Examen L3 EEA sujet – démodulation logicielle d'un signal modulé en fréquence

É. Carry, J.-M Friedt, 14 mars 2016

Pour chaque question **en caractères gras**, vous ferez valider votre réponse par une démonstration rapide à un enseignant, et transmettrez les codes sources associés par courrier électronique à `emile.carry@femto-st.fr` et `jmfriedt@femto-st.fr`. Vous rédigerez pour toutes les questions un texte donnant les réponses et les calculs sur une feuille.
Tous les documents papier et électroniques autorisés, accès aux téléphones portables proscrit.

Nous avons vu en Transmission de l'Information qu'un démodulateur de fréquence est une boucle à verrouillage de phase (PLL). Notre objectif est d'implémenter de façon logicielle l'oscillateur ajustable en tension afin d'effectuer la transposition de fréquence après échantillonnage d'un signal par le convertisseur analogique-numérique du microcontrôleur. Une application d'un tel algorithme a été proposée pour démoduler un signal acquis sur la bande FM commerciale – certes avec un microcontrôleur un peu plus puissant [1] que l'Atmega32U4 – à `https://github.com/rad1o/f1rmware/blob/master/rfapp/wfm.c`. Dans la Fig. 1, l'oscillateur ajustable en tension (VCO) n'est plus une implémentation matérielle mais une version logicielle d'une fonction proposant les même caractéristiques. Dans une PLL, La fréquence du VCO sera ajustée par l'erreur de fréquence entre le signal acquis et le signal de mélange chargé de compenser la modulation : ici, **nous nous contenterons de la boucle ouverte sans ajuster la fréquence du VCO qui sera considérée fixe**.

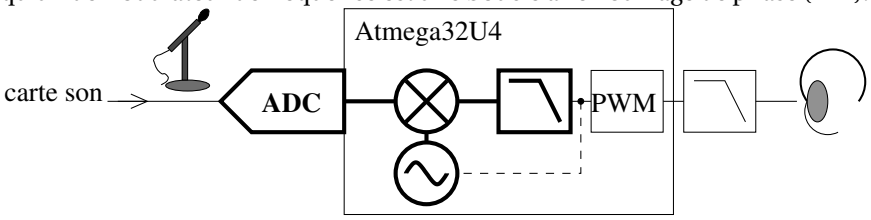

FIGURE 1: Schéma du circuit de démodulation logicielle. Seules les parties rouges nous intéressent.

1. Tout au long de cet exercice nous aurons besoin d'afficher des données sur le port USB (implémentant un protocole compatible avec un port série virtuel – CDC). **Proposer une fonction `affiche(short, char*)` qui prend un entier signé sur 16 bits en argument et renvoie une chaîne de caractère contenant sa valeur en décimal. Démontrer par l'affichage de 1234 et de -4253.** Pour rappel, la bibliothèque de communication sur port USB et son exemple associé sont à `http://jmfriedt.free.fr/LUFA_light_EEA.tar.gz`.
2. Quelle est la plage de valeurs affichables par cette fonction ?
3. Partant de l'exemple vu en TP d'acquisition d'un signal périodique, **proposer une implémentation de l'ADC qui se déclenche à intervalles de temps réguliers imposés par un *timer*. Démontrer le bon fonctionnement du programme – passant par une compilation en se liant à la fonction `affiche()` vue auparavant – par une acquisition d'un signal de fréquence fixée à 100 Hz issu de la carte son du PC.** Afin de ne pas perdre de temps à retaper du code vu en TP, nous proposons une archive à `http://jmfriedt.free.fr/exam2016_uC.tar.gz` contenant un exemple fonctionnel de lecture périodique d'ADC déclenchée sur *timer* à un rythme prédéfini.
4. Quels ajustements sur le signal issu de la carte son sont nécessaires avant d'alimenter le convertisseur analogique-numérique du microcontrôleur ? Proposer un schéma de circuit électronique effectuant ces opérations.
5. Quelle est la fréquence de déclenchement de l'interruption *timer* qui induit une conversion analogique-numérique dans le programme fourni dans l'archive ? Justifier.
6. Quelle est la plage de tensions mesurable par le convertisseur analogique-numérique ?
7. Démontrer votre analyse en traçant la transformée de Fourier (fonction `fft` de GNU/Octave) d'un signal acquis et en graduant "correctement" l'axe des abscisses. Vérifier que la position du pic correspondant au signal acquis est à l'abscisse prévue. Pour rappel, la génération d'un signal périodique sur carte son peut se faire par `gnuradio-companion`, `audacity` ou en ligne de commande par `play`.
8. Connaissant l'intervalle de temps entre deux mesures (inverse de la fréquence d'échantillonnage), modifier le programme fourni pour y implémenter (dans une fonction judicieusement choisie) un oscillateur qui **génère un signal sinusoïdal périodique à 30 Hz**. Quelle bibliothèque utiliser pour atteindre rapidement ce résultat ?
9. On notera que la représentation classique d'un signal périodique s de fréquence f échantillonné à f_e de la forme $s(n) = \sin(n \cdot f/f_e)$ est instable numériquement lorsque n devient grand. Afin de pallier à cette déficience, il est judicieux d'exprimer la phase φ du signal et de calculer (selon du pseudo-code) $\varphi += d\varphi; if\ (\varphi > 2\pi)\ \varphi -= 2\pi; oscillateur = \sin(\varphi)$; qui garantit que la phase, argument du sinus, reste comprise entre 0 et 2π et évite les calculs sur des nombres trop grands induisant des erreurs d'approximations. Exprimer $d\varphi$ en fonction des données du problème. **Modifier le programme en conséquence si nécessaire.**
10. Commenter sur les ressources occupées par l'implémentation de l'oscillateur dans le microcontrôleur, par rapport aux ressources disponibles. **Démontrer le bon fonctionnement de l'oscillateur** (on pourra par exemple générer les données sur le port USB et tracer la séquence, ou sa transformée de Fourier).
11. Comment appliquer le mélange de l'oscillateur local généré par logiciel sur les données acquises ? **Démontrer expérimentalement le mélange et tracer le spectre du signal acquis après transposition de fréquence. Est-il cohérent avec nos attentes ?**
12. Le signal observé contient plusieurs composantes spectrales, dont une qui n'est pas favorable au travail de démodulation (élimination de la porteuse par transposition de fréquence). Comment l'éliminer ? Comment implémenter trivialement un tel algorithme dans le microcontrôleur ? Quels sont les paramètres de cette implémentation ?
13. Quelle opération est mise en œuvre par le couple {PWM, filtre passe bas} sur le schéma de la Fig. 1 ? Quel périphérique matériel pourrait faire la même opération ? Ce périphérique est-il disponible sur Atmega32U4 ?

1. `https://rad1o.badge.events.ccc.de/`

Solutions

1. La fonction affiche est désormais classique :

```
int affiche(short v,char *s)
{int l=0;
 if (v<0) {s[l]='-';l++;v=-v;}
 s[l]=(v/10000)+'0';l++;v=v-(v/10000)*10000;
 s[l]=(v/1000)+'0'; l++;v=v-(v/1000)*1000;
 s[l]=(v/100)+'0'; l++;v=v-(v/100)*100;
 s[l]=(v/10)+'0'; l++;v=v-(v/10)*10;
 s[l]=(v)+'0';
 return(l); // renvoie le nombre de chars stock'es dans s
}
```

2. Cette fonction prend une valeur codée sur 16 bits signée, donc comprise entre ±32767. Il nous faut donc 5 chiffres pour afficher ce nombre en plus du signe.

3.
```
#include <avr/io.h> //E/S ex PORTB
#define F_CPU 16000000UL
#include <util/delay.h> // _delay_ms
#include "affiche.h"
#include "VirtualSerial.h"

#define N 256 // nbre de points acquis
#define fe 1000 // frequence echantillonnage
#define f 30 // frequence du melange
volatile short adc[N],flag,sinus[N];
volatile float phase=0.;

extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
extern FILE USBSerialStream;

EMPTY_INTERRUPT(TIMER4_OVF_vect);
ISR(ADC_vect)
{ if (flag<N) {adc[flag]=ADC/4-127; sinus[flag]=(short)(127.*sin(phase));flag++;}
  phase+=(2*M_PI*f/fe);if (phase>(2*M_PI)) phase-=2*M_PI;
}

void adc_init()
{ ADMUX = (1<<REFS0)+1; // quelle tension de reference ? quelle canal ?
  ADCSRB = (1<<ADTS3); // quelle source de declenchement de la mesure ?
  ADCSRA = (1<<ADEN) | (1<<ADSC) | (1<<ADATE) | (1<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
}

int main(void)
{ char s[9]; int k;
  s[0]='0';s[1]='x';s[6]='\r'; s[7]='\n'; s[8]=0;
  SetupHardware();
  CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
  GlobalInterruptEnable();

  TCCR4B=(0<<CS43)|(1<<CS42)|(1<<CS41)|(1<<CS40); // 16 MHz/64=250 kHz
  TIMSK4=(1<<TOIE4);
  TC4H = 0; OCR4C = 250; // TOP=250 => 250 kHz/250=1 kHz

  adc_init();
  sei();
  flag=0;
  while (1)
    { if (flag==N) {
        flag = 0;
        for (k=0;k<N;k++)
           {affiche(32767+(adc[k]*sinus[k]),&s[2]); fputs(s, &USBSerialStream);} // mixed ADC
        // {affiche(128+adc[k],&s[2]); fputs(s, &USBSerialStream);} // raw ADC
    }
    CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
    CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
    USB_USBTask();
    }
}
```

4. ajout d'un biais (la sortie de carte son est de valeur moyenne nulle et oscille entre valeurs positives et négatives, l'entrée de l'ADC est à valeurs positives uniquement), potentiellement amplification et adaptation d'impédance par un circuit suiveur.

5. Nous avons choisi une configuration du *timer* qui divise l'horloge qui cadence le microcontrôleur par 64, soit $16 \cdot 10^3/64 = 250$ kHz. En déclenchant la conversion chaque fois que le compteur atteint OCR4C, placer la valeur de 250 dans ce registre induit une mesure au rythme de 1 kéchantillons/seconde.
6. 0 V à V_{ref}
7. Nous vérifions que par tracé de la transformée de Fourier sur N points du signal acquis, le pic se trouve à l'abscisse $N/10$, en accord avec nos attentes (Fig. 2).

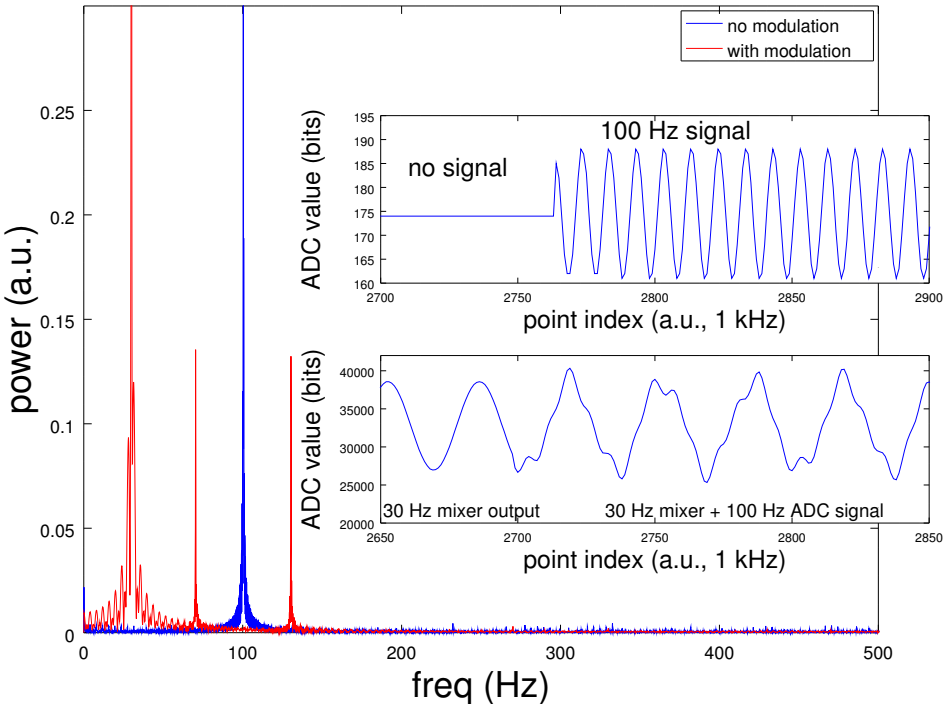

FIGURE 2 – Spectre et, en insert, mesures temporelles de signaux acquis par ADC. En haut une mesure du signal brut à 100 Hz au rythme de 1 kéchantillons/s, et en bas avec mélange par un oscillateur local numérique à 30 Hz. Dans les deux cas, le signal issu de la carte son démarre environ 50 points après l'origine du graphique.

8. dans le programme proposé auparavant, le calcul de l'oscillateur local est fait dans le gestionnaire d'interruption, au moyen de la bibliothèque mathématique `libm`. Ainsi, l'option `-lm` est ajoutée lors de la compilation du programme (bien que `gcc` sache se lier avec la bibliothèque implicitement).
9. $d\varphi = 2\pi \cdot f/fe$
10. avec un code d'environ 9,5 koctets, l'utilisation de la bibliothèque mathématique reste accessible même pour l'Atmega32U4. La taille du code croît sensiblement par rapport aux 8,3 koctets sans appel aux fonctions associées aux flottants, malgré un binaire déjà volumineux à cause de la structure `FILE` nécessaire pour communiquer par USB au travers de LUFA.
11. multiplication terme à terme : $sortie_n = VCO_n \times signal_n$ avec VCO_n incrémenté de $d\varphi$ à chaque pas de temps d'échantillonnage.
12. la somme des fréquences n'est pas favorable à la démodulation : nous ne voulons garder que la différence des fréquences. La façon la plus simple de fabriquer ce filtre passe bas est par moyenne glissante.
13. la PWM se comporte, après filtre passe-bas pour lisser le signal, comme un convertisseur numérique-analogique. Un tel périphérique n'est pas disponible sur Atmega32U4, d'où le passage par la PWM.