Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Symulacja filtru cyfrowego

Dokumentacja projektu

Alicja Misterka, Piotr Pokornowski

Warszawa, 2 czerwca 2024

Spis treści

1 Wprowadzenie

Projekt zakłada stworzenie aplikacji do symulacji filtrów cyfrowych używanych do przetwarzania sygnałów audio. Aplikacja została napisana w języku Java i umożliwia użytkownikowi składanie układu filtru z poszczególnych bloków, takich jak filtry dolnoprzepustowe (LP), górnoprzepustowe (HP) itp.

2 Specyfikacja projektowa

2.1 Cel projektu

Celem projektu jest stworzenie interaktywnej aplikacji umożliwiającej użytkownikom łatwe przetwarzanie sygnałów audio za pomocą filtrów cyfrowych.

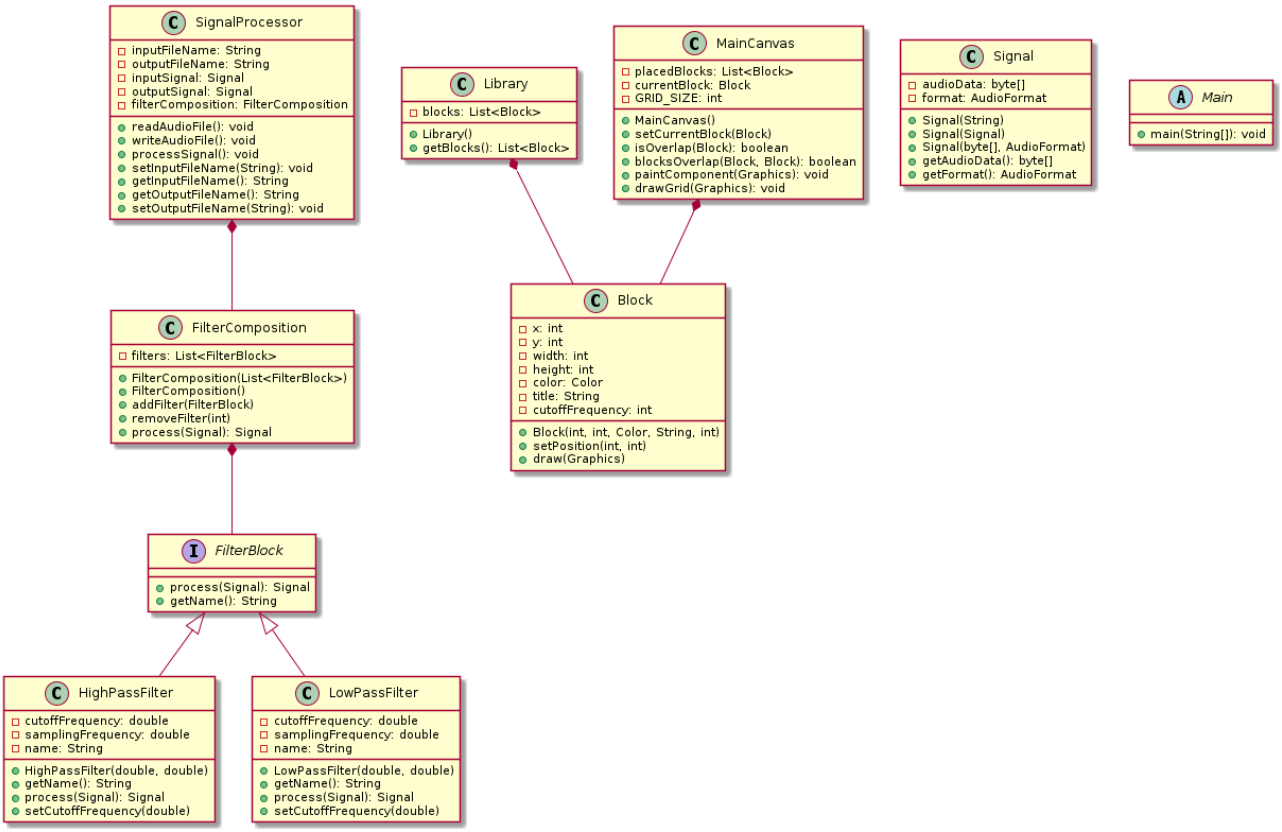
2.2 Funkcjonalności

- Dodawanie i konfigurowanie bloków filtrów - ustalanie częstotliwości odcięcia.
- Wybór pliku wejściowego do przetwarzania.
- Przetwarzanie sygnału audio za pomocą wybranych filtrów.
- Zapis przetworzonego sygnału do pliku wyjściowego.

2.3 Wymagania systemowe

- Java Development Kit (JDK) 8 lub wyższy.
- Biblioteki Java Swing do interfejsu graficznego.
- Biblioteka Java Sound do przetwarzania sygnałów audio.

3 Diagram UML



Rysunek 1: Pełny diagram UML

4 Założenia projektowe

- Użytkownik ma możliwość wyboru filtra oraz ustawienia jego parametrów.
- Interfejs graficzny pozwala na dodawanie i rozmieszczanie bloków filtrów.
- Przetwarzanie sygnału odbywa się w sposób sekwencyjny, gdzie każdy filtr przetwarza wyjście poprzedniego filtra.

5 Kod źródłowy

5.1 Klasa Block

```
1  import java.awt.Color;
2  import java.awt.Graphics;
3
4  public class Block {
5      public int x, y, width, height;
6      public Color color;
7      public String title; // Nowy atrybut na tytuĆ bloku
8      public int cutoffFrequency; // Nowy atrybut na czŹstotliwoŻ  odciŹcia
9
10     public Block(int width, int height, Color color, String title, int cutoffFrequency)
       {
11         this.width = width;
12         this.height = height;
13         this.color = color;
```

```
14          this.title = title;
15          this.cutoffFrequency = cutoffFrequency;
16      }
17
18      public void setPosition(int x, int y) {
19          this.x = x;
20          this.y = y;
21      }
22
23      public void draw(Graphics g) {
24          g.setColor(color);
25          g.fillRect(x, y, width * MainCanvas.GRID_SIZE, height * MainCanvas.GRID_SIZE);
26          g.setColor(Color.BLACK);
27          g.drawRect(x, y, width * MainCanvas.GRID_SIZE, height * MainCanvas.GRID_SIZE);
28          // Rysowanie tytu Ću
29          if (title != null && !title.isEmpty()) {
30              g.drawString(title, x + width * MainCanvas.GRID_SIZE / 2 - title.length() *
        3,
31                      y + MainCanvas.GRID_SIZE / 2);
32          }
33          g.drawString(String.valueOf(cutoffFrequency) + " Hz",
34                  x + width * MainCanvas.GRID_SIZE / 2 - String.valueOf(cutoffFrequency).
        length() * 3,
35                  y + MainCanvas.GRID_SIZE / 2 + 20);
36      }
37  }
```

Listing 1: Implementacja klasy Block

5.2 Interfejs FilterBlock

```
1  public interface FilterBlock {
2      Signal process(Signal inputSignal);
3      String getName();
4  }
```

Listing 2: Implementacja interfejsu FilterBlock

5.3 Klasa FilterComposition

```
1  import java.util.List;
2  import java.util.ArrayList;
3
4  public class FilterComposition {
5      private List<FilterBlock> filters = new ArrayList<>();
6
7      public FilterComposition(List<FilterBlock> filters) {
8          this.filters = filters;
9      }
10
11     public FilterComposition() {
12     }
13
14     public void addFilter(FilterBlock filter) {
15         filters.add(filter);
16     }
17
18     public void removeFilter(int index) {
19         filters.remove(index);
20     }
21
22     public Signal process(Signal inputSignal) {
23         Signal outputSignal = inputSignal;
24
```

```
25          for (FilterBlock filter : filters) {
26              outputSignal = filter.process(outputSignal);
27              System.out.println("Applied filter: " + filter.getName());
28          }
29
30          return outputSignal;
31      }
32  }
```

Listing 3: Implementacja klasy FilterComposition

5.4 Klasa HighPassFilter

```
1   public class HighPassFilter implements FilterBlock {
2       private double cutoffFrequency; // w Hz
3       private double samplingFrequency; // w Hz
4       private String name = "HighPassFilter";
5
6       public String getName() {
7           return name;
8       }
9
10      public HighPassFilter(double cutoffFrequency, double samplingFrequency) {
11          this.cutoffFrequency = cutoffFrequency;
12          this.samplingFrequency = samplingFrequency;
13      }
14
15      public Signal process(Signal inputSignal) {
16          byte[] audioData = inputSignal.getAudioData();
17          byte[] filteredData = applyHighPassFilter(audioData);
18          return new Signal(filteredData, inputSignal.getFormat());
19      }
20
21      private byte[] applyHighPassFilter(byte[] audioData) {
22          double[] samples = byteArrayToDoubleArray(audioData);
23          double rc = 1.0 / (2 * Math.PI * cutoffFrequency);
24          double alpha = rc / (rc + (1.0 / samplingFrequency));
25          double[] filteredSamples = new double[samples.length];
26
27          filteredSamples[0] = samples[0];
28          for (int i = 1; i < samples.length; i++) {
29              filteredSamples[i] = alpha * (filteredSamples[i - 1] + samples[i] - samples
    [i - 1]);
30          }
31
32          return doubleArrayToByteArray(filteredSamples);
33      }
34
35      private double[] byteArrayToDoubleArray(byte[] byteArray) {
36          double[] doubleArray = new double[byteArray.length / 2];
37          for (int i = 0; i < doubleArray.length; i++) {
38              doubleArray[i] = ((short) ((byteArray[2 * i + 1] << 8) | (byteArray[2 * i]
    & 0xff))) / 32768.0;
39          }
40          return doubleArray;
41      }
42
43      private byte[] doubleArrayToByteArray(double[] doubleArray) {
44          byte[] byteArray = new byte[doubleArray.length * 2];
45          for (int i = 0; i < doubleArray.length; i++) {
46              short shortValue = (short) (doubleArray[i] * 32768.0);
47              byteArray[2 * i] = (byte) shortValue;
48              byteArray[2 * i + 1] = (byte) (shortValue >> 8);
49          }
50          return byteArray;
```

```
51      }
52
53      public void setCutoffFrequency(double cutoffFrequency) {
54          this.cutoffFrequency = cutoffFrequency;
55      }
56  }
```

Listing 4: Implementacja klasy HighPassFilter

5.5 Klasa LowPassFilter

```
1   public class LowPassFilter implements FilterBlock {
2       private double cutoffFrequency; // w Hz
3       private double samplingFrequency; // w Hz
4       private String name = "LowPassFilter";
5
6       public String getName() {
7           return name;
8       }
9
10      public LowPassFilter(double cutoffFrequency, double samplingFrequency) {
11          this.cutoffFrequency = cutoffFrequency;
12          this.samplingFrequency = samplingFrequency;
13      }
14
15      public Signal process(Signal inputSignal) {
16          byte[] audioData = inputSignal.getAudioData();
17          byte[] filteredData = applyLowPassFilter(audioData);
18          return new Signal(filteredData, inputSignal.getFormat());
19      }
20
21      private byte[] applyLowPassFilter(byte[] audioData) {
22          double[] samples = byteArrayToDoubleArray(audioData);
23          double rc = 1.0 / (2 * Math.PI * cutoffFrequency);
24          double alpha = 1.0 / (1.0 + rc * samplingFrequency);
25          double[] filteredSamples = new double[samples.length];
26
27          filteredSamples[0] = samples[0];
28          for (int i = 1; i < samples.length; i++) {
29              filteredSamples[i] = alpha * samples[i] + (1.0 - alpha) * filteredSamples[i
        - 1];
30          }
31
32          return doubleArrayToByteArray(filteredSamples);
33      }
34
35      private double[] byteArrayToDoubleArray(byte[] byteArray) {
36          double[] doubleArray = new double[byteArray.length / 2];
37          for (int i = 0; i < doubleArray.length; i++) {
38              doubleArray[i] = ((short) ((byteArray[2 * i + 1] << 8) | (byteArray[2 * i]
       & 0xff))) / 32768.0;
39          }
40          return doubleArray;
41      }
42
43      private byte[] doubleArrayToByteArray(double[] doubleArray) {
44          byte[] byteArray = new byte[doubleArray.length * 2];
45          for (int i = 0; i < doubleArray.length; i++) {
46              short shortValue = (short) (doubleArray[i] * 32768.0);
47              byteArray[2 * i] = (byte) shortValue;
48              byteArray[2 * i + 1] = (byte) (shortValue >> 8);
49          }
50          return byteArray;
51      }
52
```

```
53      public void setCutoffFrequency(double cutoffFrequency) {
54          this.cutoffFrequency = cutoffFrequency;
55      }
56  }
```

Listing 5: Implementacja klasy LowPassFilter

6 Interfejs użytkownika

Aplikacja zawiera interfejs graficzny oparty na Java Swing, umożliwiający użytkownikom dodawanie i konfigurowanie filtrów, przetwarzanie sygnałów audio oraz zapisywanie wyników. Użytkownik może przeciągać i upuszczać bloki filtrów na płótnie, ustawiać parametry filtrów oraz wybierać pliki wejściowe i wyjściowe.