

**Facultatea de Automatică și Calculatoare**

**Programul de Master:**

**Sisteme Informatice Aplicate în Producție și Servicii**

**Music to your ear**

**PROIECT**

**Prelucrarea Matematică a Semnalelor**

Studenți:

Cadmiel HOLHOȘ

Emanuel MOLDOVAN

Radu PETRUȘAN

Anul I, Master SIAPS

Timișoara,

2019

Contents

[**CAP. 1 INTRODUCERE** 3](#_Toc534739501)

[1.1. Motivație 3](#_Toc534739502)

[1.2. Descriere proiect 4](#_Toc534739503)

[**CAP. 2 ARHITECTURA APLICAȚIEI** 5](#_Toc534739504)

[2.1. Arhitectură și design 5](#_Toc534739505)

[2.2. Descriere funcționalități și detalii de implementare 6](#_Toc534739506)

[2.1.1. Generare de semnale sonore sinusoidale 6](#_Toc534739507)

[2.1.2. Înregistrare de semnale sonore 8](#_Toc534739508)

[2.1.3. Prelucrare și aplicare efecte asupra semalelor sonore 9](#_Toc534739509)

[2.1.4. Vizualizare grafică 14](#_Toc534739510)

[**CAP. 3 STUDII DE CAZ** 18](#_Toc534739511)

[3.1. Generare de semnale sonore sinusoidale 18](#_Toc534739512)

[3.2. Prelucrare și aplicare efecte asupra semnalelor sonore 19](#_Toc534739513)

[3.2.1. Aplicare efect de echo 19](#_Toc534739514)

[3.2.2. Aplicare efect de redare inversă 22](#_Toc534739515)

[3.2.3. Aplicare efect de redare continuă 23](#_Toc534739516)

[3.3. Vizualizare grafică 24](#_Toc534739517)

[**CAP. 4 CONCLUZII** 25](#_Toc534739518)

[**CAP. 5 BIBLIOGRAFIE** 26](#_Toc534739519)

# **CAP. 1 INTRODUCERE**

## Motivație

## Descriere proiect

# **CAP. 2 ARHITECTURA APLICAȚIEI**

## Arhitectură și design

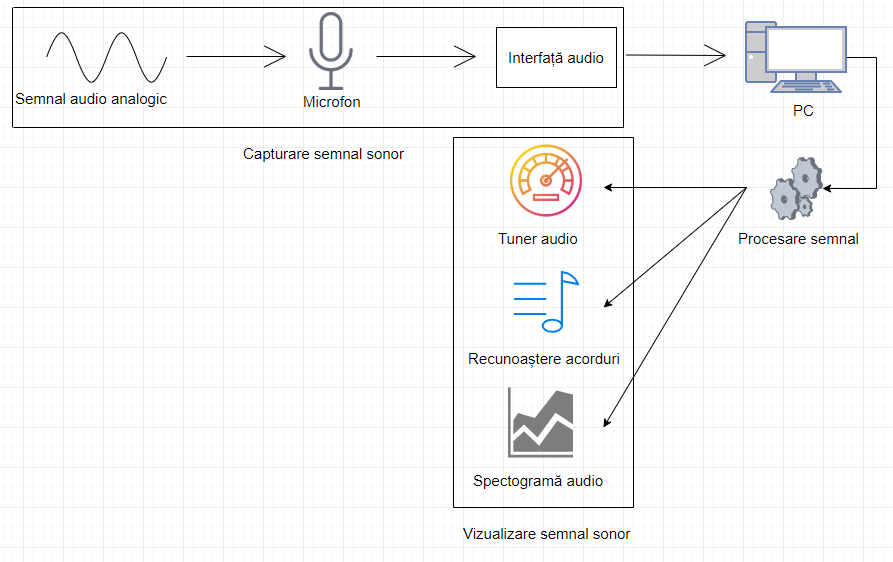


Figura 1. Arhitectura aplicației de prelucrare a semnalului sonor

Cele 3 componente de bază sunt descrise mai jos:

1. **Capturare semnal sonor** – modul care oferă posibilitatea de a înregistra sunetul provenit de la microfon pentru prelucrări viitoare.
2. **Procesare semnal sonor** – avand la dispozitie un semnal sonor, aplicăm FFT (Fast Fourier Transform) pentru a extrage frecvențele semnalului.
3. **Vizualizare semnale sonore** – aplicația oferă 3 funcționalități: tuner grafic audio, recunoaștere grafică acorduri și vizualizare grafică a spectogramei audio.

## Descriere funcționalități și detalii de implementare

În continuare vor fi prezentate principalele funcționalitați ale aplicației:

### Generare de semnale sonore sinusoidale

Fereastra de mai jos permite introducerea parametrilor pentru generarea funcției sinusoidale: frecvența și amplitudinea(faza considerată 0).

În urma genererării sunetul poate fi ascultat, dar și vizualizat în zona din dreapta, în urma apăsării butonului de stop ton.

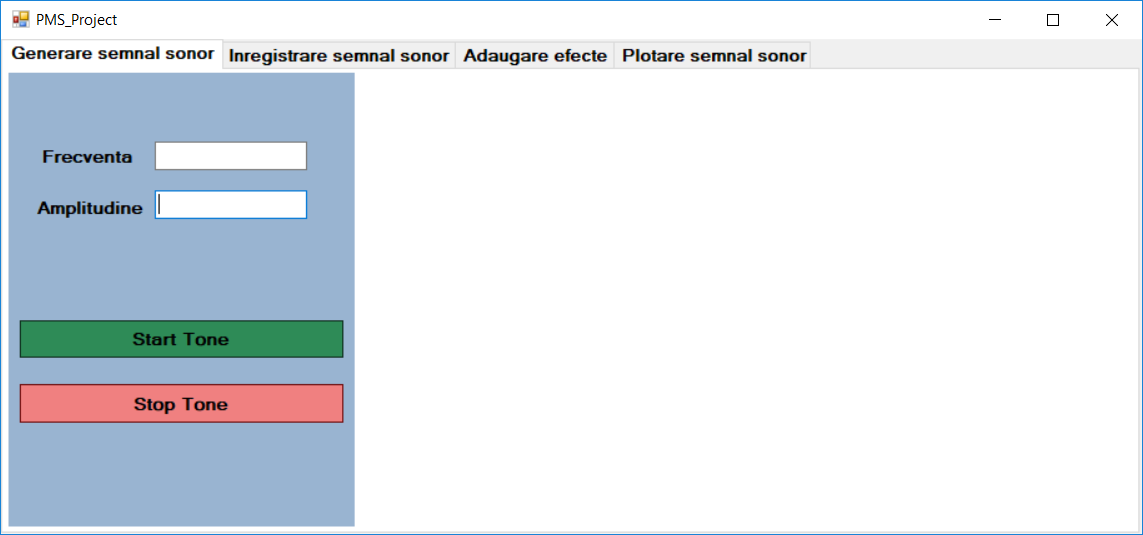


Figura 2. Fereastra generare semnal sinusoidal

O funcție sinusoidală are 3 parametri de bază care apar în ecuația undei:

1. Frecvența(f) – măsurată în Hz sau cicli pe secundă;
2. Amplitudinea(A) – este o masură a schimbării de presiune a sunetului. Se masoară în decibeli(bB) relativ la alte sunete. Scala dB este o scală logaritmică. Pătratului amplitudinii este proporțional cu energia, nivelul sau intensitatea (I) unui sunet. Diferența în decibeli între două sunete poate fi exprimată și în termeni de modificări ale intensității: 10 \* log10 (I1 / I2). Dubland intensitatea se obține o creștere de 3dB (10 \* 0.3).
3. Faza() – măsurata în grade sau radiani.

În figura de mai jos este reprezentat un ton de semnal sinusoidal cu frecvența de 1000 Hz, amplitudine 1, faza 0. O functie sinusoidală are energie doar la o frecvența, de aceea reprezentarea spectrală este doar un punct.

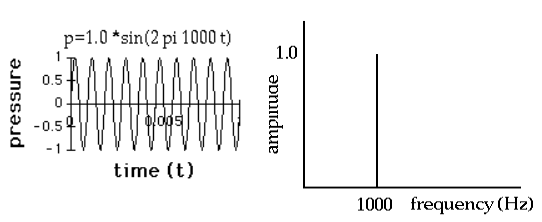


Figura 3. Reprezentarea în timp și spectrală a unei funcții sinusoidale

WaveTone tone = new WaveTone(amplitude, frequency);

stream = new BlockAlignReductionStream(tone);

output = new DirectSoundOut();

output.Init(stream);

output.Play();

Clasa ***WaveTone*** care va implementa clasa abstracta ***WaveStream*** *v*a suprascrie proprietațile și metodele acesteia:

* *Position*
* *Length*
* *WaveFormat*
* 44100 eșantioane/sec
* 16 bits/eșantion
* canalul
* Int *Read*(byte[] buffer, int offset, int count) - returnează numărul de bytes citiți:

public override int Read(byte[] buffer, int offset, int count)

{

int samples = count / 2;

for (int i = 0; i < samples; i++)

{

double sine = amplitude \* Math.Sin(Math.PI \* 2 \* frequency \* time);

time += 1.0 / 44100;

short truncated = (short)Math.Round(sine \* (Math.Pow(2, 15) - 1));

buffer[i \* 2] = (byte)(truncated & 0x00ff);

buffer[i \* 2 + 1] = (byte)((truncated & 0xff00) >> 8);

}

return count;

}

Count – numărul de bytes care se citesc. Numărul de eșantioane este egal cu jumătate din numraul de bytes care se citesc, doarece sunt 16 bits/eșantion(2bytes/eșantion).

Pentru fiecare eșantion se caluleaza funcția de undă:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |
|  |  | (2) |

Valoarea obținută se reprezintă în bits:

Se calculează cei 2 bytes corespunzători pentru un eșantion utilizând operații pe biți.

Clasa WaveTone va avea și 3 proprietăți private:

* Frecvența
* 100Hz
* 1000Hz – ton normal
* 3000Hz – ton mai inalt
* Amplitudinea - cât de puternic va fi tonul
* -1 => 1 – 1
* -0.1 => 0.1 – 0.1
* Timpul

### Înregistrare de semnale sonore

Există două modalități prin care se pot realiza fișiere de tip .wav.

* + - Sunet preluat de la microfon;
    - Sunet preluat de la boxe;

În prealabil se selectează o sursă și implicit un canal pe care are loc înregistrarea din partea de sus a primei coloane(în cazul în care microfonul va fi cel folosit).

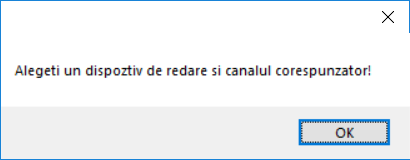


Figura 4. Mesaj aparut în cazul în care nu se alege un dispozitiv de redare la utilizarea microfonului

După care se poate apăsa pe unul din cele două butoane de culoare verde din ultimele două coloane. În urma apăsarii, utilizatorul va avea posibilitatea să selecteze locația pe disk și numele fișierui .wav unde se va salva înregistrarea. Salvarea propriu-zisă va avea loc la apasarea butonului stop corespunzător.

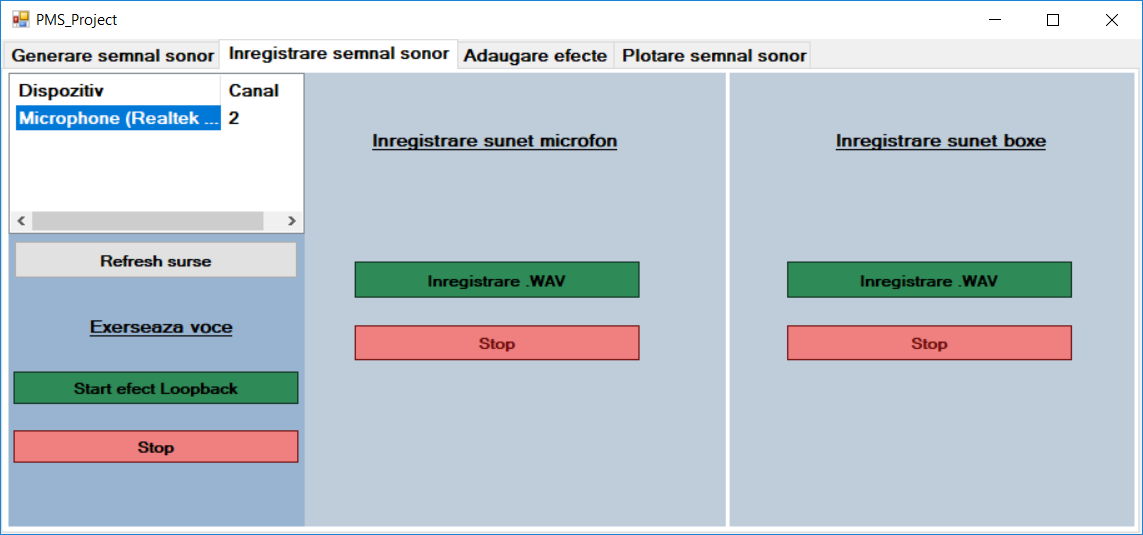


Figura 2. Modalități de înregistrare a semnalului sonor

De asemenea, fereastra mai pune la dispoziție și o secțiune în care utilizatorul va putea să își asculte propria voce, fara a produce salvarea pe disk. Se oferă posibilitatea de a exersa înainte de înregistrarea propriei voci.

### Prelucrare și aplicare efecte asupra semalelor sonore

Fereastra de mai jos pune la dispoziție interfata pentru aplicarea a trei efecte:

* + Efect de echo;
  + Redare inversă;
  + Redare continuă.

Odata ce un efect este selectat, utilizatorul va avea posibilitatea sa aleaga fisierul .wav pentru care se va aplica efectul. In urma apasarii butonului stop(disponibil la aplicarea efectului de ecou si la redarea continuă – intrucat se lasa posibilitatea acordarii unui timp suplimentar in in inregistrare pentru evidentierea efectului) pe disk la aceeasi locatie se vor salva fisierele .wav rezultate denumite dupa cum urmeaza:

Odată ce un efect este selectat, utilizatorul va avea posibilitatea să aleagă fișierul .wav pentru care se va aplică efectul. În urmă apăsării butonului stop(disponibil la aplicarea efectului de ecou și la redarea continuă – întrucât se lasă posibilitatea acordării unui timp suplimentar în în înregistrare pentru evidențierea efectului) pe disk la aceeași locație se vor salva fișierele .wav rezultate denumite după cum urmează:

|  |  |  |
| --- | --- | --- |
| Efect | Fișier original | Fișier prelucrat |
| Efect de ecou | Fișier.wav | Fișier\_echo.wav |
| Efect de redare inversă | Fișier.wav | Fișier\_reverse.wav |
| Efect de redare continuă | Fișier.wav | Fișier\_loop.wav |

Tabel 1. Denumirea fișierelor salvate pe disk dupa prelucrare

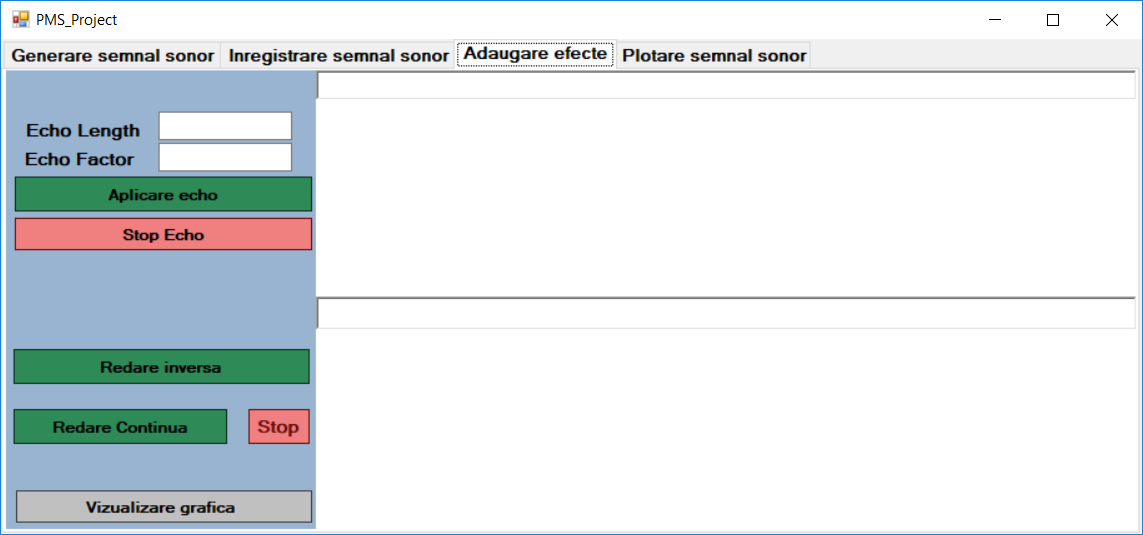


Figura 5. Fereastra adaugare efecte peste un semnal sonor

Efectul produs poate fi ascultat și de asemenea poate fi și vizualizat în partea dreaptă a ferestrei înainte și după prelucrare. Există două casete text care vor conține calea către fișierele pereche generate aflate pe disk, alăturat de graficul corespunzător.

***Efectul de ecou***

Un efect de ecou poate fi obținut pentru semnalele audio realizând următoarea ecuație diferențială,

|  |  |  |
| --- | --- | --- |
|  | , | (3) |

unde intrarea audio (discretă în timp) este x, ieșirea este y și 0 <= a <1 este o constantă reală și N este o constantă intregă. N și a determină cât durează echo-ul și cum arată.

Răspunsul impulsului h poate fi obținut permițând ca intrarea să fie un impuls, x = δ și găsirea ieșirii y = h:

|  |  |  |
| --- | --- | --- |
|  | , | (3) |
|  |  | (4) |

Din (3) și (4) rezultă:

|  |  |  |
| --- | --- | --- |
|  | , | (5) |
| Dar |  | (6) |
| Deci |  | (7) |

Continuând în acest mod, vedem că răspunsul la impuls al sistemului de ecou este impulsul original și un set infinit de ecouri (impulsuri întârziate și scalate). Acesta poate fi scrisă compact, după cum urmează, rezultatul valabil pentru toate valorile n întregi:

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

Răspunsul la impuls este reprezentat mai jos:

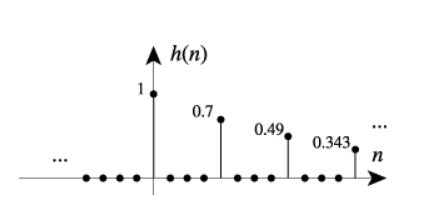


Figura 6. Răspunsul la impuls

Sistemul precedent poartă numele de sistem cu raspuns infinit la impuls(IIR) deoarece răspunsul la un impuls nu se termină niciodată complet.

În continuare se poate observa codul apelat în spatele butonului de adăugare efect:

1 WaveChannel32 wave = new WaveChannel32(new WaveFileReader(inputFile));

2 EffectStream effect = new EffectStream(wave);

3 stream = new BlockAlignReductionStream(effect);

4 for (int i = 0; i < wave.WaveFormat.Channels; i++)

5 effect.Effects.Add(new **Echo**(Echo\_length, Echo\_Factor)));

7 output = new DirectSoundOut(200);

8 output.Init(stream);

9 output.Play();

Linile 7-9 fac posibilă redarea semnalului audio în urma prelucrărilor.

Linia 2 instantiază un obiect de tip **EffectStream** pe baza fișierului dat că intrare. După care se adaugă efectul de ecou.

Mai jos se prezintă conținutul clasei **Echo** realizate:

public class Echo : IEffect

{

public int EchoLength { get; private set; }

public float EchoFactor { get; private set; }

private Queue<float> samples;

public Echo(int length, float factor)

{

this.EchoLength = length;

this.EchoFactor = factor;

this.samples = new Queue<float>();

for (int i = 0; i < length; i++) samples.Enqueue(0f);

}

public float ApplyEffect(float sample)

{

samples.Enqueue(sample);

return Math.Min(1, Math.Max(-1, sample + EchoFactor \* samples.Dequeue()));

}

}

Ea conține doi parametri publici care sunt introduși de către utilizator în cele două casete text puse la dispoziție. Aplicarea propriu-zisă a efectul se poate observă în ultima linie de cod din secțiunea de mai jos. Valoarea esantioului primit se înmulțește cu factorul de echo și se adună eșantionului existent.

Pentru redare inversă citirea din streamul de citire se face în ordine inversă, iar buffer-ul care conține fiecare byte citit este transmis spre streamul de scriere.

using (WaveFileReader reader = new WaveFileReader(open.FileName))

{

int blockAlign = reader.WaveFormat.BlockAlign;

using (WaveFileWriter writer = new WaveFileWriter(reverseFile, reader.WaveFormat))

{

byte[] buffer = new byte[blockAlign];

long samples = reader.Length / blockAlign;

for (long sample = samples - 1; sample >= 0; sample--)

{

reader.Position = sample \* blockAlign;

reader.Read(buffer, 0, blockAlign);

writer.WriteData(buffer, 0, blockAlign);

}

}

}

În continuare se poate observa codul apelat în spatele butonului de adăugare efect de redare continuă:

1 WaveFileReader reader = new WaveFileReader(open.FileName);

2 LoopStream loop = new LoopStream(reader);

3 waveOut = new WaveOut();

4 waveOut.Init(loop);

5 waveOut.Play();

Linile 3-5 fac posibilă redarea semnalului audio în urmă prelucrărilor.

Linia 2 instantiază un obiect de tip **LoopStream** pe baza streamului de citire al fișierului dat că intrare. Această clasa este derivate din clasa de baza WaveStream utilizată. Metodă de cititire Read va fi suprascrisă astfel încât să înceapă recitirea fișierului audio odată ce streamul sursă nu mai returnează date. Acest lucru presupune că stream-ul sursă dat ca parametru se oprește din citire. Notă: Metoda *Read* trebuie să returneze 0 în momentul în care a ajuns la final, altfel nu se va realiza saltul la început.

O altă posibilitate ar fi să folosim proprietatea de lungime a streamului sursă, și să reîncepem citirea de la început de îndată ce numărul de bytes așteptați au fost transmiși.

În continuare este prezentat codul sursă pentru clasa **LoopStream:**

public class LoopStream : WaveStream

{

WaveStream sourceStream;

public LoopStream(WaveStream sourceStream)

{

this.sourceStream = sourceStream;

this.EnableLooping = true;

}

public bool EnableLooping { get; set; }

public override WaveFormat WaveFormat

{

get { return sourceStream.WaveFormat; }

}

public override long Length

{

get { return sourceStream.Length; }

}

public override long Position

{

get { return sourceStream.Position; }

set { sourceStream.Position = value; }

}

public override int Read(byte[] buffer, int offset, int count)

{

int totalBytesRead = 0;

while (totalBytesRead < count)

{

int bytesRead = sourceStream.Read(buffer, offset + totalBytesRead, count - totalBytesRead);

if (bytesRead == 0)

{

if (sourceStream.Position == 0 || !EnableLooping)

{

// something wrong with the source stream

break;

}

// loop

sourceStream.Position = 0;

}

totalBytesRead += bytesRead;

}

return totalBytesRead;

}

}

### Vizualizare grafică

Fereastră de mai jos oferă posibilitatea de a deschide un fișier .wav de pe disk pentru redare și vizualizare. Se poate face pauză în timpul redării, iar vizualizarea este permisă în 2 moduri, cea anterioară și o altă plotare care permite utilizatorului să facă zoom în sau să revină la formă inițială.

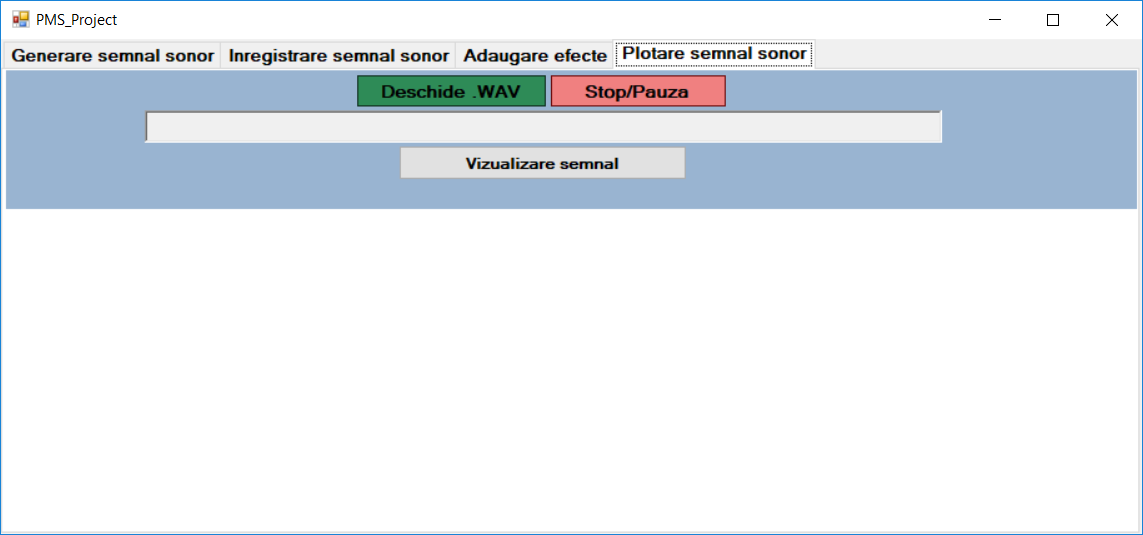


Figura 7. Vizualizare semnal sonor cu posibilitate de zoom in

În continuare este prezentat codul sursă al reprezentării grafice a semnalelor sonore sub formă de chart:

chart1.Series.Add("wave");

chart1.Series["wave"].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.FastLine;

chart1.Series["wave"].ChartArea = "ChartArea1";

WaveChannel32 wave = new WaveChannel32(new WaveFileReader(richTextBox1.Text));

byte[] buffer = new byte[16384];

int read = 0;

while (wave.Position < wave.Length)

{

read = wave.Read(buffer, 0, 16384);

for (int i = 0; i < read / 4; i++)

{

chart1.Series["wave"].Points.Add(BitConverter.ToSingle(buffer, i \* 4));

}

}

Pentru a doua modalitate de afișare se crează un nou control sub forma de clasă cu denumirea **CustomWaveViewer** cu constructorul următor:

public CustomWaveViewer()

{

// This call is required by the Windows.Forms Form Designer.

InitializeComponent();

this.DoubleBuffered = true;

this.PenColor = Color.DodgerBlue;

this.PenWidth = 1;

}

și trei callbackuri pentru acțiunile inițiate de mouse:

**MouseDown** inițiază procesul de redimensionare a graficului. Pentru o bună vizualizare se desenează o linie verticală în zona în care s-a dat click.

protected override void OnMouseDown(MouseEventArgs e)

{

if (e.Button == System.Windows.Forms.MouseButtons.Left)

{

startPos = e.Location;

mousePos = new Point(-1, -1);

mouseDrag = true;

DrawVerticalLine(e.X);

}

base.OnMouseDown(e);

}

La mișcarea mouse-ului coordonatele trasării liniei verticale se ajustează.

protected override void OnMouseMove(MouseEventArgs e)

{

if (mouseDrag)

{

DrawVerticalLine(e.X);

if (mousePos.X != -1) DrawVerticalLine(mousePos.X);

mousePos = e.Location;

}

base.OnMouseMove(e);

}

La evenimentul de **MouseUp** porțiunea selectată este ajustată în așa fel ca ea să ocupe întreaga zonă disponibilă.

protected override void OnMouseUp(MouseEventArgs e)

{

if (mouseDrag && e.Button == System.Windows.Forms.MouseButtons.Left)

{

mouseDrag = false;

DrawVerticalLine(startPos.X);

if (mousePos.X == -1) return;

DrawVerticalLine(mousePos.X);

int leftSample = (int)(StartPosition / bytesPerSample + samplesPerPixel \* Math.Min(startPos.X, mousePos.X));

int rightSample = (int)(StartPosition / bytesPerSample + samplesPerPixel \* Math.Max(startPos.X, mousePos.X));

Zoom(leftSample, rightSample);

}

else if (e.Button == System.Windows.Forms.MouseButtons.Middle)

FitToScreen();

base.OnMouseUp(e);

}

Efectul de zoom-in și redimensionare poate fi observat mai jos, se pot vedea cele două linii verticale, porțiune care va fi ajustată în așa fel ca ea sa ocupe întreaga zonă disponibilă, după cum se vede în figura 8.



Figura 8. Efectul de redimensionare al graficului

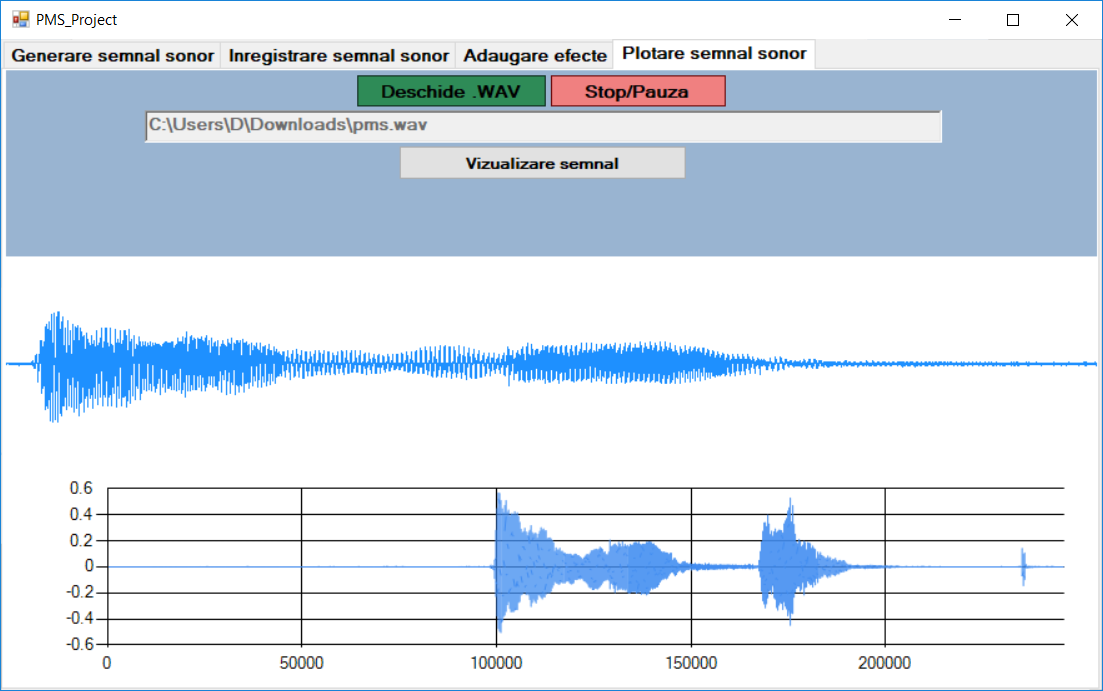


Figura 9. Rezultat redimensionare

# **CAP. 3 STUDII DE CAZ**

## Generare de semnale sonore sinusoidale

În continuare sunt prezentate două semnale sinusoidale(de tip ton) generate folosind primul tab disponibil al aplicației.

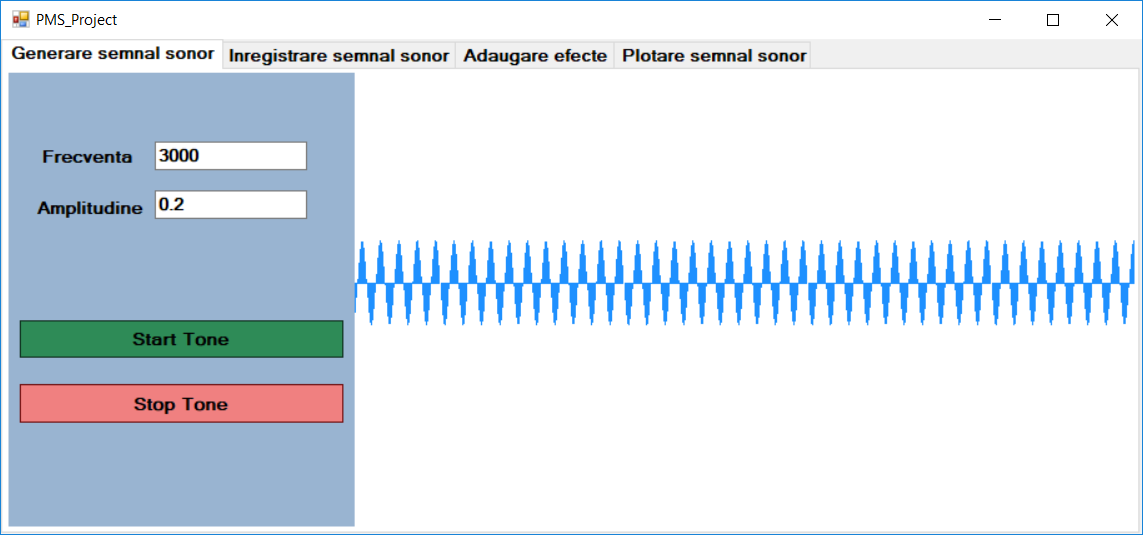


Figura 10. Generare ton cu frecvența de 3000 HZ și amplitudine 0.2

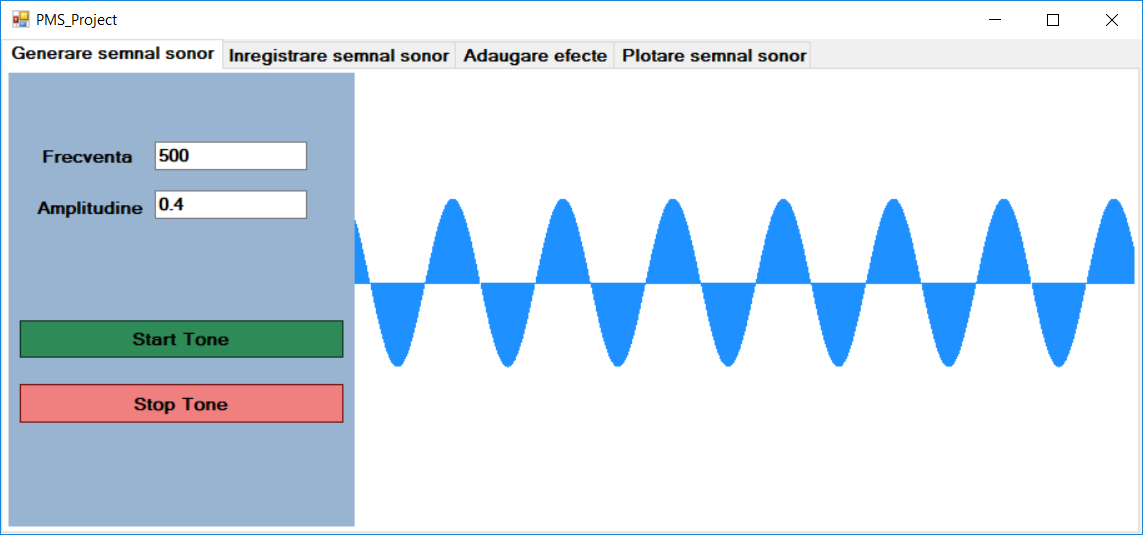


Figura 11. Generare ton cu frecvența de 500 HZ și amplitudine 0.4

## Prelucrare și aplicare efecte asupra semnalelor sonore

În continuare vor fi prezentate formele de undă ale semnalelor obtinuțe în urma prelucrărilor prezentate.

### Aplicare efect de echo

Efectul de echo peste semnalul util se poate observa în figurile următoare cu diverși parametri:

* Length: 20000
* Factor: 0.5

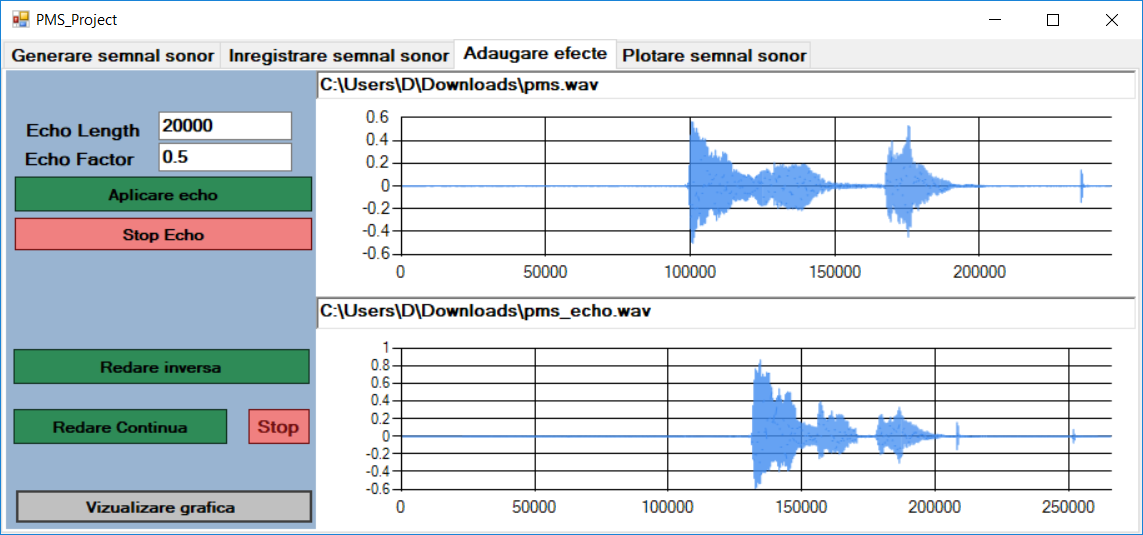


Figura 12. Vizualizare efect de ecou aplicat peste un semnal sonor

* Length: 20000
* Factor: 1

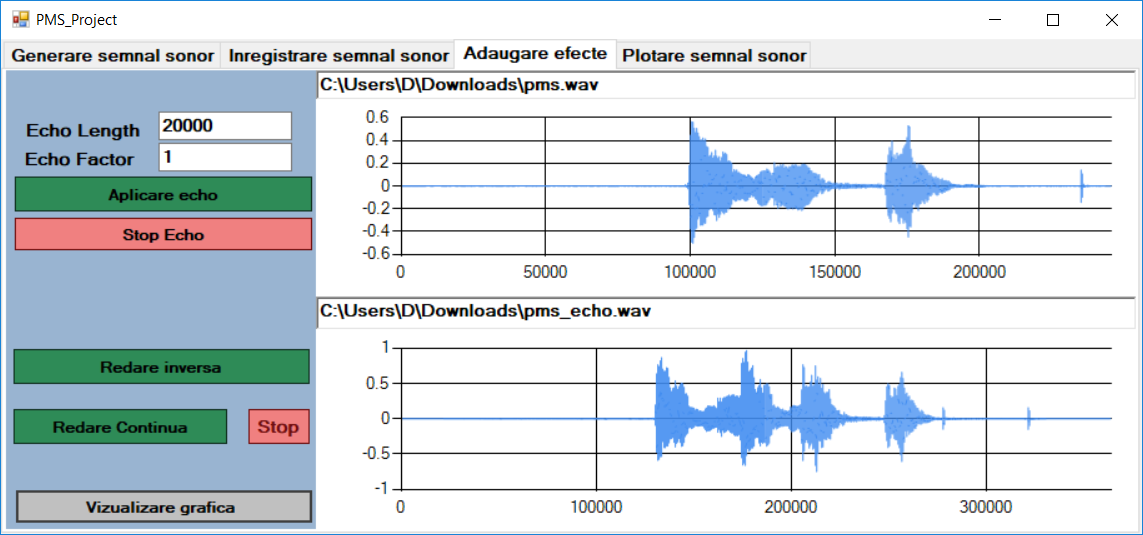


Figura 13. Vizualizare efect de ecou aplicat peste un semnal sonor

* Length:10000
* Factor: 1

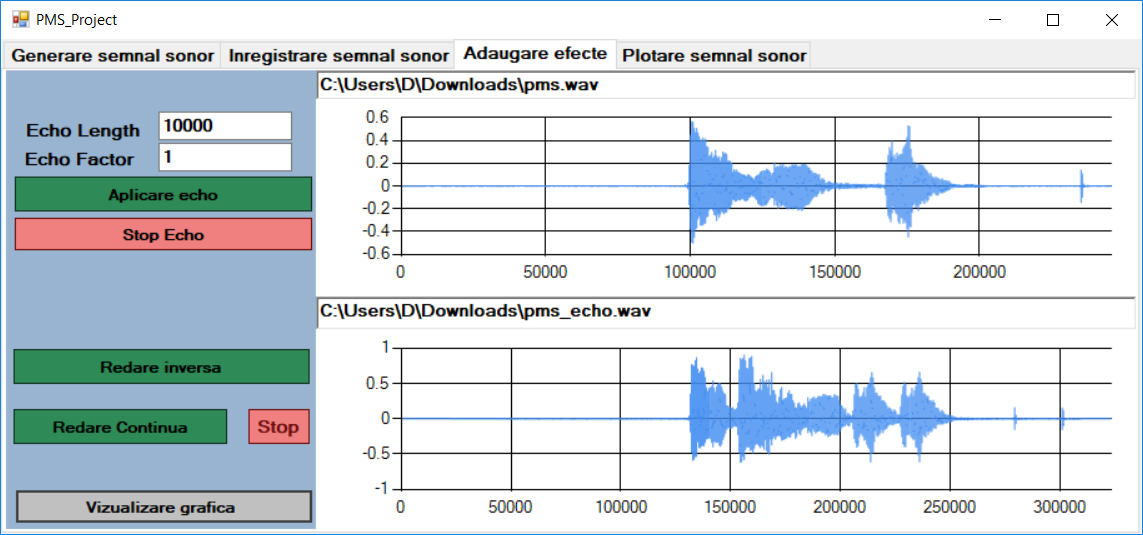


Figura 14. Vizualizare efect de ecou aplicat peste un semnal sonor

* Length: 20000
* Factor: 0.5

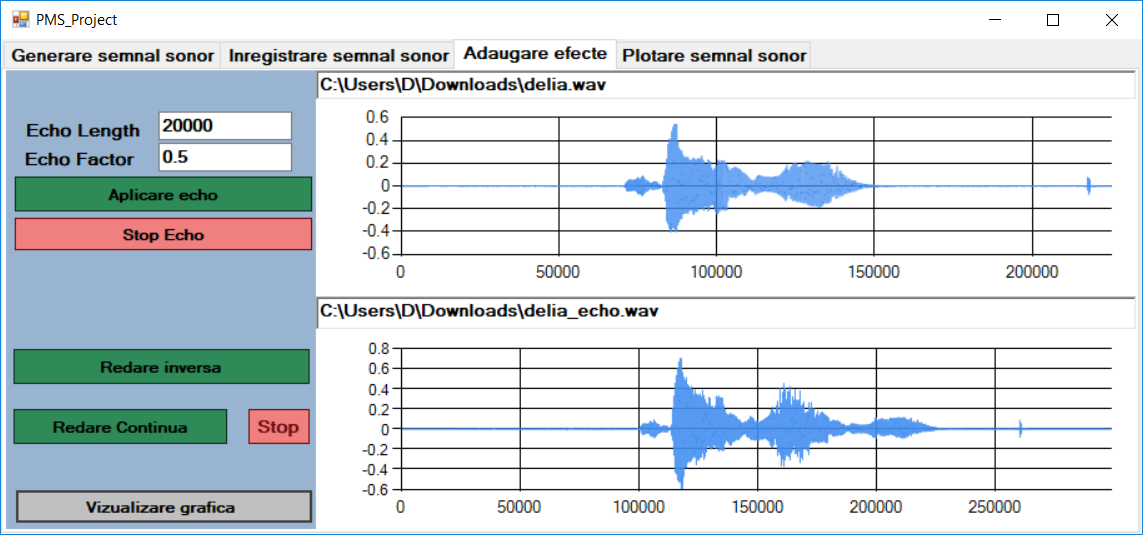


Figura 15. Vizualizare efect de ecou aplicat peste un semnal sonor

* Length: 20000
* Factor: 1

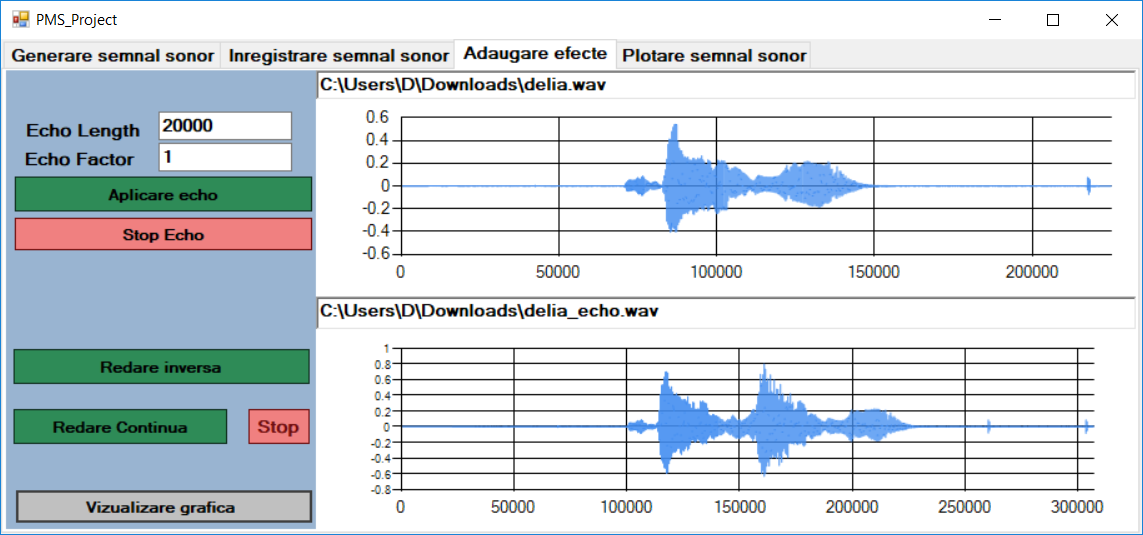


Figura 16. Vizualizare efect de ecou aplicat peste un semnal sonor

* Length:10000
* Factor: 1

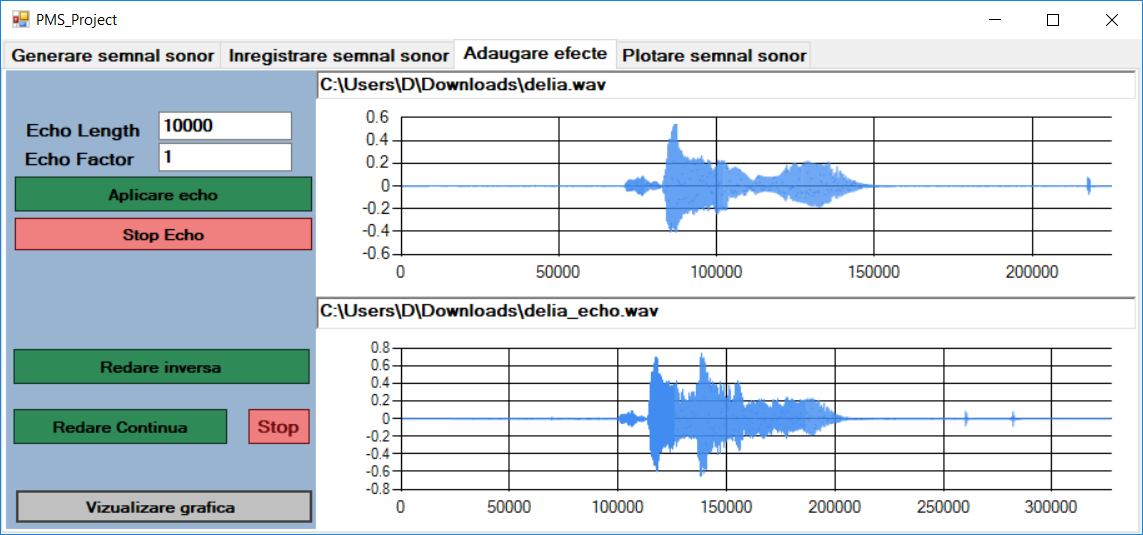


Figura 17. Vizualizare efect de ecou aplicat peste un semnal sonor

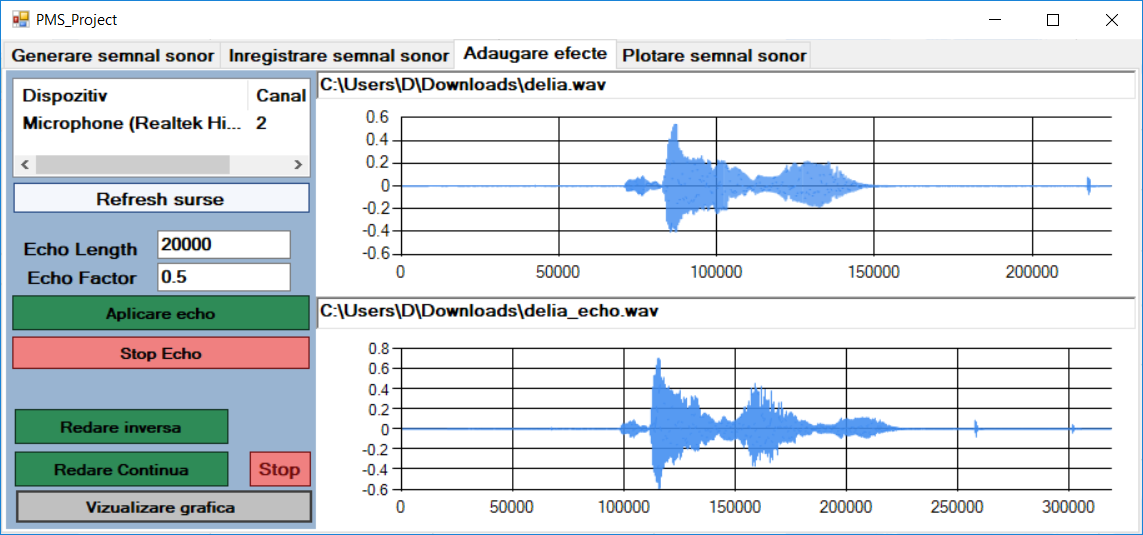


Figura 18. Vizualizare efect de ecou aplicat peste un semnal sonor 2

### Aplicare efect de redare inversă

Efectul de redare inversă peste semnalul util se poate observa în figurile următoare:

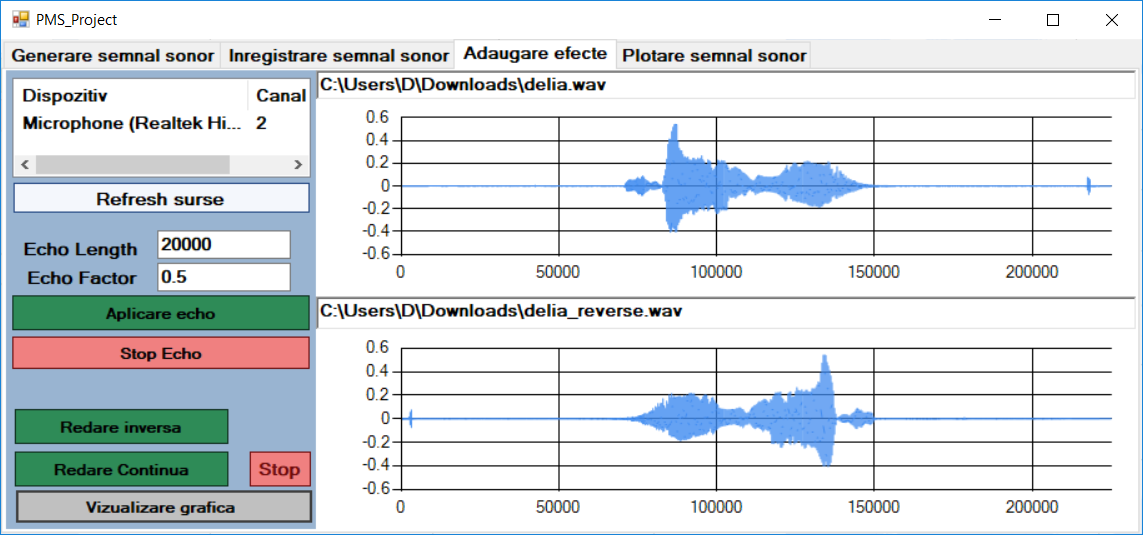


Figura 19. Redare inversă a unui semnal sonor

### Aplicare efect de redare continuă

Efectul de redare continuă peste semnalul util se poate observa în figurile următoare:

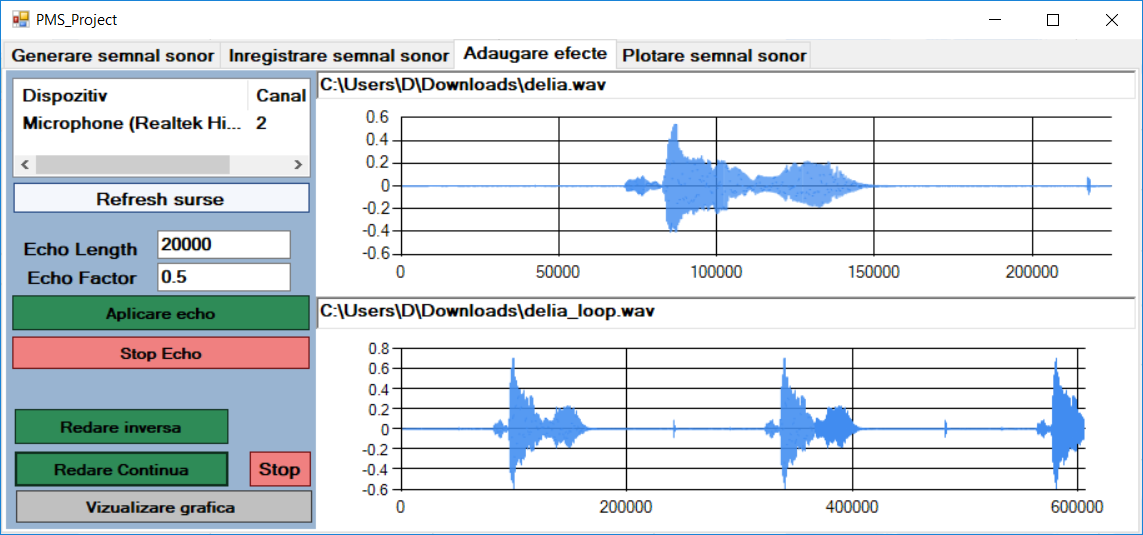


Figura 20. Redare continuă a unui semnal sonor

## Vizualizare grafică

Vizualizare semnal sonor cu posibilitate de redimensionare:

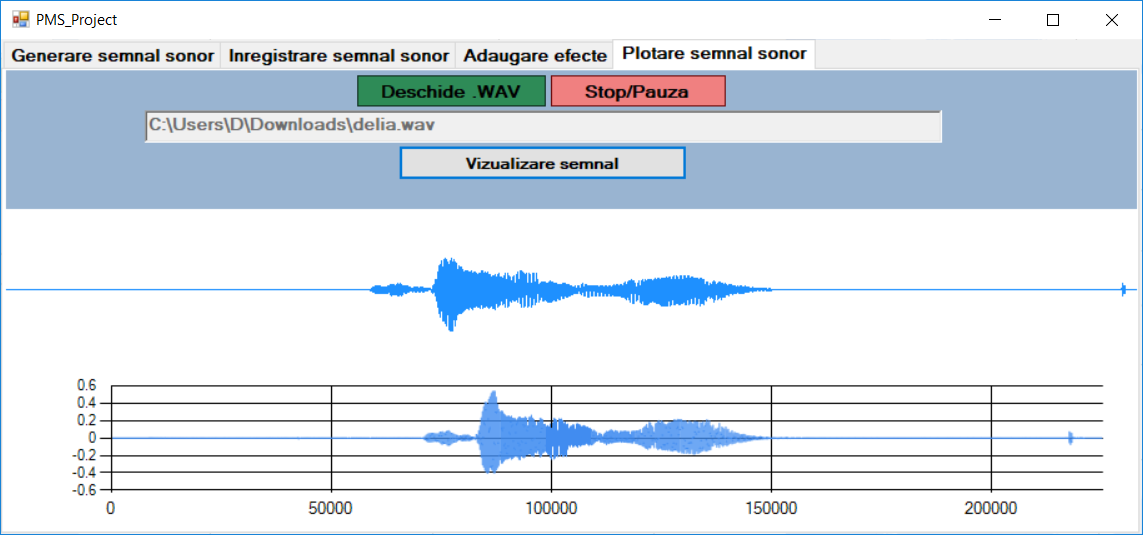


Figura 3. Vizualizare semnal sonor cu posibilitate de dimensionare

# **CAP. 4 CONCLUZII**

În concluzie, întreaga lucrare a venit cu o soluție de procesare completă de sunet care pornește de la înregistrarea semnalelor sonore, aplicarea de prelucrări matematice asupra acestora și până la vizualizarea efectelor produse.

Totodată se poate spune că efectul central în jurul căruia s-au construit celelalte funcționalități este efectul de ecou. Acesta poate fi asemănat cu un sistem de tip IIR, sistem întâlnit sub denumirea de **filtru recursiv** deoarece are ieşirea dependentă de valoarea curentă şi valorile anterioare ale intrării cât şi de valorile anterioare ale ieşirii, de unde şi afirmaţia că aceste filtre sunt **sisteme cu reacţie inversă**.

Aplicația constituie un punct de pornire care prezintă modalități de bază care pot fi aplicate în scopul procesarii semnalelor sonore.

În cadrul acestei aplicații se face reprezentarea semnalelor sonore în domeniul timp, însă ținând cont de faptul că semnalele reale întâlnite în viața de zi cu zi nu sunt deterministe, ci aleatoare o direcție de urmat în viitor ar fi reprezentarea și sub formă spectrală a frecvențelor semnalului.

# **CAP. 5 BIBLIOGRAFIE**

[www1] <http://www.giawa.com/tutorials/src/>

[www2] <http://www.lifesci.sussex.ac.uk/home/Chris_Darwin/Perception/Lecture_Notes/Hearing1/hearing1.html>

[www3] <http://www.etc.ugal.ro/imunteanu/Csemnale_bmk.pdf>

[www4] <https://archive.codeplex.com/?p=naudio>

[www5] <http://mark-dot-net.blogspot.com/2011/04/how-to-use-wavefilewriter.html>

[www6] <https://stackoverflow.com/questions/11793310/how-to-add-echo-effect-on-audio-file-using-objective-c/11804367#11804367>

[www7] <http://mark-dot-net.blogspot.com/2009/10/looped-playback-in-net-with-naudio.html>

[www8] <https://ptolemy.berkeley.edu/eecs20/week7/echo.html>

[www9] <https://ptolemy.berkeley.edu/eecs20/week7/echoResponse.pdf>

[www10] <http://etd.lib.metu.edu.tr/upload/12618426/index.pdf>