Description

Le but de cette fonction est de d'identifier une trame correctement formatée dans le signal audio. Dans le schéma-bloc de la communication audio, elle se situe après les comparateurs. Elle est appelée une fois par période d'échantillonnage.

Son prototype est :

```
int FskDetector(int detLow, int detHigh);
```

Ses paramètres sont les sorties des deux comparateurs. Ces signaux sont définis comme suit :

> *detLow* (*detHigh*) vaut 1 si le signal audio capté par le micro contient la fréquence *f0* (*f1*), dans le cas contraire, il vaut 0.

De ces deux paramètres, *fskDetector()* déduit l'état du signal, qui peut prendre 4 valeurs :

detLow	***detHigh***	**Etat du signal audio**
0	0	Silence
0	1	Bit 1
1	0	Bit 0
1	1	Bruit

La fonction renvoie les 10 bits de données de la trame (dans un *int*) si une trame correcte a été identifiée ; elle renvoie 0 sinon.

Principe de détection

Pour commencer, introduisons la notion d'*OverSampling Ratio (OSR)* : c'est le rapport entre la fréquence d'échantillonnage et la fréquence des bits. Ce rapport donne le nombre d'échantillons du signal qui compose un bit de la trame. Pour que le décodage se passe correctement, il faut que ce rapport soit un nombre entier.

Détection du début de la trame

Pour détecter le début de la trame, on attend simplement d'avoir l'état *BIT0*. Nous utilisons cette détection pour nous synchroniser sur le début de la trame.

Pour cela, nous utiliserons la variable *timer* qui nous servira à compter le nombre d'échantillons composant un bit (soit *OSR*).

Détection de la valeur d'un bit

Pour notre fonction, un bit est représenté par *OSR* échantillons successifs.

Pour déterminer si le bit est un '0', elle compte le nombre d'échantillons où le signal est dans l'état *BIT0* (dans la variable *countL*). Après *OSR* échantillons, elle compare *countL* au seuil *FSK_MIN_SAMPLES_NB*.

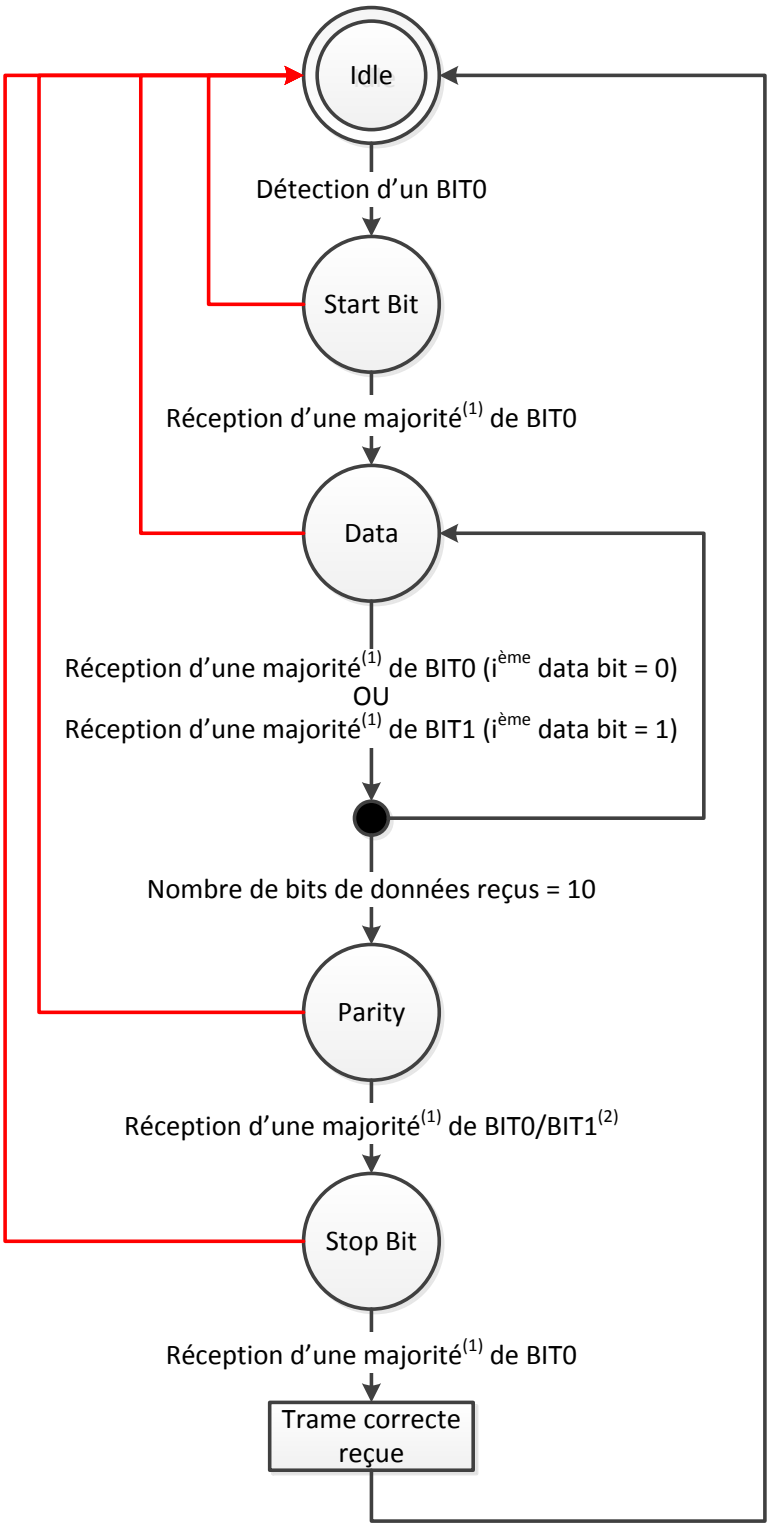
Elle fait de même pour déterminer si le bit est un '1', en utilisant la variable *countH*.

Dans le programme de test, *FSK_MIN_SAMPLES_NB* est égal à *OSR*, ce qui implique que, sur la durée d'un bit, tous les échantillons du signal doivent être dans l'état *BIT0*(*BIT1*) pour que le bit soit considéré comme un '0'('1'). Il faudra probablement ajuster sa valeur expérimentalement.

Si *countL* et *countH* sont tous deux inférieurs à *FSK_MIN_SAMPLES_NB*, le bit est considéré comme indéterminé. Cela implique que la trame est corrompue et la machine d'état passe directement dans l'état *IDLE*, sans essayer de décoder le reste de la trame.

Machine d'état

Comme le montre ce qui précède, le décodage d'une trame de bits est un processus séquentiel, notre fonction doit donc implémenter une machine d'état



(1) : Le seuil définissant une majorité acceptable est à définir (entre OSR/2 et OSR)
(2) : L'état correct du bit de parité (BIT0/BIT1) dépend du nombre de '1' dans les bits de données (impair/pair)

Configuration

Dans le fichier *fskDetector.h*, on trouve le prototype de la fonction et la définition de plusieurs constantes qui permettent de configurer la fonction :

- MESSAGE_LENGTH est le nombre de bits de données de la trame (10 dans notre cas)
- SAMPLING_FREQ est la fréquence d'échantillonnage du signal audio, en Hz
- BIT_FREQ est la fréquence des bits de la trame, en Hz
- OS Rest l'OverSampling Ratio (cf. i-dessus). Il est égal à SAMPLING_FREQ/BIT_FREQ)
- FSK_MIN_SAMPLES_NB est le nombre d'échantillons minimum pour considérer un bit valide.

Tests

Test 1

Le projet *testFskDetector1.X* permet de tester la fonction en lui fournissant les échantillons d'une trame (définie par la variable *trame*).

Deux boucles *for* imbriquées simulent la chaîne d'acquisition audio :

- La 1ère boucle parcoure les 13 bits de la trame et en déduit les valeurs de *detLow* et *detHigh*
- La 2ème boucle appelle *OSR* fois *fskDetector()*, ce qui correspond à un bit.

fskDetector() est donc appelée 13xOSR fois sur la durée d'une trame.

Lors du dernier appel de *fskDetector()*, elle doit renvoyée le message décodé, si tout va bien. Dans ce cas, le programme allume une LED.

Ce ts nous a permis de vérifier que notre fonction détecte les trames correctes et ignore les trames incorrectes.

Test 2

Le projet *testFskDetector2.X* est basé sur le projet précédent. On lui a ajouté une 3ème boucle pour pouvoir tester plusieurs trames successivement.

Ce test a validé le fait que notre fonction peut détecter plusieurs trames successives, pour autant qu'elles soient séparées par une période de silence (ou de bruit).

En effet, comme la machine d'état de *fskDetector()* est réinitialisée dès qu'elle a détecté que la trame est erronée, elle considérera le prochain bit 0 qu'elle rencontrera comme le start bit d'une nouvelle trame.

Par exemple, si on lui fournit la trame

Péambule	Données		Postambule	
Start Bit	Ordre	Paramètre	Parité	Stop bit
0	00	00001111	1	0

Dont le bit de parité est faux, la machine d'état considérera le stop bit comme le start bit d'une nouvelle trame. En pratique, cela ne pose pas de problème car nous n'enverrons qu'une trame à la fois, *fskDetector()* détectera que le "bit" suivant est du silence et réinitialisera sa machine d'état.

Par contre, cela nous a obligé à ajouter une boucle envoyant un bit de silence entre chaque trame testée pour que *fskDtector()* soit dans l'état IDLE au début de chaque trame.