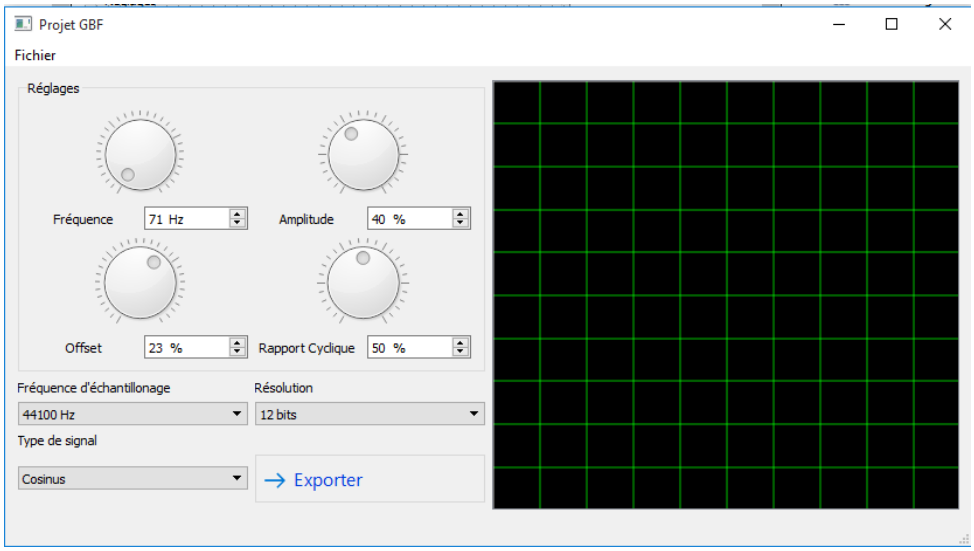
Solutions à mettre en œuvre

Objectif : Créer une application permettant de générer des fichiers Wave contenant des signaux simples (sinus, cosinus, carré, triangle).

➢ Interface homme machine

1. L'interface homme machine



2. Explications des relations de l'IHM

Voir en annexe le diagramme des slots/signaux.

➢ Explication mathématique des signaux

1. Généralité sur les signaux

Dans le projet, cinq classes ont été développées pour la partie « création des signaux ». La classe « GBF_Signal » est la classe mère. Elle contient tous les attributs et les méthodes liées à la création des signaux. Les attributs de cette classe sont les suivants :

- m_Frequency : Fréquence en Hz du signal
- m_Offset : Composante continue du signal
- m_Amplitude : Amplitude crête à crête du signal
- m_Waveform : Contient les points calculés du signal
- m_Type : Contient un enum relatif au type de signal (1 : Carré, 2 : Sinus , 3 : Cosinus, 4 : triangle)

Des classes filles héritent des propriétés et des méthodes de la classe mère. Seule la classe carrée dispose d'un attribut supplémentaire m_DutyCycle de façon à agir sur le rapport cyclique de ce signal en particulier.

Seulement une période est calculée pour chaque signal généré. Le motif sera ensuite répété en fonction du paramètre « Temps du signal » lors de la génération du fichier Wave.

2. Le signal triangle

Ci-dessous le code C++ commenté développé pour la fonction triangle.

```
void GBF_TriangleWave::ComputeWaveform(int Resolution, int SamplingFrequency)
{
    int NbSamples = 0;                                  // Création variable nombre d'échantillons
    int MaximumAmplitude = 0;                           // Création variable amplitude maximale
    int CurrentSample = 0;                              // Création variable echantillon en cours de calcul

    NbSamples = SamplingFrequency/m_Frequency;          // Définition du nombre d'échantillon en fonction de la fréquence d'échantillonnage et de la fréquence
    MaximumAmplitude = pow(2, Resolution)/2 -1;         // Définition de l'amplitude maximale en fonction de la résolution
    m_Waveform.clear();                                 // RAZ de l'attribut contenant la waveform

    for(int i = 0; i < NbSamples; i++)                  // Boucle de calcul pour chaque échantillon
    {
        if(i < NbSamples/2)                             // Calcul des échantillons pour la descente du signal triangle
        {
            CurrentSample = int(-1 * m_Amplitude * MaximumAmplitude + (4*m_Amplitude*MaximumAmplitude/(NbSamples))*i + m_Offset * MaximumAmplitude);

            if(CurrentSample > MaximumAmplitude)                    // Si saturation positive
                CurrentSample = MaximumAmplitude;                   // Ecretage positif
            else if(CurrentSample < MaximumAmplitude * -1)          // Si saturation négative
                CurrentSample = MaximumAmplitude * -1;              // Ecretage négatif

            m_Waveform.push_back(CurrentSample);
        }

        else                                            // Calcul des échantillons pour la montée du signal triangle (même principe)
        {
            CurrentSample = int(m_Amplitude * MaximumAmplitude + (4*m_Amplitude*MaximumAmplitude/(NbSamples))*(i - NbSamples/2)*-1 + m_Offset * MaximumAmplitude)

            if(CurrentSample > MaximumAmplitude)
                CurrentSample = MaximumAmplitude;
            else if(CurrentSample < MaximumAmplitude * -1)
                CurrentSample = MaximumAmplitude * -1;

            m_Waveform.push_back(CurrentSample);        // Ajout de l'échantillon precedement calculé a la fin de la waveform

        }
        }
    }
}
```

3. Le signal carré

Ci-dessous le code C++ commenté développé pour la fonction carré.

```
void GBF_SquareWave::ComputeWaveform(int Resolution, int SamplingFrequency)
{
    int NbSamples = 0;                                  // Création variable nombre d'échantillons
    int MaximumAmplitude = 0;                           // Création variable amplitude maximale
    int CurrentSample = 0;                              // Création variable echantillon en cours de calcul

    NbSamples = SamplingFrequency/m_Frequency;          // Définition du nombre d'échantillon en fonction de la fréquence d'échantillonnage et de la fréquence
    MaximumAmplitude = pow(2, Resolution)/2 -1;         // Définition de l'amplitude maximale en fonction de la résolution
    m_Waveform.clear();                                 // RAZ de l'attribut contenant la waveform


    for(int i = 0; i < NbSamples; i++)                  // boucle de calcul de chaque echantillon

    {
        if(i < m_DutyCycle*NbSamples)                   // Calcul des échantillon de la partie haute du signal
        {
            CurrentSample = int(m_Amplitude * MaximumAmplitude + m_Offset * MaximumAmplitude);      // Formule de calcul partie haute
            if(CurrentSample > MaximumAmplitude)                                                    // Si saturation positive
                CurrentSample = MaximumAmplitude;                                                   // Ecretage positif
            else if(CurrentSample < MaximumAmplitude * -1)                                          // Si saturation négative
                CurrentSample = MaximumAmplitude * -1;                                              // Ecretage négatif

            m_Waveform.push_back(CurrentSample);                                // Ajout de l'échantillon precedement calculé a la fin de la waveform
        }

        else                                                                    // Calcul des échantillon de la partie haute du signal
        {
            CurrentSample = int(-1 * m_Amplitude * MaximumAmplitude + m_Offset * MaximumAmplitude); // Formule de calcul partie basse
            if(CurrentSample > MaximumAmplitude)                                                    // Si saturation positive
                CurrentSample = MaximumAmplitude;                                                   // Ecretage positif
            else if(CurrentSample < MaximumAmplitude * -1)                                          // Si saturation négative
                CurrentSample = MaximumAmplitude * -1;                                              // Ecretage négatif

           m_Waveform.push_back(CurrentSample);                                 // Ajout de l'échantillon precedement calculé a la fin de la waveform
        }
    }
}
```

4. Le signal sinus et cosinus

Ci-dessous le code C++ commenté développé pour la fonction composition du cosinus. La fonction sinus a été développée sur le même modèle en remplaçant simplement le cos par un sinus.

```
void GBF_CosinusWave::ComputeWaveform(int Resolution,int SamplingFrequency)
{
    int NbSamples = 0;                                  // Création variable nombre d'échantillons
    int MaximumAmplitude = 0;                           // Création variable amplitude maximale
    int CurrentSample = 0;                              // Création variable echantillon en cours de calcul

    NbSamples = SamplingFrequency/m_Frequency;          // Définition du nombre d'échantillon en fonction de la fréquence d'échantillonnage et de la fréquence
    MaximumAmplitude = pow(2, Resolution)/2 -1;         // Définition de l'amplitude maximale en fonction de la résolution
    m_Waveform.clear();                                 // RAZ de l'attribut contenant la waveform

     for(int i = 0; i < NbSamples; i++)                 // boucle de calcul de chaque echantillon
     {

         CurrentSample = m_Amplitude *MaximumAmplitude*cos(i*2*M_PI/NbSamples)+m_Offset;       // Calcul de l'échantillon en fonction du signal

         if(CurrentSample > MaximumAmplitude)                           // Si saturation positive
             CurrentSample = MaximumAmplitude;                          // Ecretage positif
         else if(CurrentSample < MaximumAmplitude * -1)                 // Si saturation négative
             CurrentSample = MaximumAmplitude * -1;                     // Ecretage négatif

         m_Waveform.push_back(CurrentSample);                           // Ajout de l'échantillon precedement calculé a la fin de la waveform

     }
}
```

➢ L'exportation du signal généré en fichier Wave

1. La trame à mettre en œuvre

Le remplissage du fichier respecte le format standard Wave :

```
[Bloc de déclaration d'un fichier au format WAVE]
   FileTypeBlocID  (4 octets) : Constante «RIFF»  (0x52,0x49,0x46,0x46)
   FileSize        (4 octets) : Taille du fichier moins 8 octets
   FileFormatID    (4 octets) : Format = «WAVE»  (0x57,0x41,0x56,0x45)

[Bloc décrivant le format audio]
   FormatBlocID    (4 octets) : Identifiant «fmt »  (0x66,0x6D, 0x74,0x20)
   BlocSize        (4 octets) : Nombre d'octets du bloc - 16  (0x10)

   AudioFormat     (2 octets) : Format du stockage dans le fichier (1: PCM, ...)
   NbrCanaux       (2 octets) : Nombre de canaux (de 1 à 6, cf. ci-dessous)
   Frequence       (4 octets) : Fréquence d'échantillonnage (en hertz) [Valeurs standardisées : 11025, 22050, 44100 et éventuellement 48000 et 96000]
   BytePerSec      (4 octets) : Nombre d'octets à lire par seconde (i.e., Frequence * BytePerBloc).
   BytePerBloc     (2 octets) : Nombre d'octets par bloc d'échantillonnage (i.e., tous canaux confondus : NbrCanaux * BitsPerSample/8).
   BitsPerSample   (2 octets) : Nombre de bits utilisés pour le codage de chaque échantillon (8, 16, 24)

[Bloc des données]
   DataBlocID      (4 octets) : Constante «data»  (0x64,0x61,0x74,0x61)
   DataSize        (4 octets) : Nombre d'octets des données (i.e. "Data[]", i.e. taille_du_fichier - taille_de_l'entête  (qui fait 44 octets normalement).
   DATAS[] : [Octets du Sample 1 du Canal 1] [Octets du Sample 1 du Canal 2] [Octets du Sample 2 du Canal 1] [Octets du Sample 2 du Canal 2]
```

On retrouve trois blocs distincts qui forment l'en-tête de 44 octets :

- Un bloc de déclaration sur 12 octets
- Un bloc décrivant le format sur 24 octets
- Un bloc de données

2. La solution choisie

Lors de l'appui sur le bouton "Enregistrer", la méthode "ExportWAV" de la classe "GBF_Generator" sera exécutée. Cette méthode utilisera des attributs de la classe "GBF_Generator" mis à jour par l'IHM:

- m_Name : Nom du fichier
- m_Directory : Répertoire de création du fichier
- m_Time : Durée du signal à créer

Les caractéristiques suivantes du signal seront accessibles par l'intermédiaire de la classe "GBF_Signal " :

- m_Frequency : Fréquence en Hz du signal
- m_Waveform : Contient les points calculés du signal

m_Frequency nous permettra de calculer la période, et donc calculer le nombre de duplication nécessaire de m_Waveform pour atteindre la durée d'enregistrement demandée. Le nombre de période nécessaire sera arrondie à l'entier inférieur pour simplifier le code.Lorsque les données du signal seront copiées, nous pourrons calculer la taille exacte du fichier et renseigner le champ correspondant dans l'entête.

A noté que certains champsde l'en-tête seront fixés :

- Constante "RIFF" en héxa
- Format "WAVE" en héxa
- Identifiant "fmt" en hexa
- Format de stockage dans le fichier : 1 pour PCM
- Nombre canaux = 2 (stéréo)
- Format audio = 1 pour PCM
- Constante "data" en héxa

➢ Annexes

Diagramme de slots/signaux