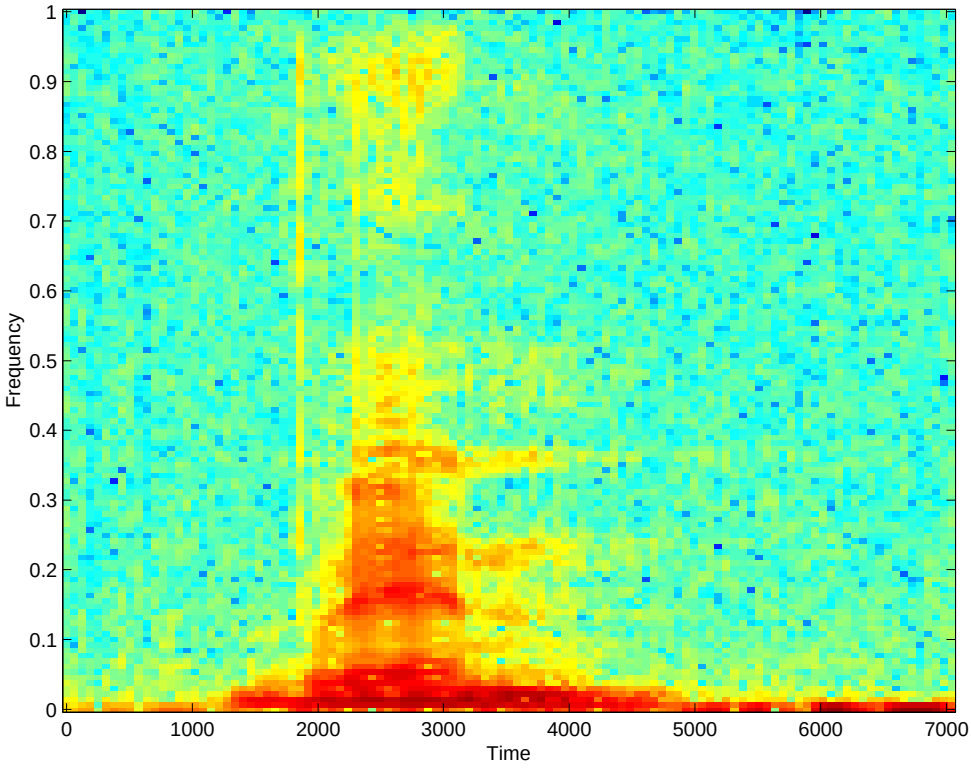

Figure 1.3. Spectrogram of a word ‘when’

Figure 1.3 is the spectrogram of the sample word ‘when’, when it is spoken completely as a whole. There we can observe a continuous spectral arrangement.

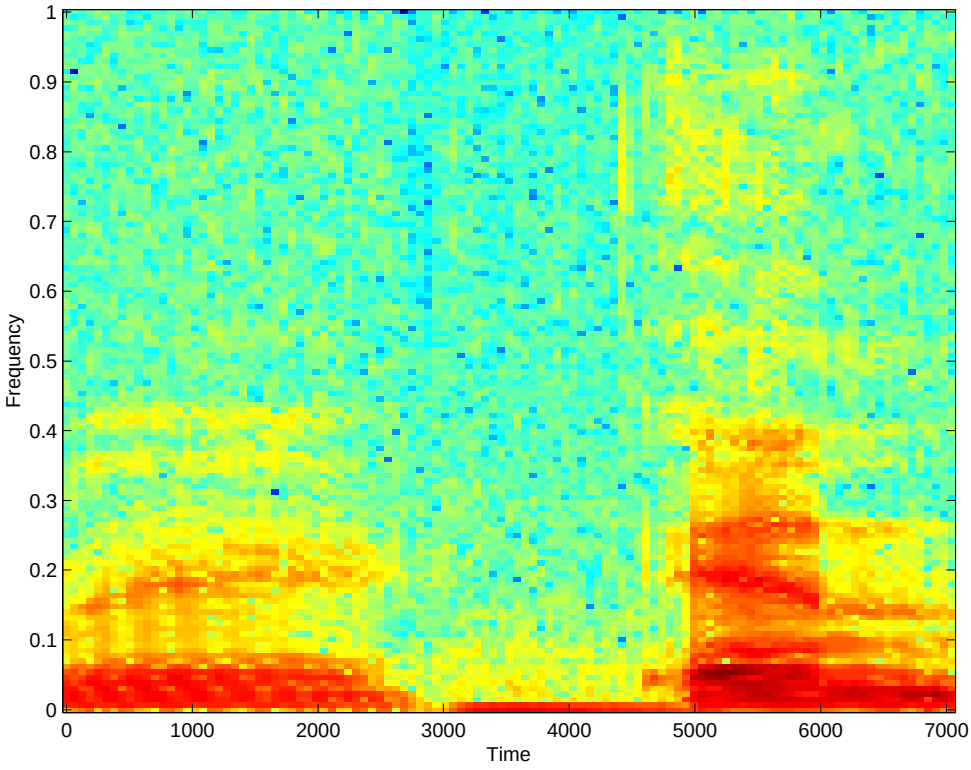

Figure 1.4. Spectrogram of a concatenated word ‘when’ formed by concatenating segments ‘whe’ and ‘en’

Figure 1.4 represents the spectrogram of the concatenated word ‘when’. In the instance where the word ’when’ is concatenated using two audio segments ‘whe’ and ‘en’, discontinuous spectral lines can be observed. This is because of the time gap between the boundaries of formants ‘whe’ and ‘en’.

Synthesized text-to-speech inevitably sounds unnatural and weird. However, it is very good for character voices that are supposed to be robots, aliens, or maybe even foreigners. For an application which cannot afford to have recordings of all the possible dialogs or if the dialogs cannot be recorded ahead of time, then text-to-speech remains the only alternative [7].

1.5 Summary

This thesis presents an architecture and implementation of a concatenative speech synthesis algorithm on a FPGA device and its performance is measured. Chapters 1 and 2 discuss about the introduction to speech synthesis, literature concerning to speech synthesis, concatenative speech synthesis systems, techniques to improve the quality of speech, and techniques to obtain phase coherence in concatenative speech synthesis.

Chapter 3 extensively discusses about the algorithm used as part of this thesis work. This chapter also describes the target architecture developed to implement the proposed algorithm. It also contains the top level digital design as well as the modular VHDL design for the synthesis process. Chapter 4 discusses about the implementation of the synthesis algorithm and the experimental results. Chapter 5 discusses conclusions and suggestions.

CHAPTER 2

RELATED LITERATURE

2.1 Review of prior work

In this section a brief overview of existing literature and research work in speech synthesis is presented. As the central topic is based on concatenative speech synthesis, the literature is limited to it.

Text to speech synthesis is a process, where input text after natural language processing and digital signal processing is converted to output speech. The general text-to-speech synthesis system is shown as a block diagram in the figure 2.1.

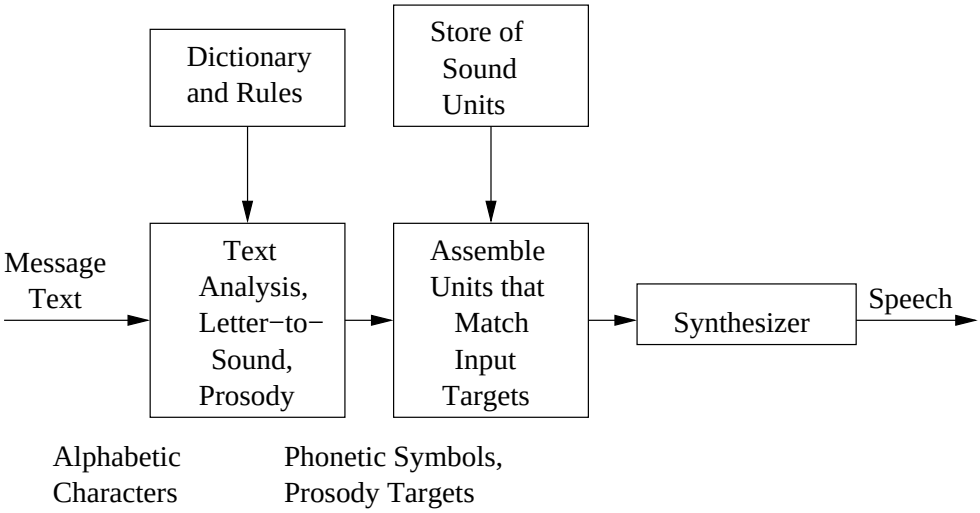

Figure 2.1. Block diagram of the text-to-speech synthesis process

The first block represents a text analysis module that takes text and converts it into a set of phonetic symbols and prosody targets. The input text is first analyzed and then transcribed. The transcribed text goes through a syntactic parser which with the help of a pronunciation dictionary generates a string of phonemes. With available transcribed text, syntactic, and phonological information prosody module generates targets. The second block searches for the input targets in the

store of sound units and assemble the units that match input targets. Then the assembled units are fed to a synthesizer that generates the speech waveform [8, 9].

There are three important characteristics that define a good a speech synthesis system

- Good concatenation method.
- Good database of synthesis units.
- Ability to produce natural prosody across the concatenated units.

An important factor in concatenative speech synthesis is the availability of efficient search techniques, which enable us to search very large number of available sound segments in real time for the optimal output word [10]. In concatenative speech synthesis there exists problems such as storage problem, intelligibility, etc.

Becker and Poza [11] proposed a model of concatenative speech synthesis system called *Dyad synthesis system*. A dyad is defined as the representation of a last half of one sound and first half of the next sound. Based on some theoretical grounds, Wang and Peterson [12] calculated that an infinitely large set of messages could be produced using approximately 800 dyads for monotonic speech and 8500 dyads for intonated speech. The intonation of the speech can be improved by enlarging the inventory of the audio segments. By this dyad synthesis system any English message can be constructed from a stored inventory of audio segments that can be stored in the available RAMs in the market. There are many approaches that have been proposed to improve speech synthesis. These are described in detail in Section 2.2.

Bulut, Narayanan, and Srydal [2] has carried out an experiment of synthesizing speech consisting of various emotions. They tried to synthesize four emotional states anger, happiness, sadness, and neutral using a concatenative speech synthesizer. As part of that experiment they constructed a search database of sounds with the separately recorded inventories of different emotions such as anger, happiness, sadness and neutral. In their results they classified anger as inventory dominant and sadness and neutral as prosody dominant whereas the results were not conclusive enough for happiness. They conclude that various combinations of inventory and prosody of basic emotions may synthesize different emotions.

2.2 Review of proposed approaches to improve concatenation

The common problems in concatenative speech synthesis are

- Occurrence of unseen text. It is a commonplace occurrence in text-to-speech synthesis, as it is practically not possible to obtain acoustic representations for all the possible contexts that could occur in speech because there exists a vast combinations of text, memory constraints etc.

- Spectral discontinuity at concatenation points. This is explained in detail in the previous chapter.

Based on Hidden Markov Models, prosodic parameters, and contextual label, Low and Vaseghi [13] have proposed a method for selecting speech segments for the database. They also proposed a method for synthesizing an unseen text from the existing speech units. Combination of above mentioned methods enables to solve the unseen text problem, as well as it satisfies the memory constraints. They also discussed about making use of linear interpolation techniques to obtain spectral smoothing at the concatenation point.

Chappell and Hansen [7] discussed about performing effective concatenative speech synthesis by smoothing the transitions between the speech segments with a limited database of audio segments. They talk about using various techniques such as optimal coupling, waveform interpolation, linear predictive pole shifting, and psychoacoustic closure in spectral smoothing. After the smoothing processing is done, the output speech can be approximated to the desired speech characteristics.

2.2.1 Spectral smoothing

This technique is used to smooth the transitions between concatenated speech segments. The general approach of this technique is to take one frame of speech from the edge of each segment and interpolate them [7]. Determining the circumstances in which smoothing is to be performed is an important issue of this technique.

2.2.2 Optimal coupling

This is a very cost efficient spectral smoothing technique. This method enables the boundaries of speech segments to be moved to provide the best fit with adjacent segments. Basically in this technique a measure of mismatch is tested at a number of possible segment boundaries until the closest fit is found.

2.2.3 Waveform interpolation

In this technique a waveform is interpolated in either the time or frequency domains [7]. The waveform is interpolated between the frames at the edges of speech segments, so that smoothed data can be inserted between them.

Lukaszewicz and Karjalainen [14] experimented many ways to overcome difficulties in concatenating speech sample waveforms. They introduced a new method of speech synthesis called as *microphonemic method*. The fundamental idea of this method is to apply pitch changes for intonations and transitions by mixing parts of neighboring phoneme prototypes.

2.2.4 Removing linear phase mismatches

One important issue in the concatenative speech synthesis is the synchronization of acoustic units. In the context of concatenative speech synthesis there are two types of phase mismatches [15], such as

- *Linear phase mismatch*: This refers to the interframe mismatch.
- *System phase mismatch*: This occurs around the concatenation point. It introduces noise between harmonic peaks, thus effecting the output quality.

Stylianou [10, 15] proposed a neural network based Harmonics plus Noise Model(HNM) towards the modification of fundamental frequency and duration of speech signals. The author extensively discussed about the issues that arise when this model is used to concatenate and smooth between acoustic units in a speech synthesizer. This HNM model represents speech as a sum of sinusoids plus a filtered and time-modulated noise. O'Brien and Monaghan [16, 17] proposed an alternative harmonic model that uses a bandwidth expansion technique which corrects pitch as well as time

scale to represent the aperiodic elements of the speech signal. They applied this technique to concatenative speech synthesis and the results obtained compares favorably to implementations of other common speech synthesis methods. They also presented a smoothing algorithm that corrects phase mismatches at unit boundaries.

Wouters and Macon [18] discuss about minimizing audible distortions at the concatenation point, thus attain smooth concatenation of speech segments. Even with the largest sample space of speech segments there exists a possibility that end of one segment may not match with the beginning of the next segment. Therefore the residual discontinuities have to be smoothed to the largest extent possible to attain a quality output speech. Wouters and Macon [18] proposed a technique called *unit fusion* which controls the spectral dynamics at the concatenation point between two speech segments by fusing the segments together. They also proposed a signal processing technique based on sinusoidal modeling that modifies spectral shape thus enabling quality resynthesis of units. Tatham and Lewis [19, 20] in their paper discussed about a high level modular text-to-speech synthesis system called SPRUCE. SPRUCE was based on naturalness factor in the synthesis which is normally obtained by modeling intonation, rhythm, and variability.

2.3 Summary

A survey of various approaches proposed in text-to-speech synthesis are presented. An overview of various techniques proposed such as optimal coupling, waveform interpolation, spectral smoothing, etc. to improve concatenative speech synthesis are presented.

CHAPTER 3

PROPOSED SYNTHESIS ALGORITHM AND ARCHITECTURE

Concatenative speech synthesis is capable of producing natural and intelligible speech whose quality closely matches to the voice quality of the speaker who recorded the audio segments from which the concatenation units are drawn. Concatenative synthesis has the advantages of being simple to implement and requiring a relatively small database of speech. This chapter discusses about the proposed speech synthesis algorithm. The speech synthesis algorithm described here is part of the text-to-speech synthesis. The block diagram of the algorithm shown in the Figure 3.1.

The first block in the diagram represents the *acoustic library* where all the audio recordings are stored. The second block represents the *parsing unit* which breaks an input word or a target word into small segments to be searched in the acoustic library. The third and final block represents the *concatenating unit* which concatenates the matched audio segments to synthesize the output word. The detailed explanation of the speech synthesis system and proposed speech synthesis algorithm is given in the following sections.

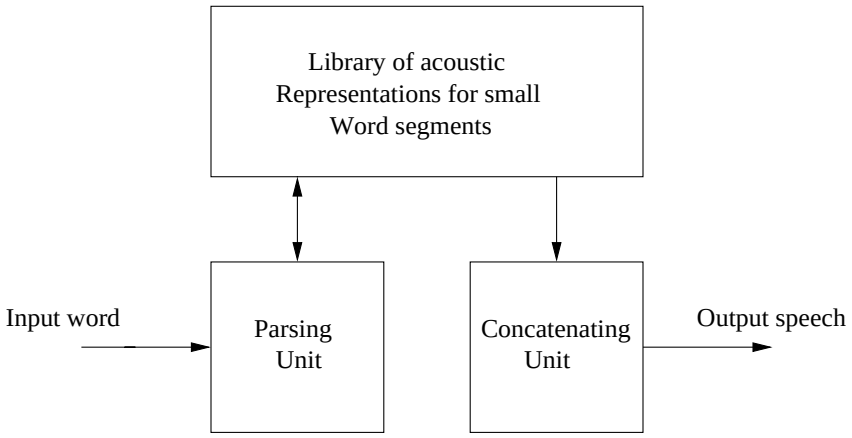

Figure 3.1. Block diagram of the proposed speech synthesis algorithm

The speech synthesis system primarily consists of three units.

- Acoustic library

- Parsing unit
- Concatenation unit

3.1 Acoustic library

This library contains the utterances from a single speaker recorded for the small word segments of interest. As it is not possible to account for the all the possible combinations of words, we have recorded a few selected words of sizes four, three, and two. This library consists of around 250 such representations. The duration of such audio segments is standardized to 0.50 seconds.
There are some important considerations to be taken care of during the construction of acoustic library, such as

- Since recording is often done in several sessions, it is necessary to maintain the recording conditions constant to maintain spectral or amplitude continuity. The recording devices such as microphone, sound card, etc. should remain the same throughout all the recording sessions. Changes in recording conditions may cause spectral or amplitude discontinuity [21].
- High quality speech synthesis can be obtained with a sample set of large number of utterances, but it requires large memory. If we can restrict the text read by the target speaker is representative of the text to be used in our application, we can account for memory considerations without sacrificing the quality of speech synthesis.

3.2 Parsing unit

This unit breaks input word into smaller segments to be searched for the matching acoustic representations in the acoustic library. The parser first takes the first four letters of the input word, in case no corresponding audio segment is found then it just takes first three letters and again goes through the search process. If the search for the first three letter segment fails, this process is repeated for the first two letters of the input word.

3.3 Concatenating unit

This unit concatenates the matched segments to form a single unit to form the synthesized output. The synthesized output can be further processed and smoothed to get a more refined output, which closely resembles the target word. This unit doesn't perform the smoothing process.

3.4 Speech synthesis algorithm

- First step in the speech synthesis algorithm is parsing an input word. The input word is fed to parsing unit. The parser takes the input word and breaks into smaller segments depending upon the acoustic library which is explained later.

- The parser first takes the first four letters of the input word and then searches for the acoustic unit corresponding to that segment. If corresponding acoustic unit is found in the library then it is sent to the concatenating unit. In case it fails to find the match, then it takes just first three letters and repeats the same procedure. When the search fails for the first three letter segment, this process is carried out for the first two letters as well, as explained in the following example. Let us assume 'stampede' as the input word. The parsing unit first takes the first four letters of the input word and forms the segment 'stam'. It searches for the corresponding audio segment in the acoustic library. If it is found then it proceeds with the rest of the input word in the same manner. In case no corresponding audio segment is found, then it searches for the segment 'sta' and proceeds as explained above. In case there is no corresponding acoustic unit is found, it also sends a message that the library doesn't contain the required segment.

- Once the above mentioned step for a single segment is finished, same procedure is repeated for the rest of the segments of the input word. When all the audio segments corresponding to the input word are available in the library, then the concatenating unit concatenates them to form the output word.

- All the matched audio segments are first read as .wav files by a MATLAB program. Then it concatenates all the matched segments and synthesizes the output word.

The algorithm was implemented in C language. It makes use of a MATLAB program to implement the concatenation of the audio segments. The implementation of the algorithm in C is provided in the appendix A.

3.5 Architectural design of the synthesis system

A digital system is basically a sequential circuit which consists of interconnected flip-flops and various combinational logic gates. Digital systems are always designed with a hierarchical and modular approach. The digital system is divided into modular subsystems which perform some functions. The modules are designed from functional blocks such as registers, counters, decoders, multiplexers, buses, arithmetic elements, flip-flops, and primary logic gates such as or, and, inverter etc. Interconnecting all the modular subsystems through control and data signals forms the digital system [22].

The architectural body of any digitally designed system consists of two main modules, such as

- Controller
- Datapath

The top level block diagram of a general digital system which shows the relationship between a controller and a datapath is given in the Figure 3.2. The first block represents the controller and the second block represents the datapath. The *datapath* performs the data-processing operations where as the *controller* determines the sequence of those operations. Control signals activate various data-processing operations. The controller communicates with the datapath through these control signals. At first controller sends a sequence of control signals to datapath and in turn receives status signals from datapath. For various operations controller uses variable control signal sequences. The controller and datapath also interact with the other parts of digital system such as memory and I/O units through control inputs, control outputs, data inputs, and data outputs. Datapath performs many register transfer operations, microoperations etc. Register transfer operations correspond to the movement of the data stored in the registers and the processing performed on that data. On the other hand microoperations correspond to the operations such as load, add, count, subtract performed on the data stored in the registers. Controller provides the sequence for the microoperations [22].

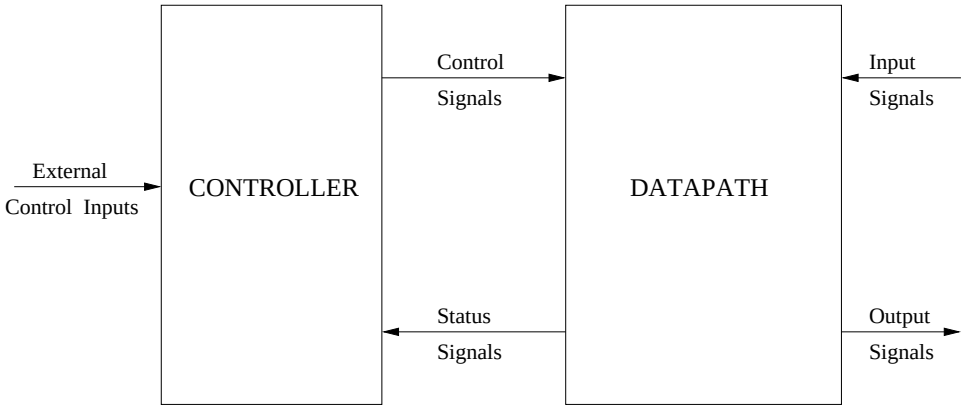

Figure 3.2. The top-level block diagram of general digital system consisting controller and datapath

In this section, the details of the speech synthesis system, designed as part of this thesis are presented. The architectural design hierarchy of the speech synthesis system consists three steps.

- Pseudo-code representation of the algorithm
- State diagram and controller design
- Datapath design

The pseudo-code representation is one of the most useful ways of representing the algorithm. We formulate the pseudo-code for the algorithm presented in the previous chapter, and the pseudo-code for the implementation of the algorithm is given in Section 3.6. The controller design is the first step in the architecture design of any digital system. At first, a state flow graph is developed from the pseudo-code explained in Section 3.6. From the state flow graph we design the finite state machine or the controller. Controller controls the datapath at specific control points in the datapath. The detailed design process for the controller is explained in Section 3.7. The last step in the architectural design hierarchy is the design of the datapath. The datapath consists of all combinational and sequential logical units such as counters, logic gates etc., memory elements and I/O elements etc. The detailed datapath design is presented in the Section 3.8.

3.6 Pseudo-code of the algorithm

(1) Initialize and load the acoustic library, which consists all the audio recordings;

(2) Load the target or input word;

```
(3)     Initialize the register counter K;
(4)     While ( For all the characters in target word ) do
(5)       {
(6)     Divide the target word into phones of size two starting from the first character;
(7)       } /* end for (4) */
(8)     For all the phones of size two
(9)           {
(10)      search the acoustic library for all the phones of size two;
(11)      if(matching phones are found) then
(12)                {
(13)      found <= 1;
(14)      Read the output data for all the phones ;
(15)                    {
(16)      send the start signal to audioplayer when player_done <= 0;
(17)      when player_done <= 1 implies audioplayer has finished playing the phone;
(18)                    } /* end for (14) */
(19)      reset the register counter K and go onto the next phone;
(20)                } /* end if (11) */
(21)      if( no matching phones are found) then
(22)                {
(23)      dfound <= 1;
(24)      reset the register counter K -> reset_K <= 1;
(25)                } /* end if (20) */
(26)      repeat the procedure for all the phones of the target word;
(27)           } /* end for (8) */
```

3.6.1 Description of the pseudo-code

All the audio recordings (phones) are loaded in the acoustic library (line 01). After acoustic library has been compiled then, load the target word or input word (line 02) which has to be synthesized. Line 03 determines the initialization of the register counter K which is used to count all the phones loaded in the acoustic library. Once the target word is loaded, parse the target word into small segments of two letters (line 04 to line 06), here we assume that the target word has even number of letters in it. After target word is divided into small segments, for all the segments of the target word a search operation for the corresponding phones in the acoustic library (line 08 and line 10) is performed. While searching for the corresponding audio recording for a small target word segment, if a matching phone is found (line 11 to line 13) then it sends a start signal to the audio player (line 16) which starts to play the entire audio recording of that particular phone. When the audio player finishes its operation then it sets a signal *player_done* to 1 (line 17). It indicates that the play operation for a single phone is finished. Now register counter K resets to 0 (line 19). The process is repeated for other segments of the target word (line 26). In case, no matching phone for a segment is available in the acoustic library, then a signal *dfound* is set to 1 (line 23). Once an operation of a single segment is done, then the process is repeated for all the remaining segments of the target word.

3.7 Design of the controller

Usually, a state machine design starts with the general word description of what we want the machine to do. The following step is to build the state table, transition table, and excitation table. Then the excitation equations, the equations used for the inputs of flip-flops are derived from the excitation table and so do the output equations from the transition table. Once the output and excitation equations are derived, the rest of the controller design procedure resembles a normal combinatorial circuit design, and connecting them to the flip-flops.

There are three methods to design a finite state machine [23].

- Random logic. It is the easiest method for the design of a finite state machine. Reassignment of the states or state machine optimization is not permitted.

- Use directives to guide the logic synthesis tool to improve or modify the state assignment.
- Use a special state machine compiler to optimize the state machine. It is the most accurate and difficult method to use for finite state machine design.

State encoding is the primary and most important step in finite state machine design. Poor choice of state codes result in too much logic and it makes the controller too slow. In digital design the state encoding can be performed in different ways such as

- Adjacent encoding
- One hot encoding
- Random encoding
- User-specified encoding
- Moore encoding

The encoding style for state machines depends on the synthesis tool we are using. Some synthesis tools encode better than others depending on the device architecture and the size of the decode logic. We can declare the state vectors or let the synthesis tool determine the vectors. As we are using Xilinx tools, they use the enumerated type of encoding to specify the states and use the Finite State Machine (FSM) extraction commands to extract and encode the state machine as well as to perform state minimization and optimization algorithms.

3.7.1 Procedure for the controller design

Sequential circuit depends on the clock as well as a certain set of specifications which form the logic for the design. The design of the sequential circuit consists of choosing the flip-flops and finding a combinational circuit structure which along with the flip-flops produces the circuit that meets the specifications. The sequence of steps to be followed to derive the finite state machine, are mentioned below as well as shown in the Figure 3.3.

- From the statement of the problem the state diagram is obtained. The state diagram is given in the Figure 3.4 which was derived from the psuedo-code explained in Section 3.6.

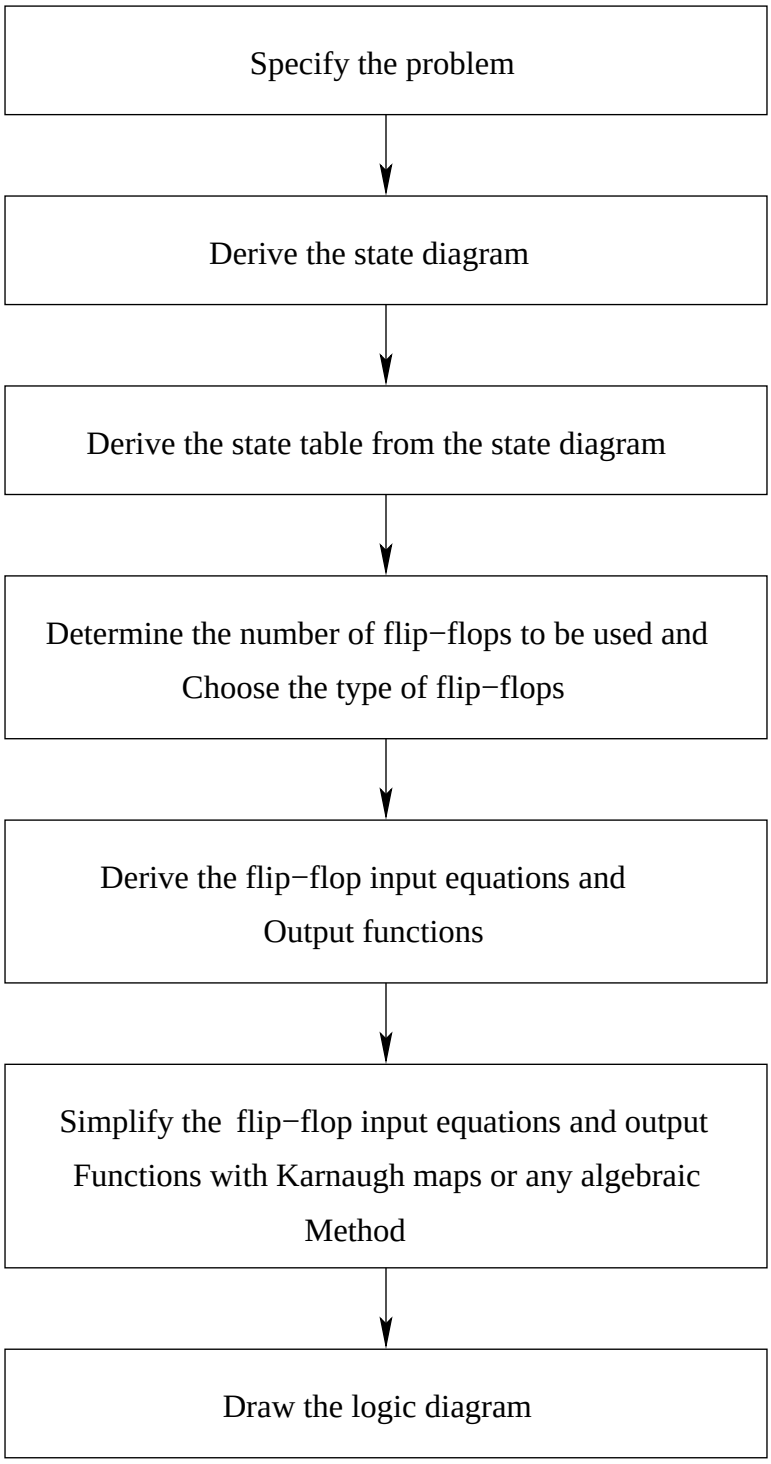

Figure 3.3. Design procedure for the finite state machine

- As a combinational circuit is fully specified by a truth table a sequential circuit is specified by a state table. The state table consisting of all the inputs and outputs of the controller and their behavior is formulated from the state diagram given in the Figure 3.4.

- A synchronous sequential circuit is a combination of flip-flops and combinational gates, it is necessary to obtain the flip-flop equations. The flip-flop input equations are derived from the next state entries in the encoded state table. Positive edge triggered D-type flip-flops are used in the controller design as they ignore the pulse when it is constant and triggers only during a positive transition of the clock pulse. Some flip-flops trigger at negative transition(1-to-0) transition and some trigger at positive transition(0-to-1). As part of the controller design, the input equations 3.1, 3.2, and 3.3 are derived.

- Similarly, the output equations are derived from the output entries in the state table. For all the outputs from the controller the output equations are obtained. In case of this design, the equations 3.4, 3.5, 3.6, 3.7, 3.8 are output equations which are given later.

- If possible, simplify the flip-flop input equations as well as output equations. Karnaugh maps can be used to simplify the equations to avoid cumbersome logic circuit structure.

- As the last step the logic diagram with the flip-flops and combinational gates is drawn, according to the requirements mentioned in the input and output equations.

3.7.2 Design of the state diagram and state table

The state diagram is given in the Figure 3.4. It consists of eight states namely S0, S1, S2, S3, S4, S5, S6 and S7. The state S0 is the initial state. The system starts when signal $ready$ is set to 1. In state S1 target word and acoustic library is loaded for the system to be synthesized. When $load$ signal is 1 target word is loaded and the system goes to the next state. Thus loaded target word is parsed into small segments externally depending on the $incr\_I$ signal. In state S2 it assigns the first segment of the target word to X. In state S3 it checks the library. Here we assign a counter K, which is used to count the number of phones existing in the library. In the following state S4 it performs the search process. The target word segment X is compared to the elements in the library.

In case a matching phone is found then cmp signal goes high and system enters into state S6 else it goes to state S5, which increments the register counter K and continues the search process. In state S6 the audio recording corresponding to the matched phone is read through an audioplayer program. Once this task is finished it sets a signal $player\_done$ to 1 and goes to state S7 where the register counter K is reset and segment count I is incremented. The next segment of the target word is assigned to X and again follows the same procedure. This procedure is followed for the all the segments of the target word.

A state table is defined as the enumeration of the functional relationships between the inputs, outputs, and the flip-flop states of a sequential circuit [22]. It primarily consists of four sections, labeled *inputs, present state, next state,* and *outputs*. The *inputs* section provides all the possible values for the given input signals for each possible present state. The *present state* section shows states of flip-flops A, B, and C at any given time t. The *next state* section provides the states of the flip-flops A, B, and C on clock cycle later at time $t + 1$. The *outputs* section provides us the values of the all the output signals at time t for every combination of present state and inputs.

In general in a sequential circuit of m number of flip-flops and n number of inputs then we will have 2^{m+n} rows in the state table. The binary values for the next state are directly derived from flip-flop input equations. The state table is given in the table 3.1. We have got rid of the some unused states, so we have the less number of rows as per the formula given above. The detailed procedure for the remaining part of the controller design is explained in the later sections.

It is very important to have a look at all the controller signals involved and their functionality. Here we list the controller inputs and outputs and their functional abilities. The brief information about the input and output signals and their purpose is given below.

- $ready$ is an input signal. When $ready$ is 0, the system is in the idle state. The system gets started only when $ready$ is set to 1.

- $load$ is an input signal. When $load$ is set to 0, the system is in the process of loading the target word as well as the the input phones that exist in the acoustic library. When $load$ is 1, it implies that target word is loaded into the system.

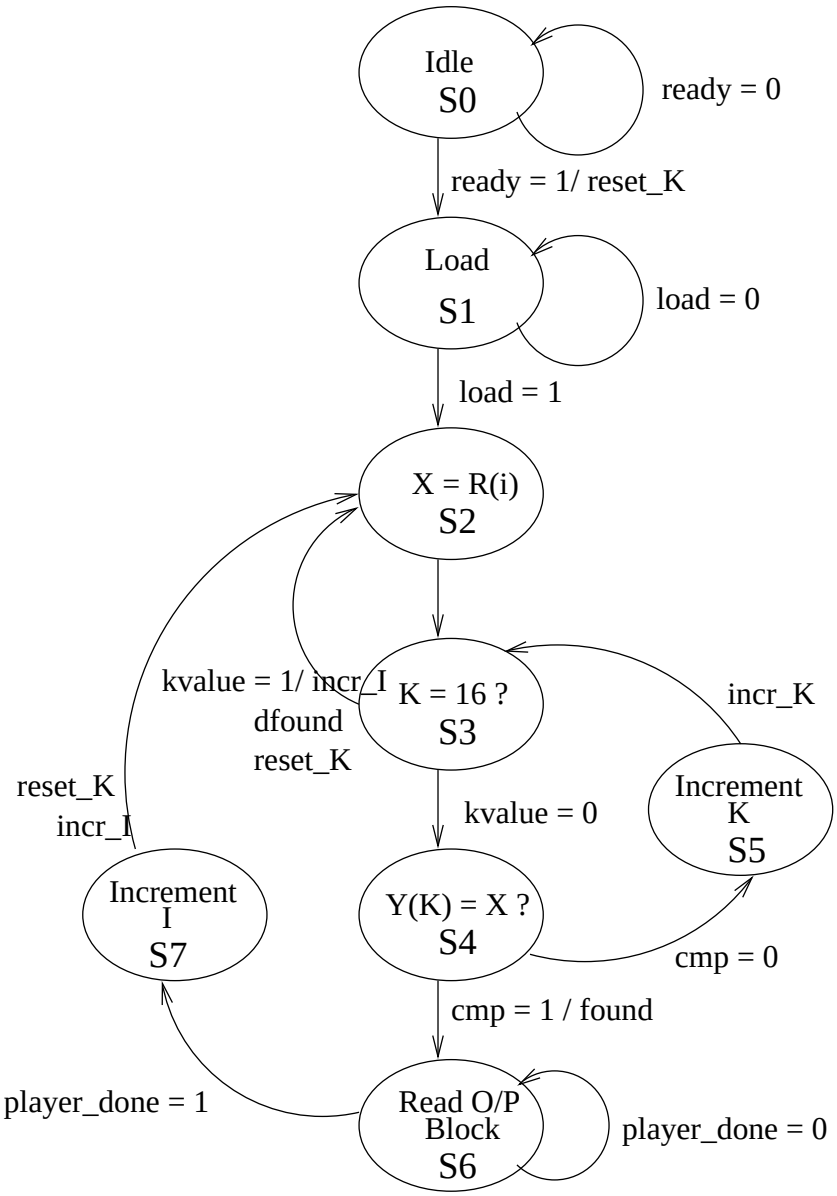

Figure 3.4. State diagram for the controller

- $kvalue$ is also an input signal. When it is set to 1, it denotes that all the acoustic library elements are compared to a particular target word segment. When it is 0, we still have some more entries in the library to be compared to.

- cmp is an input signal. It is the output of the comparator which compares the target word segments and acoustic library units. When it is 1, it indicates that a matching phone is found for that particular target word segment. When it is 0, it implies no matching phone is available in the library.

- $player\_done$ is also an input signal. It is the output of the audio player program which we use to play the audio recording for a particular phone when a matching phone is found in the library. When it is set to 0, the recording is still being played. When it is 1, it implies the audio recording is finished and the system goes to the next state.

- $reset\_K$ is an output signal. It is used to reset the register counter K. Initially, when the system enters the state S1, the counter is reset. It also goes high when search is finished irrespective of the result of the search. When a matching phone is found for a particular segment, then the counter is reset to 0. The search continues for the other segments, if there is no fit available in the library, then it skips that particular segment and set to 0 to begin the search process for the other segments of the target word.

- $incr\_I$ is an output signal. This signal is used to replace the one target word segment for which the search has finished with another target word segment for which the search has to be done. When it is 1, it implies that search for one segment is done and the search for the next segment begins.

- $incr\_K$ is also an output signal. It is the increment signal which increments the register counter K. When a particular entry in the library when compared to the target word segment, is not the right match then the register counter gets incremented so that the next available entry in the library can be compared to the target word segment. Counter K gets incremented till it reaches the maximum available entries in the acoustic library are compared.

- $found$ is an output signal. It is the output of the comparator which compares the target word segments and the acoustic library entries. When there exists a matching phone in the library to that of a particular target word segment, it is set to 1.

- $dfound$ is also an output signal. It is set to 1, when there exists no matching phone in the library for a particular target word segment. $dfound$ is set to high, if $kvalue$ is 1, and there is no right match for a particular target word segment.

3.7.3 Flip-flop input and output equations

The three input equations for the D flip-flops can be derived from the next state values in the state table. If possible, we simplify the equations using Karnaugh maps or any algebraic methods.

The input equations for the state flip-flops are

$$A^+ = S_3 * \overline{kvalue} + S_4 + S_6 \tag{3.1}$$

$$B^+ = S_1 * load + S_2 + S_3 * kvalue + S_4 * cmp + S_5 + S_6 + S_7 \tag{3.2}$$

$$C^+ = S_0 * ready + S_1 * \overline{load} + S_2 + S_4 * \overline{cmp} + S_5 + S_6 * player\_done \tag{3.3}$$

The output equations can be obtained from the binary values of the output entries in the state table. If possible output equations are simplified as well. The output equations are

$$reset\_K = S_0 * ready + S_3 * kvalue + S_7 \tag{3.4}$$

$$incr\_I = S_3 * kvalue + S_7 \tag{3.5}$$

$$incr\_K = S_5 \tag{3.6}$$

$$found = S_4 * cmp \tag{3.7}$$

$$dfound = S_3 * kvalue \tag{3.8}$$

The next step, after obtaining all the input and output equations is building the logic diagram using those equations as building blocks. Along with all input signals we also have external signals such as clk and $reset$. $reset$ signal is used to initialize the state of the flip-flops. This signal is different from the output signal $reset\_K$, which is used to reset the register counter K. This master

Table 3.1. State table for the controller

$ready$	$load$	$kvalue$	cmp	$player\_done$	Present State	Next State	ABC	$A^+B^+C^+$	$reset\_K$	$incr\_I$	$incr\_K$	$found$	$dfound$
0	×	×	×	×	S_0	S_0	000	000	×	×	×	×	×
1	×	×	×	×	S_0	S_1	000	001	1	×	×	×	×
×	0	×	×	×	S_1	S_1	001	001	×	×	×	×	×
×	1	×	×	×	S_1	S_2	001	010	×	×	×	×	×
×	×	×	×	×	S_2	S_3	010	011	×	×	×	×	×
×	×	0	×	×	S_3	S_4	011	100	×	×	×	×	×
×	×	×	0	×	S_4	S_5	100	101	×	×	×	×	×
×	×	×	1	×	S_4	S_6	100	110	×	×	×	1	×
×	×	×	×	×	S_5	S_3	101	011	×	×	1	×	×
×	×	1	×	×	S_3	S_2	011	010	1	1	×	×	1
×	×	×	×	0	S_6	S_6	110	110	×	×	×	×	×
×	×	×	×	1	S_6	S_7	110	111	×	×	×	×	×
×	×	×	×	×	S_7	S_2	111	010	1	1	×	×	×

$reset$ avoids the system starting in an unused state. In most of the cases the flip-flops are initially reset to 0, but depending on the requirements in some cases they may be reset to 1. The port interface diagram for the controller is given in Figure 3.5.

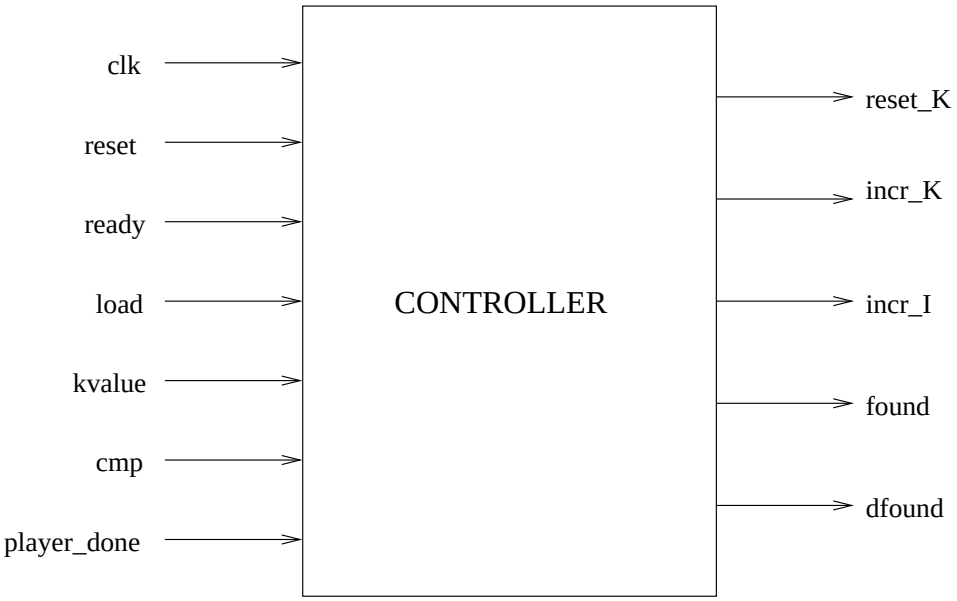

Figure 3.5. Pin diagram of the controller

3.8 Datapath design for the system

A datapath consists a digital logic that implements various microoperations. This digital logic involves buses, multiplexers, decoders, logic gates, and processing circuits [24]. The datapaths are best defined by their registers and the operations such as shift, count, load, etc. performed on the binary data stored in them. These operations are called *register transfer* operations. They are defined as the movement of data stored in the registers and the processing performed on the data [22]. The datapath for the synthesis system is given in Figure 3.6.

3.8.1 Components of the datapath

The datapath of this synthesis system is divided into five main functional blocks, they are

- Input module

- RAM
- 16-bit Comparator
- Register counter K
- Output module

3.8.1.1 Input module

The basic functionality of this module is to fetch the target word and break into two letter segments, so that they can be searched in the acoustic library for their respective audio recordings. Initially when signal $load$ is 1 the target word (input) is loaded into the system. Input module consists of two signals $control$ and $incr\_I$. When $control$ is 1, the input module asserts the first segment to be searched for the corresponding phone in the library. Once the search for this segment is finished, then $incr\_I$ signal goes high. When $incr\_I$ is 1, then the input module assigns the next immediate segment. This process continues till all the segments of the target word are searched in the acoustic library.

3.8.1.2 RAM

This is basically a register file which stores the information about the starting address of the audio recordings stored in the RAM of a FPGA device as well as the phones. This register file contains 16 registers, each register contains 35-bit wide information. The address width of the audio recordings in the RAM of FPGA is 19 bits. The phones in the acoustic library has a width of 16 bits. To scan through all the registers in the RAM we make use of a counter K. During the search process counter K provides the address of the registers in the register file to be compared with that of target word segments. RAM has a signal sel which basically implements read/write operation of a register file. When it is 0, the data is loaded into the register file and when 1, data is read.

When there exists a matching phone for a target word segment then data stored in the registers with a size of 35 bits is split into two chunks of 16 bits data and 19 bits data respectively. The 19 bits data chunk provides the address information about the corresponding audio recording for that

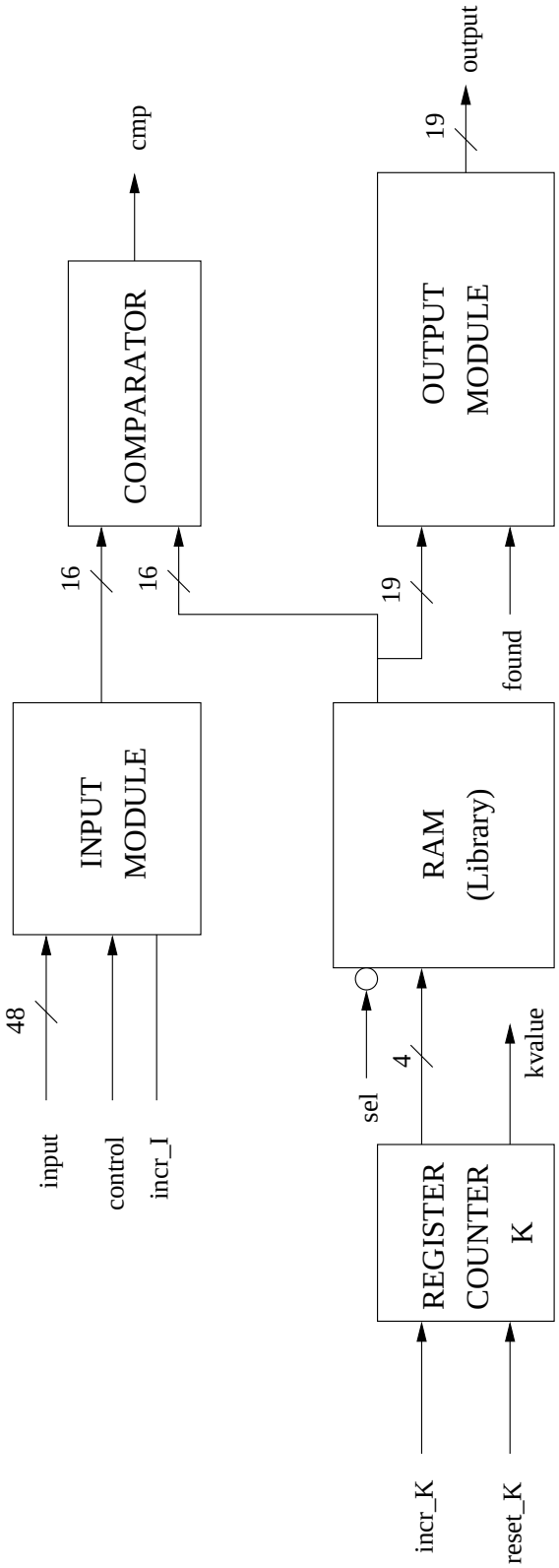

Figure 3.6. RTL schematic of the datapath

particular phone stored in the RAM of FPGA. This data chunk is provided to the output module. The 16 bits data chunk contains that particular phone, which is provided to the 16 bit comparator.

3.8.1.3 Comparator

This compares the target word segments to that of the phones stored in the acoustic library. As we know that input module breaks the target word into segments of 16 bit data width. The register file which contains the all the phones provide us the phone to be compared with the target word segment. If they match, then cmp signal is set to 1 and the remaining 19 bits of the data stored in the same register is sent to the output module. Output module provides this output data as the starting address to the audio player module. The audio player starts playing the audio data starting from the address provided by the output module to the end of that particular audio recording.

3.8.1.4 Register counter K

: This counter provides the address location of a particular phone when it is compared to a target word segment. It helps us through scanning through the library. This counter has two input signals $Reset\_K$ and $Incr\_K$. $Reset\_K$ is used to reset the counter and $Incr\_K$ is used increment the counter. The counter is reset when there exists no matching phone for a particular target word segment. It is also reset when there is a matching phone for a particular segment, and proceeds to the next target word segment. Counter gets incremented, when the comparison of a particular target word segment to a acoustic library entry (phone) fails, then the counter gets incremented so that the phone available in the next register of the register file is compared to the target word segment.

3.8.1.5 Output module

This module provides us the starting address of the audio recording stored in the RAM of the FPGA device, which we use as a target device for this experiment. When there exists a matching phone for a particular target word segment, then as it is discussed, in the controller design section $found$ signal goes high. When $found$ is high then starting address is loaded to the output module, which provides the starting address to the audio player which plays out the audio recording. To put it briefly, the starting address of an audio recording is stored in the register file along with the

corresponding phone. When that particular phone is a right match for the target word segment, then the starting address stored with that phone is loaded to the player and it is played out as an audio signal.

3.8.2 Functional description of the datapath

The above five functional blocks constitute the datapath of the synthesis system. Briefly, the datapath functionality is described in this subsection. The functionality of the datapath can be divided into three phases, such as

- Loading phase
- Searching phase
- Output phase

At first, in the loading phase the target word, which is to be synthesized is loaded into the input module along with the acoustic library. When loading is done, $load$ is set to 1. This indicates loading phase is finished. The next phase is searching phase. In this phase, the target word is parsed into segments of size 2 (16-bit binary). Once loading is done an external signal $control$ is set to 1, so that the first segment of the target word is loaded for the search process. The comparator compares the target word segment to the entries in the acoustic library. Because of memory constraints we are assuming there are just 16 entries in the library, depending upon the size of the memory library can be expanded. Counter K is used to scan through all the available phones in the library. If there exists a matching entry to that of the target word segment then signal cmp goes high. It implies there is a matching phone available in the library for the target word segment, which also sets signal $found$ to 1. The register file also contains the information about the audio recording corresponding to all of the entries stored in the library. In the output phase, when $found$ goes high, then starting address of the audio file stored in the FPGA memory is provided to the output module. Then output module provides starting address to the audio player program which plays the audio file. The pin diagram of the datapath is given in the Figure 3.7.

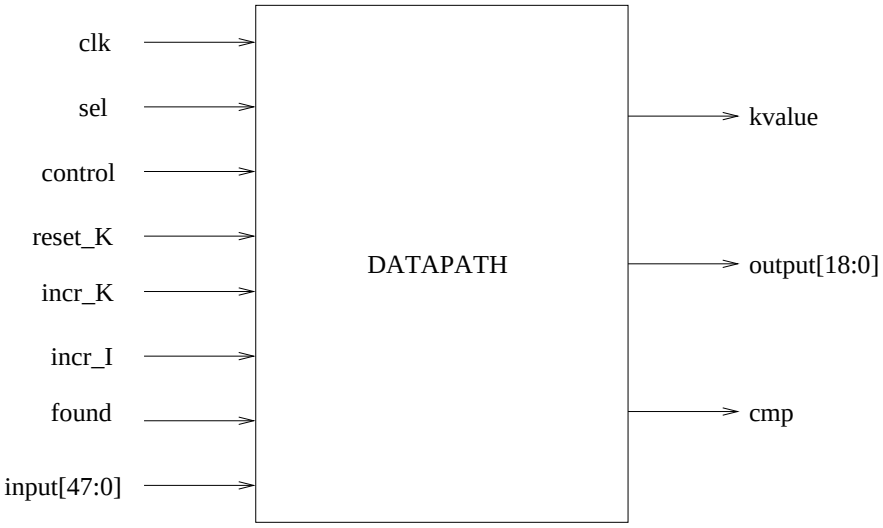

Figure 3.7. Pin diagram of the datapath

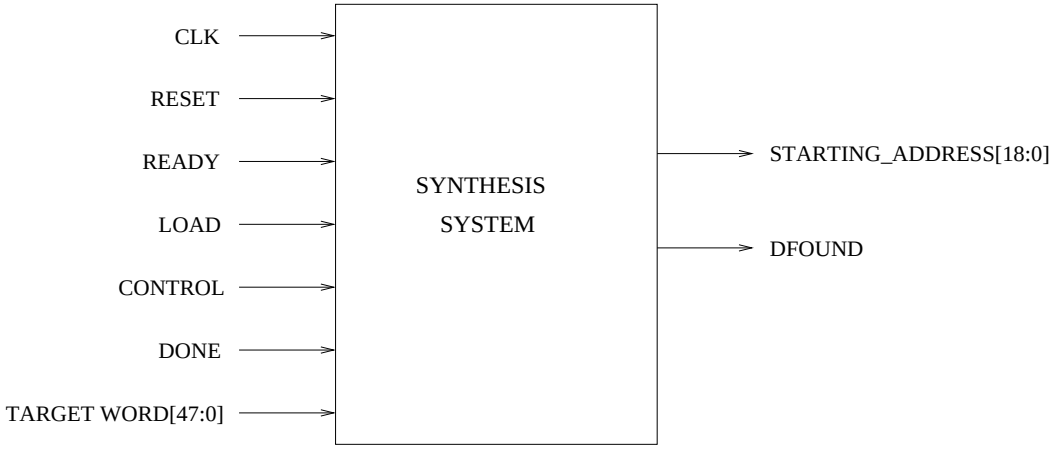

Figure 3.8. Pin diagram of the synthesis system

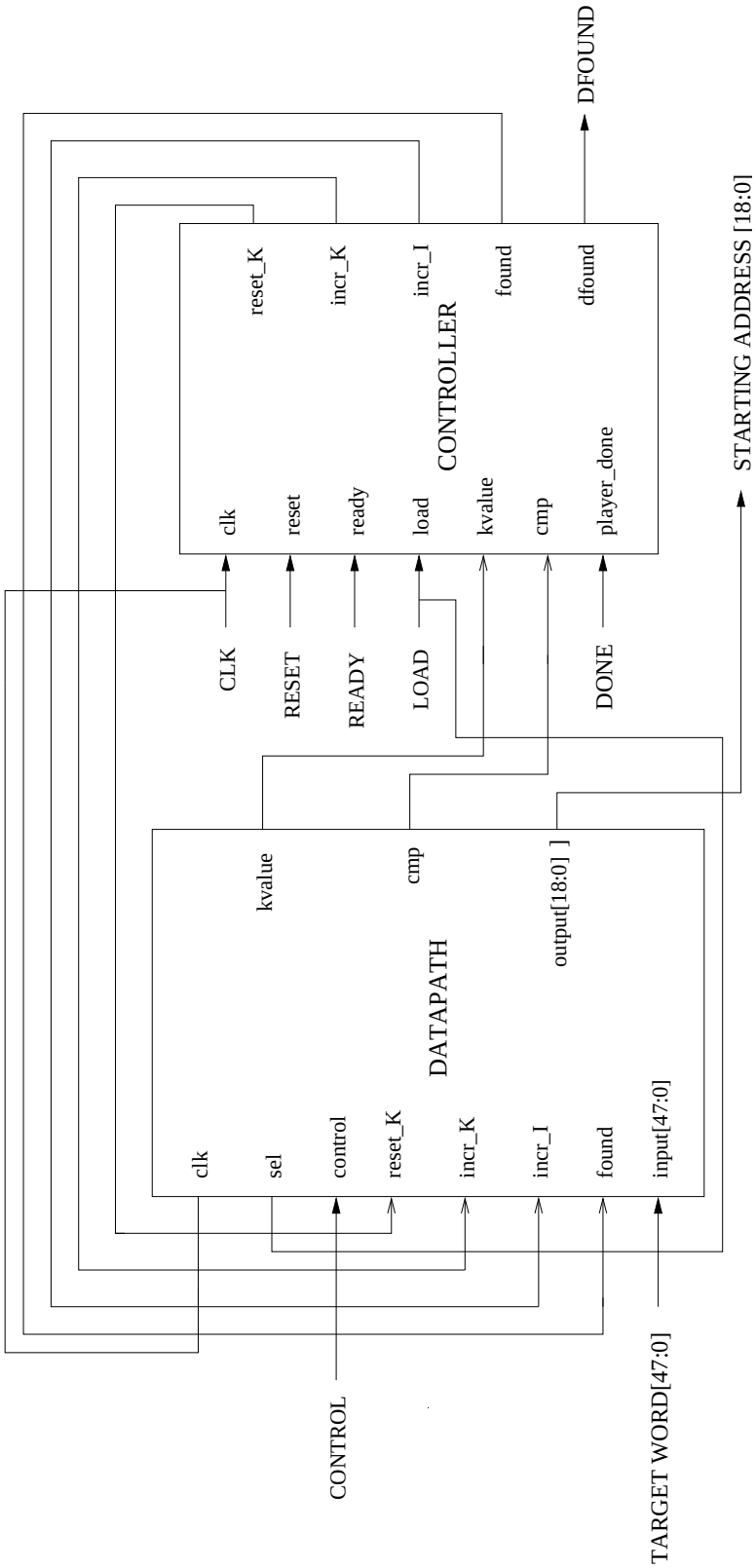

Figure 3.9. RTL schematic diagram of the synthesis system

3.9 Schematic diagram of the synthesis system

In previous sections, the design of the controller and the datapath were discussed. The schematic design of the synthesis system is given in the figure 3.9. It is formed by combining the controller and datapth explained earlier in previous sections. The pin diagram of the synthesis system is given in the figure 3.8. As given in the figure it has only two output signals $dfound$ and $starting\_address$. $dfound$ is mostly unused in the implementation of the architecture. The implementation of the synthesis system is done in VHDL which is discussed extensively in the next chapter.

3.10 Summary

A detailed explanation of the algorithm is presented. The detailed architectural design procedure as well as the architecture to implement the algorithm is described. The software implementation of the algorithm is presented in appendix A. The hardware implementation is presented in the next chapter.

CHAPTER 4

IMPLEMENTATION OF THE ARCHITECTURE AND EXPERIMENTAL RESULTS

This chapter discusses how the system architecture is implemented as well as the experimental results compiled during the course of this research work. The complexity of the contemporary integrated circuits made design automation an essential part of the vlsi design methodologies [25]. Designing an integrated circuit and ensuring that it operates to the specifics is a practically impossible task without the use of computer aided (CAD) tools. CAD tools perform various functions such as synthesis, implementation, analysis, verification, testability, depending upon their design specifications. Analysis and Verification tools examine the behavior of the circuit. Synthesis and implementation tools generate and optimize the circuit schematics or layout. Testability tools verify the functionality of the design. For design proposed in the previous chapter we make use of Xilinx foundation tools to perform the implementation and verification.

For the entire architecture explained in the previous chapter the structural VHDL modules have been designed. The hierarchy of the implementation is given in Figure 4.1.

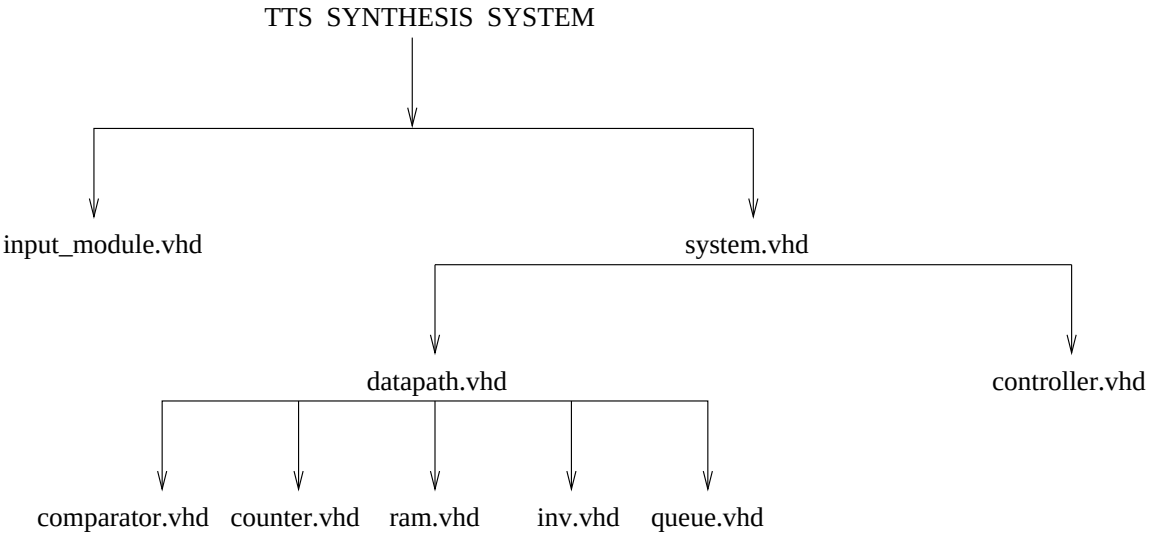

Figure 4.1. Hierarchy of the implementation of the system architecture in VHDL

4.1 Structural VHDL design for the architecture

As shown in Figure 4.1, the text-to-speech synthesis system has two basic components.

- $input\_module.vhd$ module provides the target word to the synthesis system. This module basically performs the multiplexer operation.
- $system.vhd$, its basic functionality is to take the input word from the input module and perform the rest of the synthesis process. The system module further divided into two controller and datapath modules. Datapath module consists of five other components such as
  - $comparator.vhd$
  - $counter.vhd$
  - $ram.vhd$
  - $inv.vhd$
  - $queue.vhd$

$controller.vhd$ is designed using state editor of the Xilinx foundation tools. It uses enumerated state encoding for the states. It interacts with the datapath and ensures that the functionality of the system is prevailed. Its functional simulation is provided in Figure 4.3.

$comparator.vhd$ compares two 16 bits of binary data. Its main functionality is to compare the target word segments with the acoustic library elements stored in the RAM.

$counter.vhd$ is a 8-bit behavioral counter module. It is used to provide the addresses of a particular acoustic library elements stored in the RAM.

$ram.vhd$ is a behavioral module of size of 8 x 35. It has an address 8-bit wide and data of 35-bit long. It stores a phone of width 16-bits and the starting address of its audio representation stored in the RAM of the FPGA board. It can store 256 (2^8) phones. Due to memory constraints, the RAM of the FPGA (XSV-800) board can only store 10 seconds of data. Because of this memory constraints, we experimented with the 20 audio representations for 20 selected phones.

$inv.vhd$ is a basic inverter module which inverts the input signal.

$queue.vhd$, it takes the target word from the input module and breaks into segments of 16-bit width. Depending upon the controller signals, it sends the 16-bit data segments to be compared with that of the phones stored in the memory.

All the components of the synthesis system have been verified and tested for their respective functionality individually at various instances.

4.2 Functional description of the synthesis system

Initially, the input module provides the target word to the queue module. When the controller sends the $load$ signal to 1 then queue module provides the first segment of the target word to be searched in the library. The 16-bit target word segment is compared with the phones stored in the RAM module with the use of comparator and counter. Counter provides the address of the phones. When there exists a matching phone corresponding to that of the target word segment then, it sets $found$ signal to 1.

To play the audio recordings corresponding to the phones we make use of a VHDL project, called "audioproject" [26]. This project allows recording and playback of just over 10 seconds of sound, and has options to play back the sound at doubled speed, reversed, etc. This project is to be used with the XSV board v1.0, produced by XESS Corp. This board contains a Virtex FPGA from Xilinx Inc. and support circuitry for communicating with a wide variety of external devices. This project is developed internally by students working for the School of Computer Science and Electrical Engineering in the University of Queensland, Australia [26].

When $found$ is set to 1 then the starting address of the phone is passed to the player module of the audioproject. Then the player module reads the entire data of the audio recording corresponding to that phone. Once the player finishes it's operation it sets $player\_done$ signal to 1 indicating it has done playing the data. The functionality of the player module is given in the Figure 4.2. The player plays the data till it reaches the ending address of the phone. Once the player finishes its operation the next immediate task is to search for the next target word segment. When $player\_done$ signal sets to 1 then the controller module sets $incr\_i$ signal to 1 in turn queue module sends the next target word segment to be searched in the acoustic library. The controller operation is shown in the waveform 4.3. This procedure is repeated, till all the target word segments are searched for in the acoustic library.

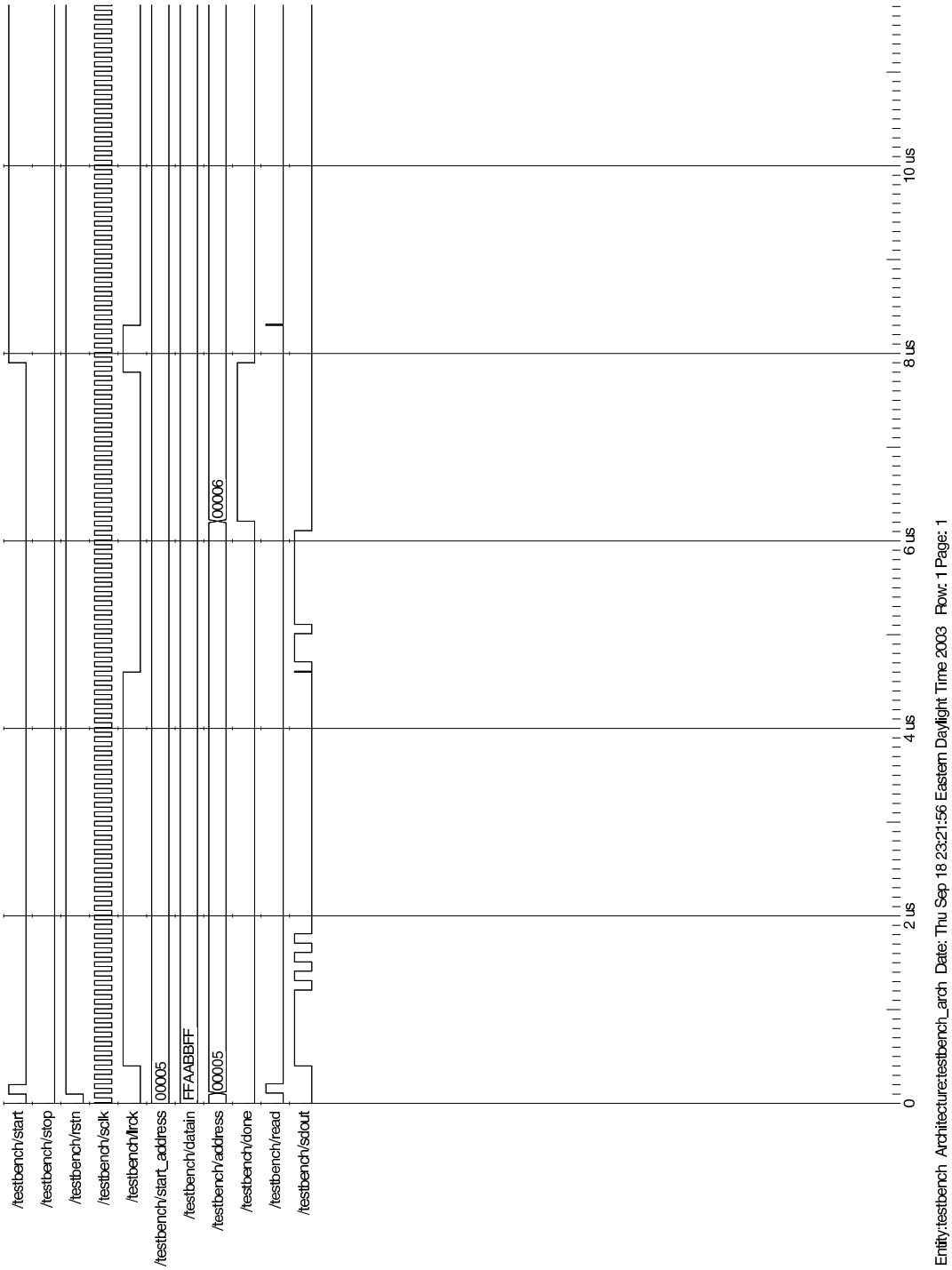

Figure 4.2. Output waveform of the audio player

4.3 Synthesis and simulation results

Simulation is performed by setting test values in the testbench. The design is synthesized using Xilinx tools and it is simulated with Modelsim simulator. The simulation results are given in the Figures 4.4, 4.5, 4.6 for three different target words. In the output waveforms we can notice that when the player module is done($player\_done$ -> 1) playing the data for a particular phone, then it goes to next segment. If another phone is found ($found$ -> 1) then it gets the $starting\_address$ of it in the RAM and sends it to the player module. Then the player module repeats the procedure performed earlier. Once the functionality of the design is verified, the design is synthesized targeting it to the Virtex FPGA (v800hq240-4). The design mapping, place and route operations are performed as part of the implementation step of the flow engine of Xilinx tools. In the next step the bitstream file to be downloaded onto the FPGA on the XSV board is generated. We make use of XSTOOLs Version 4.0.2, to download the bitstream file onto the FPGA. These tools are provided by Xess corporation. For the utilities that download and exercise the XS40, XS95, XSA, and XSV Boards for Win95, Win98, WinME, WinNT, Win2000, and WinXP.

Figure 4.3 explains the behavior of the controller. It can be observed that, when cmp goes to 1 then $found$ goes to 1. Once a particular phone is found, then $incr\_i$ goes high indicating that the next target word segment to be searched for in the acoustic library. Figures 4.4, 4.5, and 4.6 are the output waveforms generated by Modelsim for three target words *bamite*, *gemini*, and *devote* respectively. In figure 4.4, the target word *bamite* is broken into segments *ba*, *mi*, and *te*. It first searches for *ba* in the library, if it finds the matching phone then it provides the $starting\_address$ 7EE00 to the player.vhd module of the audioproject. The player module plays out the audio data corresponding to the phone *ba* and sets $player\_done$ to 1. Then *mi* is searched in the library, when found it returns $starting\_address$ 50540 to the audioproject. This process is continued for segment *te* and it returns $starting\_address$ 00000. Similarly, the search is performed for the other target word segments.

Various simple words such as bone, when, gyration, vote, byte, dose, bane, mine, nine etc. have been synthesized with this mechanism. The spectrograms of the words are obtained and observed the difference between spoken words and synthesized words. There is a distinct spectral mismatch

observed between them. The spectrograms of the synthesized as well as spoken words ‘goose’ and ‘byte’ are given in Figures 4.8, 4.7, 4.9, and 4.10.

4.3.1 Device utilization data summary

The design is synthesized targeting it to the Virtex FPGA (v800hq240-4) on the XSV-800 board. The device utilization data is given in the Tables 4.1 and 4.2. Table 4.1 gives the the utilization data of the synthesis system without the audioproject module and table 4.2 gives the utilization data of the system with audioproject.

Table 4.1. The utilization data summary of the system

Number of Slices:	69	out	of	9408	0%
Number of Slice Flip Flops:	69	out	of	18816	0%
Number of 4 input LUTs:	127	out	of	18816	0%
Number of bonded IOBs:	27	out	of	170	15%
Number of BRAMs:	6	out	of	28	21%
Number of GCLKs:	1	out	of	4	25%

Table 4.2. The utilization data summary of the system with audioproject

Number of Slices:	235	out	of	9408	0%
Number of Slice Flip Flops:	264	out	of	18816	0%
Number of 4 input LUTs:	332	out	of	18816	0%
Number of bonded IOBs:	91	out	of	170	15%
Number of BRAMs:	6	out	of	28	21%
Number of GCLKs:	1	out	of	4	25%

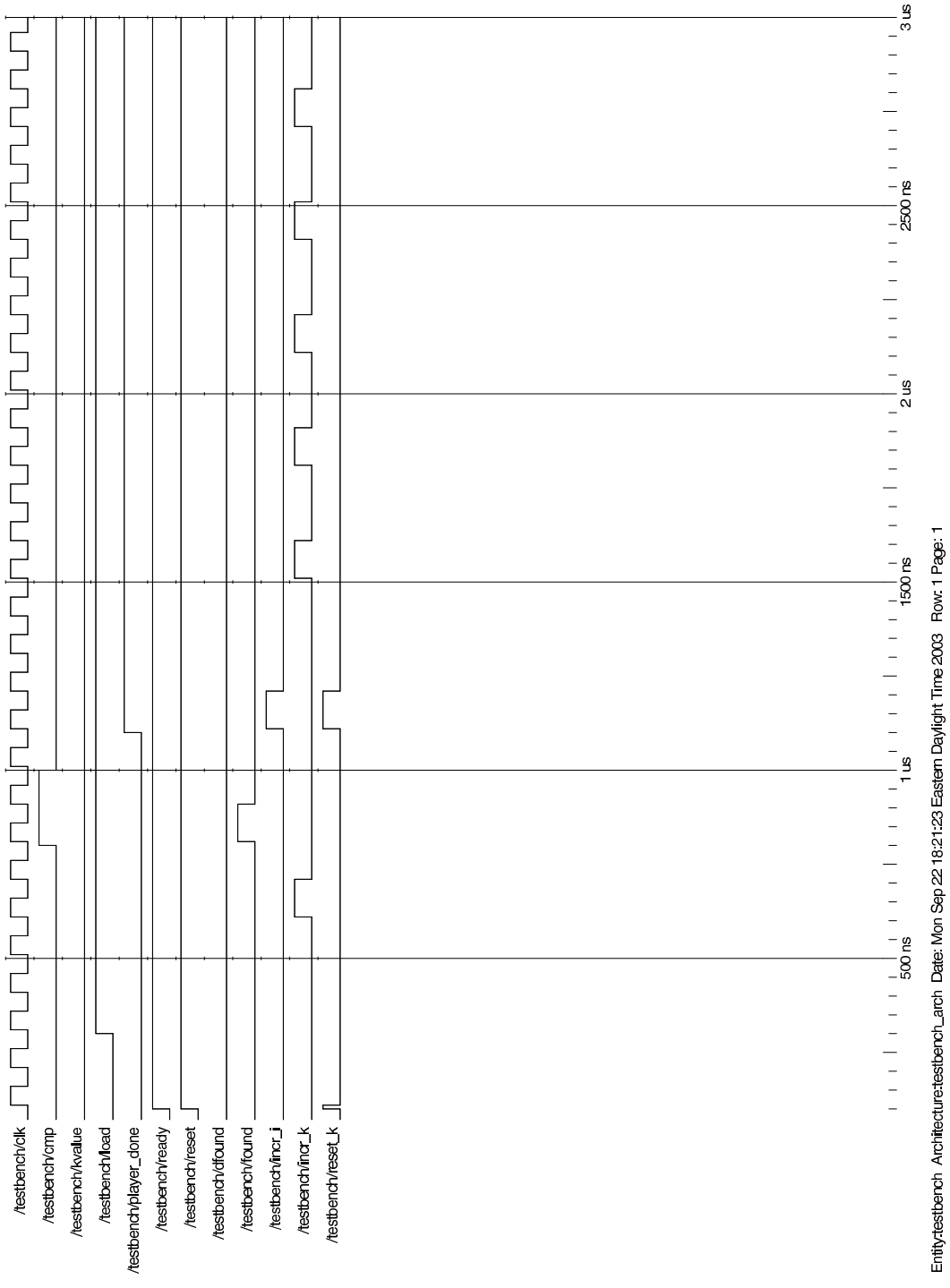

Figure 4.3. Sample output waveform of the controller

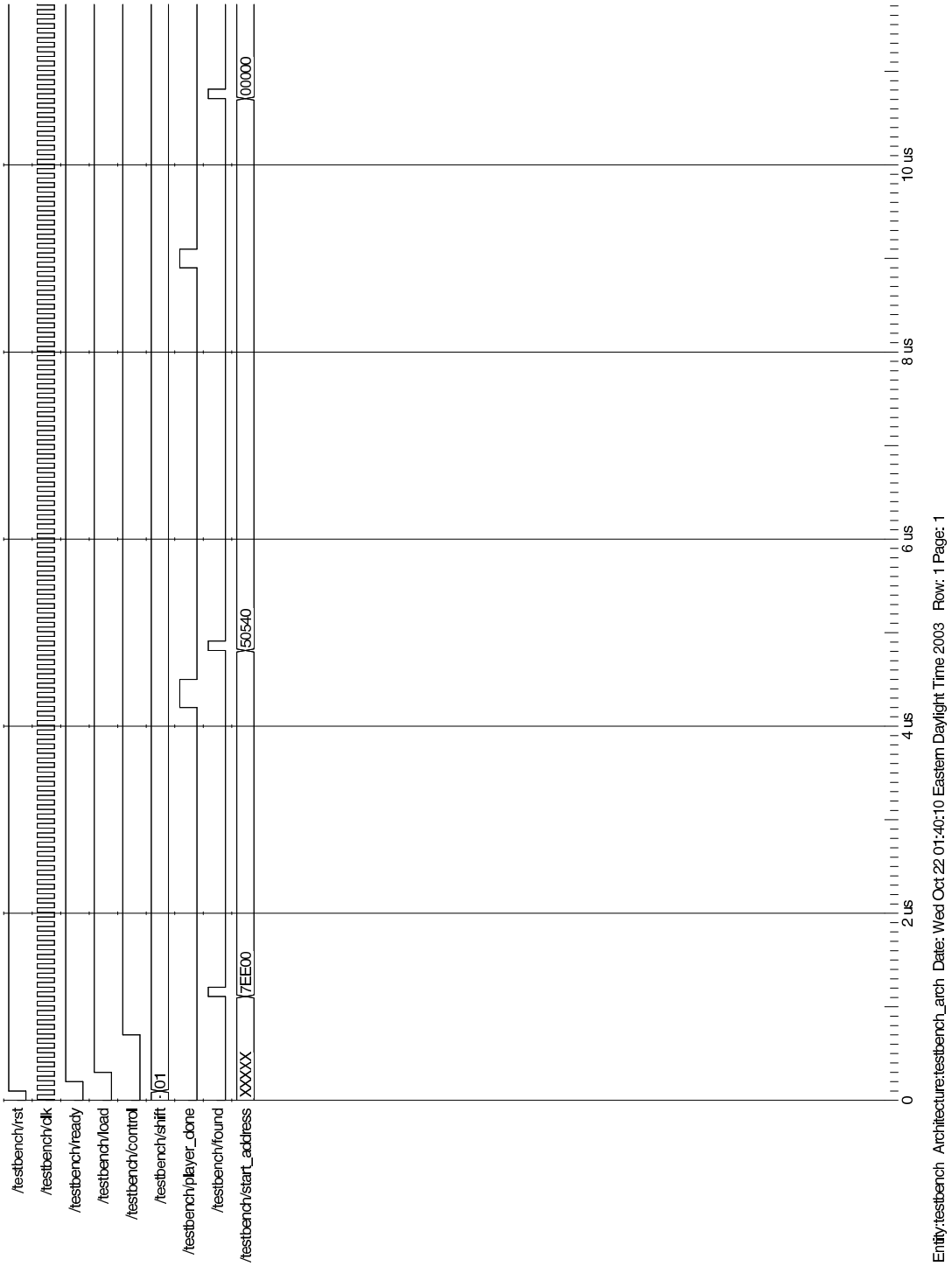

Figure 4.4. Sample output waveform 1 of the synthesized word ‘bamite’

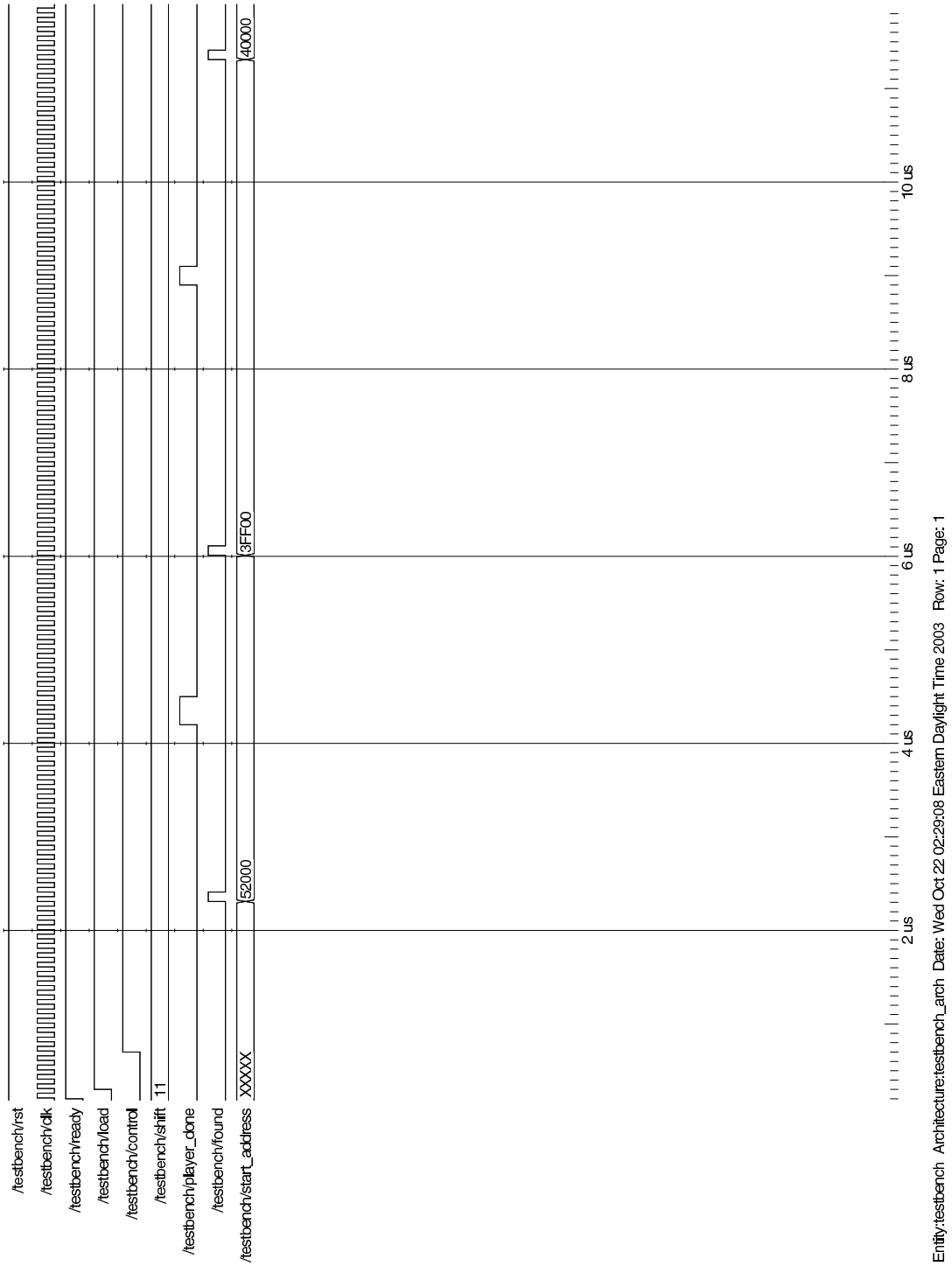

Figure 4.5. Sample output waveform 3 of the synthesized word ‘gemini’

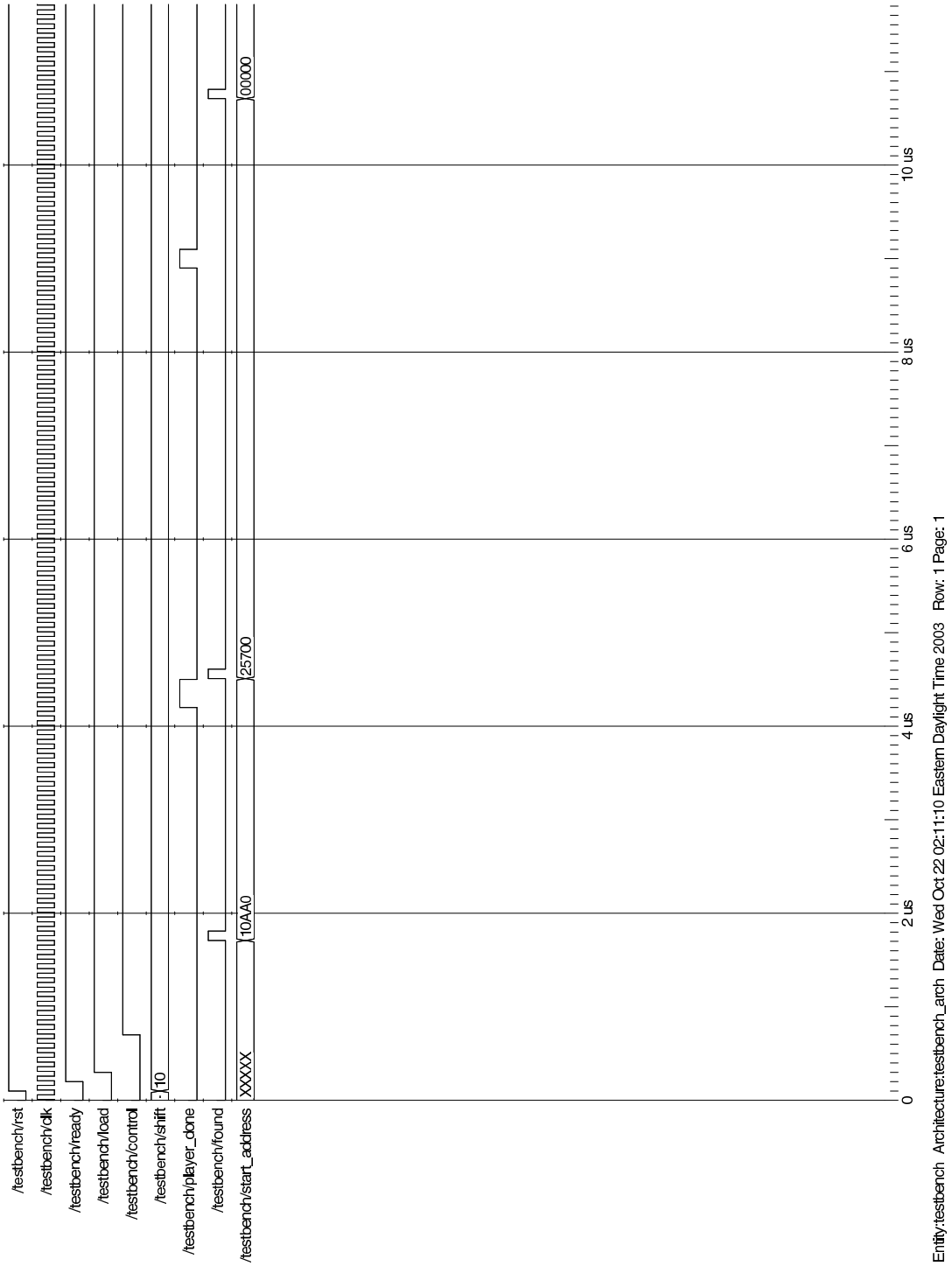

Figure 4.6. Sample output waveform 2 of the synthesized word ‘devote’

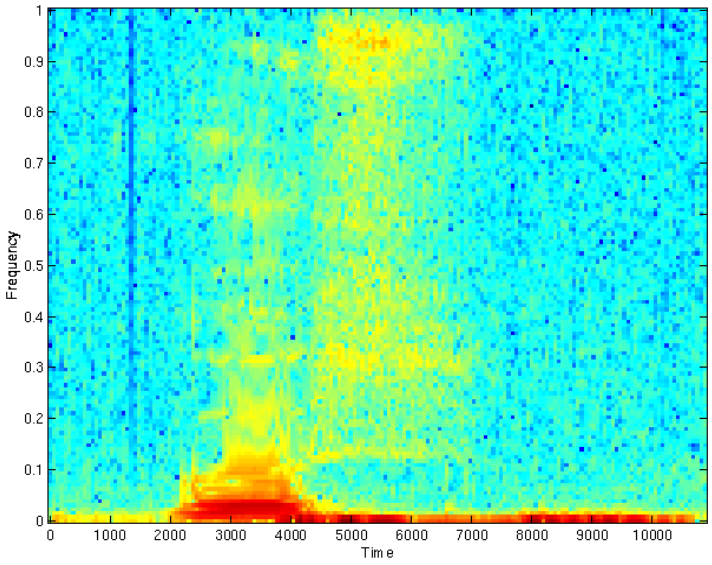

Figure 4.7. Spectrogram of a spoken word ‘goose’

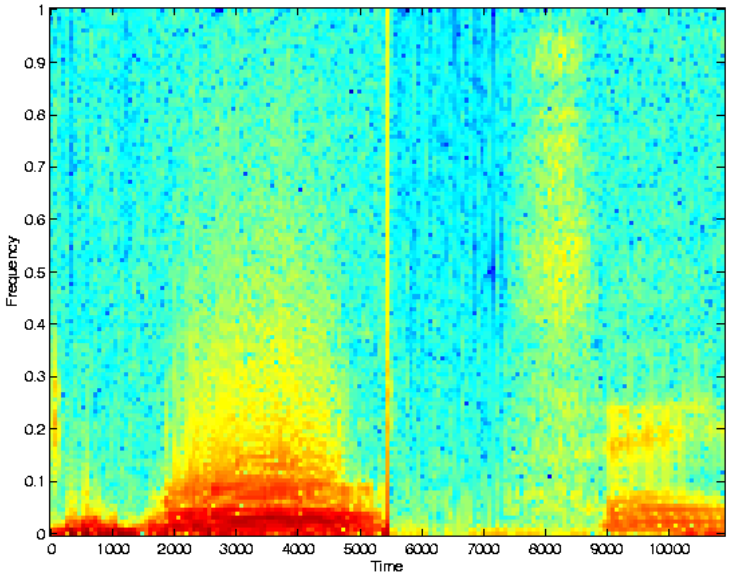

Figure 4.8. Spectrogram of synthesized word ‘goose’

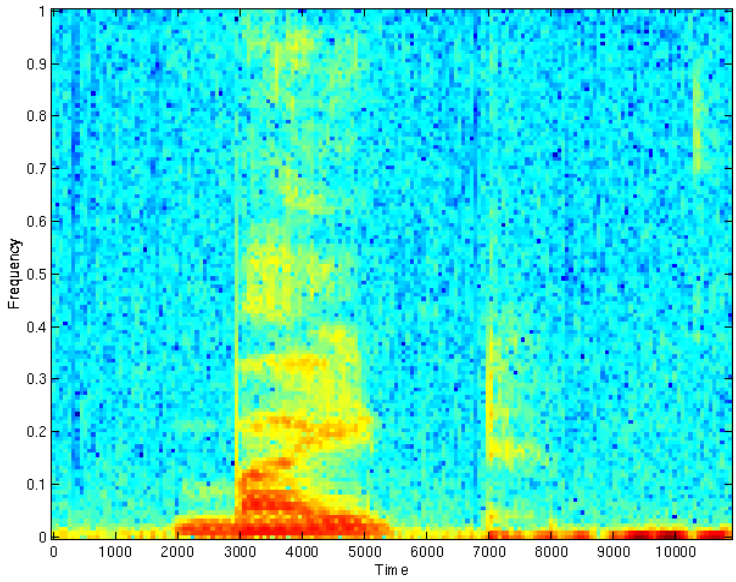

Figure 4.9. Spectrogram of a spoken word ‘byte’

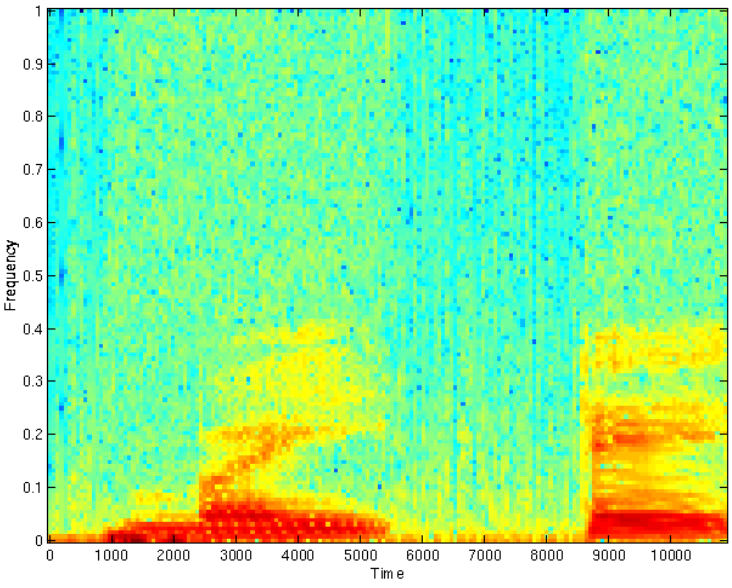

Figure 4.10. Spectrogram of synthesized word ‘byte’

CHAPTER 5

CONCLUSION

We conclude from this work that the proposed concatenative speech synthesis algorithm is an effective and easy method of synthesizing speech. Truly speaking, this method is not exactly a speech synthesis system, but it is one of the most commonly used text-to-speech systems around. In concatenative synthesizer, the designer provides recordings for phrases and individual words. The engine pastes the recordings together to speak out a sentence or phrase. The advantage of this mechanism is that it retains the naturalness and understandability of the speech. On the other hand the downside of this process is that it doesn't address the problem of spectral discontinuity. This problem can be reduced by techniques such as spectral smoothing, manual adjustment of unit boundaries, use of longer units to reduce the number of concatenations etc. The downside of using spectral smoothing is that it decreases the naturalness of the resulting speech. We do not address this problem in this implementation, which can be further improved upon in future. Controlling prosodic features has been found very difficult and the synthesized speech still sounds usually synthetic or monotonic. Techniques such as Artificial Neural Networks and Hidden Markov Models have been applied to speech synthesis. These methods have been useful for controlling the synthesizer parameters, such as duration, fundamental frequency, etc.

The concatenative speech synthesis process has been modeled as a sequential circuit and it is synthesized and implemented in VHDL. We made use of Xilinx tools and XS tools to download the design onto the XSV-800 board which contains a FPGA(v800hq240-4). Its functionality has been verified and illustrated with the waveforms. In future, visual text-to-speech will be the state of the art, which is defined as the synchronization of facial image with the synthetic speech.

REFERENCES

[1] E. J. Yannakoudakis and P. J. Hutton, *Speech Synthesis and Recognition Systems*, Ellis Horwood Limited, 1987.

[2] Murtaza Bulut, Shrikanth S. Narayanan, and Ann K. Syrdal, "Expressive speech synthesis using a concatenative synthesizer," in *Proceedings of International Conference on Spoken Language Processing*, sept 2002.

[3] M. H. O'Malley, "Text-to-Speech conversion Technology," *IEEE Computer Journal*, vol. 23, no. 8, pp. 17–23, Aug 1990.

[4] A. Acero, H. Hon, and X. Huang, *Spoken Language Processing: A guide to Theory, Algorithm, and System Development*, Prentice Hall PTR, 2001.

[5] J. P. Olive, A. Greenwood, and J. S. Coleman, *Acoustics of American English Speech: a Dynamic Approach*, Springer-Verlag, 1993.

[6] M. Plumpe, A. Acero, H. Hon, and X. Huang, "HMM-Based Smoothing For Concatenative Speech Synthesis," in *The 5th International Conference on Spoken Language Processing*, Nov-Dec 1998.

[7] David Chappell and John H. L. Hansen, "Spectral Smoothing for Concatenative Speech Synthesis," in *International Conference on Spoken Language Processing*, Dec 1998, pp. 1935–1938.

[8] Richard V. Cox, Candace A. Kamm, Lawrence R. Rabiner, and Juergen Schroeter, "Speech and Language Processing for NExt-Millennium Communications Services," *Proceedings of The IEEE*, vol. 88, no. 8, pp. 1314–1337, Aug 2000.

[9] J. M. Pickett, J. Schroeter, C. Bickley, A. Syrdal, and D. Kewelyport, *The Acoustics of Speech Communication*, Allyn and Bacon, Boston, MA, 1998.

[10] B. H. Juang, " Why speech synthesis? (in memory of Prof. Jonathan Allen, 1934-2000) ," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 1, pp. 1–2, Jan 2001.

[11] R. Baker and Fausto Poza, "Natural speech from a computer," in *Proceedings of ACM National Conference*, 1968, pp. 795–800.

[12] W. S. Y. Wang and G. E. Peterson, "A study of the building blocks in speech," *Journal Acoustical Society of America*, vol. 30, pp. 743, 1958.

[13] Phuay Hui Low and Saeed Vaseghi, "Synthesis of unseen context and spectral and pitch contour smoothing in concatenated text to speech synthesis," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 2002, pp. 469–472.

[14] K. Lukaszewicz and M. Karjalainen, "A Microphonemic method of speech synthesis," in *International Conference on Acoustics, Speech, and Signal Processing*, Apr 1987, pp. 1426–1429.

[15] Y. Stylianou, " Removing linear phase mismatches in concatenative speech synthesis ," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 3, pp. 232–239, Mar 2001.

[16] D. O'Brien and A. I. C. Monaghan, "Concatenative synthesis based on a harmonic model," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 1, pp. 11–20, Jan 2001.

[17] R. J. McAulay and T. F. Quatieri, "Speech analysis/synthesis based on a sinusoidal representation," *IEEE Transactions on Acoustics, Speech, Signal Processing*, vol. 34, pp. 744–754, Aug 1986.

[18] J. Wouters and M. W. Macon, " Control of spectral dynamics in concatenative speech synthesis," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 1, pp. 30–38, jan 2001.

[19] Mark Tatham and Eric Lewis, "Improving text-to-speech synthesis," in *International Conference on Spoken Language Processing*, Oct 1996, pp. 1856–1859.

[20] E. Lewis and M. A. A. Tatham, "SPRUCE- A New Text-to-Speech Synthesis System," in *Proceedings 2nd European Conference on Speech Communication and Technology*, 1991, pp. 1235–1238.

[21] Y. Stylianou, "Assessment and correction of Voice Quality Variabilities in Large Speech Databases for Concatenative Speech Synthesis," in *International Conference on Acoustics, Speech and Signal Processing*, Mar 1999, pp. 377–380.

[22] M. Morris Mano and Charles R. Kime, *Logic and Computer Design Fundamentals*, Prentice Hall, Upper Saddle River, NJ 07458, 2001.

[23] M. J. Smith, *Application Specific Integrated Circuits*, Addison-Wesley, 1997.

[24] John L. Hennessy and David A. Patterson, *Computer Organization and Design The Hardware/software Interface*, Morgan Kaufmann Publishers, Inc., San Fransico, CA, 1994.

[25] Jan M. Rabaey, *Digital Integrated circuits: A design Perspective*, Prentice-Hall of India Private Limited, 2002.

[26] Jorgen Pedderson, *Audio Project*, http://www.itee.uq.edu.au/∼peters/xsvboard/audio/audio.htm, 2001.

APPENDICES

Appendix A

```
    /* The program to implement the speech synthesis algorithm */
#include <string.h>
#include <strings.h>
#include <math.h>
#include <stdio.h>
#include <engine.h>
#define MAXPHONEMES 500
#define MAXIDLEN 32

    /* Main program */
int main()
{
     int i, j;
    Engine *ep;
    char S[MAXIDLEN];
    char answer[MAXIDLEN];
    char main_word[MAXIDLEN];
    char *init_word;
    char temp2[MAXIDLEN];
    char *ptr;
    char temp[MAXIDLEN];
    char token[10*MAXIDLEN], token2[10*MAXIDLEN];
    char newsubstr[MAXIDLEN];
    char lib[MAXPHONEMES][MAXIDLEN];
    int count=0, index=0, libsize=0, found=0, charcount=0, base=0;
    int phoneme_size=0;
    FILE *fp;
    /* Read the phonemes.txt file */
```

Appendix A (Continued)

```
fp=fopen(”phonemes.txt”, ”r”);
index=0;
while(fscanf(fp, ”%s”, lib[index]) != EOF)
{
      index++;
}
libsize=index;
for(index=0; index <libsize; index++)
 {
      printf(”%s ”, lib[index]);
 }

      /* ask for the input word */
printf(” Enter the string to be parsed: ”);
scanf(”%s”, S);

     /* Calling the MATLAB program */
if (!(ep = engOpen(”\0”)))
    {
      fprintf(stderr, ” Can’t start MATLAB engine”);
       return EXIT_FAILURE;
    }
count=1;
charcount=0;
base=0;
while(charcount < strlen(S))
 {
    for(phoneme_size =4; phoneme_size ≥1; phoneme_size–)
```

Appendix A (Continued)

```
{
  for(i=0; i<phoneme_size; i++)
   {
       temp[i] = S[base+i];
   }
  temp[i]='\0';
  printf("Looking in Library for string: %s of length %d ", temp, strlen(temp));
  found=0;
  for (i=0; i< libsize; i++)
    {
      strcpy(temp2,lib[i]);
     init_word = temp2;
     ptr = strchr(init_word, ',');
     if(ptr) *ptr = '\0';
      strcpy (main_word,init_word);
      /* if (strncmp(temp,main_word,strlen(temp))==0)*/
     if (strcmp(temp,main_word)==0)
          {
            strcpy (answer, main_word);
             printf(" main_word = %s", main_word);
             printf(" The output is %s ", answer);
             found = 1;
          }
     if(found == 0)
         {
           if (ptr)
             {
                 init_word = ptr +1;
```

Appendix A (Continued)

```
                ptr = strchr(init_word, ',');
                while(ptr)
                  {
                     *ptr = '\0';
                     /*if (strncmp(temp,init_word,strlen(temp))==0)*/
                     if (strcmp(temp,init_word)==0)
                       {
                          strcpy (answer, main_word);
                          found = 1;
                       }
                     if(found == 1) break;
                      init_word = ptr+1;
                      ptr = strchr(init_word,',');
                   }
              }
          }
      if(found == 1)
         {
           printf(" The answer is %s ",answer);
          printf("Found a phoneme in the library : %s ", lib[i]);
          sprintf(token, "[X%d,fs, nbits] = wavread('sounds/%s.wav');", count, answer);
           printf("token = %s", token);
           engEvalString(ep, token);
           count++;
          found=1;
           base=base+phoneme_size;
           charcount = charcount + phoneme_size;
          break;
```

Appendix A (Continued)

```
                }
            }
          if(found == 1) break;
       }
     if(found == 0)
        {
           printf("Failed to find a matching phoneme for %s ", temp);
           printf("Aborting!!");
           engClose(ep);
           exit(1);
        }
   }
/* engEvalString(ep, "[X,fs,nbits] = wavread('ca.wav');");
 engEvalString(ep, "[Y,fs,nbits] = wavread('ar.wav');");
 engEvalString(ep, "C = [X;Y];");*/

     printf("count = %d ", count);
 strcpy(token,"[");
 for(i=1; i< count; i++)
 {
    if(i==1)
     {
        sprintf(newsubstr, "X%d", i);
     }
   else
     {
        sprintf(newsubstr, ";X%d", i);
     }
```

Appendix A (Continued)

```
        printf("%s", newsubstr);
        strcpy(token,strcat(token, newsubstr));
        printf("%s", token);
      }
    strcpy(token, strcat(token,"]"));
    printf("token = %s", token);
    sprintf(token2, "C = %s;", token);
    printf("token2 = %s", token2);
    engEvalString(ep, token2);
    engEvalString(ep, "wavwrite(C,fs,nbits,'word.wav');");
    engEvalString(ep, "close;");
    engClose(ep);
}
```