A MODEL SPEECH ANALYSIS - SYNTHESIS SYSTEM USING MATLAB AND SYSTEMC

- Vigil Varghese

CONTENTS

CHAPTER 1

ABSTRACT

ESL design methodology specifies having an executable model as its first step. In the digital design of a speech analysis-synthesis system a MATLAB model is developed first. After examining the feasibility of the idea a SYSTEMC model is developed. The model is an interdisciplinary one, in that, data is exchanged to and fro between the MATLAB and SYSTEMC domains. The design is done for minimum phase systems. The characteristics of the cepstrum, which makes them suitable for a variety of speech processing tasks, are also explored. Additional features that make the model a more complete one are also dealt with in detail.

CHAPTER 2

INTRODUCTION

Speech processing has come a long way from the early telephone transducers to the present automatic speech recognition (ASR) systems and speech synthesizers. This tremendous growth in a span of four decades can be largely attributed to the developments in the field of digital signal processing. By sampling a continuous time signal such as speech, one can make use of the immense computational capabilities of a computer. Such computational power gives the freedom to implement any speech processing algorithm using the conventional computers. But while trying to implement such algorithms in hardware, one encounters a number of problems.

In such situations, two of the most recent technologies that are being developed come to the rescue. One is Electronic System Level (ESL)[1] design and the other is SYSTEMC[2]. Their contributions in the development of a digital system can never be underestimated. ESL is a design methodology that aids in faster development of digital systems by establishing a set of loosely bound rules and a succinct framework. SYSTEMC on the other hand satisfies the requirements laid down by ESL in addition to providing a platform for hardware – software co-design.

Speech analysis – synthesis system essentially consist of two modules. One of which is a speech analyzer and the other a speech synthesizer. In the remainder of the document, basics required in developing a speech analysis – synthesis system is given along with the MATLAB and SYSTEMC models.

[1] ESL has been around for about 15 years, but still it is not well understood
[2] Currently in its second version

CHAPTER 3

HOMOMORPHIC SIGNAL PROCESSING

Homomorphic signal processing is a class of nonlinear signal processing techniques. This class of techniques has found application in a number of fields, including image enhancement, speech analysis and seismic exploration.

Homomorphic systems are based on a generalization of the class of linear systems. Linear shift invariant systems are important because they are relatively easy to analyze and characterize, leading to rather elegant and powerful mathematical representations, and because it is possible to design linear shift invariant systems to perform a variety of useful signal processing functions. For example, if we are given a signal that is a sum of two component signals whose Fourier transforms occupy different frequency bands, then it is possible to separate the two components with a linear filter. The fact that linear systems are relatively easy to analyze and are useful in separating signals combined by addition is a direct consequence of the property of superposition, which defines the class of linear systems. This observation leads to the consideration of classes of nonlinear systems that obey a generalized principle of superposition. Such systems are called homomorphic systems.

3.1 GENERALIZED SUPERPOSITION

The principle of superposition as it is stated for the linear systems requires that, if L is the system transformation, then for any two inputs $s_1(n)$ and $s_2(n)$ and any scalar α,

$$L[s_1(n) + s_2(n)] = L[s_1(n)] + L[s_2(n)] \qquad 3.1a$$

and

$$L[\alpha\, s_1(n)] = \alpha\, L[s_1(n)] \qquad 3.1b$$

To allow the separation of signals that are nonlinearly combined, Oppenheim introduced the concept of generalized superposition. To generalize the principle, let us denote by
a rule for combining inputs with each other (e.g. addition, multiplication, convolution, etc.) and by : a rule for combining inputs with scalars. Similarly, ○ will denote a rule for

combining system outputs and ¬ a rule for combining outputs with scalars. Then, with H denoting the system transformation, the equations (3.1) are generalized as given below

$$H[s_1(n) \quad s_2(n)] = H[s_1(n)] \circ H[s_2(n)]$$

3.2a

and

$$H[\alpha : s_1(n)] = \alpha \neg H[s_1(n)] \qquad 3.2b$$

Such systems are said to obey a generalized principle of superposition with an input operation   and an output operation ○. Such systems are depicted as in Fig. 3.1. Linear systems are a special case for which   and ○ are addition and : and ¬ are multiplication.

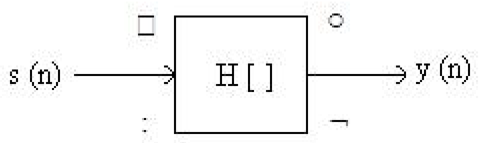


Fig. 3.1: Representation of a homomorphic system

3.2 CANONIC REPRESENTATION

It can be shown that if the system inputs constitute a vector space with   and : corresponding to vector addition and scalar multiplication and the system outputs constitute a vector space with ○ and ¬ corresponding to vector addition and scalar multiplication, then all systems of this class can be represented as a cascade of three systems, as shown in Fig. 3.2.

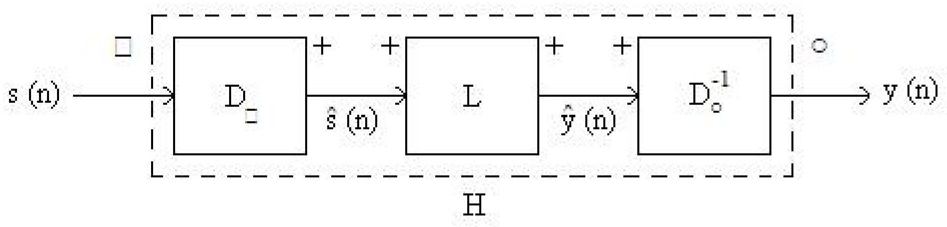


Fig. 3.2: Canonic representation of homomorphic systems

This cascade of fig. 3.2 is referred to as the canonic representation of homomorphic systems. The first system, D , has the property that

$$D\ \ [s_1(n) \qquad s_2(n)] = D\ \ [s_1(n)] + D\ \ [s_2(n)] = \hat{s}_1(n) + \hat{s}_2(n)$$

3.3a

and

$$D\ \ [\alpha : s_1(n)] = \alpha\, D\ \ [s_1(n)] = \alpha\, \hat{s}_1(n)$$

3.3b

It can be observed that D obeys a generalized principle of superposition where the input operation is and the output operation is +. The effect of the system D is to transform the combination of the signals $s_1(n)$ and $s_2(n)$ according to the rule into a conventional linear combination of corresponding signals D $[s_1(n)]$ and D $[s_2(n)]$. The system L is a conventional linear system, so

$$L[\hat{s}_1(n) + \hat{s}_2(n)] = L[\hat{s}_1(n)] + L[\hat{s}_2(n)] = \hat{y}_1(n) + \hat{y}_2(n) \qquad 3.4a$$

and

$$L[\alpha\, \hat{s}_1(n)] = \alpha\, L[\hat{s}_1(n)] = \alpha\, \hat{y}_1(n) \qquad 3.4b$$

Finally, the system $D_\circ^{-1}$ transforms from addition to ○, so

$$D_\circ^{-1}[\hat{y}_1(n) + \hat{y}_2(n)] = D_\circ^{-1}[\hat{y}_1(n)] \circ D_\circ^{-1}[\hat{y}_2(n)] = y_1(n) \circ y_2(n) \qquad 3.5a$$

and

$$D_\circ^{-1}[\alpha\, \hat{y}_1(n)] = \alpha \neg D_\circ^{-1}[\hat{y}_1(n)] = \alpha \neg y_1(n) \qquad 3.5b$$

Since the system D is fixed by the operations and :, it is characteristic of the class and is therefore called the characteristic system for the operation . Similarly, $D_\circ$ is the characteristic system for the operation ○. Furthermore, it is clear that all homomorphic systems with the same input and output operations differ only in the linear part.

This result is of fundamental importance, because it implies that once the characteristic systems for the class have been determined, we are left with a linear filtering problem. The practical importance of homomorphic systems for speech processing lies in their capability of transforming nonlinearly combined signals to additively combined signals so that linear filtering can be performed. This capability stems from the fact that homomorphic systems can be expressed as a cascade of three homomorphic sub-systems, as shown in Fig. 3.2.

3.3 HOMOMORPHIC SYSTEMS FOR CONVOLUTION

There are a multitude of signal processing problems where signals are combined by convolution. In communicating or recording in a multipath or reverberant environment, the effect of the distortion introduced can be modeled in terms of noise convolved with the desired signal. In speech processing, it is often of interest to isolate the effects of vocal tract impulse response and excitation which, at least on a short time basis, can be considered as having been convolved to form the speech waveform.

A common approach to separating the components of such signals, i.e. to deconvolution, is linear inverse filtering. Unfortunately, since linear systems are not matched to the structure of convolutional combinations, inverse filtering requires detailed knowledge of one of the component signals. As an alternative, we are led to consider a class of homomorphic systems which obeys a generalized principle of superposition for convolution.

3.3.1 CANONIC SYSTEM FOR CONVOLUTION

Consider a sequence s(n) combined by convolution of two sequences $s_1(n)$ and $s_2(n)$, i.e.,

$$s(n) = s_1(n) * s_2(n) \qquad 3.6$$

The canonic form for homomorphic systems, under convolution is depicted in Fig. 3.3.

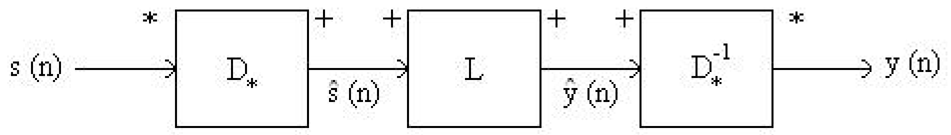


Fig. 3.3: Canonic system for convolution

The characteristic system, D_*, has the property

$$D_*[s_1(n) * s_2(n)] = D_*[s_1(n)] + D_*[s_2(n)] = \hat{s}_1(n) + \hat{s}_2(n) \qquad 3.7a$$

and

$$D_*[\alpha : s_1(n)] = \alpha\, D_*[s_1(n)] = \alpha\, \hat{s}_1(n) \qquad 3.7b$$

The inverse characteristic system, D_*^{-1}, is given by

$$D_*^{-1}[\hat{y}_1(n) + \hat{y}_2(n)] = D_*^{-1}[\hat{y}_1(n)] * D_*^{-1}[\hat{y}_2(n)] = y_1(n) * y_2(n) \qquad 3.8a$$

and

$$D_*^{-1}[\alpha\ \hat{y}_1(n)] = \alpha : D_*^{-1}[\hat{y}_1(n)] = \alpha : y_1(n) \qquad 3.8b$$

In theory, any system that obeys superposition for addition can be used as a linear system L in the canonic system of the Fig. 3.3.

CHAPTER 4

SPEECH ANALYSIS-SYNTHESIS SYSTEM

Many speech analysis-synthesis systems are directed toward a separation of the speech excitation function and the vocal tract impulse response. Typically, the excitation function is characterized by a measurement of the pitch period, and the vocal tract impulse response is characterized by samples of the spectral envelope. In speech analysis, the underlying parameters of the speech model are estimated, and in synthesis, the waveform is reconstructed from the model parameter estimates. It will not be an exaggeration to say that the characteristic system D_* and its inverse D_*^{-1} can be viewed as speech analysis and synthesis systems.

The model presented here performs the operations of the systems D_* and D_*^{-1}. It is different from the conventional speech analysis-synthesis systems, because in conventional systems, speech parameters are estimated during analysis and they are used to reconstruct the original speech waveform during synthesis. In conventional analysis systems, the speech parameters such as pitch and voicing or unvoicing estimate are obtained along with the vocal tract impulse response. In synthesis systems, these parameters are used to generate the original speech waveform.

The model presented here differs from the conventional one, as it does not calculate the pitch and voicing or unvoicing estimate. The output of the system D_* is passed on as it is to the input of the inverse system D_*^{-1}. The purpose of this model is to highlight the characteristics of the output of D_*. However, with few modifications to the model it can be made like the conventional system. Later it will shown that from the output of the system D_*, how easily the pitch and voicing or unvoicing estimate can be obtained. Since the speech waveform can be viewed as a convolution of the vocal tract impulse response and the excitation function, homomorphic deconvolution can be used to separate them.

4.1 PHASE CONSIDERATIONS

We can classify the vocal tract impulse response with respect to the poles and zeros of the Z transform of the sequence. The three general classifications are given below

4.1.1 MINIMUM PHASE SEQUENCES

For minimum phase sequences the poles and zeros are inside the unit circle. The sequences are of the form[3]

$$V(z) = \frac{\prod_{k=1}^{mi} (1-a_k z^{-1})}{\prod_{k=1}^{pi} (1-b_k z^{-1})} \qquad 4.1$$

where $|a_k|$ and $|b_k|$ are less than unity.

4.1.2 MAXIMUM PHASE SEQUENCES

For maximum phase sequences the poles and zeros are outside the unit circle. The sequences are of the form

$$V(z) = \frac{\prod_{k=1}^{mo} (1-a_k z)}{\prod_{k=1}^{po} (1-b_k z)} \qquad 4.2$$

where $|a_k|$ and $|b_k|$ are less than unity.

4.1.3 MIXED PHASE SEQUENCES

For mixed phase sequences the poles and zeros can be either inside or outside the unit circle. The sequences are of the form

$$\prod^{mi} (1-a_k z^{-1}) \prod^{mo} (1-b_k z)$$

[3] Here the linear phase term z^r and the constant A are considered as unity

$$V(z) = \frac{{}_{k=1} \qquad\qquad {}_{k=1}}{\prod_{k=1}^{pi} (1-c_k z^{-1}) \prod_{k=1}^{po} (1-d_k z)} \qquad 4.3$$

where $|a_k|$, $|b_k|$, $|c_k|$ and $|d_k|$ are less than unity.

4.2 ANALYZER CONFIGURATION

Mathematical representation of the analyzer (which is nothing but the characteristic system D_*)[4] can be done either by using Z transform or Discrete Time Fourier Transform (DTFT). But since Discrete Fourier Transform (DFT) is more robust and computationally feasible (because of FFT), DFT will be used for further analysis. The DFT is a sampled (in frequency) version of the DTFT of a finite-length sequence and provides a good approximation to the analysis. The figure below shows the speech analyzer configuration.

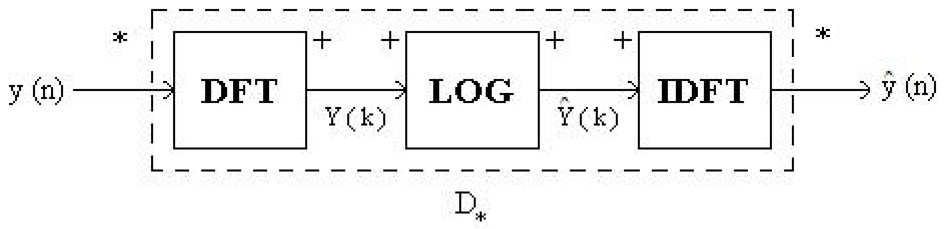


Fig. 4.1: Speech analyzer

Here y(n) is a speech sample. It is the convolution of periodic impulse train x(n) and vocal tract impulse response v(n) i.e.,

$$y(n) = x(n) * v(n) \qquad 4.4$$

DFT of y(n) is given by

$$Y(k) = \sum_{n=0}^{N-1} y(n)\, e^{-j(2\pi k/N)n} \qquad 4.5$$

Taking log on eq. 4.5 we get

$$\hat{Y}(k) = \log|Y(k)| + j \arg\{Y(k)\} \qquad 4.6$$

Finally taking IDFT on eq. 4.6 we get

[4] The reasons have been explained at the beginning of chapter 4

$$\hat{y}(n) = 1/N \sum_{k=0}^{N-1} \hat{Y}(k)\, e^{j(2\pi k/N)n} \qquad 4.7$$

Since DFT is a finite length sequence, using DFT instead of DTFT results in time domain aliasing. The effect of time domain aliasing can be made negligible by using a large value for N.
As mentioned earlier, this is just meant to be a model and so the pitch and voicing or unvoicing estimate along with the low quefrency[5] components, are not calculated, as would be done in conventional analysis systems.

4.2.1 COMPLEX LOGARITHM

A more serious problem in computation of the output is the computation of the complex logarithm. The real and imaginary parts of the complex logarithm by writing the logarithm in polar form, is given as

$$\log[Y(k)] = \log(|Y(k)|\, e^{j \arg[Y(k)]}) = \log(|Y(k)|) + j \arg[Y(k)] \qquad 4.8$$

Then, if $Y(k) = X(k)V(k)$, we want the logarithm of the real parts and the logarithm of the imaginary parts to equal the sum of the respective logarithms. The real part is the logarithm of magnitude and, for the product $X(k)V(k)$, is given by

$$\log(|Y(k)|) = \log(|X(k)V(k)|) = \log(|X(k)|) + \log(|V(k)|) \qquad 4.9$$

provided that $|X(k)| > 0$ and $|V(k)| > 0$. In this case there is no problem with uniqueness and additivity of the logarithms. The imaginary part of the logarithm is the phase of the Fourier transform and requires more careful consideration. As with the real part, we want the imaginary parts to add

$$\arg[Y(k)] = \arg[X(k)V(k)] = \arg[X(k)] + \arg[V(k)] \qquad 4.10$$

The relation in eq. 4.10 , however, generally does not hold due to the ambiguity in the definition of phase, i.e., $\arg[Y(k)] = PV\{\arg[Y(k)]\} + 2\pi k$, where k is any integer value, and where PV denotes the principle value of the phase which falls in the interval $[-\pi , \pi]$.

[5] Will be discussed shortly

Since an arbitrary multiple of 2π can be added to the principal phase values of X(k) and V(k) , the additivity property generally does not hold. The way out of this problem is to unwrap the phase, in order to ensure continuity, by using phase unwrapper algorithms.

4.2.2 REAL AND COMPLEX CEPSTRUM

In the eq. 4.7, $\hat{y}(n)$ is known as the complex cepstrum. As it is evident from the equation, the real and the imaginary parts of the logarithm have been used for the computation of the complex cepstrum. If the imaginary part, which is nothing but the phase, is discarded and we consider only the logarithm of the magnitude, $\log|Y(k)|$, then we get the real cepstrum. For minimum phase sequences, this approximation does not seem to affect the output much. Hence in the model to be presented here, minimum phase sequences are considered and consequently the phase has been discarded. In short, real cepstrum has been calculated. It can be seen that the real part of the complex cepstrum, denoted as c(n), is given by $c(n) = (\hat{y}(n) + \hat{y}(-n))/2$, which is nothing but the real cepstrum. Applying an inverse transform to a log-spectrum makes the real and complex cepstra a function of the time index n. This time index is sometimes referred to as “quefrency”. There are certain techniques for approximating the complex cepstrum from the real cepstrum and these will be dealt with later.

4.3 SYNTHESIZER CONFIGURATION

By performing the inverse of the operations performed during analysis, we obtain the original speech waveform. Although, this is not how a conventional speech synthesis system would work, for illustrating the characteristics of the cepstrum and other concepts, this model would suffice. The figure below shows the speech synthesizer configuration

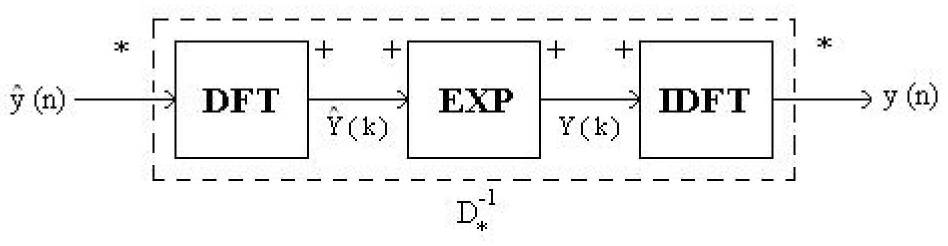


Fig. 4.2: Speech synthesizer configuration

The mathematical representation of the synthesizer (which is nothing but the inverse characteristic system D_*^{-1}) follows simply from the representation of the analyzer. By definition,

$$D_*^{-1}[D_*[y(n)]] = y(n) \qquad 4.11$$

There is no problem of additivity or uniqueness with the inverse exponential operator since $e^{a+b} = e^a e^b$. Thus, along with the forward and inverse DFT's, addition is unambiguously mapped back to convolution.

CHAPTER 5

MATLAB MODEL

In the previous chapters, the basics required for developing a speech analysis-synthesis system was given. In the present chapter a MATLAB model will be developed. As mentioned earlier, MATLAB allows for testing the "proof of concept" in addition to generating synthetic speech and verifying the output. The figure below shows the MATLAB model

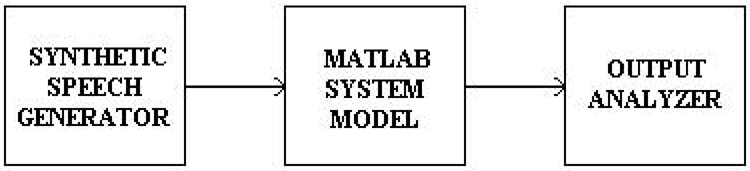


Fig. 5.1: MATLAB model of speech analysis-synthesis system

5.1 SYNTHETIC SPEECH GENERATOR

The synthetic speech generator, creates synthetic speech by convolving the vocal tract impulse response v(n) with the periodic impulse train x(n). Since, for minimum phase sequences discarding the phase does not affect the output much, the following program creates impulse response sequence which is minimum phase. The MATLAB program which generates synthetic speech is given below:

MATLAB PROGRAM:

```
N=1024;    % N for FFT and IFFT
Fs=8000;   % Sampling rate = 8000 Hz
```

```
% Vocal tract response: v[n]
                              % Poles
p1 = .1*exp(j*.3*pi);
p2 = -.8*exp(-j*.8*pi);
p3 = -.2*exp(j*.2*pi);
p4 = .32*exp(-j*.6*pi);
                              % Zeros
z1 = .83*exp(j*.5*pi);
z2 = -.73*exp(-j*.1*pi);

K = 1;
P = [p1, p2, p3, p4]';
Z = [z1, z2]';
[B,A] = zp2tf(Z,P,K);
impulse = zeros(1, 100);
impulse(1) = 1.0;
v = filter(B,A,impulse);
t = [v(3:length(v)), 0, 0];
v = t;
% Glottal pulse: g[n]
                              % Two poles outside unit circle
eq = (.9).^(0:99);
g = conv(eq, eq);
g = g(length(g):-1:1);
g = g(180:199);

% Impulse train: p[n]

p = zeros(1, 1000);
beta = 0.70;
p(1) = 1.0;
p(201) = beta^1;
p(401) = beta^2;
```

```
p(601) = beta^3;
p(801) = beta^4;
```

% Glottal pulse train: x[n]

```
x = conv(p, g);
```

% Speech: y[n] = x[n]*v[n]

```
y = filter(B,A,x);
wavplay(y);
```

% Quantize the speech values

```
smax=max(y);
slen=length(y);
for i=1:slen
   y(i)=y(i)/(smax+1);
end
```

%Sample the speech with a sampling rate of Fs hz

```
wavwrite(y,Fs,16,'synthetic');
speech=wavread('synthetic');
pause(.75);
original=(smax+1)*speech;
wavplay(original);
```

% Set N for FFT and IFFT

```
if slen<N
```

```
    original(slen:N)=0;
end
if slen>N
     original=original(1:N);
end
```

% Display speech waveform

```
Ts=(1/Fs);
n=[(1*Ts):Ts:(N*Ts)];
plot(n,original);
title('Synthetic speech waveform');
xlabel('Time in seconds');
ylabel('Amplitude');
```

% Save the values in a file

```
fid=fopen('speech_values.txt','w');
fprintf(fid,' %d ',N);
fprintf(fid,' %d ',Fs);
fprintf(fid,' %f ',original);
fclose(fid);
```

5.2 MATLAB SYSTEM-MODEL

The speech analysis-synthesis system was first created using MATLAB in order to test the proof of concept. The MATLAB system model comprises of the speech analyzer and synthesizer mentioned earlier. After developing a complete MATLAB model a SYSTEMC model will be developed. The MATLAB program of the speech analysis-synthesis system is given below:

MATLAB PROGRAM:

% Get the speech values stored in the file

```
fid=fopen('speech_values.txt','r');
```

```
N1=fscanf(fid,'%d',1);
Fs1=fscanf(fid,'%d',1);
val=fscanf(fid,'%f');

%Speech analysis

trans=fft(val,N1);
mag=abs(trans);
sep=log(mag);
cepstrum=ifft(sep,N1);

%Store cepstrum values in file

fid=fopen('cepstrum.txt','w');
fprintf(fid,' %f ',cepstrum);
fclose(fid);

%Speech synthesis

inver=fft(cepstrum);
ab=abs(inver);
com=exp(ab);
fin=ifft(com);

%Store final values in file

fid=fopen('finvalues.txt','w');
fprintf(fid,' %d',N1);
fprintf(fid,' %d',Fs1);
fprintf(fid,' %f ',fin);
fclose(fid);
```

5.3 OUTPUT ANALYZER

The values generated by the previous stage, is displayed in the appropriate format by the output analyzer. The MATLAB program which does this is given below:

MATLAB PROGRAM:

```
%Get cepstrum values from file

fid=fopen('cepstrum.txt','r');
cepo=fscanf(fid,'%f');
fclose(fid);

%Get final output values from file

fid=fopen('finvalues.txt','r');
No=fscanf(fid,'%d',1);
Fso=fscanf(fid,'%d',1);
fino=fscanf(fid,'%f');
fclose(fid);

% Determine sampling period

Tso=(1/Fso);
no=[(1*Tso):Tso:(No*Tso)];

% Plot cepstrum

figure,plot(no,cepo);
title('Cepstrum');
xlabel('Quefrency in seconds');
ylabel('Amplitude');

%Plot final output waveform

figure,plot(no,fino);
title('Output speech waveform');
```

```
xlabel('Time in seconds');
ylabel('Amplitude');

wavplay(fino);
```

CHAPTER 6

SYSTEMC MODEL

In the previous chapter a MATLAB model was developed. In the present one, a SYSTEMC model will be developed which does the same function as the MATLAB model. The SYSTEMC model is given below:

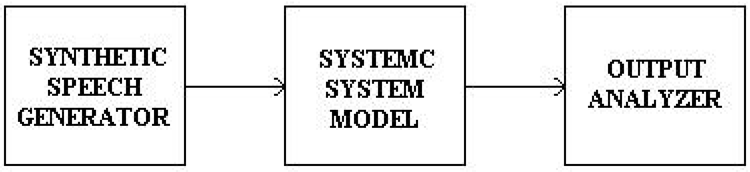


Fig. 6.1: SYSTEMC model of speech analysis-synthesis system

In the above figure, the "synthetic speech generator" and the "output analyzer" are same as that of the MATLAB model. The speech analysis-synthesis system was, however, developed in SYSTEMC. The SYSTEMC program for the SYSTEMC system-model is given below:

<u>SYSTEMC PROGRAM</u>:

```
//|||||||||||||||||| HEADER FILE (hfilter.h) ||||||||||||||||||

#include<systemc.h>
#include<math.h>
#include<stdio.h>
```

```
SC_MODULE(hfilter)
{
        sc_event bitrevre,sta,butter,revimag,maglog,butteri,bitrevi;
        sc_event div,bitrevfil,butterfil,revimagfil;
        sc_event expon,bitrevfinal,butterfinal;

        SC_CTOR(hfilter);

        void bit_reverse_real(void);
        void num_stage(void);
        void butterfly(void);
        void bit_reverse_img(void);
        void abslog(void);
        void butterflyi(void);
        void bit_reverse_reali(void);
        void divN(void);
        void bit_reverse_real_fil(void);
        void butterfly_fil(void);
        void bit_reverse_img_fil(void);
        void expo(void);
        void bit_reverse_real_final(void);
        void butterflyfinal(void);
        void divNfinal(void);
};
```

//||||||||||||||||| *IMPLEMENTATION FILE (hfilter.cpp)* ||||||||||||||||

```
#include "hfilter.h"
#define PI 3.1428571

int N,stage;
float xr[1024],xi[1024],ablog[1024],ablogi[1024],res[256],cep[1024],fin[1024];

hfilter::hfilter(sc_module_name hfil_module):sc_module(hfil_module)
```

```
{
        SC_THREAD(bit_reverse_real);
        SC_THREAD(bit_reverse_img);
        SC_THREAD(num_stage);
        SC_THREAD(butterfly);
        SC_THREAD(abslog);
        SC_THREAD(bit_reverse_reali);
        SC_THREAD(butterflyi);
        SC_THREAD(divN);
        SC_THREAD(bit_reverse_real_fil);
        SC_THREAD(butterfly_fil);
        SC_THREAD(bit_reverse_img_fil);
        SC_THREAD(expo);
        SC_THREAD(bit_reverse_real_final);
        SC_THREAD(butterflyfinal);
        SC_THREAD(divNfinal);
}
//****************************** ANALYZER ****************************

//-------- BIT REVERSE REAL COEFFICIENTS FOR FFT OF ANALYZER ---------

void hfilter::bit_reverse_real(void)
{
        wait(SC_ZERO_TIME);
        int i,j,k;
        float tr;
        j=0;
        for(i=0;i<(N-1);i++)
        {
                if(i<j)
                {
                        tr=xr[j];
                        xr[j]=xr[i];
                        xr[i]=tr;
                }
                k=N/2;
                while(k<=j)
                {
                        j=j-k;
                        k=k/2;
                }
                j=j+k;
        }
        bitrevre.notify();
}
```

//---------- FIND THE NUMBER OF STAGES FOR FFT OF ANALYZER -----------

```
void hfilter::num_stage(void)
{
        wait(SC_ZERO_TIME);
        wait(bitrevre);
        int irem;
        stage=0;
        irem=N;
        while (irem>1)
        {
                irem=irem/2;
                stage=stage+1;
        }
        sta.notify();
}
```

//------------- BUTTERFLY COMPUTATION FOR FFT OF ANALYZER ------------

```
void hfilter::butterfly(void)
{
        wait(SC_ZERO_TIME);
        wait(sta);
        int M,M2,l,j,i,ip,sign=1;
        float ur,ui,wr,wi,temp,tr,ti;
        M=1;
        for(l=1;l<=stage;l++)
        {
                M2=M;
                M=M*2;
                ur=1.0;
                ui=0;
                wr=cos(PI/M2);
                wi=sign*sin(PI/M2);
                for(j=0;j<M2;j++)
                {
                        i=j;
                        while(i<N)
                        {
                                ip=i+M2;
                                tr=xr[ip]*ur-xi[ip]*ui;
                                ti=xi[ip]*ur+xr[ip]*ui;
                                xr[ip]=xr[i]-tr;
                                xi[ip]=xi[i]-ti;
                                xr[i]=xr[i]+tr;
                                xi[i]=xi[i]+ti;
                                i=i+M;
                        }
                        temp=ur*wr-ui*wi;
                        ui=ui*wr+ur*wi;
                        ur=temp;
                }
        }
butter.notify();
}
```

//----- BIT REVERSE IMAGINARY COEFFICIENTS FOR FFT OF ANALYZER -----

```
void hfilter::bit_reverse_img(void)
{
        wait(SC_ZERO_TIME);
        wait(butter);
        int i;
        float ti;
        for(i=1;i<(N/2);i++)
        {
                        ti=xi[i];
                        xi[i]=xi[N-i];
                        xi[N-i]=ti;
        }
        revimag.notify();
}
```

//-------------- MAGNITUDE AND LOGARITHM IN ANALYZER ----------------

```
void hfilter::abslog(void)
{
        wait(SC_ZERO_TIME);
        wait(revimag);
        int i;
        float mag,temp;
        for(i=0;i<N;i++)
        {
                mag=sqrt(pow((xr[i]),2)+pow((xi[i]),2));
                if (mag==0)
                {
                        mag=1;
                }
                temp=log(mag);
                ablog[i]=temp;
        }
        maglog.notify();
}
```

//-------- BIT REVERSE REAL COEFFICIENTS FOR IFFT OF ANALYZER --------

```
void hfilter::bit_reverse_reali(void)
{
        wait(SC_ZERO_TIME);
        wait(maglog);
        int i,j,k;
        float tr;
        j=0;
        for(i=0;i<(N-1);i++)
        {
                if(i<j)
                {
                        tr=ablog[j];
                        ablog[j]=ablog[i];
                        ablog[i]=tr;
                }
                k=N/2;
                while(k<=j)
                {
                        j=j-k;
                        k=k/2;
                }
                j=j+k;
        }
        for(i=0;i<N;i++)
        {
                ablogi[i]=0.0;
        }
        bitrevi.notify();
```

```
}

//----------- BUTTERFLY COMPUTATION FOR IFFT OF ANALYZER ------------

void hfilter::butterflyi(void)
{
        wait(SC_ZERO_TIME);
        wait(bitrevi);
        int M,M2,l,j,i,ip,sign=-1;
        float ur,ui,wr,wi,temp,tr,ti;
        M=1;
        for(l=1;l<=stage;l++)
        {
                M2=M;
                M=M*2;
                ur=1.0;
                ui=0;
                wr=cos(PI/M2);
                wi=sign*sin(PI/M2);
                for(j=0;j<M2;j++)
                {
                        i=j;
                        while(i<N)
                        {
                                ip=i+M2;
                                tr=ablog[ip]*ur-ablogi[ip]*ui;
                                ti=ablogi[ip]*ur+ablog[ip]*ui;
                                ablog[ip]=ablog[i]-tr;
                                ablogi[ip]=ablogi[i]-ti;
                                ablog[i]=ablog[i]+tr;
                                ablogi[i]=ablogi[i]+ti;
                                i=i+M;
                        }
                        temp=ur*wr-ui*wi;
```

```
                    ui=ui*wr+ur*wi;
                    ur=temp;
                }
            }
butteri.notify();
}
```

//------------ DIVIDE EACH TERM BY N FOR IFFT OF ANALYZER ------------

```
void hfilter::divN(void)
{
        wait(SC_ZERO_TIME);
        wait(butteri);
        int i;
        float temp;
        for(i=0;i<N;i++)
        {
                ablog[i]=ablog[i]/N;
                temp=ablog[i];
                cep[i]=temp;
        }
        div.notify();
}
```

//************************* SYNTHESIZER ****************************

//------- BIT REVERSE REAL COEFFICIENTS FOR FFT OF SYNTHESIZER -------

```
void hfilter::bit_reverse_real_fil(void)
{
        wait(SC_ZERO_TIME);
        wait(div);
        int i,j,k;
        float tr;
        j=0;
        for(i=0;i<(N-1);i++)
        {
                if(i<j)
                {
                        tr=ablog[j];
                        ablog[j]=ablog[i];
                        ablog[i]=tr;
                }
                k=N/2;
                while(k<=j)
                {
                        j=j-k;
                        k=k/2;
                }
                j=j+k;
        }
        for(i=0;i<N;i++)
        {
                xr[i]=ablog[i];
        }
        for(i=0;i<N;i++)
        {
                xi[i]=0.0;
        }
        bitrevfil.notify();
}

//---------- BUTTERFLY COMPUTATION FOR FFT OF SYNTHESIZER ------------

void hfilter::butterfly_fil(void)
{
        wait(SC_ZERO_TIME);
        wait(bitrevfil);
        int M,M2,l,j,i,ip,sign=1;
        float ur,ui,wr,wi,temp,tr,ti;
        M=1;
        for(l=1;l<=stage;l++)
        {
                M2=M;
                M=M*2;
                ur=1.0;
                ui=0;
                wr=cos(PI/M2);
                wi=sign*sin(PI/M2);
                for(j=0;j<M2;j++)
                {
                        i=j;
                        while(i<N)
                        {
                                ip=i+M2;
                                tr=xr[ip]*ur-xi[ip]*ui;
                                ti=xi[ip]*ur+xr[ip]*ui;
                                xr[ip]=xr[i]-tr;
                                xi[ip]=xi[i]-ti;
                                xr[i]=xr[i]+tr;
                                xi[i]=xi[i]+ti;
                                i=i+M;
                        }
                        temp=ur*wr-ui*wi;
                        ui=ui*wr+ur*wi;
                        ur=temp;
                }
        }
butterfil.notify();
}
```

//--- BIT REVERSE IMAGINARY COEFFICIENTS FOR FFT OF SYNTHESIZER --

```
void hfilter::bit_reverse_img_fil(void)
{
        wait(SC_ZERO_TIME);
        wait(butterfil);
        int i;
        float ti;
        for(i=1;i<(N/2);i++)
        {
                ti=xi[i];
                xi[i]=xi[N-i];
                xi[N-i]=ti;
        }
        revimagfil.notify();
}
```

//------------------- EXPONENTIATION IN SYNTHESIZER ------------------

```
void hfilter::expo(void)
{
        wait(SC_ZERO_TIME);
        wait(revimagfil);
        int i;
        float temp,mag;
        for(i=0;i<N;i++)
        {
            mag=sqrt(pow((xr[i]),2)+pow((xi[i]),2));
            temp=exp(mag);
            xr[i]=temp;
        }
        expon.notify();
}
```

//------ BIT REVERSE REAL COEFFICIENTS FOR IFFT OF SYNTHESIZER -------

```
void hfilter::bit_reverse_real_final(void)
{
        wait(SC_ZERO_TIME);
        wait(expon);
        int i,j,k;
        float tr;
        j=0;
        for(i=0;i<(N-1);i++)
        {
                if(i<j)
                {
                        tr=xr[j];
                        xr[j]=xr[i];
                        xr[i]=tr;
                }
                k=N/2;
                while(k<=j)
                {
                        j=j-k;
                        k=k/2;
                }
                j=j+k;
        }

        for(i=0;i<N;i++)
        {
                xi[i]=0.0;
        }
        bitrevfinal.notify();
}
```

//---------- BUTTERFLY COMPUTATION FOR IFFT OF SYNTHESIZER -----------

```
void hfilter::butterflyfinal(void)
{
        wait(SC_ZERO_TIME);
        wait(bitrevfinal);
        int M,M2,l,j,i,ip,sign=-1;
        float ur,ui,wr,wi,temp,tr,ti;
        M=1;
        for(l=1;l<=stage;l++)
        {
                M2=M;
                M=M*2;
                ur=1.0;
                ui=0;
                wr=cos(PI/M2);
                wi=sign*sin(PI/M2);
                for(j=0;j<M2;j++)
                {
                        i=j;
                        while(i<N)
                        {
                                ip=i+M2;
                                tr=xr[ip]*ur-xi[ip]*ui;
                                ti=xi[ip]*ur+xr[ip]*ui;
                                xr[ip]=xr[i]-tr;
                                xi[ip]=xi[i]-ti;
                                xr[i]=xr[i]+tr;
                                xi[i]=xi[i]+ti;
                                i=i+M;
                        }
                        temp=ur*wr-ui*wi;
                        ui=ui*wr+ur*wi;
                        ur=temp;
                }
        }
```

```
butterfinal.notify();
}

//---------- DIVIDE EACH TERM BY N FOR IFFT OF SYNTHESIZER -----------

void hfilter::divNfinal(void)
{
        wait(SC_ZERO_TIME);
        wait(butterfinal);
        int i;
        for(i=0;i<N;i++)
        {
                fin[i]=xr[i]/N;
        }
}

//****************************** MAIN PROGRAM ***********************

int sc_main(int argc, char* argv[])
{
        int i,Fs;
        float temp;
        FILE *fp;
        FILE *fg;

        struct in
        {
                float real;
                int real1;
        };
        struct in invalues;

        struct ceps
        {
```

```
        float cep;
    };
    struct ceps cepstrum;

    struct res
    {
        float re;
        int   re1;
    };
    struct res final;
```

// Get values from the speech file

```
    fp= fopen("speech_values.txt","r");
    fscanf(fp," %d",&invalues.real1);
    N=invalues.real1;

    for(i=1;i<N;i++)
    {
        if(i==1)
        {
            fscanf(fp," %d",&invalues.real1);
            Fs=invalues.real1;
        }
        else
        {
        fscanf(fp," %f",&invalues.real);
        xr[i]=invalues.real;
        }
    }
    fclose(fp);

    for(i=0;i<N;i++)
    {
```

```
                temp=xr[i+2];
                xr[i]=temp;
        }
```

//Start the SYSTEMC simulation engine after instantiating the module

```
        hfilter hfilter_inst("hfilter_inst");
        sc_start();
```

// Write cepstrum values to file

```
        fp= fopen("cepstrum.txt","w");

        for(i=0;i<N;i++)
        {
                cepstrum.cep=cep[i];
                fprintf(fp,"  %f",cepstrum.cep);
        }
        fclose(fp);
```

// Write final values to file

```
        fg= fopen("finvalues.txt","w");

        final.re1=N;
        fprintf(fg,"  %d",final.re1);
        final.re1=Fs;
        fprintf(fg,"  %d",final.re1);

        for(i=0;i<N;i++)
        {
                final.re=fin[i];
                fprintf(fg,"  %f",final.re);
        }
        fclose(fg);

        return 0;
}
```

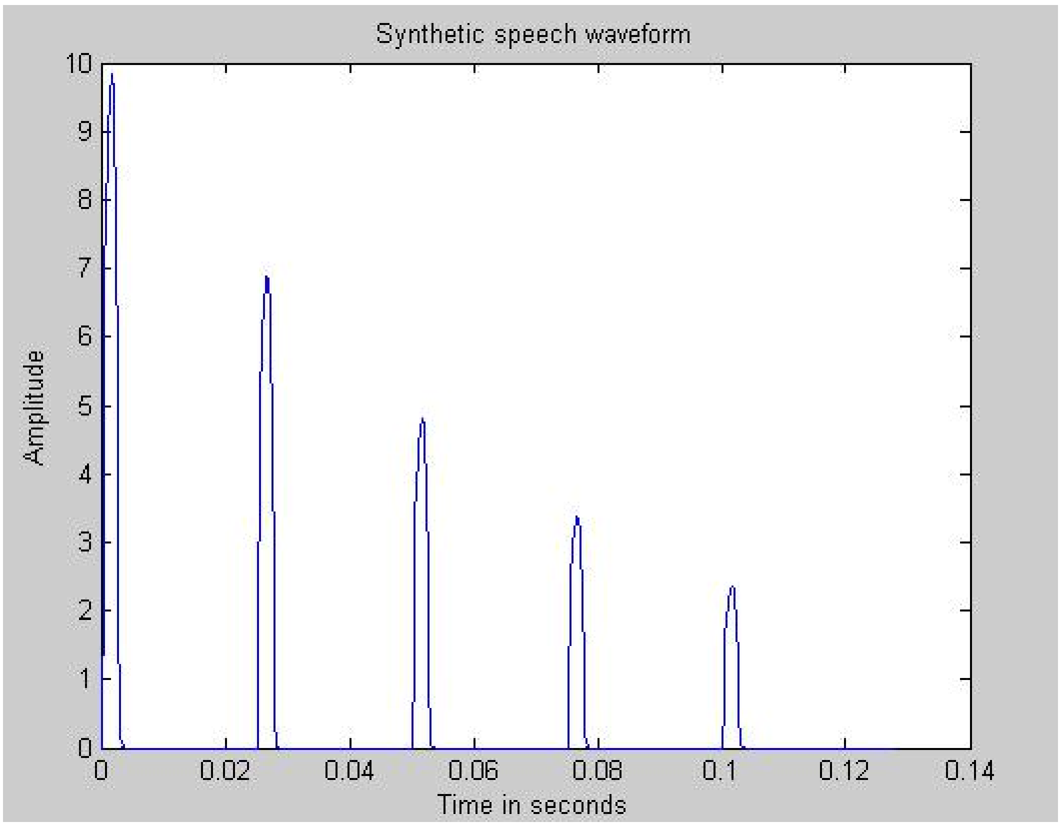
CHAPTER 7

RESULTS AND OBSERVATIONS

The output from the synthetic speech generator is sampled at Fs Hz and contains N points. The resulting values are stored in a file “speech_values.txt”. This is then passed on to the MATLAB and SYSTEMC model respectively. The output from these models namely, the cepstrum values and the final processed values are stored in the files “cepstrum.txt” and “finvalues.txt”. These files are used by the output analyzer to display the cepstrum and processed speech waveform.

7.1 MATLAB MODEL

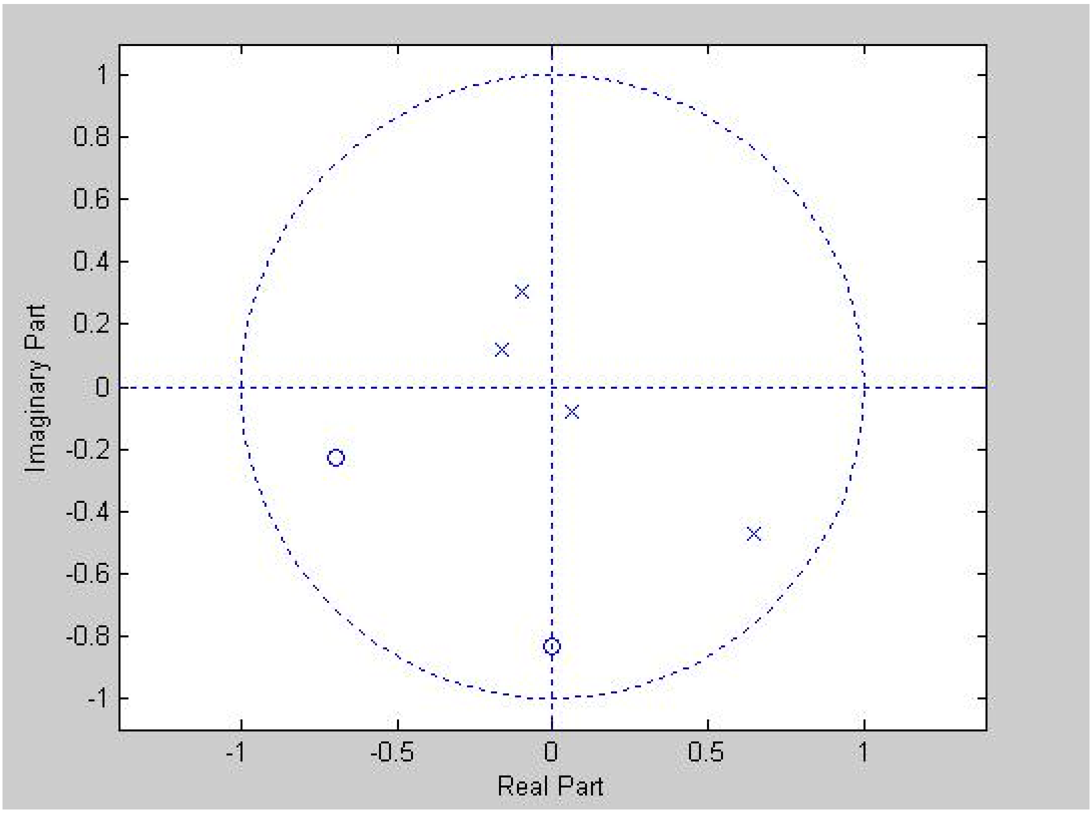
The output from the synthetic speech generator is shown below



Graph 7.1: Synthetic speech waveform

It can be seen that the waveform is periodic on a short time basis. Hence it is the voiced form of the speech and consequently, the pitch can be detected. As mentioned earlier, a minimum phase analysis was performed, which allowed us to discard the phase without compromising much on the accuracy.

The pole zero plot of the vocal tract impulse response is given below

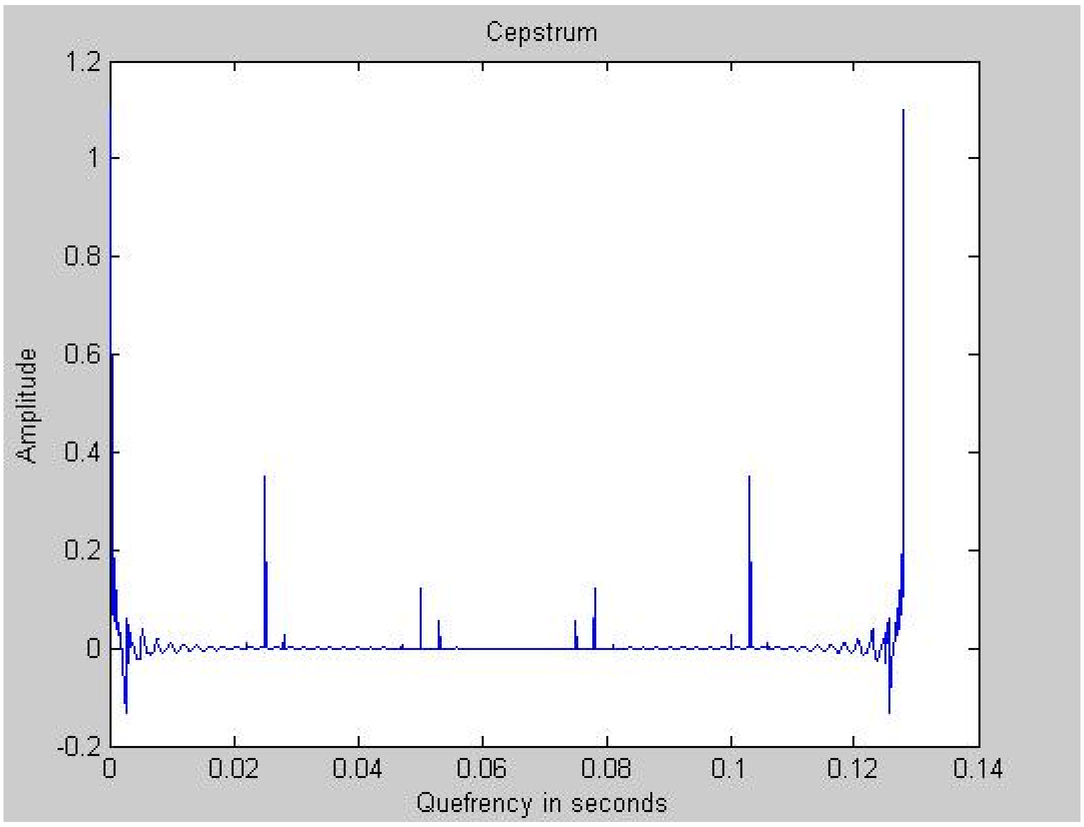


Graph 7.2: Pole zero plot of vocal tract response

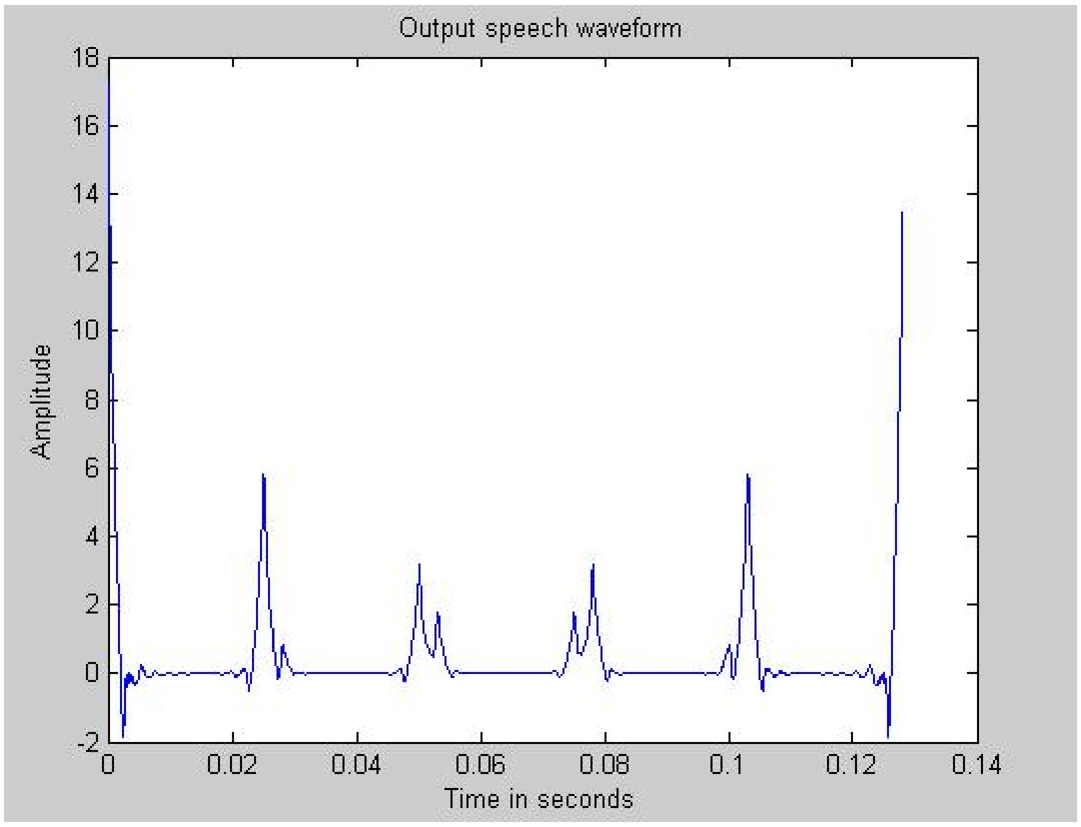
It can be seen that all the poles and zeroes are inside the unit circle. Hence the vocal tract response is minimum phase and consequently the analysis is a minimum phase analysis.

The cepstrum values, as computed by the MATLAB model, when plotted by the output analyzer appears as given in Graph 7.3. From the graph it can be clearly seen that, there are periodic pulses at multiples of pitch period. This again reiterates the fact that the synthetic speech generated was that of the voiced type. At low quefrency region, we can see the presence of the vocal tract response up to a duration of (N/8 * Ts) seconds from the origin. N is the number of points used to compute the DFT and Ts is the sampling period. Another fact that can be observed is that at high quefrency region there is some sort of aliasing that is happening. This could be made negligible by using large values of N.

The final speech waveform, after all the processing has been done is shown in Graph 7.4.



Graph 7.3: Cepstrum from MATLAB model

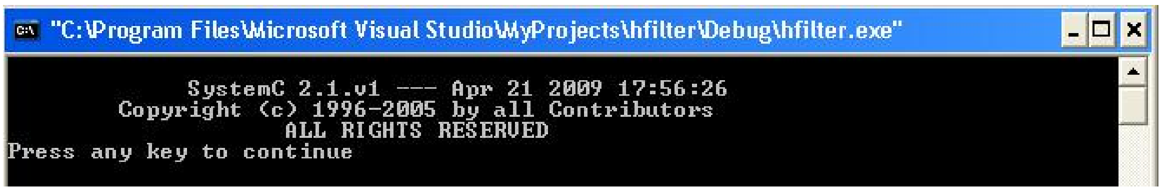


Graph 7.4: Processed speech waveform from MATLAB model

The above graph is similar to Graph 7.1, but not an exact replica. That is understandable because we have made a lot of assumptions in getting to the final model. But the speech sounds produced by both, are nearly the same and are practically indistinguishable.

7.2 SYSTEMC MODEL

The SYSTEMC model was tested using Microsoft Visual C++ 6.0 environment. The output screen after executing the program is given below



```
"C:\Program Files\Microsoft Visual Studio\MyProjects\hfilter\Debug\hfilter.exe"

             SystemC 2.1.v1 --- Apr 21 2009 17:56:26
        Copyright (c) 1996-2005 by all Contributors
                    ALL RIGHTS RESERVED
Press any key to continue
```

Fig. 7.1: SYSTEMC output display

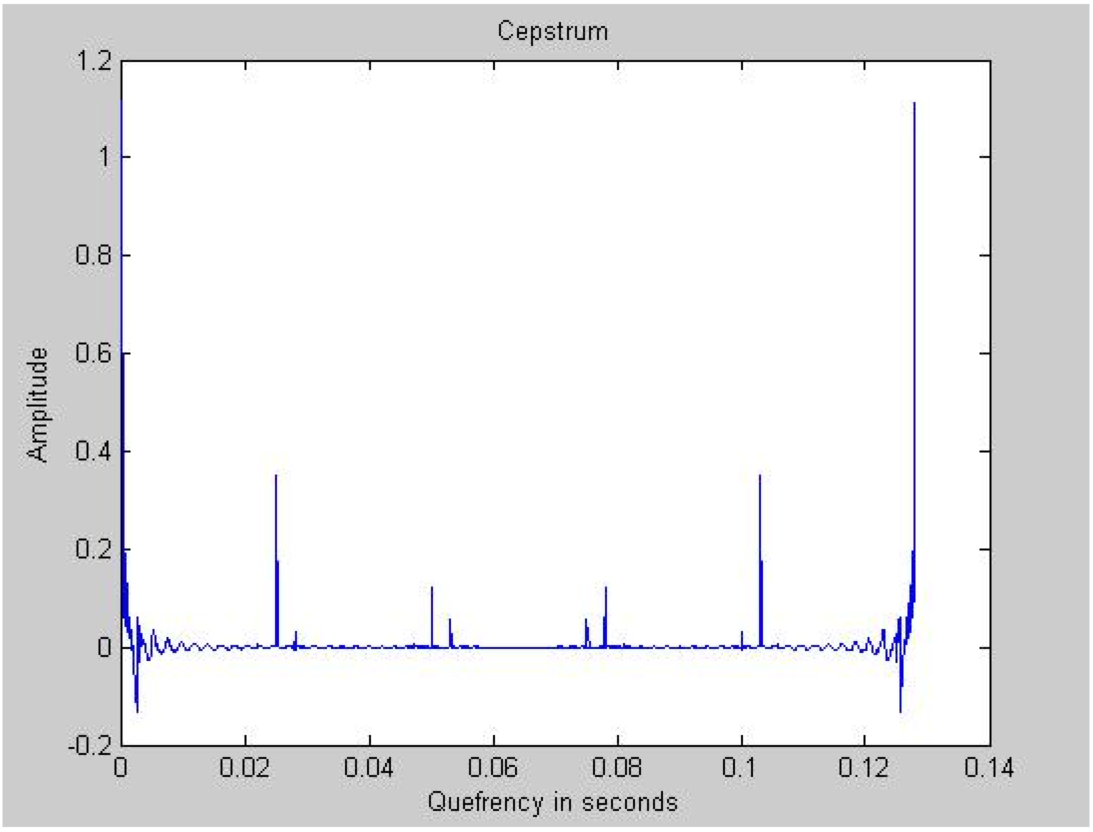
No results are displayed because we are storing them in files.

The synthetic speech generated using the MATLAB model is passed on to the SYSTEMC system-model. The output from the SYSTEMC model is analyzed using the output analyzer of the MATLAB model.

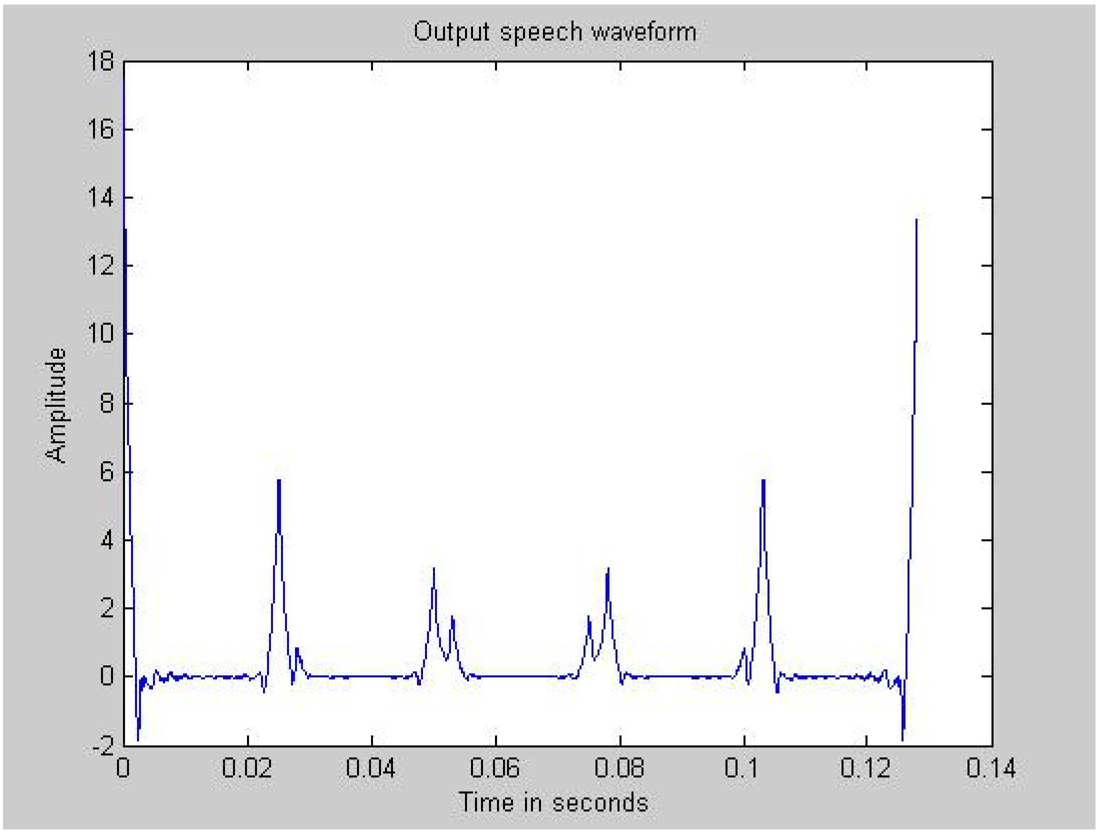
The cepstrum values generated using the SYSTEMC model are plotted using the output analyzer and is given in Graph 7.5.

The final speech waveform after complete processing has been done is given in Graph 7.5.

By comparing the graphs 7.5 and 7.6 with graphs 7.3 and 7.4, we can see that they are similar, which indicates the success of the SYSTEMC model.



Graph 7.5: Cepstrum from SYSTEMC model



Graph 7.6: Processed speech waveform from SYSTEMC model

CHAPTER 8

FURTHER IMPROVEMENTS

The basic model presented in the previous sections can be extended to perform a variety of different functions. Some of them are briefly discussed in this chapter.

8.1 PITCH DETECTION

There are several methods to determine the pitch of a speech waveform. The simplest one makes use of the cepstrum. By observing the cepstrum computed using the models discussed so far, we can conclude that there are distinct peaks occurring in a periodic fashion in the high quefrency region. By detecting such peaks, pitch can be established.

8.2 VOICED OR UNVOICED ESTIMATION

Voicing or unvoicing can be determined from the periodicity of the speech waveform. If the waveform is periodic, then it is of voiced form. Otherwise it is of unvoiced form. Alternately, if the pitch can be determined, then the speech segment is said to be voiced, else it is unvoiced.

8.3 ESTIMATION OF COMPLEX CEPSTRUM

In real cepstrum we discarded the phase, hence to estimate the complex cepstrum we have to estimate the phase. One way to do this would be to take the Hilbert transform of the log-magnitude, which would give a good enough approximation to the phase.

Another approach to determine the complex cepstrum is based on the phase considerations discussed earlier. It can be noted that the complex cepstrum of minimum phase sequences are right sided and that of maximum phase sequences are left sided. Hence for minimum phase sequences the complex cepstrum can be estimated from the real cepstrum as

$$\hat{y}(n) = \begin{cases} c(n), & n = 0, N/2 \\ 2c(n), & 1 \leq n < N/2 \\ 0, & N/2 < n \leq N-1 \end{cases} \qquad 8.1$$

For maximum phase sequences

$$\hat{y}(n) = \begin{cases} 0, & 1 \le n < N/2 \\ c(n), & n = 0, N/2 \\ 2c(n), & N/2 < n \le N-1 \end{cases} \qquad 8.2$$

8.4 MAXIMUM AND MIXED PHASE ANALYSIS

Minimum phase analysis can be performed on maximum phase and mixed phase sequences by using an exponential operator at the input to flip the zeroes and poles outside the unit circle to inside the unit circle. The output is then scaled according to the exponential operator.

If the input y(n) is a maximum or mixed phase sequence, then the input to the system is

$$w(n) = \alpha^n \, y(n) \qquad 8.3$$

α is real and positive.

The output from the system is finally scaled back to the original form

$$\hat{y}(n) = \alpha^n \, \hat{w}(n) \qquad 8.4$$

CHAPTER 9

CONCLUSION

By using the ESL design methodology, chance of failure of a system is reduced. Since an executable model is developed at the beginning of the design methodology, the specifications can be put to test before developing an actual digital system. As SYSTEMC provides a uniform environment for hardware-software co-design the processed output can be stored for further analysis in the design stage. Also data exchange between multiple domains is facilitated by SYSTEMC.

MATLAB is indeed a very useful tool for quick design of any signal processing system. The concept and the inputs and outputs can be tested in MATLAB before taking it to the digital domain. MATLAB provides a single environment for processing data from different domains such as speech and text.

Finally, the real cepstrum that was obtained in the model has many distinct characteristics. These characteristics facilitate the use of cepstrum for a variety of applications including homomorphic filtering, removal of echoes, restoration of acoustic recordings and automatic speech recognition systems.

CHAPTER 10

BIBLIOGRAPHY

1. Brian Bailey, Grant Martin and Andrew Piziali, *ESL design and verification – A prescription for electronic system level methodology*, Elsevier, Morgan Kaufmann publishers, 1st edition, 2007.
2. David C. Black and Jack Donovan, *SYSTEMC: From the ground up*, Kluwer academic publishers, 1st edition, 2004.
3. Alan V. Oppenheim and Ronald W. Shaffer, *Digital signal processing*, Pearson education Asia, First Indian reprint, 2002.
4. Thomas F. Quatieri, *Discrete-Time Speech signal processing – Principles and practice*, Pearson education, Third impression, 2007.
5. Ben Gold and Nelson Morgan, *Speech and audio signal processing – processing and perception of speech and music*, John wiley & sons, inc, 2000.
6. Lawrence R. Rabiner and Rownald W. Schafer, *Introduction to digital speech processing*, Now publishers, Volume 1, 2007.
7. Alan V. Oppenheim, *Speech analysis-synthesis system based on homomorphic filtering*, The journal of acoustic society of America, Vol. 45, No. 2, 458-465, February 1969.