UNIVERSITÉ PAUL SABATIER

Systèmes Temps Réel

Compte Rendu de TP SUJET -

Auteurs :
David TOCAVEN
Najib FARHI
Lucien RAKOTOMALALA

Encadrant :
Hamid DEMMOU

19 décembre 2017

Table des matières

Chapitre 1

TP 1 : Initiation a un OS temps Réel basé sur Linux

1.1 Mesures sous Linux

1.1.1 Programme *carrelinux − comedi.c*

Ce premier programme est un générateur de signal carré. Il va nous permettre d'analyser les réponses temps réel de en étant basé sur Linux. Notre première analyse du programme donne :

- Fonction *Void out* : envoie un signal inverse celui qu'elle envoyait précédemment. La fréquence semble être défini ailleurs dans le programme. Le signal est envoyé vers le port 0 de la carte E/S initialisé dans la fonction *main*.
- *Carte E/S* : la librairie *Comedio* permet d'ouvrir la connexion série avec la carte d'entrée sortie puis d'initialiser le sens du port.
- *main* : Initialisation d'une structure de temps dans le main qui sera utilisé dans la boucle infinie du programme principal.

Le programme entre ensuite dans une boucle infinie dans laquelle il attend un temps *TIMER_ABSTIME* pour ensuite appeler la fonction *out*. Après, le programme calcule le *next shot*, i.e le prochain déclenchement, qu'il devra attendre lors du recommencement de la boucle.

Avec ce recueil d'informations, nous sommes capable de définir la fréquence du signal du signal carré que nous allons observer : elle est égale à 2 fois le *next shot* calculé dans la boucle infinie : ce calcul est : $next\_shot = 50000 + t$, la variable t appartient à la structure de temps et elle est défini en nanosecondes donc :

$$f = 2 \times 50000ns \Leftrightarrow f = 100\mu s \tag{1.1}$$

1.1.2 Mesures des période du signal

Pour mesurer les modifications de période, nous avons crée deux variables de type *timespec* : une qui mesure le temps précédent le sleep, une qui mesure a la fin de l'instance *while*(1). La mesure de la demi-période du signal carré δ est alors la différence entre le temps du début de la boucle et le temps en fin de boucle.

Pour permettre un affichage correct, nous avons introduit dans notre programme une fonction qui permet d'écrire dans un fichier les 5000 dernières δ mesurées et qui sera appelé dès que le signal **Ctrl+C** grâce à l'instruction :

```
signal(SIGINT, IntHandler)
```

Nous utilisons ensuite le fichier de mesure en .res pour un affiche avec un script *gnuplot*. Nous observons les résultats suivants :

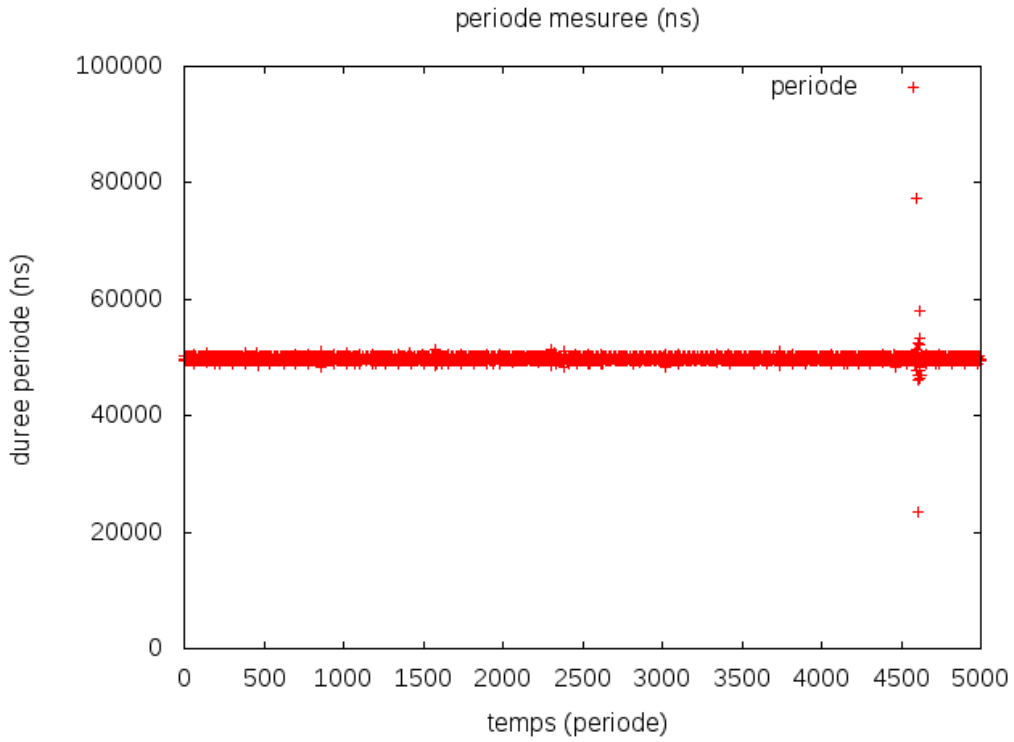

FIGURE 1.1 – Mesure des périodes sans perturbation

Pour cette première observation, nous n'avons pas touché le système d'exploitation pendant la mesure des périodes et cependant, nous remarquons que celles ci ont eu out de même quelques légères perturbations. Nous allons maintenant effectué quelques perturbations pendant que le relevé s'effectue.

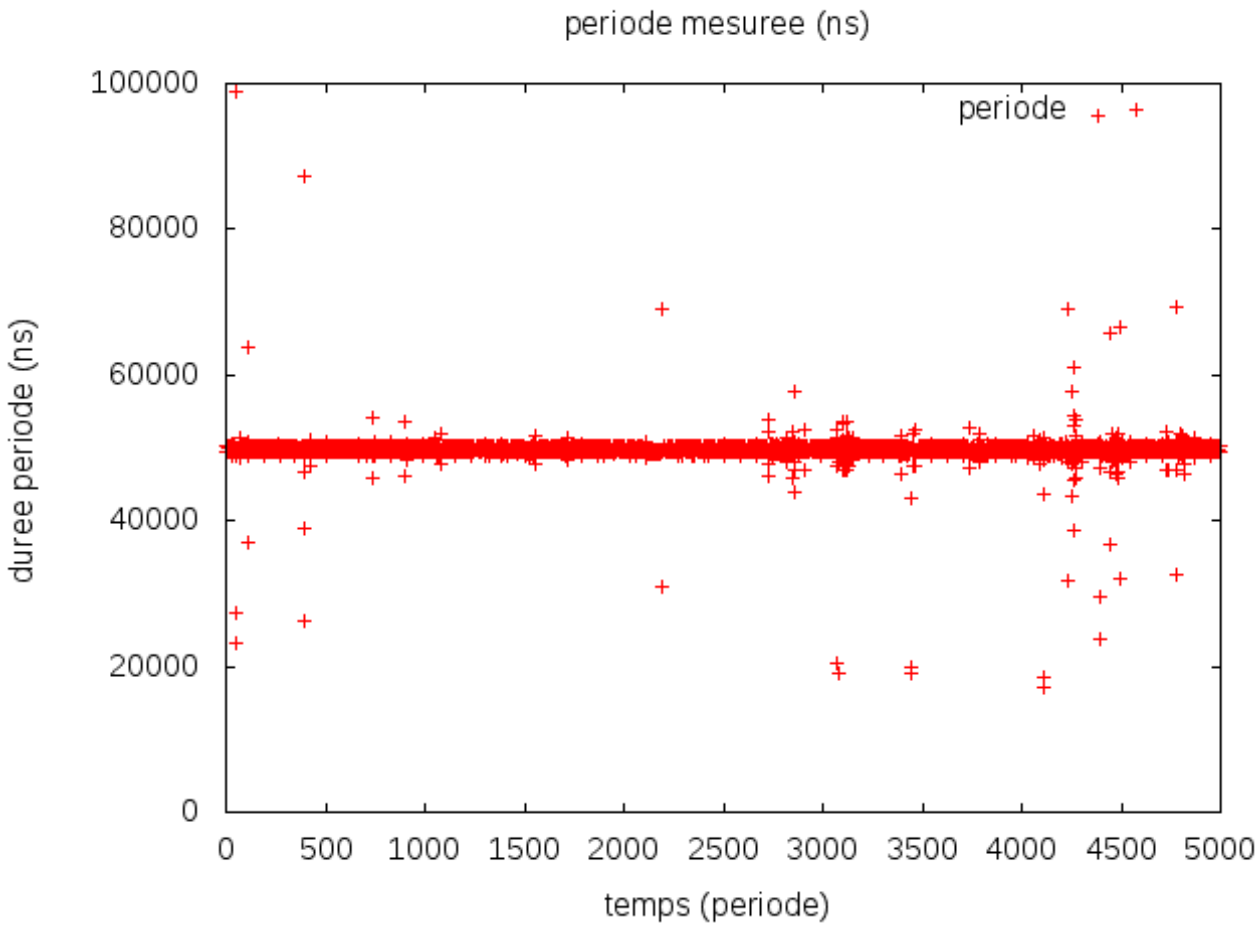

FIGURE 1.2 – Mesures des périodes avec quelques perturbations

Nous notons ici que les petites perturbations commencent à avoir de grosse conséquence sur la demi période du signal. Ces perturbations se notent aussi sur l'oscilloscope où nous observons des modifications du signal. Pour terminer cette étude, nous effectuons une dernière mesure des périodes dans laquelle nous demandons au système d'exploitation une concaténation de fichier volumineux.

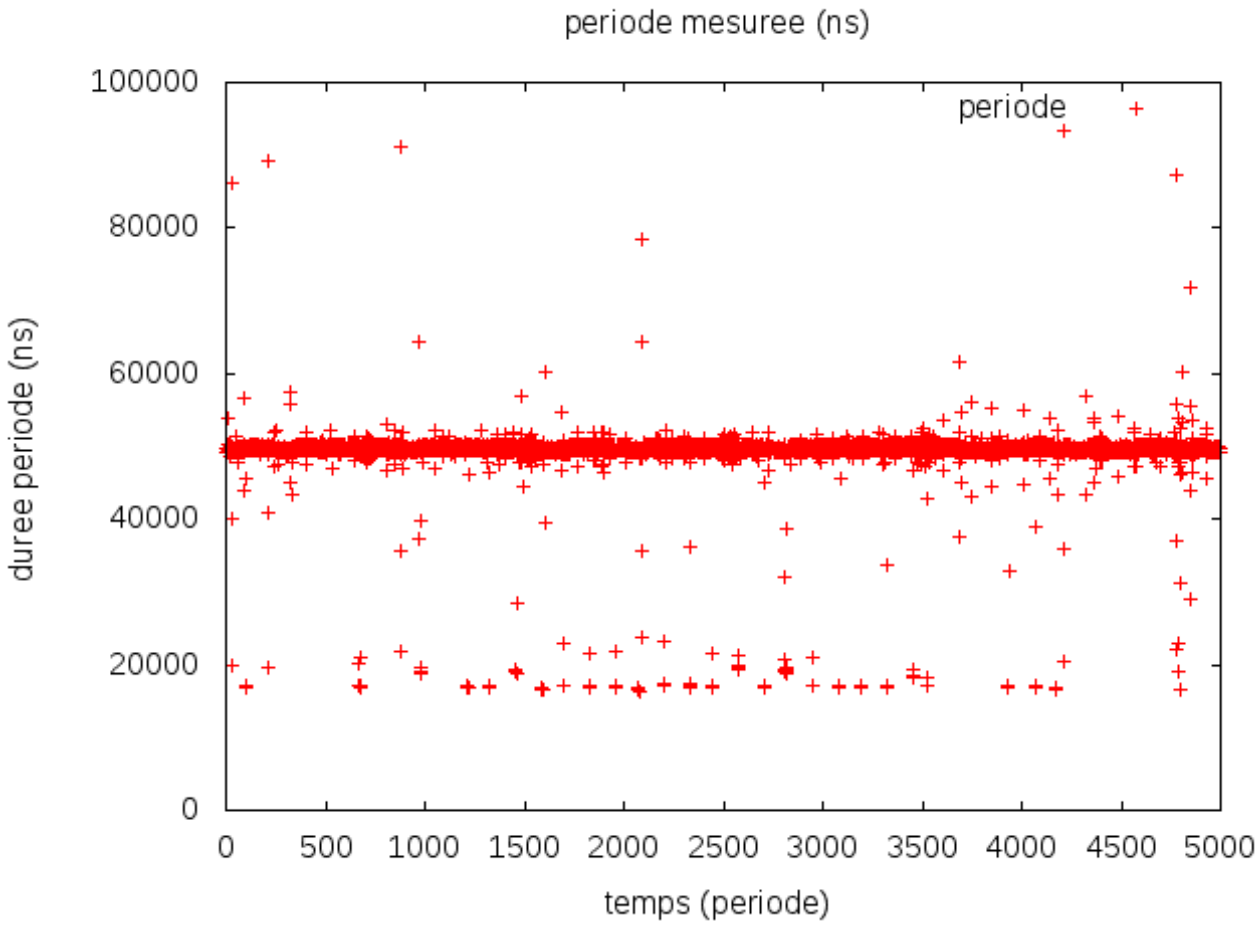

FIGURE 1.3 – Mesures des périodes avec la commande "cat /dev/zer > file2delete" en parallèle

Nous observons que la dispersion des périodes est beaucoup plus importante. Le système d'exploitation n'a pas réussi à ordonnancer notre programme avec l'instruction que nous lui avons demandé. De telles différences ne sont pas acceptables pour des applications ou des systèmes Temps Réel. Le système d'exploitation Linux seul ne peut pas être utilisé tel quel pour le Temps réel, nous devons ajouter au noyau la capacité à préempter des tâches avec l'architecture logicielle RTAI.

1.2 Mesures sous RTAI

1.2.1 L'environnement RTAI Kernel

Voir en annexe les fichiers des différents fichiers

Dans cette partie, nous avons refait la fonctionnalité précédente, c'est-à-dire un générateur de signal carré à fréquence fixe de 100 microsecondes, sous la forme d'un processus temps réel.

Dans un premier temps, nous avons modifié les fichiers *go*, *stop* et *Makefile* afin qu'ils correspondent avec le nom du fichier a compiler. En plus de cela, dans le *Makefile*, nous avons ajouté une règle *clean* afin de détruire les fichiers objets de la précédente compilation afin d'être sur que les compilations soient bien effectuées (le décalage temporel des ordinateurs causant parfois problèmes en salle de TP). Nous avons aussi, dans la règle *default*, ajouté une commande afin de rendre exécutable l'exécutable généré.

Dans un second temps, nous avons complété le processus RTAI (fichier *squelet.c*).

Nous avons complété la fonction *init_module* afin qu'elle crée et initialise la fonction *Tache1*. Cette configuration fait d'elle une tâche périodique, de fréquence $50\mu s$. De cette façon, la *Tache1* est appelée toute les demi-périodes du signal à générer. Nous avons aussi créé un compteur *now* qui permet relever le temps présent et de lancer la tâche *Tache1* en même temps que celle-ci est rendue périodique. Il y est également effectué l'ouverture de la carte d'entrée/sortie et l'initialisation d'un timer nécessaire à la mesure du temps dans la tâche *Tache1*.

La tâche *Tache1*, sert a générer le signal carré et a générer la mesure et l'écriture dans une FIFO de chaque période. Elle est organisée en deux parties :

Une initialisation où sont créé 3 entiers : *voie*, *composant* et *delta_i* servants respectivement a désigner le numéro de la voie du FIFO où nous écrivons, a désigner le numéro de son composant et a stocker la valeur de la période du signal généré. Dans une seconde partie, contenue dans une boucle infinie afin que celle-ci se répète indéfiniment. Au début de cette partie, nous relevons le temps présent grâce au timer global décrit précédemment. Il est stocké dans *delta_i*. Ensuite, nous générons la valeur haute de sortie sur le port 0. Nous attendons la fin de la période de la tâche (la moitié de celle du signal) et ensuite nous faisons de même pour la valeur basse du signal et nous récupérons le temps courant et calculons la période du signal. La dernière étape est d'écrire dans une fifo la valeur.
Une troisième tâche est décrite dans ce fichier, *cleanup_module* qui est lancée en fin d'exécution permet d'arrêter les timers et de détruire les tâches.

Dans *writeToFile.c*, qui est lancée dans *stop*, nous lisons les 5000 valeurs de la FIFO précédente et les stockons dans un fichier *"erreur.res"*.

Nous n'avons pas réussi a lire les données dans la FIFO mais les résultats attendus sont que l'application lancée sur le noyau RTAI présente une meilleure robustesse aux actions exécutées depuis le système d'exploitation et, donc, que la période du signal généré présente une plus faible variation autour de 100 microsecondes.

Chapitre 2

Conception d'un logiciel Temps réel

Dans cette partie du rapport, nous allons vous présenter nos travaux durant la séance de TP du 18/12/2017 pendant laquelle nous avons développé un logiciel temps réel. Ce logiciel doit permettre de générer deux signaux périodiques qui peuvent varier en fréquence, en amplitude et en phase. Dans une première partie, nous allons vous présenter comment nous avons choisit de séparer notre problème en tâches temps réel, puis nous implémenterons notre solution. Enfin, nous évaluerons les performances de notre système pour déterminer les points sur lesquels notre logiciel doit être améliorer.

2.1 Position du problème

Nous avons décider de séparer notre logiciel en 2 tâches temps réel :

— *void generateur* : fonction de haute priorité qui s'occupe de générer les 2 signaux sinusoïdaux.

— *void lecture* : fonction qui va lire les valeurs en entrées (potentiomètre et numérique) pour recalculer les signaux.

Au vue de ces deux tâches, il est impératif de décider maintenant de la différence de priorité entre ces deux taches principale. Il nous semble approprier, à ce stade de l'implémentation, de donner plus de priorité à la fonction *void generateur* pour permettre un rafraichissement de la valeur envoyé sur le convertisseur numérique analogique suffisante. La fonction *void lecture* sera alors appelée moins régulièrement que la génération des signaux.

2.1.1 Génération du signal périodique

Les calculs dans un programme temps réel ne sont généralement pas souhaitables : il demande trop de temps, selon leur complexité, au processeur. Selon cette affirmation, pour générer les signaux sinusoïdaux, nous ne devons faire aucun calcul de sin dans le programme principal. Nous devons générer d'une autre manière un série de 50 valeurs comprises entre [-1 ;1] qui représente notre signal sinusoïdal qui n'a pas subit de modifications.

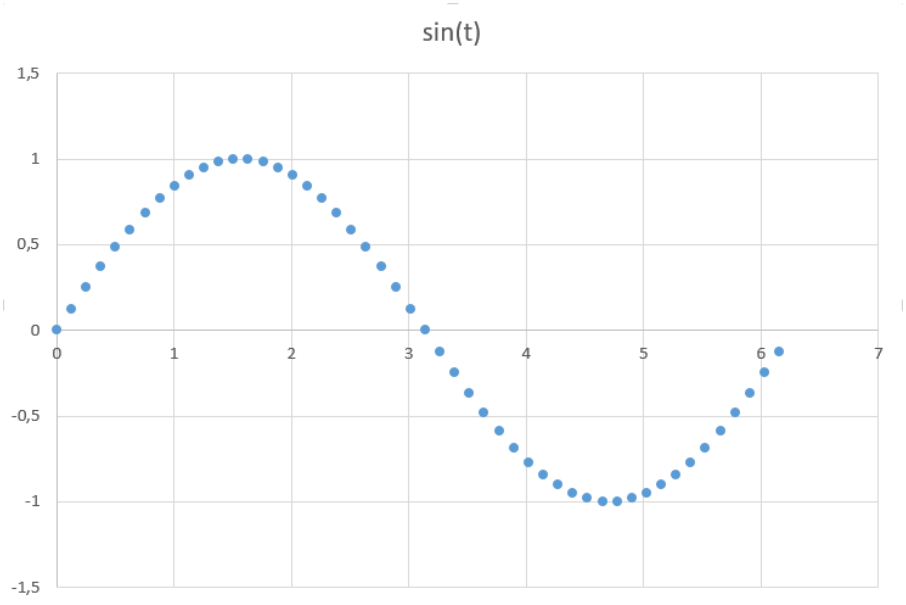

FIGURE 2.1 – Signal sinusoïdal enregistré dans le tableau

2.1.2 Conversion des valeurs analogique vers numérique

Pour pouvoir sortir un signal analogique, le programme RTAI utilise un convertisseur numérique-analogique 16 bits avec la librairie *comedio*. Il nous faut adapter les valeurs en V que nous souhaitons envoyer à ces valeurs numériques. Nous allons utiliser la loi suivante qui a été calculé en sachant que le signal peut avoir une amplitude max de [-10V ; 10V]. Avec ces informations, nous trouvons :

$$S_{data} = \frac{65535}{20} \times V + 32767$$

2.1.3 Envoie des signaux sur le CNA

L'implémentation de la tâche de haute priorité nécessite de commencer par l'initialisation RTAI de celle-ci. Nous décidons de mettre une fréquence fixe pour commencer de 1ms

```
int init_module(void)
{
 // Initialisation de la carte d'E/S
  cf = comedi_open("/dev/comedi0");
  if(cf == NULL)
  {
    comedi_perror("Comedi fails to open");
    return -1;
  }

    // OUTPUTS

  rt_set_oneshot_mode();
  // Lancement du timer

  now =  start_rt_timer(0); // creation timer de periode  50ms
  now = rt_get_time();
    // Lancement des taches
      // lecture


     // generateur
  rt_task_init( &generateur_ptr,  /* Le prt de tache */
          generateur1,       /* Nom de la tache */
          1,     /* valeur du parametre X */
          2000,    /* Taille de la pile necessaire pour la tache (memoire tempo
              utilisee) */
          0,     /* Priorite */
          1,       /* Calcul flottant */
          0);      /* choix d'un signal ou non  */


  rt_task_make_periodic(  &generateur_ptr,  /* Pointeur vers la tache */
        now,     /* Instant de depart */
        nano2count(s1_period*ms/SIZETAB));     /* Periode */

  return 0;
}
```

La tâche de génération de signaux consiste à exécuter dans une boucle infinie la lecture de la case du tableau des valeurs possibles du sinus, convertir cette valeur pour le CNA de 16bits et l'envoyer avec *comedi_data_write*. Cette boucle attend ensuite sa prochaine ré-activation pour recommencer ces opérations en décalant sa lecture des données générés dans 2.1.1.

```
void generateur1 (long int x)
{
```

```
    unsigned int  data; //[0 ;2^32 -1]

    while (1)//loop
    {
        data = (unsigned int)(32767.5 + 65535/20*(s1_alpha*signal_sin[s1_i])); //
            A*sin(2pif)
        comedi_data_write(cf, CNA, CHAN_0, RANGE, AREF, data);
        s1_i = (s1_i+1)%SIZETAB;                    // Passer toute les cases du
            tableau
        rt_task_wait_period();                      // Jusqu'a la prochaine periode
    }
}
```

2.1.4 Observation des résultats

Nous observons deux signaux échantillonnés sur l'écran de notre oscilloscope qui sont similaires. Il semble que l'implémentation du reste de l'application soit possible.

2.2 Implémentation de la modification des signaux

2.2.1 Division de la tâches *void generateur*

Il est nécessaire de modifier le code précédent pour respecter les contraintes générales. Celles-ci nous demandent de donner la possibilité de modifier la fréquence des signaux qui dépend elle-même de la fréquence de l'unique tâche *generateur*. Pour modifier la fréquence d'un seul signal de manière indépendante, il est donc nécessaire de séparer a tâche *generateur* en deux taches temps réel *genrateur1* et *genrateur2*.

Pour l'initialisation de ces tâches, nous avons séparé la fonction d'initialisation présenté dans la partie précédente. Nous avons cependant remarqué un problème de superposition des tâches, car les tâches doivent avoir un décalage dans leurs lancement sinon la génération des 2 signaux ne se superpose pas correctement.

2.2.2 Tache de lecture des informations

Maintenant que les signaux sont indépendants, nous pouvons commencer l'implémentation de la tâche moins récurrente : *void lecture*. Cette fonction va utiliser plusieurs types de signaux :

a : valeur binaire pour modifier l'amplitude.
p : valeur binaire pour modifier la phase.
f : valeur binaire pour modifier la fréquence.
n : valeur binaire pour sélectionner le signal à modifier.
a : valeur binaire pour sélectionner la sensibilité.
potar : valeur analogique du potentiomètre

Ces valeurs doivent être récupéré sur la carte E/S connecté avec la librairie *comedio*. Les variables binaire seront récupérées avec *comedi_dio_read* et les variables analogiques seront lus par *comedi_data_read*. Nous devons ensuite appliquer sur les variables analogiques une loi de conversion pour ne retenir que la valeur que nous souhaitons. Ces modifications se calculent en fonction du Convertisseur Analogique Numérique 16 bits utilisé pour convertir la tension du potentiomètre,pour une valeur 0 lu sur le CAN, nous devons obtenir 0V et pour une valeur de $2^{16} - 1 = 65535$ nous devons obtenir 10V. La loi de conversion s'écrit donc :

$$V = \frac{10}{65535} \times CAN \tag{2.1}$$

2.2.3 Application de ces informations aux signaux

Une fois les valeurs du CAN et des interrupteurs lus, nous allons l'appliquer sur les signaux en sachant que :

$$S = A\sin(2\pi f + \phi) \tag{2.2}$$

où on l'on retrouve A l'amplitude du signal que nous règlerons avec **a**, la phase ϕ qui sera réglé avec **p** et la fréquence f qui sera modifié avec **f**. Cependant, nous n'avons pas accès au calcul du sin, nous allons donc devoir procéder d'une autre manière.

Le premier paramètres que nous choisissons de modifier est l'amplitude. Cette modification est la plus simple à applquer car elle ne demande pas l'accès au calcul du sinus. Nous écrivons donc la ligne :

```
data = (unsigned int)(32767.5 + 65535/20*(s1_alpha*signal_sin[s1_i]));
```

avec la variable *s1_alpha* qui est la valeur de l'amplitude et *data* la valeur envoyé sur le CNA.

Pour avoir une influence sur la fréquence du signal généré, nous devons pouvoir modifier la fréquence à laquelle la fonction de génération des signaux est appelé. C'est possible grâce à la fonction RTAI *rt_task_make_periodic* dont nous choisirons le paramètres de *période* en fonction de la valeur récupéré sur le potentiomètre.

De même pour la modification de phase, nous devons jouer avec un paramètre qui n'est pas accessible. Nous proposons de modifier la lecture du tableau pour permettre un décalage de l'émission du signal.

Pour savoir quelle modifications doit être appliqué sur le signal, nous utilisons alors les valeurs binaire **p**, **f** et **a** reçus en début de la fonction *lecture*. Une succession de condition *if* nous permet d'appliquer la modification souhaité. Cet enchainement est ensuite en-capsulé dans une autre condition *if* qui va dépendre du bit **n**, pour choisir sur que signal les modifications seront appliquées.

2.2.4 Observation des résultats

Nous avons réussi à implémenter la première application sur le signal, la modification de l'amplitude. Les premiers test se sont retrouvé être très concluant et nous souhaitons passer à l'implémentation des autres fonction et modification. Mais le manque de temps et les problèmes RTAI nous ont retardés.

En effet, les premiers programmes que nous avons exécuté ont eu un effet radical sur le RTOS. Le programme prenait totalement la main sur le système d'exploitation en ne laissant aucun espace au système Linux qui tourne en parallèle. A partir de ce moment, il est impossible d'arrêter le programme et de faire autre chose avec l'OS. Pour corriger ce problème, il faut éteindre brutalement le RTOS pour le relancer et vérifier quel sont les problèmes du programme qui font qu'il est non préemptible et pourquoi il utilise toute les ressources du processeur.

Annexes

Annexe 1 - TP 1

Code partie 1

```
/* compile using  "gcc -o swave swave.c -lrt -Wall"  */
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <sched.h>
#include <signal.h>
#include <sys/io.h>
#include </usr/local/src/comedilib/include/comedilib.h>

#define NSEC_PER_SEC    1000000000
#define DIO 2
#define SIZETAB 5000

comedi_t *cf;

/* array who contain all outputs periodes */
   unsigned int deltat[SIZETAB];

/* the struct timespec consists of nanoseconds
 * and seconds. if the nanoseconds are getting
 * bigger than 1000000000 (= 1 second) the
 * variable containing seconds has to be
 * incremented and the nanoseconds decremented
 * by 1000000000.
 */
static inline void tsnorm(struct timespec *ts)
{
    while (ts->tv_nsec >= NSEC_PER_SEC) {
        ts->tv_nsec -= NSEC_PER_SEC;
        ts->tv_sec++;
    }
}

/* increment counter and write to parallelport */
void out()
{
    static unsigned char state=0;
    comedi_dio_write(cf,DIO,0,state);
   state=!state;
}

/* for print the datas into a file */
void IntHandler(int sig)
{
  FILE *file;
  int i;
   file = fopen("/home/m2istr_13/Documents/TP_RTAI/M2ISTR_RTAI/tp1_mesures_TR/I/
```

```
      delta.res","w");
  if(file==NULL)
  {
    fprintf(stderr,"Erreur de creation du fichier\n");
  }
  for(i=0;i<SIZETAB;i++)
  {
    fprintf(file,"%d %d\n",i,deltat[i]);
  }
  fclose(file);
  exit(0);
}

/*****************************  MAIN
   ****************************************/
int main()
{
   struct timespec t, /* for output signal */
        t1,       /* for get the time at the beginning of the loop */
          t2;          /* for get the time at the end of the loop */

   /* default interval = 50000ns = 50us
    * cycle duration = 100us
    */
   int interval=50000,
       i = 0; /* for moving into deltat */

   /*    attatch the Ctr+C action to print file  */
   signal(SIGINT,IntHandler);
   cf=comedi_open("/dev/comedi0");
   if(cf==NULL)
   {
      comedi_perror("Comedi fails to open");
      return -1;
   }

   // Configure le device ANALOG_OUTPUT pour envoyer les donnees signaux
   comedi_dio_config(cf,DIO,0,COMEDI_OUTPUT);
   comedi_dio_config(cf,DIO,1,COMEDI_OUTPUT);

   /* get current time */
   clock_gettime(0,&t);

   /* start after one second */
   t.tv_sec++;

   while(1){
      /* wait untill next shot */
      clock_gettime(0,&t1);
      clock_nanosleep(0, TIMER_ABSTIME, &t, NULL);
      /* do the stuff */
      out();
      /* calculate next shot */
      t.tv_nsec+=interval;
      tsnorm(&t);
      clock_gettime(0,&t2);
      deltat[i] = t2.tv_nsec - t1.tv_nsec + t2.tv_sec - t1.tv_sec;
      i=(i+1)%SIZETAB;
   }
   return 0;
}
```

Code Partie 2

Script du bash go

```
sudo /sbin/insmod /usr/realtime/modules/rtai_hal.ko
sudo /sbin/insmod /usr/realtime/modules/rtai_lxrt.ko
sudo /sbin/insmod /usr/realtime/modules/rtai_fifos.ko

sudo modprobe comedi
sudo modprobe kcomedilib


sudo /sbin/insmod /usr/realtime/modules/rtai_comedi.ko
sudo modprobe ni_pcimio


# Remplacer test.ko par le module compile
sudo /sbin/insmod ./squelet.ko
```

Script du bash go

```
# Remplacer test par le module compile
sudo /sbin/rmmod squelet
```

Script du bash stop

```
# Remplacer test par le module compile
sudo /sbin/rmmod squelet
```

Script du Makefile

```
# Remplacer foo par le fichier objet (.o) compile
obj-m   := tp2_gen_sign.o

KDIR    := /lib/modules/$(shell uname -r)/build
PWD             := $(shell pwd)
EXTRA_CFLAGS := -I/usr/realtime/include -I/usr/include/ -I/usr/local/include -
    D__IN_RTAI__ -ffast-math -mhard-float


default: clean
        $(MAKE) -lcomedi -lm -C $(KDIR) SUBDIRS=$(PWD) modules
        chmod +x $(obj-m)

clean :
        rm -f   $(obj-m)
```

Source de l'application RTAI

Listing 2.1 – squelet.c

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_fifos.h>
#include <rtai_proc_fs.h>
#include <comedilib.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Squelette de programme RTAI et carte ni-6221");
MODULE_AUTHOR("RAKOTOMALALA FARHI TOCAVEN");

// times unit
#define ms  1000000
#define microsec 1000
// min-max output signal values
#define CARRE_HIGH 1.22
#define CARRE_LOW 0

#define CAN 0
#define CNA 1
#define DIO 2
#define CHAN_0 0
#define CHAN_1 1
#define CAN_RANGE 1 // [-5, +5]
#define PERIOD 100 // microsecond
#define SIZETAB 5000
#define deltat 0    // FIFO

static RT_TASK Tache1_Ptr; // Pointeur pour la tache 1
static RT_TASK IT_handler_Ptr; // Pointeur pour la tache de reprise de main

comedi_t *cf; // la carte
RTIME curTime; // un timer pour mesurer les dif de temps

/****************** tache1 ***************************/
void Tache1 (long int x)
{
  int voie    = x,
      composant   = DIO,
    delta_i   = 0;
  while (1)
  {
    curTime = rt_get_time(); // read beginning time
    delta_i=(int)curTime; // keep current time
    comedi_dio_write(cf,composant,voie,CARRE_HIGH);
    rt_task_wait_period();
    comedi_dio_write(cf,composant,voie,CARRE_LOW);
    rt_task_wait_period();
    curTime = rt_get_time(); // read final times
    delta_i = (int)curTime - delta_i; // calc period
    if(rtf_put(deltat,&delta_i,SIZETAB)<0)
    {
      printk("probleme ecrite fifo : /dev/rtf0\n");
    }
```

```
  }
}

/****************** init *************************/

int init_module(void)
{
  RTIME now ;
    // Initialisation de la carte d'E/S
  cf = comedi_open("/dev/comedi0");
  if(cf == NULL)
  {
    comedi_perror("Comedi fails to open");
    return -1;
  }
  // init FIFO pour mesures
  // Configurer le device DIGITAL_INPUT pour recevoir les donnees signaux
  // et DIGITAL_OUTPUT pour envoyer les donnees signaux
  comedi_dio_config(cf,DIO,0,COMEDI_OUTPUT);
    comedi_dio_config(cf,DIO,1,COMEDI_OUTPUT);
  rt_set_oneshot_mode();
  // Lancement du timer

  now =  start_rt_timer(0); // creation timer
  curTime = start_rt_timer( nano2count(0));
  now = rt_get_time();
    // Lancement des taches
  rt_task_init(&Tache1_Ptr,    /* Le prt de tache */
          Tache1,      /* Nom de la tache */
          1,     /* valeur du parametre X */
          2000,    /* Taille de la pile necessaire pour la tache (memoire tempo
              utilisee) */
          0,     /* Priorite */
          0,       /* Pas de calcul flottant */
          0);      /* choix d'un signal ou non  */


  rt_task_make_periodic( &Tache1_Ptr,  /* Pointeur vers la tache */
        now,     /* Instant de depart */
        nano2count(PERIOD*microsec/2));    /* Periode */


  rtf_create(deltat, SIZETAB);
  return 0;
}

/****************** cleanup ***************************/
void cleanup_module(void)
{
  stop_rt_timer();
  rt_task_delete(&Tache1_Ptr);
 }
```

Source du processus de récupération de données

Listing 2.2 – writeToFile.c

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#define SIZETAB 5000
int main()
{
  int fifo1;
  int i;
  int *data;//[SIZETAB];
  if(fifo1 =open("/dev/rtf0",O_RDONLY) < 0)
  {
          printf("probleme ouverte fifo : /dev/rtf0\n");
          exit(1);
  }
  read(fifo1, data, SIZETAB*sizeof(int));

  FILE *f;
  f = fopen("erreur.res","w");

  for(i=0;i<SIZETAB;i++)
  {
  printf("%d\n",i);
    //read(&fifo1,i,sizeof(int));
    fprintf(f, "%d",data[i]);
    i++;
  }
  fclose(f);
  return 0;
}
```

Annexe 2 - TP 2

Code source du TP 2 tp2_gen_sign

```
/****************** includes ********************/

#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_fifos.h>
#include <rtai_proc_fs.h>
#include <comedilib.h>


/****************** Licences ********************/
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Squelette de programme RTAI et carte ni-6221");
MODULE_AUTHOR("RAKOTOMALALA FARHI TOCAVEN");


/****************** defines ********************/
#define ms  1000000

#define microsec 1000



#define CAN 0
#define CNA 1
#define DIO 2
#define CHAN_0 0
#define CHAN_1 1

#define PORT_s 0
#define PORT_f 1
#define PORT_p 2
#define PORT_a 3
#define PORT_n 4

#define CAN_RANGE 1 // [-5, +5]
#define SIZETAB 50
#define AREF AREF_GROUND  // indique la reference de masse a utiliser
#define RANGE 0
#define LECTURE_PERIOD 500 // milisecond

/****************** Variables globales ********************/

static RT_TASK generateur1_ptr; // Pointeur pour la tache generateur1
```

```
static RT_TASK generateur2_ptr; // Pointeur pour la tache generateur2
static RT_TASK lecture_ptr; // Pointeur pour la tache lecture

static RT_TASK IT_handler_Ptr; // Pointeur pour la tache de reprise de main

RTIME now ;
comedi_t *cf; // la carte

float signal_sin[50] =
    {0,0.1253332336,0.2486898872,0.3681245527,0.4817536741,0.5877852523,0.6845471059,0.7705

0.8443279255,0.9048270525,0.9510565163,0.9822872507,0.9980267284,0.9980267284,0.9822872507

0.8443279255,0.7705132428,0.6845471059,0.5877852523,0.4817536741,0.3681245527,0.2486898872

-0.2486898872,-0.3681245527,-0.4817536741,-0.5877852523,-0.6845471059,-0.7705132428,-0.844

-0.9510565163,-0.9822872507,-0.9980267284,-0.9980267284,-0.9822872507,-0.9510565163,-0.904

-0.7705132428,-0.6845471059,-0.5877852523,-0.4817536741,-0.3681245527,-0.2486898872,-0.125


unsigned int s     = 1; // sensibilite ( s=1 : "s=0"      ; s=100 :"s=1" )
// signal 1
int    s1_i     = 0; // Indice de l'echantillon a generer.
int    s1_phi     = 0; // Dephasage du signal
float    s1_alpha   = 1; // coeficient d'amplification.
unsigned int s1_period   = 10; // periode ( s=0 :       ; s=1 : )

// signal 2
int    s2_i     = 0; // Indice de l'echantillon a generer.
int    s2_phi     = 0; // Dephasage du signal
float    s2_alpha   = 1; // coeficient d'amplification.
unsigned int s2_period   = 10; // periode ( s=0 :       ; s=1 : )
/****************** tache generateur **************************/
void generateur1 (long int x)
{
  unsigned int  data; //[0 ;2^32 -1]

  // init CHAN
  s1_i = s1_phi;

    while (1)//loop
    {
      data = (unsigned int)(32767.5 + 65535/20*(s1_alpha*signal_sin[s1_i])); //
          A*sin(2pif)

      comedi_data_write(cf, CNA, CHAN_0, RANGE, AREF, data);
      s1_i = (s1_i+1)%SIZETAB;
      rt_task_wait_period();
    }
}

/****************** tache generateur **************************/
void generateur2 (long int x)
{
  unsigned int  data; //[0 ;2^32 -1]

  // init CHAN
  s2_i = s2_phi;
    while (1)//loop
```

```
    {
      data = (unsigned int)(32767.5 + 65535/20*(s2_alpha*signal_sin[s2_i])); //
          A*sin(2pif)
      comedi_data_write(cf, CNA, CHAN_1, RANGE, AREF, data);
      s2_i = (s2_i+1)%SIZETAB;
      rt_task_wait_period();
    }
}
/****************** tache lecture ***************************/

void lecture (long int x)
{
unsigned int  s0,
    f,
    p,
    a,
    n;

 while (1)//loop
  {
  s0=0;
  f=0;
  p=0;
  a=0;
  n=0;

  // lecture digit
  comedi_dio_read(cf,DIO,PORT_n,&n); // pour n
  comedi_dio_read(cf,DIO,PORT_s,&s0); // pour s
  comedi_dio_read(cf,DIO,PORT_p,&p); // pour p
  comedi_dio_read(cf,DIO,PORT_a,&a); // pour a
  comedi_dio_read(cf,DIO,PORT_f,&f); // pour f
  // sensibilitee
  s = (s0==0)*1 + (s0==1)*100;


  //rt_printk("n%d, s%d, p%d, a%d, f%d|n",n,s,p,a,f);

    if(a==1)
    {
      comedi_data_read(cf,CAN,0,1,AREF,&a); //range = 1 : [0 ; 5]
      if(n==0)
      {
        s1_alpha=10.0/65535.0*a;
      }
      else
      {
        s2_alpha=10.0/65535.0*a;
      }
    }
    else if(p==1)
    {
      comedi_data_read(cf,CAN,0,1,AREF,&p); //range = 1 : [0 ; 5]
      if(n==0)
      {
        s1_i=((int)(s1_i+0.0007629511*p-0.0015260252))%SIZETAB;
      }
      else
      {
        s2_i=((int)(s2_i+0.0007629511*p-0.0015260252))%SIZETAB;
      }
```

```
    }



  rt_task_wait_period();
 }
}

/****************** init_module ***************************/

int init_module(void)
{
    // Initialisation de la carte d'E/S
  cf = comedi_open("/dev/comedi0");
  if(cf == NULL)
  {
    comedi_perror("Comedi fails to open");
    return -1;
  }

  // INPUTS
    // Configurer le device DIGITAL_INPUT pour recevoir les donnees signaux
    // et DIGITAL_OUTPUT pour envoyer les donnees signaux
    comedi_dio_config(cf,DIO,PORT_s,COMEDI_INPUT); // port 0 pour s
      comedi_dio_config(cf,DIO,PORT_f,COMEDI_INPUT); // port 1 pour f

      comedi_dio_config(cf,DIO,PORT_p,COMEDI_INPUT); // port 2 pour p
      comedi_dio_config(cf,DIO,PORT_a,COMEDI_INPUT); // port 3 pour a
    comedi_dio_config(cf,DIO,PORT_n,COMEDI_INPUT); // port 4 pour n
    // OUTPUTS

  rt_set_oneshot_mode();
  // Lancement du timer

  now =  start_rt_timer(0); // creation timer de periode  50ms
  now = rt_get_time();
    // Lancement des taches
      // lecture

  rt_task_make_periodic( &lecture_ptr,    /* Pointeur vers la tache */
        now,    /* Instant de depart */
        nano2count(LECTURE_PERIOD*ms));    /* Periode */


      // generateur 1
  rt_task_init( &generateur1_ptr,    /* Le prt de tache */
          generateur1,       /* Nom de la tache */
          1,    /* valeur du parametre X */
          2000,    /* Taille de la pile necessaire pour la tache (memoire tempo
              utilisee) */
          0,    /* Priorite */
          1,      /* Calcul flottant */
          0);      /* choix d'un signal ou non  */


  rt_task_make_periodic( &generateur1_ptr,    /* Pointeur vers la tache */
        now,    /* Instant de depart */
        nano2count(s1_period*ms/SIZETAB));    /* Periode */
```

```
      // generateur 2
  rt_task_init( &generateur2_ptr,    /* Le prt de tache */
          generateur2,       /* Nom de la tache */
          1,     /* valeur du parametre X */
          2000,    /* Taille de la pile necessaire pour la tache (memoire tempo
              utilisee) */
          0,     /* Priorite */
          1,       /* Calcul flottant */
          0);      /* choix d'un signal ou non  */

  now = rt_get_time(); // IMPORTANT : decalage des deux horlogues pour eviter
      perturbations sur sorties
           // (chevauchement de taches)
  rt_task_make_periodic(  &generateur2_ptr,    /* Pointeur vers la tache */
        now,     /* Instant de depart */
        nano2count(s2_period*ms/SIZETAB));     /* Periode */



  return 0;
}

/****************** cleanup ************************/
void cleanup_module(void)
{
  stop_rt_timer();
rt_task_delete(&generateur1_ptr);
rt_task_delete(&generateur2_ptr);
rt_task_delete(&lecture_ptr);
 }
```